Веб-приложения в комбинаторном стиле, Giraffe

Карасева Елизавета Олеговна
liza.1610@mail.ru

19.05.2023

Комбина́торное программирование

- Разновидность функционального программирования
  - Без явного упоминания аргументов
  - Вместо переменных — комбинаторы и их композиция
  - Без λ-абстракций!
- Компактные программы
- Не лучшая читабельность

Немного истории

- Комбинаторная логика (1924, Шейнфинкель)
- Популяризация парадигмы (1977, Джон Бэкус)
- FP, APL, J, K
  - avg =. +/ % #
- Бесточечный стиль

Веб-программирование на F#

- Почему F#?
  - Быстрота, краткость, совместимость, кроссплатформенность, ...
- Интеграции
  - SafeStack, WebSharper, Fable
- Веб-фреймворки
  - Suave, Bolero, Falco, ServiceStack
  - Saturn -> Giraffe -> ASP.NET Core
- Веб и юнит тестирование
  - Canopy, Expecto, FsCheck

ASP.NET Core и Giraffe

- ASP.NET Core
  - Создание и развертывание веб-приложений
  - Интеграция с платформами и библиотеками
  - Работает на .NET Core, F# ”из коробки”
  - Поддержка Microsoft и сообществом разработчиков

- Giraffe
  - Библиотека F#
  - Похож на Suave
  - Компактное дополнение ASP.NET Core
  - Разработка расширенных веб-приложений в функциональном подходе
  - Создание и повторное использование блоков ASP.NET Core

Немного подробностей

- HttpHandler
  - type HttpFuncResult = Task<HttpContext option>
    type HttpFunc = HttpContext -> HttpFuncResult
    type HttpHandler = HttpFunc -> HttpContext -> HttpFuncResult
- Конвейер Giraffe — дополнение к конвейеру ASP.NET Core
- Комбинаторы
  - compose (>=>)
  - choose [
    route ”/foo”>=> text ”Foo”
    route ”/bar”>=> text ”Bar”
    ]

Немного подробностей x2

- warbler
  - let app = GET >=> path ”/”>=> OK (string DateTime.Now)
  - let app = GET >=> path ”/”
    >=> warbler (fun _ -> OK (string DateTime.Now))
- Task и Task<’T>
- Логирование
- Обработка ошибок
  - type ErrorHandler = exn -> ILogger -> HttpHandler

HTTP Verbs

```
let submitFooHandler : HttpHandler =
    setStatusCode 200 >=> text "Hello World"

let submitBarHandler : HttpHandler =
    // Do something

let webApp =
    choose [
        // Filters for GET requests
        GET  >=> choose [
            route "/foo" >=> text "Foo"
            route "/bar" >=> text "Bar"
        ]
        // Filters for POST requests
        POST >=> choose [
            route "/foo" >=> submitFooHandler
            route "/bar" >=> submitBarHandler
        ]
        // If the HTTP verb or the route didn't match return a 404
        RequestErrors.NOT_FOUND "Not Found"
    ]

let someHttpHandler : HttpHandler =
    fun (next : HttpFunc) (ctx : HttpContext) ->
        if HttpMethods.IsPut ctx.Request.Method then
            // Do something
        else
            // Do something else
        // Return a Task<HttpContext option>
```

Немного подробностей x3

- route, routeCi, routex, routeBind, routeStartsWith, subRoute, ...
- Model Binding & Response Writing
  - JSON, XML, HTML, Forms, Query Strings, ...
- Загрузка и чтение файлов
- Аутентификация и авторизация
- If-Match, If-Modified-Since

Тестирование

- Необходимый импорт
  - open Microsoft.AspNetCore.Builder
    open Microsoft.AspNetCore.TestHost
    open Microsoft.AspNetCore.Hosting
    open System.Net.Http
- Создаем тестовый хост
- Создаем вспомогательную функцию для отправки тестовых запросов
- Пишем тесты

Giraffe View Engine

- Веб-представления на основе HTML или XML
- Автоматическая компиляция во время сборки
- На F# и для F#
- html [атрибуты] [данные]
- Обработчики событий

View

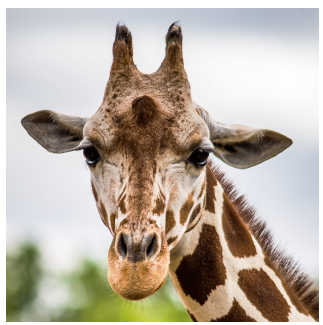
```
<html>
    <head>
        <title>Giraffe Sample</title>
    </head>
    <body>
        <h1>I |> F#<\h1>
        <p class="some-css-class", id="someId">Hello World<\p>
    </body>
</html>
```

Giraffe View Engine format

```
let indexView =
    html [] [
        head [] [
            title [] [ str "Giraffe Sample" ]
        ]
        body [] [
            h1 [] [ str "I |> F#" ]
            p [ _class "some-css-class"; _id "someId" ] [
                str "Hello World"
            ]
        ]
    ]
```

Building first app

- dotnet new console -lang F#
- dotnet add package Giraffe
- dotnet add package Giraffe.ViewEngine
- clck.ru/34RLd9
- dotnet run
- https://localhost:5000



Подробнее можно прочитать тут

- Комбинаторный стиль
  - https://en.wikipedia.org/wiki/Function-level_programming
  - https://dic.academic.ru/dic.nsf/ruwiki/1365190
- Веб-приложения на F#
  - https://fsharp.org/guides/web/
- Giraffe
  - Документация по ASP.NET Core
  - Документация по Giraffe
  - Документация по Giraffe.ViewEngine
  - Автор решил удалить курс именно 18.05, но пока материалы доступны в архиве