Урок Спринт 1.3

HTTP сервер на Go

Сегодня пришло время создать свой сервер, который сможет обрабатывать запросы пользователей. Вместе с Go сделать это будет легче лёгкого!

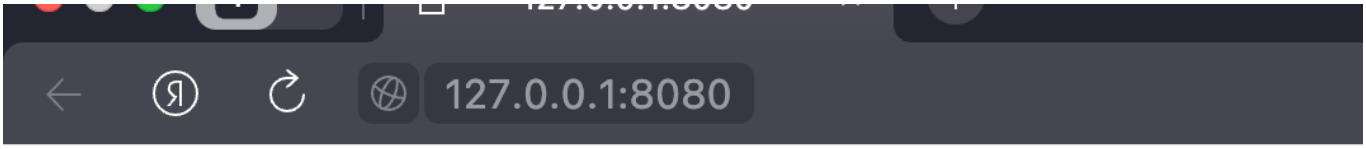
Простой веб-сервер на Go выглядит так:

```
package main

import (
    "fmt"
    "net/http"
)

func main() {

    http.HandleFunc("/", func(w http.ResponseWriter, r *http.Request) {
        fmt.Fprintf(w, "Hello World!")
    })

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

Этот код создаёт веб-сервер, который будет получать входящие запросы на порту 8080 и отправлять ответ "Hello World!" на любой запрос.

Дать запрос этому серверу можно, если перейти по **ссылке** или если использовать `curl`:

```
curl http://127.0.0.1:8080/
```




```
Hello World!
```

Обработка запросов с помощью http.Handler и http.HandleFunc

http.Handler и **http.HandleFunc** — это два способа обработки HTTP-запросов в Go.

`http.Handler` — это интерфейс, который определяет метод `ServeHTTP` и с его помощью принимает `http.ResponseWriter` и `*http.Request`. Объект, который реализует этот интерфейс, можно использовать для обработки HTTP-запросов. Например, `http.FileServer` реализует `http.Handler`, а эта функция обслуживает статические файлы из указанной директории.

`http.HandleFunc` — это функция, которая принимает путь URL и функцию-обработчика для этого пути. У функции-обработчика должна быть сигнатура `func(http.ResponseWriter, *http.Request)`. Эта функция будет вызываться для каждого запроса на указанный путь URL. Функция-обработчик может использовать `http.ResponseWriter`, чтобы записывать ответ, и `*http.Request`, чтобы получать информацию о запросе. Пример с использованием `http.Handler`:

```go
package main

import (
    "fmt"
    "net/http"
)

type MyHandler struct{}

func (h *MyHandler) ServeHTTP(w http.ResponseWriter, r *http.Request) {
    fmt.Fprint(w, "Hello, World!")
}

func main() {
    handler := &MyHandler{}
```

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

`http.HandleFunc` — это функция, которая принимает путь URL и функцию-обработчик для этого пути. Функция-обработчик должна иметь сигнатуру `func(http.ResponseWriter, *http.Request)`. Эта функция будет вызываться для каждого запроса на указанный путь URL. Функция-обработчик может использовать `http.ResponseWriter` для записи ответа и `*http.Request` для получения информации о запросе.

Пример с использованием `http.HandleFunc`:

```
package main

import (
    "fmt"
    "net/http"
)

func helloHandler(w http.ResponseWriter, r *http.Request) {
    fmt.Fprint(w, "Hello, World!")
}

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

Маршрутизация запросов с использованием `http.ServeMux`

`http.ServeMux` — это механизм маршрутизации. Он определяет, какие обработчики будут вызываться для каждого URL-адреса на сервере. Такой механизм полезен разработчикам веб-приложений, которые должны обрабатывать множество запросов на разные URL-адреса. Например:

```
type apiHandler struct{}

func (apiHandler) ServeHTTP(http.ResponseWriter, *http.Request) {}

func main() {
    mux := http.NewServeMux()
    mux.Handle("/api/", apiHandler{})
    mux.HandleFunc("/", func(w http.ResponseWriter, req *http.Request) {
        if req.URL.Path != "/" {
            http.NotFound(w, req)
            return
        }
        fmt.Fprintf(w, "Welcome to the home page!")
    })
    // ...
}
```

Так вы можете задать несколько URL префиксов для разных адресов.

Справка