Servant
A Type-Level DSL for Web APIs

Julian K. Arni

Curry On, Prague, July 6th 2015

Outline

Section 1

Rethinking Web Frameworks

The Main Idea

The Main Idea

Web API descriptions should be central

The Main Idea

Web API descriptions should be central

1. Accords with development practice

The Main Idea

Web API descriptions should be central

1. Accords with development practice
2. Clarifies behaviour of programs

The Main Idea

Web API descriptions should be central

1. Accords with development practice
2. Clarifies behaviour of programs
3. Is a reusable component

An Example

Counter API

```
GET  /        obtain the current value
POST /step    increment the current value
```

An Example

Counter API

```
GET  /       obtain the current value
POST /step   increment the current value
```

```
type GetCounter = Get '[JSON] CounterVal
type StepCounter = "step" :> Post '[] ()
type CounterAPI = GetCounter :<|> StepCounter
```

An Example

An Example

Counter API

```
type GetCounter = Get '[JSON] CounterVal
type StepCounter = "step" :> Post '[] ()
type CounterAPI = GetCounter :<|> StepCounter
```

An Example

Counter API

```
type GetCounter = Get '[JSON] CounterVal
type StepCounter = "step" :> Post '[] ()
type CounterAPI = GetCounter :<|> StepCounter
```

```
newtype CounterVal = CounterVal {getCounterVal :: Int}
  deriving (ToJSON)

counterAPI :: Proxy CounterAPI
counterAPI = Proxy

instance ToSample CounterVal where
  toSample _ = Just (CounterVal 42)
```

Documentation

Documentation

```haskell
counterDocs :: String
counterDocs = markdown $ docs counterAPI
```

```
## GET /

#### Response:

- Status code 200
- Headers: []

- Supported content types are:

    - ‘application/json‘

- Response body as below.

‘‘‘javascript
42
‘‘‘
```

Client Functions

Client Functions

```
getCounter' :<|> stepCounter'
 = client counterAPI ( BaseUrl Http "localhost" 8000 )
```

Client Functions

```
getCounter' :<|> stepCounter'
 = client counterAPI ( BaseUrl Http "localhost" 8000 )

getCounter' :: EitherT ServantError IO CounterVal
setCounter' :: EitherT ServantError IO ()
```

Safe links

Safe links

```
safeLink counterAPI (Proxy :: Proxy StepCounter)
-- > "step"
```

Safe links

```
safeLink counterAPI (Proxy :: Proxy StepCounter)
-- > "step"

safeLink counterAPI (Proxy :: "doesntexist" :> Get '[] ())
-- > type error
```

Server

```
getCounter :: TVar CounterVal -> Server GetCounter
getCounter ctr = liftIO $ readTVarIO ctr

stepCounter :: TVar CounterVal -> Server StepCounter
stepCounter ctr
  = liftIO $ atomically $ modifyTVar ctr (+ 1)
```

Server

```
getCounter :: TVar CounterVal -> Server GetCounter
getCounter ctr = liftIO $ readTVarIO ctr

stepCounter :: TVar CounterVal -> Server StepCounter
stepCounter ctr
  = liftIO $ atomically $ modifyTVar ctr (+ 1)

server :: TVar CounterVal -> Server CounterAPI
server ctr = getCounter ctr
       :<|>  stepCounter ctr
```

Server

```haskell
getCounter :: TVar CounterVal -> Server GetCounter
getCounter ctr = liftIO $ readTVarIO ctr

stepCounter :: TVar CounterVal -> Server StepCounter
stepCounter ctr
  = liftIO $ atomically $ modifyTVar ctr (+ 1)

server :: TVar CounterVal -> Server CounterAPI
server ctr = getCounter ctr
       :<|>  stepCounter ctr

main = do
  initCtr <- newTVar 0
  run 8000 (serve counterAPI $ server initCtr)
```

Section 2

Type-level DSLs

A Simple DSL

```
Add (Add One One) One
```

A Simple DSL

```
Add (Add One One) One
```

```
data Expr = Add Expr Expr
          | One

eval :: Expr -> Int
eval One = 1
eval (Add x y) = eval x + eval y
```

A Simple DSL

```
Add (Add One One) One
```

A Simple DSL

```
Add (Add One One) One
```

```
class Eval x where
    eval :: Proxy x -> Int
```

A Simple DSL

```
Add (Add One One) One
```

```
class Eval x where
    eval :: Proxy x -> Int

data One
data Add a b
instance Eval One where
    eval _ = 1
instance (Eval a, Eval b) => Eval (Add a b) where
    eval _ = eval (Proxy :: Proxy a)
          +  eval (Proxy :: Proxy b)
```

Type-level Advantages

1. Extensible

Type-level Advantages

1. Extensible
2. Right side of phase distinction

A Fancier DSL

A Fancier DSL

```
data Hole
```

A Fancier DSL

```
data Hole

class Eval' a where
    type Value a r :: *
    eval' :: Proxy a -> (Int -> r) -> Value a r
```

A Fancier DSL

```
data Hole

class Eval' a where
    type Value a r :: *
    eval' :: Proxy a -> (Int -> r) -> Value a r

instance Eval' One where
    type Value One r = r
    eval' _ ret = ret 1
```

A Fancier DSL

```
data Hole

class Eval' a where
    type Value a r :: *
    eval' :: Proxy a -> (Int -> r) -> Value a r

instance Eval' One where
    type Value One r = r
    eval' _ ret = ret 1

instance (Eval' a, Eval' b) => Eval' (Add a b) where
    type Value (Add a b) r = Value a (Value b r)
    eval' _ ret = eval' (Proxy :: Proxy a) (\v1 ->
                  eval' (Proxy :: Proxy b) (\v2 ->
                  ret (v1 + v2)))
```

A Fancier DSL

```
data Hole

class Eval' a where
    type Value a r :: *
    eval' :: Proxy a -> (Int -> r) -> Value a r

instance Eval' One where
    type Value One r = r
    eval' _ ret = ret 1

instance (Eval' a, Eval' b) => Eval' (Add a b) where
    type Value (Add a b) r = Value a (Value b r)
    eval' _ ret = eval' (Proxy :: Proxy a) (\v1 ->
                  eval' (Proxy :: Proxy b) (\v2 ->
                  ret (v1 + v2)))

instance Eval' Hole where
    type Value Hole r = Int -> r
    eval' _ ret n = ret n
```

A Fancier DSL

```
type Succ = Add One Hole

succ = eval' (Proxy :: Proxy Succ) -- :: (Int -> Int)
succ 5 -- > 6
```

A Fancier DSL

```
type Sum = Add Hole Hole

sum = eval' (Proxy :: Proxy Sum) -- :: (Int -> Int -> Int)
sum 5 10 -- > 15
```

Section 3

Servant DSL

Servant Grammar

Servant Grammar

```
api          ::=  api :<|> api
             |    item :> api
             |    method

item         ::=  symbol
             |    ReqBody      ctypes type
             |    Capture      symbol stype
             |    ...
```

```
method      ::=  Get      ctypes type
            |    Put      ctypes type
            |    Post     ctypes type
            |    Delete   ctypes type
            |    Raw
            |    ...

type        ::=  <Haskell Types>

ctypes      ::=  '[ctype, ...]

ctype       ::=  PlainText
            |    JSON
            |    HTML
            |    ...
```

HasServer

```
class HasServer api where
  type Server api :: *
  route :: Proxy api -> Server api -> RoutingApplication
```

HasServer

```
class HasServer api where
  type Server api :: *
  route :: Proxy api -> Server api -> RoutingApplication

type RoutingApplication  =
       Request
   ->  (RouteResult Response -> IO ResponseReceived)
   ->  IO ResponseReceived

data RouteResult a = NotMatched | Matched a
```

HasServer - GetText

HasServer - GetText

```
data GetText t

instance Show t => HasServer (GetText t) where
  type Server (GetText t) = EitherT ServantErr IO t

  route  ::  Proxy (GetText t) -> Server (GetText t)
         ->  RoutingApplication
  route _ handler request respond
    |   pathIsEmpty request
    &&  requestMethod request == methodGet = accept
    |   otherwise = respond NotMatched
```

HasServer - GetText

```
where
accept = do
  e <- runEitherT handler
  respond $ case e of
    Right t    ->  Matched $ responseLBS ok200
                     [("Content-Type", "text/plain")]
                     (fromString (show t))
    Left err   ->  Matched $ responseServantErr err
```

HasServer - Alternative

```
data a :<|> b = a :<|> b
infixr 8 :<|>

instance  (HasServer api1, HasServer api2) =>
          HasServer (api1 :<|> api2) where

  type  Server (api1 :<|> api2) =
          Server api1 :<|> Server api2

  route _ (handler1 :<|> handler2) request respond =
    route (Proxy :: Proxy api1) handler1 request $ \ r ->
      case r of
        Matched result  ->  respond (Matched result)
        NotMatched      ->  route (Proxy :: Proxy api2)
                              handler2 request respond
```

HasServer - Capture

HasServer - Capture

```
instance (KnownSymbol sym, FromText t, HasServer api)
      => HasServer (Capture sym t :> api) where

  type Server (Capture sym t :> api) = t -> Server api

  route _ handler request respond =
    case processedPathInfo request of
      p : ps | Just v <- (fromText p :: Maybe t)
         ->  forward ps v
      _  ->  respond NotMatched
    where
    forward ps v = route (Proxy :: Proxy api) (handler v)
                      (request { pathInfo = ps }) respond
```

References

Slides	`https://github.com/jkarni/curry-on-servant`
Type-level DSL	Lämmel, Ralf and Ostermann, Klaus, *Software Extension and Integration with Type Classes*, GPCE, 2006.
Servant WGP paper	`http://haskell-servant.github.io/posts/2015-05-25-servant-paper-wgp-2015.html`
Servant website	`http://haskell-servant.github.io/`