Making web services better with servant

Denis Redozubov, @rufuse

December 6, 2015

Slides & code

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

Server Example

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

apiHandler = return 42

fapi :: Proxy API
fapi = Proxy

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

Why Servant

- True Separations of concerns
- Type Safety
- Unified way of bringing API to server, clients, documentation

How we use servant

- Servant servers (1)
- Generate mock servers to test client-server connectivity
- Generated clients
- We love type safety
- (1) is possible to use in a Yesod subsite, we do that

What is an API description?

```
-- get all orders
"/order" - GET
-- get one order
"/order/:order_id/" - GET
-- add new order
"/order" - PUT / Request 'application/json'
 with Order object(see schema)
-- add payment to the order
"/order/:order_id/payment" - GET
```

APIs can be constructed with programming languages

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

Monoids side-story

```
-- 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  = []
  (<>) = (++)
```

Alternative side-story

```
-- 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 can be created of smaller parts

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

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

-- It may be a part of bigger API or a router!
orderAPI = getOrderAPI <|> addOrderAPI
```

API is a type in 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
-- the rest of the stuff we need
-- to construct the order could be here
:> Put '[JSON] Int
```

Servant servers infer handler types from the API

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

getOrder :: Int -> Server Order

addOrder :: Order -> Server Int

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

What can we infer from the API type

- server handlers
- full haskell client
- thin js/ruby/etc clients
- mock servers

servant-0.5

- auth combinators - basic auth and JWT support
- improved router with ‘Delayed‘ check etc
- servant-foreign as a universal backend to codegen libraries
- mandatory query params

Possible integrations

- API Blueprint
- Swagger
- JSON Schema validations

Notable projects

- servant-swagger
- verdict

Plan of action

- review a backend
- define an API
- construct a CRUD server
- derive a haskell client
- generate a js client
- implement a mock server

Implementation

An article on implementing servant:
`http://www.well-typed.com/blog/2015/11/implementing-a-minimal-version-of-haskell-servant/`

contacts

- http://twitter.com/rufuse
- http://bananasandlenses.net