Data Validation with JSON Schema

BayNode Meetup, August 4, 2015

http://www.meetup.com/BayNode/events/217469902/

Tony Pujals @subfuzion

Peter Svetlichny @p_svetlichny

JSON

JSON is a lightweight, text-based, language-independent, **data exchange format** for the portable representation of **structured data**.

Official ECMA specification describes it in precise detail, but essentially:

* easy format for machines to generate and parse
* relatively easy for humans to read and write

JSON maps well to JavaScript object notation syntax

Serialize

```
// generate JSON text from JavaScript object
JSON.stringify(obj);
```

Deserialize

```
// parse JSON text to create JavaScript object
obj = JSON.parse(text)
```

Using JSON, programmers can describe **data structures** comprised of (potentially nested) **collections** of **unordered name/value pairs** and **ordered lists of values**

```
{
    "id": 1,
    "name": "A green door",
    "price": {
      "amount": 12.50,
      "currency": "USD"
    },
    "tags": ["home", "green"]
}
```

The structure starts with the declaration of an object (contained within the outermost (first and last) left **{** and right **}** brace pairs containing a number of **unordered** members. Each member is comprised of a name/value pair, where the **name is a string** and the **value can be any of the following**:

- `null`
- `true` or `false`
- string, number, object, array

Note that while a JSON document contains data represented in a particular data structure, **JSON is only concerned** with governing how that structure is **formatted**, not with enforcing any particular structure. The same data could be alternatively expressed in valid JSON as follows:

```
{
    "product-id": "1",
    "name": "A green door",
    "cost": "$12.50",
    "description": {
      "type": "residential",
      "color": "green"
    }
}
```

Expressing structure and type information

As humans, **either representation** makes sense to us, and since both are formatted properly according to the rules of JSON, either one can be successfully parsed by a machine as well. As long as **both sender and receiver agree** on the actual **structure** of the data along with the **types** of values represented, either one can be successfully **processed**.

How do the sender and receiver coordinate on the **schema**, which is **metadata** about the structure of fields of data as well as valid values for those fields, of the JSON data?

One way is to provide **official documentation** and make it available to programmers. Here are two examples:

- Twitter's documentation on the format of Tweets
- Facebook's documentation on the format of a Post

By studying the documentation, programmers can use the information to **write code** that will be able to successfully **exchange data** with Twitter or Facebook. This code can even be shared in **libraries** that ensure schema rules are respected when generating or processing JSON instances.

However, it seems that there should be something better than an ad hoc approach to schema representation. A **formal, standardized approach is desirable** not only for **eliminating** potential **ambiguity** about what it means to be a valid schema, but because it allows a schema definition to be fed into validators that can **automate** the **validation** of JSON data.

JSON Schema

JSON Schema is intended to be a **clear, human- and machine-readable format** for **describing** what constitutes **valid** JSON data.

The "clear, human- and machine-readable format" that JSON Schema uses to describe JSON data structures is of course **JSON**.

Here is a simple schema example from the JSON Schema website example page:

https://gist.github.com/tonypujals/3e4b5e3866595de95b1e

```
{
    "$schema": "http://json-schema.org/draft-04/schema#",
    "title": "Product",
    "description": "A product from Acme's catalog",

    "type": "object",

    "properties": {
        "id": {
            "description": "The unique identifier for a product",
            "type": "integer"
        },
...
}
```

This schema example has six properties, called *keywords*:

1. $schema
2. title
3. description
4. type
5. properties
6. required

The first three are not required. The **$schema** keyword declares that this is a schema based on the draft v4 specification, and the **title** and **description** keywords are simply informational.

The last three keywords (**type**, **properties**, **required**) declare **constraints** for valid JSON objects intended to represent product data.

These keywords declare that a JSON data document must **satisify** the following **constraints** to be considered a **valid instance** of this schema:

- `id`, `name`, and `price` properties must be present
- `id` must be of type `integer`
- `name` must be of type `string`
- `price` must be of type `number` with a value greater than zero
- `tags` is an optional array with `string` values; if present, there must be at least one element and all elements must be unique.

Using this example, a schema validator can be used to verify that the following JSON data satisfies all constraints to be a valid instance:

```
{
    "id": 1,
    "name": "A green door",
    "price": 12.50,
    "tags": ["home", "green"]
}
```

Based on the schema, the JSON examples shown earlier would **not** be valid instances because of both **structural** and **type** differences.

JSON Schema is a specification split into **three parts**. The latest draft of the specification is **v4** and it can be found both on the IETF site as Internet Drafts and on the official JSON Schema website here.

1. JSON Schema Core (defines basic foundation, such as core definitions and terminology
2. JSON Schema Validation (defines the validation keywords)
3. JSON Hyper-Schema (defines hyperlink and hypermedia keywords)

Access to **specifications** is obviously important for those **implementing** a JSON Schema **validator** (a tool or library used to validate JSON instances against a JSON schema).

The site does provide a few JSON Schema examples:

http://json-schema.org/examples.html

But for those looking for a **nice tutorial** for learning how to **read and write** JSON Schema, **Michael Droettboom** of the Space Telescope Science Institute has created a very nice open source **online book** here:

http://spacetelescope.github.io/understanding-json-schema/

Json Schema Lint is a nice online tool to practice writing and test JSON Schemas:

http://jsonschemalint.com/draft4/

JSON Schema validation constraints

http://json-schema.org/latest/json-schema-validation.html

Schema constraints that can be applied to **any** schema:

- `type`
- `enum`
- `allOf`
- `anyOf`
- `oneOf`
- `not`

Constraints can be applied to **object** types:

- `properties`
- `additionalProperties`
- `patternProperties`
- `maxProperties`
- `minProperties`
- `required`
- `dependencies`

Constraints can be applied to **numeric** types:

- `multipleOf`
- `maximum` and `exclusiveMaximum`
- `minimum` and `exclusiveMinimum`

Constraints can be applied to **string** types:

- `maxLength`
- `minLength`
- `pattern`

Constraints can be applied to **array** types:

- `items`
- `additionalItems`
- `maxItems`
- `minItems`
- `uniqueItems`

JSON Schema Test Suite

Public repository containing a set of JSON objects that **implementors** of **JSON Schema validation libraries** can use to **test** their **validators**.

Tests are **language agnostic** and should require **only** a JSON parser.

https://github.com/json-schema/JSON-Schema-Test-Suite

I contributed the node branch to simplify things for Node.js/ JavaScript developers.

All **node-specific** support (`package.json`, API, and mocha tests) are maintained on the node branch, which is **periodically refreshed** from the `develop` branch.

https://github.com/json-schema/JSON-Schema-Test-Suite/blob/node/NODE-README.md

The test suite is available as an npm package.

```
npm install json-schema-test-suite
```

JSON Schema Builder

You can build JSON Schemas is by hand. One alternative is to generate **syntactically correct** JSON Schemas using `json-schema-builder`.

`json-schema-builder` is a node package for generating JSON Schemas using a **fluent** JavaScript API.

https://github.com/atomiqio/json-schema-builder

Peter Svetlichny and I work for a startup called **Atomiq**. We wrote `json-schema-builder` as part of our microservice tooling, and we released the repo today as open source on GitHub, and published it to npm

```
npm install --save json-schema-builder
```

Our team is building another approach to generating JSON Schemas. We're getting ready to open source **Phase** next week.

Phase is a simple **domain-specific language** (DSL) for generating schemas. Under the hood, it leverages `json-schema-builder`.

https://github.com/atomiqio/phase

Phase example

```
Pet {
   id integer @format('int64')
   name string @required
   category $Category
   photoUrls [string] @required
   tags [$Tag]
   status string @description('pet status in the store')
                 @enum('available', 'pending', 'sold')
}

Category {
  id integer @format('int64')
  name string
}

Tag {
  id integer @format('int64')
  name string
}
```

The previous Phase example transforms to this JSON Schema:

https://gist.github.com/tonypujals/fcc98f5ed82ce3e59f8f

Using a JSON Schema validator

See the list at:

https://github.com/json-schema/JSON-Schema-Test-Suite#javascript

I like

- https://github.com/zaggino/z-schema
- https://github.com/geraintluff/tv4

Demo

Demo

This presentation and demo code are on GitHub:

https://github.com/tonypujals/demo-json-schema