Valkyrie

v0.1.2 2024-05-12 GPL-3.0-only

Type safe type validation

James R. Swift
tinger <me@tinger.dev>

https://github.com/JamesxX/valkyrie

This package implements type validation, and is targetted mainly at package and template developers. The desired outcome is that it becomes easier for the programmer to quickly put a package together without spending a long time on type safety, but also to make the usage of those packages by end-users less painful by generating useful error messages.

Table of contents

Part I.
Example usage

```
#let template-schema = z.dictionary((
  title: z.content(),
  abstract: z.content(default: []),
  dates: z.array(z.dictionary((
    type: z.content(),
    date: z.string()
  ))),
  paper: z.schemas.papersize(default: "a4"),
  authors: z.array(z.dictionary((
    name: z.string(),
    corresponding: z.boolean(default: false),
    orcid: z.string(optional: true)
  ))),
  header: z.dictionary((
    journal: z.content(default: [Journal Name]),
    article-type: z.content(default: "Article"),
    article-color: z.color(default: rgb(167,195,212)),
    article-meta: z.content(default: [])
  )),
));

#z.parse(
  (
    title: [This is a required title],
    paper: "a3",
    authors: ( (name: "Example"),)
  ),
  template-schema,
)
```

```
(
  title: [This is a required title],
  paper: "a3",
  authors: (
    (name: "Example", corresponding: false, orcid: none),
  ),
  abstract: [],
  dates: (),
  header: (
    journal: [Journal Name],
    article-type: "Article",
    article-color: rgb("#a7c3d4"),
    article-meta: [],
  ),
)
```

Part II.
Documentation

II.1. Terminology

As this package introduces several type-like objects, the Tidy style has had these added for clarity. At present, these are `schema` (to represent type-validating objects), `z-ctx` (to represent the current state of the parsing heuristic), and `scope` (an array of strings that represents the parent object of values being parsed). `internal` represents arguments that, while settable by the end-user, should be reserved for internal or advanced usage.

Generally, users of this package will only need to be aware of the `schema` type.

II.2. Specifig language

The key words “MUST”, “MUST NOT”, “REQUIRED”, “SHALL”, “SHALL NOT”, “SHOULD”, “SHOULD NOT”, “RECOMMENDED”, “MAY”, and “OPTIONAL” in this document are to be interpreted as described in RFC 2119.

II.3. Use cases

The interface for a template that a user expects and that the developer has implemented are rearly one and the same. Instead, the user will apply common sense and the developer will put in somewhere between a token- and a whole-hearted- attempt at making their interface intuitive. Contrary to what one might expect, this makes it more difficult for the end user to correctly guess the interface as different developers will disagree on what is and isn’t intuitive, and what edge cases the developer is willing to cover.

By first providing a low-level set of tools for validating primitives upon which more complicated schemas can be defined, Valkyrie handles both the micro and macro of input validation.

II.4. Parsing functions

#parse(⟨object⟩, ⟨schemas⟩, ⟨ctx⟩: auto, ⟨scope⟩: ("argument",)) ⟶ any | none

Validates an object against one or more schemas. **WILL** return the given object after validation if successful, or none and **MAY** throw a failed assertion error.

Argument

⟨object⟩ any

Object to validate against provided schema. Object **SHOULD** statisfy the schema requirements. An error **MAY** be produced if not.

Argument

⟨schemas⟩ array | schema

Schema against which object is validated. Coerced into array. **MUST** be an array of valid valkyrie schema types.

Argument

⟨ctx⟩: auto z-ctx

ctx passed to schema validator function, containing flags that **MAY** alter behaviour.

Argument

⟨scope⟩: ("argument",) scope

An array of strings used to generate the string representing the location of a failed requirement within object. **MUST** be an array of strings of length greater than or equal to 1

II.5. Schema definition functions

For the sake of brevity and owing to their consistency, the arguments that each schema generating function accepts are listed in the table below, followed by a description of each of argument.

	any	**array**	**boolean**	**color**	**content**	**date**	**dictionary**	**either**	**number, integer, float**	**string, ip, email**	**tuple**	**choice**
body		✓					✓	✓			✓	✓
name	✱	✱	✱	✱	✱	✱	✱	✱	✱	✱	✱	✱
optional	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓
default	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓
types	✓	✱	✱	✱	✱	✱	✱	✱	✱	✱	✱	
assertions	✓	✓	✓	✓	✓	✓	✱		✓	✱	✓	✱
pre-transform	✓	✓	✓	✓	✓	✓	✱		✓	✓	✓	✓
post-transform	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓

✓ Indicates that the argument is available to the user. ✱ Indicates that while the argument is availble to the user, it may be used internally or may hold a default value.

Argument

⟨name⟩: "unknown" str

Human-friendly name of the schema for error-reporting purposes.

Argument

⟨optional⟩: false bool

Allows the value to have not been set at the time of parsing, without generating an error.

> If used on a dictionary, consider adding default values to child schemas instead.

> If used on a array, consider relying on the default (an empty array) instead.

Argument

⟨default⟩: none any

The default value to use if object being validated is none.

> Setting a default value allows the end-user to omit it.

Argument

⟨types⟩: () array

Array of allowable types. If not set, all types are accepted

Argument

⟨assertions⟩: () array

Array of assertions to be tested during object validation. see (LINK TO ASSERTIONS)

> Assertions cannot modify values

Argument

⟨pre-transform⟩: (self,it)=>it function

Transformation to apply prior to validation. Can be used to coerce values.

Argument

⟨post-transform⟩: (self,it)=>it function

Transformation to apply after validation. Can be used to reshape values for internal use

#any(..⟨args⟩) ⟶ schema

Generates a schema that accepts any input as valid.

#array(⟨schema⟩, ..⟨args⟩) ⟶ any | none

Argument

⟨schema⟩ — schema

Schema against which to validate child entries. Defaults to #any().

#boolean(..⟨args⟩) ⟶ schema

Generates a schema that accepts only booleans as valid.

#color(..⟨args⟩) ⟶ schema

Generates a schema that accepts only colors as valid.

#content(..⟨args⟩) ⟶ schema

Generates a schema that accepts only content or string as valid.

#date(..⟨args⟩) ⟶ schema

Generates a schema that accepts only datetime objects as valid.

II.6. Special Generator functions

II.6.1. Array

II.6.2. Dictionary

II.6.3. Either

II.6.4. Tuple

II.7. Coercions

II.8. Assertions

Part III.
Advanced Documentation

III.1. Validation heuristic

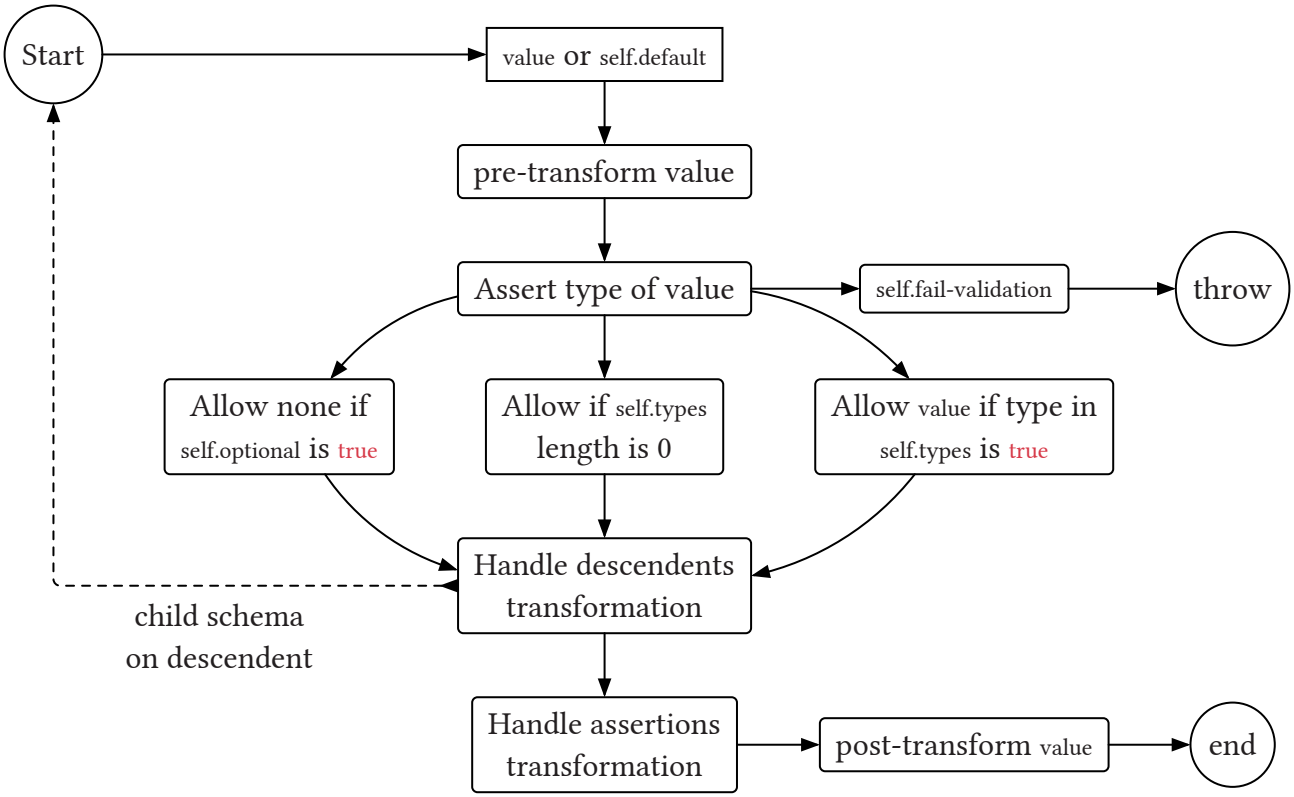

Figure 1: Flow diagram representation of parsing heuristic when validating a value against a schema.

Part IV.
Index