APS Core Lib

User Guide

1.0.0

Tommy Svensson

aps-core-lib

MapJsonDocValidator

This takes a schema (made up of a `Map<String, Object>` , see below) and another `Map<String, Object>` representing the JSON. So the catch here is that you need a JSON parser that allows you to get the content as a Map. The Vertx JSON parser does. This uses `Map` since it is generic, does not need to hardcode dependency on a specific parser, and maps are very easy to work with in Groovy.

Useage

```
 private Map<String, Object> schema = [
        "meta/header": "meta",
        header_1: [
                type_1      : "service",
                "meta/type" : "metadata",
                address_1   : "?aps\\.admin\\..*",
                classifier_1: "?public|private"
        ],
        body_1  : [
                action_1: "get-webs"
        ],
        reply_0: [
                webs_1: [
                        [
                                name_1: "?.*",
                                url_1: "?^https?://.*",
                                no1_0: "#1-100",
                                no2_0: "#<=10",
                                no3_0: "#>100",
                                no4_0: "#1.2-3.4"
                        ]
                ]
        ]
] as Map<String, Object>

private MapJsonDocValidator verifier = new MapJsonDocValidator( validStructure: schema )

...

verifier.validate(myJsonMap)
```

This will throw a runtime exception on validation failure.

Schema

Keys

<key>_0 - The key is optional.

<key>_1 - The key is required.

Values

"?regexp"

The '?' indicates that the rest of the value is a regular expression. This regular expression will be applied to each value.

"<hash><range>"

This indicates that this is a number and defines the number range allowed. The following variants are available:

"#from-to" : This specifies a range of allowed values, from lowest to highest.

"#<=num" : This specifies that the numeric value must be less than or equal to the specified number.

"#>=num" : This specifies that the numeric value must be larger than or equal to the specified

number.

"#<num" : This specifies that the numeric value must be less than the specified number.

"#>num" : This specifies that the numeric value must be larger than the specified number.

Note: Both floating point numbers and integers are allowed.

"bla"

This requires values to be exactly "bla".

Example

```
Map<String, Object> struct = [
   header_1: [
      type_1      : "service",
      address_1   : "aps.admin.web",
      classifier_0: "?public|private"
   ],
   body_1  : [
      action_1: "get-webs"
   ],
   reply_0: [
      webs_1: [
         [
            name_1: "?.*",
            url_0: "?^https?://.*",
            someNumber_0: "#0-100" // Also valid: ( ">0" "<100" ) ( ">=0" "<=100" )
         ]
      ]
   ]
]
```

MapJsonSchemaMeta

This class scans a MapJson schema as defined by MapJsonDocValidator and extracts a list of MapJsonEntryMeta instances for each value in a MapJson structure living up to the schema.

From these the following can be resolved:

- The name of a value.
- The type of a value.
- Is the value required ?
- The constraints of the value. If this starts with '?' then the rest is a regular expression. If not the value is a constant, that is, the value has to be exactly as the constraint string.

This is not used by the MapJsonDocValidator when validating! This is intended for GUI configuration editors to use to build a configuration GUI producing valid configurations.

Usage:

```
Map<String, Object> schema
...
new MapJsonSchemaMeta(schema).mapJsonEntryMetas.each { MapJsonEntryMeta mjem -> ... }
```

Mapo

Yes, I had a problem coming up with a good name for this!

This wraps a structured Map that looks like a JSON document, containing Map, List, and other 'Object's as values.

A key is a String containing branches separated by '.' characters for each sub structure.

It provides a method to collect all value referencing keys in the map structure with full key paths.

It provides a lookup method that takes a full value key path and returns a value.

Note that since this delegates to the wrapped Map the class is also a Map when compiled!

Here is an example (in Groovy) that shows how to lookup and how to use the keys:

```
Mapo mapo = new Mapo<>(
        [
                header: [
                        type      : "service",
                        address   : "aps.admin.web",
                        classifier: "public",
                        enabled   : true
                ],
                body  : [
                        action: "get-webs"
                ],
                reply : [
                        webs: [
                                [
                                        name: "ConfigAdmin",
                                        url : "http://localhost:8080/aps/ConfigAdminWeb",
                                ],
                                [
                                        name: "RemoteServicesAdmin",
                                        url : "https://localhost:8080/aps/RemoteSvcAdmin"
                                ]
                        ]
                ]
        ] as Map<String, Object>
) as Mapo

assert mapo.lookup( "header.type" ).toString() == "service"
assert mapo.lookup( "header.address" ).toString() == "aps.admin.web"
assert mapo.lookup( "header.classifier" ).toString() == "public"
assert mapo.lookup( "body.action" ).toString() == "get-webs"
assert mapo.lookup( "reply.webs.[0].name") == "ConfigAdmin"
assert mapo.lookup( "reply.webs.[1].name") == "RemoteServicesAdmin"
assert mapo.lookup( "reply.webs.[1].url") == "https://localhost:8080/aps/RemoteSvcAdmin"

mapo.withAllKeys { String key ->
    println "${key}"
}

// will produce:
header.type
header.address
header.classifier
header.enabled
body.action
reply.webs.[2].name
reply.webs.[2].url
```

Note that the values are API-wise of type Object! This is because it can be anything, like a String, Map, List, Number (if you stick to JSON formats) or any other type of value you put in there.

Also note the indexes in the keys in the example. It is not "webs[0]" but "webs.[0]"! The index is a reference name in itself. The keys returned by getAllKeys() have a number between the '[' and the ']' for List entries. This number is the number of entries in the list. The MapPath class (used by this class) can be used to provide array size of an array value.