**Обслуживание статических сайтов с помощью Go**

**Последнее обновление:** 29 февраля 2020 г.

Недавно я переместил сайт, который вы сейчас читаете, из приложения Sinatra/Ruby на (почти) статический сайт, обслуживаемый Go. Итак, пока это свежо в моей голове, вот объяснение принципов создания и обслуживания статических сайтов с помощью Go.

Давайте начнем с простого, но реального примера: обслуживание стандартных файлов HTML и CSS из определенного места на диске.

Начните с создания каталога для хранения проекта:

$ mkdir static-site

$ cd static-site

Затем добавьте main.goфайл для хранения нашего кода и несколько простых файлов HTML и CSS в staticкаталоге.

$ touch main.go

$ mkdir -p static/stylesheets

$ touch static/example.html static/stylesheets/main.css

Файл: static/example.html

<!doctype html>

<html>

<head>

<meta charset="utf-8">

<title>A static page</title>

<link rel="stylesheet" href="/stylesheets/main.css">

</head>

<body>

<h1>Hello from a static page</h1>

</body>

</html>

Файл: static/stylesheets/main.css

body {color: #c0392b}

Как только эти файлы созданы, код, который нам нужен для запуска, становится удивительно компактным:

Файл: main.go

package main

import (

"log"

"net/http"

)

func main() {

fs := http.FileServer(http.Dir("./static"))

http.Handle("/", fs)

log.Println("Listening on :3000...")

err := http.ListenAndServe(":3000", nil)

if err != nil {

log.Fatal(err)

}

}

Давайте пройдем через это.

Сначала мы используем эту [http.FileServer()](https://golang.org/pkg/net/http/" \l "FileServer)функцию для создания обработчика, который отвечает на все HTTP-запросы содержимым данной *файловой системы* . Для нашей файловой системы мы используем staticкаталог, относящийся к нашему приложению, но вы можете использовать любой другой каталог на вашем компьютере (или любой объект, реализующий [http.FileSystem](https://golang.org/pkg/net/http/" \l "FileSystem)интерфейс). Затем мы используем [http.Handle()](https://golang.org/pkg/net/http/" \l "Handle)функцию для регистрации файлового сервера в качестве обработчика всех запросов и запускаем сервер, прослушивающий порт 3000.

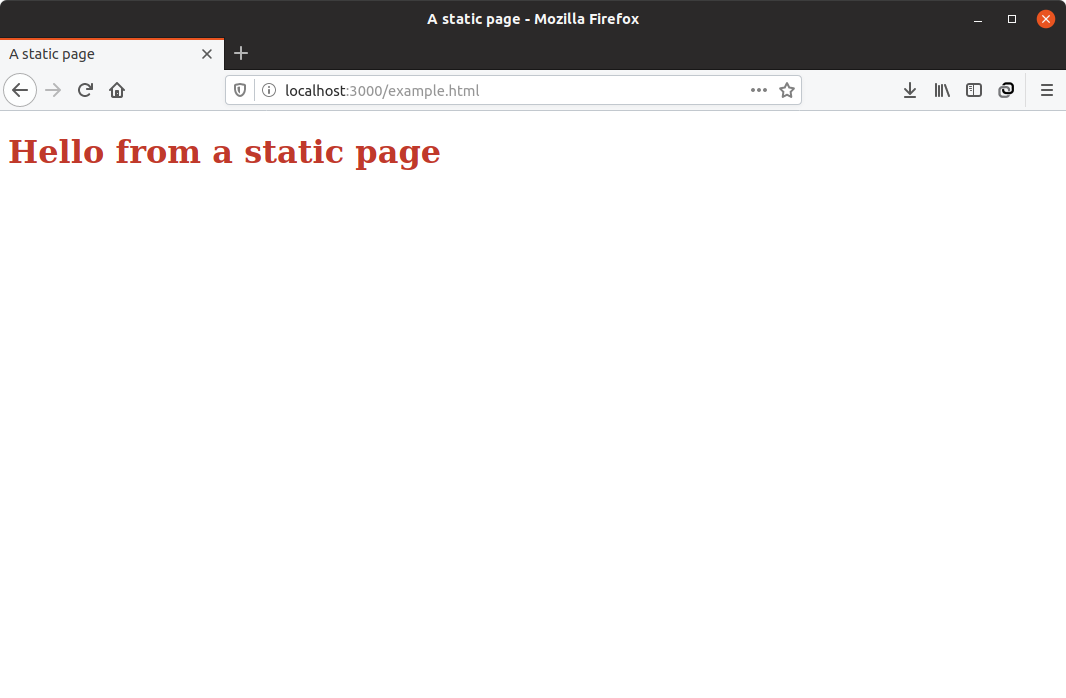
Стоит отметить, что в Go шаблон "/"соответствует всем путям запроса, а не только пустому пути.

Идем дальше и запускаем приложение:

$ go run main.go

Listening on :3000...

И открыть <http://localhost:3000/example.html>в браузере. Вы должны увидеть созданную нами HTML-страницу с большим красным заголовком.



**Почти статические сайты**

Если вы создаете много статических HTML-файлов вручную, повторение шаблонного содержимого может оказаться утомительным. Давайте рассмотрим использование [html/template](https://golang.org/pkg/html/template/)пакета Go для размещения общей разметки в файле *макета .*

На данный момент *все* запросы обрабатываются нашим файловым сервером. Давайте немного изменим наше приложение, чтобы файловый сервер обрабатывал только те пути запросов, которые начинаются с шаблона /static/.

Файл: main.go

...

func main() {

fs := http.FileServer(http.Dir("./static"))

http.Handle("/static/", http.StripPrefix("/static/", fs))

log.Println("Listening on :3000...")

err := http.ListenAndServe(":3000", nil)

if err != nil {

log.Fatal(err)

}

}

Обратите внимание, что, поскольку наш staticкаталог установлен как корень файловой системы, нам нужно удалить /static/префикс из пути запроса *перед* поиском в файловой системе для данного файла. Делаем это с помощью [http.StripPrefix()](https://golang.org/pkg/net/http/" \l "StripPrefix)функции.

Если вы перезапустите приложение, вы должны найти файл CSS, который мы сделали ранее доступным по адресу <http://localhost:3000/static/stylesheets/main.css>.

Теперь давайте создадим templatesкаталог, содержащий layout.htmlфайл с общей разметкой и example.htmlфайл с некоторым содержимым страницы.

$ mkdir templates

$ touch templates/layout.html templates/example.html

Файл: templates/layout.html

{{define "layout"}}

<!doctype html>

<html>

<head>

<meta charset="utf-8">

<title>{{template "title"}}</title>

<link rel="stylesheet" href="/static/stylesheets/main.css">

</head>

<body>

{{template "body"}}

<footer>Made with Go</footer>

</body>

</html>

{{end}}

Файл: templates/example.html

{{define "title"}}A templated page{{end}}

{{define "body"}}

<h1>Hello from a templated page</h1>

{{end}}

Если вы раньше использовали шаблоны в других веб-фреймворках или языках, надеюсь, это должно показаться вам знакомым.

Шаблоны Go — в том виде, в котором мы их здесь используем — по сути, это просто именованные текстовые блоки, окруженные тегами {{define}}и . {{end}}Шаблоны можно встраивать друг в друга с помощью {{template}}тега, как мы делали выше, где layoutшаблон встраивает как шаблоны, так titleи body.

Давайте обновим код приложения, чтобы использовать их:

Файл: main.go

package main

import (

"html/template"

"log"

"net/http"

"path/filepath"

)

func main() {

fs := http.FileServer(http.Dir("./static"))

http.Handle("/static/", http.StripPrefix("/static/", fs))

http.HandleFunc("/", serveTemplate)

log.Println("Listening on :3000...")

err := http.ListenAndServe(":3000", nil)

if err != nil {

log.Fatal(err)

}

}

func serveTemplate(w http.ResponseWriter, r \*http.Request) {

lp := filepath.Join("templates", "layout.html")

fp := filepath.Join("templates", filepath.Clean(r.URL.Path))

tmpl, \_ := template.ParseFiles(lp, fp)

tmpl.ExecuteTemplate(w, "layout", nil)

}

Так что же здесь изменилось?

Сначала мы добавили пакеты html/templateи pathв оператор импорта.

Затем мы указали, что все запросы, *не* принятые статическим файловым сервером, должны обрабатываться с помощью новой serveTemplate()функции (если вам интересно, Go сопоставляет шаблоны на основе длины, причем более длинные шаблоны имеют приоритет над более короткими).

В serveTemplate()функции строим пути к файлу макета и файлу шаблона, соответствующему запросу. Вместо ручной конкатенации мы используем [filepath.Join()](https://golang.org/pkg/path/filepath/" \l "Join), преимущество которой заключается в объединении путей с использованием правильного разделителя для вашей ОС.

Важно отметить, что поскольку путь URL-адреса является ненадежным пользовательским вводом, мы используем [filepath.Clean()](https://golang.org/pkg/path/filepath/" \l "Clean)его для очистки пути URL-адреса перед его использованием.

(Обратите внимание, что несмотря на то, что *объединенный путь*filepath.Join() автоматически проходит через , для предотвращения атак с обходом каталога вам необходимо вручную дезинфицировать любые ненадежные входные данные *перед* их объединением.)filepath.Clean()

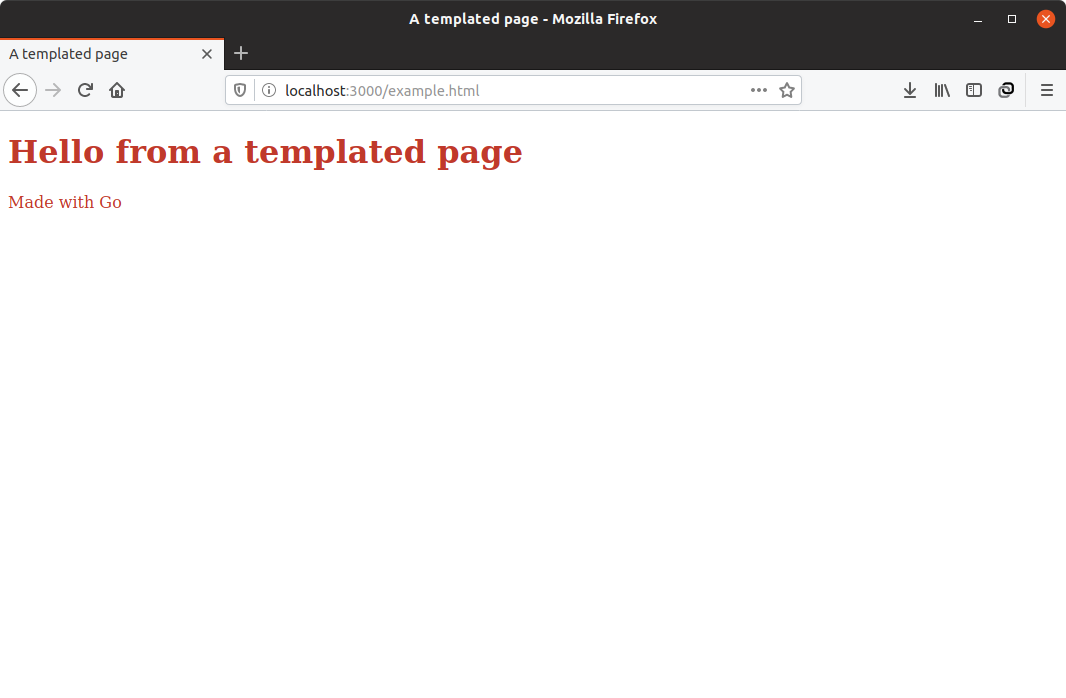
Затем мы используем эту [template.ParseFiles()](https://golang.org/pkg/text/template/" \l "Template.ParseFiles)функцию для объединения запрошенного шаблона и макета в *набор шаблонов* . Наконец, мы используем [template.ExecuteTemplate()](https://golang.org/pkg/text/template/" \l "Template.ExecuteTemplate) функцию для отображения именованного шаблона в наборе, в нашем случае layouttemplate.

Перезапустите приложение:

$ go run main.go

Listening on :3000...

И открыть <http://localhost:3000/example.html>в браузере. Вы должны увидеть разметку из всех шаблонов, объединенных вместе, например:



Если вы используете инструменты веб-разработчика для проверки ответа HTTP, вы также увидите, что Go автоматически устанавливает правильные заголовки Content-Typeи для нас.Content-Length

Наконец, давайте сделаем код немного более надежным. Мы должны:

* Отправьте 404ответ, если запрошенный шаблон не существует.
* Отправьте 404ответ, если запрошенный путь к шаблону является каталогом.
* Отправьте 500ответ, если функции template.ParseFiles()или template.ExecuteTemplate()выдают ошибку, и запишите подробное сообщение об ошибке.

Файл: main.go

package main

import (

"html/template"

"log"

"net/http"

"os"

"path/filepath"

)

func main() {

fs := http.FileServer(http.Dir("./static"))

http.Handle("/static/", http.StripPrefix("/static/", fs))

http.HandleFunc("/", serveTemplate)

log.Println("Listening on :3000...")

err := http.ListenAndServe(":3000", nil)

if err != nil {

log.Fatal(err)

}

}

func serveTemplate(w http.ResponseWriter, r \*http.Request) {

lp := filepath.Join("templates", "layout.html")

fp := filepath.Join("templates", filepath.Clean(r.URL.Path))

// Return a 404 if the template doesn't exist

info, err := os.Stat(fp)

if err != nil {

if os.IsNotExist(err) {

http.NotFound(w, r)

return

}

}

// Return a 404 if the request is for a directory

if info.IsDir() {

http.NotFound(w, r)

return

}

tmpl, err := template.ParseFiles(lp, fp)

if err != nil {

// Log the detailed error

log.Println(err.Error())

// Return a generic "Internal Server Error" message

http.Error(w, http.StatusText(500), 500)

return

}

err = tmpl.ExecuteTemplate(w, "layout", nil)

if err != nil {

log.Println(err.Error())

http.Error(w, http.StatusText(500), 500)

}

}