Делаем свою жизнь проще с Servant

Denis Redozubov, @rufuse

December 6, 2015

Слайды и код

https://github.com/dredozubov/hello-servant/tree/ruhaskell

Минимальный сервер

```
type API = "42" :> Get '[JSON] Int

apiHandler = return 42

api :: Proxy API
api = Proxy

main = run 8080 (serve api apiHandler)
```

Почему Servant?

- четкое распределение ролей
- type safety
- Способ объединить API для серверов, клиентов, документации

Как мы используем servant

- http сервера (1)
- Генерация мок-серверов для теста соединения между сервером и клиентом
- Сгенерированные клиенты
- (1) возможно использовать как subsite в др. фреймворках(e.g. Yesod)
- (+ Мы любим type safety)

Что такое описание API?

```
-- получить все заказы
"/order" - GET
-- получить один заказ
"/order/:order_id/" - GET
-- добавить один заказ
"/order" - PUT / Request 'application/json'
 with Order object(see schema)
-- добавить платеж к заказу
"/order/:order_id/payment" - GET
```

API можно описать с помощью языка программирования

```
api = Get "order" orders
  <|> Put "order" order
  <|> Get "order/:id" order
  <|> Get "order/:id/payment" payments
```

Отступление 1: Моноиды

```
-- Laws:
-- a <> mempty = a
-- mempty <> a = a
-- a <> (b <> c) = (a <> b) <> c
class Monoid a where
  mempty        :: a
  a <> b :: a -> a -> a

instance [] a where
  mempty  = []
  (<>) = (++)
```

Отступление 2: Alternative

```
-- Laws:
-- a <|> empty = a
-- empty <|> a = a
-- a <|> (b <|> c) = (a <|> b) <|> c
class Applicative f => Alternative a where
  empty   :: a
  x <|> y :: a -> a -> a

data Maybe a = Just a | Nothing

instance Alternative Maybe where
  empty = Nothing
  Nothing <|> r = r
  l       <|> _ = l
```

API возможно комбинировать из частей

```
getOrderAPI = Get "order"
  `respondsWith` (jsonOf orders)

addOrderAPI = Put "order"
  `takes` (jsonOf order)
  `respondsWith` (jsonOf id)

-- это может быть частью большего API или роутера!
orderAPI = getOrderAPI <|> addOrderAPI
```

API это тип в servant

```
type API = "order" :> Get '[JSON] [Order]
      :<|> "order" :> Capture "order_id" Int
   :> Get '[JSON] Order
      :<|> "order" :> ReqBody '[JSON] Order
   :> Put '[JSON] Int
      :<|> "order"
:> Capture "from_params"
:> QueryParam "first_name" FirstName
:> QueryParam "last_name" LastName
-- ...
-- Остаток параметров опущен
:> Put '[JSON] Int
```

Сервер наследует сигнатуры хендлеров из API

```
-- we have only types here
getOrders :: Server [Order]

getOrder :: Int -> Server Order

addOrder :: Order -> Server Int

addOrderFromParams :: FirstName
    -> LastName
    -> ...
    ...
    -> Server Int
```

Что мы можем "извлечь" из API?

- хендлеры для серверов
- полный haskell-клиент
- тонкие js/ruby/etc клиенты
- mock-сервера

servant-0.5

- auth комбинаторы - basic auth и JWT support
- улучшенные роутеры с ‘Delayed‘ проверками и пр.
- servant-foreign - универсальный бэкенд для генерации кода на других языках
- обязательные query params

Варианты интеграций

- API Blueprint
- Swagger
- JSON Schema

Проекты стоящие упоминания

- servant-swagger
- verdict

План действий

- обзор backend
- определяем API
- конструируем CRUD server
- "выводим" клиент на haskell
- генерация js-клиента
- реализуем mock server

Implementation

Статья про имплементацию servant:
http://www.well-typed.com/blog/2015/11/implementing-a-minimal-version-of-haskell-servant/

contacts

- `http://twitter.com/rufuse` (@rufuse)
- `http://bananasandlenses.net` Подкаст "Бананы и линзы"