Language Proposal: WASP

Get your API up and running in a few lines.

September 30th, 2015

COMS W4115 - Programming Languages and Translators

John Chung jjc2254 - The Heart (Language Guru)
Neel Vadoothker nv2264 - The Muscle (Testing)
Dustin Burge db2987- The Brains (System Architect)
Tingting Li tl2617 - The Boss (Manager)

Introduction

WASP stands for Web API Specification Protocol. It's a way of describing RESTful APIs in a way that's highly readable by both humans and computers. WASP focuses on cleanly describing resources, methods, parameters, responses and other HTTP constructs that form the basis for modern APIs that obey the RESTful constraints. It is meant to be simple enough that a single object definition is enough to create a RESTful backend for that object.

There are two primary use cases for employing WASP. The first is for creating simple RESTful APIs for CRUD (Create, Retrieve, Update, Delete) operations on custom objects. The second is for creating simple calculation based APIs (eg. days until, unit converter, etc.). It is these two use cases that inspired the language and will drive the evolution of WASP throughout its development.

Using WASP to implement a RESTful API is meant to be as simple as defining a struct in C. The details of implementing request routing, object storage, handling HTTP status codes, etc. are abstracted in favor of the quickest and simplest path to deployment. WASP will be compiled to Go because of its robust built-in networking libraries and the ease of deploying Go server applications.

Language Features

WASPs primary feature is the ability to define complex custom objects as collections of supported primitive types (int, float, boolean, string, list) and previously defined custom objects.

Endpoints can be defined to PUT/POST/GET/DELETE those objects simply by using the Endpoint keyword.

Objects that do not have an associated endpoint are defined with the Object keyword.

The endpoints created with WASP accept and return JSON formatted text in the message body.

All objects will have an implicit id of type int. This id is returned as part of the response of any HTTP request, but is not required on the POST request for a new object.

Each of these primitive variables can be declared as optional with the keyword opt. If a variable is not declared as optional and is missing in a request's JSON object, the appropriate error code will be returned.

WASP is statically typed.

Functions in WASP are defined with the Func keyword.

There is no limit to the number of parameters a function can accept, but it must return only a single value (either a primitive type or a custom object). There is no void return type, because functions are not meant to be written only for their side effects.

Depending on time constraints, we will attempt to store our data in a persistent database (eg. MongoDB, Postgres, etc). Minimally, data will be stored as lists of objects that persist only as long as the server instance.

For control flow, WASP has while loops with break statements, and standard if, else if, and else statements.

Example Programs

---

1. Hello_World.wasp: This simple example will start up a server and create an endpoint '/HelloWorld' and also post data. To test our REST API we use cURL to execute GET, and would retrieve JSON data.

```
1  Endpoint HelloWorld(GET:) {
2     string message = "Hello World."
3  }
```

```
$ curl http://localhost:5000/HelloWorld
```

```
{
    "message": "Hello World."
}
```

2. instagram.wasp: This example shows several different methods for retrieving objects via custom text queries for filtering search results. Note that WASP also allows POST requests to update the database by using JSON object definitions.

```
Object Image {
        string username
        string imageUrl
        string description
        string geo
}

func list[Image] ImageSearch(string q) {
        return Image.filter(description ~= q);
}

Endpoint TextSearch(GET: string q) {
        list[Image] images = ImageSearch(q);
}
```

```
$ curl -H "Content-Type: application/json" -X POST -d '{"username": "cooldude", "imageUrl": "https://www.instagram.com/uri=1241", "description": "an askew latte", "geo": "lat:1241 long:1512"}, {"username": "abc123", "imageUrl": "https://www.instagram.com/uri=1433", "description": "a picture of six lattes", "geo": "lat:6733 long:0989"}''
http://localhost:5000/TextSearch
```

```
$ curl http://localhost:5000/TextSearch?q=latte
```

```
{
    [
        {
            "username": "cooldude",
            "imageUrl": "https://www.instagram.com/uri=1241",
            "description": "an askew latte",
            "geo": "lat:1241 long:1512"
        },
        {
            "username": "abc123",
            "imageUrl": "https://www.instagram.com/uri=1433",
            "description": "a picture of six lattes",
            "geo": "lat:6733 long:0989"

        }
    ]
}
```

The following lines of code will create a server that accepts GET, PUT, POST, and DELETE HTTP requests at the endpoint `/Person`. City is defined with the Object keyword (as opposed to the Endpoint keyword), so it will not create a route that accepts HTTP requests.

```
Object City {
    string name;
    string state;
    int zipcode;
}

Endpoint Person {
    string name;
    int age;
    opt float gpa;
    City hometown;
}
```

A GET request at this endpoint (with a valid id) will return JSON like the following.

```
{
    name: "Billy"
    age: 29
    hometown: {
                    name: "Houston"
                    state: "Texas"
                    zipcode: 77004
                }
}
```

The following 3 lines of code will create a server that accepts only GET HTTP requests with a single argument at the endpoint `/CelsiusToFahrenheit`.

```
func float CToF(float c) {
    return c * 9.0 / 5.0 + 32.0
}

Endpoint CelsiusToFahrenheit(GET: float celsius) {
    float fahrenheit = CToF(celsius);
}
```

A GET request at the endpoint /CelsiusToFahrenheit will require a message body formatted as follows…

```
{
    "celsius": 75.3
}
```

… and will return JSON like the following.

```
{
    "fahrenheit": 167.5
}
```