Type the Web with Servant!

Andres Löh

BOB, Berlin, 19 February 2016 — Copyright © 2016 Well-Typed LLP

Servant

An embedded domain-specific language for describing Web APIs in Haskell (using several modern extensions).

Created by:

- Alp Mestanogullari
- Sönke Hahn
- Julian K. Arni

Features

- Type-safe
- As little “boilerplate code” as possible
- Extensible

Features

- Type-safe
- As little “boilerplate code” as possible
- Extensible

Tools to create:

- Web servers / services
- Clients
- Client functions in other languages
- Mock servers and clients
- Type-safe links
- Documentation
- …

What is a Web API?

Describes:

- what requests are valid,
- what extra information is requested and its format, such as:
  - request body,
  - request headers,
  - parameters,
- what is returned by the request and in what format.

What is a (Haskell) type?

Describes:

- what inputs a piece of code expects,
- the format of the inputs,
- what is returned by a piece of code and in what format.

What is a (Haskell) type?

Describes:

- what inputs a piece of code expects,
- the format of the inputs,
- what is returned by a piece of code and in what format.

Conceptually, Web APIs are types.

Types vs. terms

Haskell is statically typed:

- every term is assigned a type (inference + checking),
- only if all terms have valid types, the program is executed.

Types vs. terms

Haskell is statically typed:

- every term is assigned a type (inference + checking),
- only if all terms have valid types, the program is executed.

Type errors:

- happen at compile-time,
- do not prevent, but reduce runtime errors,

An example Servant API description

Informal:

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

```
type Counter  =  Get '[JSON] Int
            :<|> "step" :> Post '[JSON] Int
```

An example Servant API description

Informal:

```
GET   /          obtain the current value
POST  /step      increment counter
POST  /step/:n   increment counter by n
```

```
type Counter  =  Get '[JSON] Int
            :<|> "step" :> Post '[JSON] Int
            :<|> "step" :> Capture "n" Int
                        :> Post '[JSON] Int
```

A type?

```
type Counter  =  Get '[JSON] Int
             :<|> "step" :> Post '[JSON] Int
             :<|> "step" :> Capture "n" Int
                         :> Post '[JSON] Int
```

- Lives on the Haskell type level.
- Does not directly have code associated with it.

A type?

```
type Counter  =  Get '[JSON] Int
            :<|> "step" :> Post '[JSON] Int
            :<|> "step" :> Capture "n" Int
                        :> Post '[JSON] Int
```

- Lives on the Haskell type level.
- Does not directly have code associated with it.
- Contains sufficient information to compute other types!

Type-level computation

Compute types from other types, all at compile time:

```
type Counter  =  Get '[JSON] Int
             :<|> "step" :> Post '[JSON] Int
             :<|> "step" :> Capture "n" Int
                         :> Post '[JSON] Int
```

Type-level computation

Compute types from other types, all at compile time:

```
type Counter  =  Get '[JSON] Int
             :<|> "step" :> Post '[JSON] Int
             :<|> "step" :> Capture "n" Int
                         :> Post '[JSON] Int
```

Server and client (simplified):

```
type Counter' = (IO Int, IO Int, Int -> IO Int)
```

Type-level computation

Compute types from other types, all at compile time:

```
type Counter  =  Get '[JSON] Int
             :<|> "step" :> Post '[JSON] Int
             :<|> "step" :> Capture "n" Int
                         :> Post '[JSON] Int
```

Server and client (simplified):

```
type Counter' = (IO Int, IO Int, Int -> IO Int)
```

Other information is used to do all the tedious work.

Describing a server

```
type Counter  =  Get '[JSON] Int
            :<|> "step" :> Post '[JSON] Int
            :<|> "step" :> Capture "n" Int
                        :> Post '[JSON] Int
```

For free:

- Route requests to the right (sub-)handler.
- Send status codes for illegal requests.
- Extract and parse request parameters (and handle errors).
- Construct the response from the Haskell value.

Need to supply:

```
type Counter' = (IO Int, IO Int, Int -> IO Int)
```

(And info about where to run the server.)

Describing a client

```
type Counter  =  Get '[JSON] Int
             :<|> "step" :> Post '[JSON] Int
             :<|> "step" :> Capture "n" Int
                         :> Post '[JSON] Int
```

Need to supply info about where the server is running.

We obtain:

```
type Counter' = (IO Int, IO Int, Int -> IO Int)
```

For free:

- Construct the right request depending on the code we use.
- Send the request to the server and obtain the response.
- Extract the result value from the response.

Generating documentation

```
type Counter  =  Get '[JSON] Int
             :<|> "step" :> Post '[JSON] Int
             :<|> "step" :> Capture "n" Int
                         :> Post '[JSON] Int
```

For free:

- What the valid requests are.
- What types the inputs and outputs have.
- Status codes.

Need to supply: Additional textual information.

Generating documentation

```
type Counter  =  Get '[JSON] Int
             :<|> "step" :> Post '[JSON] Int
             :<|> "step" :> Capture "n" Int
                         :> Post '[JSON] Int
```

For free:

- What the valid requests are.
- What types the inputs and outputs have.
- Status codes.

Need to supply: Additional textual information.

Simplified:

```
type CounterDocs = (String, String, (String, String))
```

The Servant approach

EDSL:

- Abstraction and modularity.
- Extensibility.

API types:

- Possibly different implementations of the same API.
- Compatibility of e.g. a server and a client.
- A lot of functionality for free.
- Can concentrate on writing the interesting code.
- Safety and ease of refactoring.

Types as a helpful guide

Types are not the enemy.

Types as a helpful guide

Types are not the enemy.

Types limit and guide the programming process:

- polymorphism helps us to focus on the right inputs and outputs,
- types of inputs and outputs help us to know the “shapes” they can have,
- more and more allow even interactive program development (inspired by dependently typed languages).

Types as a helpful guide

Types are not the enemy.

Types limit and guide the programming process:

- polymorphism helps us to focus on the right inputs and outputs,
- types of inputs and outputs help us to know the “shapes” they can have,
- more and more allow even interactive program development (inspired by dependently typed languages).

Servant brings this philosophy to web development as well.

The Servant API description language

Seen:

- Nesting.
- Choice.
- Strings.
- Captures.
- HTTP verbs.
- Content types.

Also:

- Request and response headers.
- Request body.
- Query parameters.
- ...

Current and future work

New in 0.5:

- Efficient routing.
- Better error handling.
- More predefined content types.

Other ongoing work:

- Authentication.
- Integration with other web frameworks.
- Client code generation for several non-Haskell languages.
- …

https://haskell-servant.github.io

Questions?

andres@well-typed.com

Interested in Haskell training?

Subscribe to our annnouncement mailing list at

http://www.well-typed.com/cgi-bin/mailman/listinfo/events

or look at

http://www.well-typed.com/services_training