Valkyrie

v0.2.1 2024-07-14 MIT

Type safe type validation

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

https://github.com/typst-community/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

#alignment(..⟨args⟩) ⟶ schema

Generates a schema that accepts only aligment objects as valid.

#angle(..⟨args⟩) ⟶ schema

Generates a schema that accepts only angles as valid.

#any(..⟨args⟩) ⟶ schema

Generates a schema that accepts any input as valid.

#array(⟨schema⟩, ..⟨args⟩) ⟶ schema

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.

#bytes(..⟨args⟩) ⟶ schema

Generates a schema that accepts only bytes 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.

#dictionary(⟨aliases⟩: (:), ⟨schema⟩, ..⟨args⟩) ⟶ schema

Argument

⟨aliases⟩: (:) — dictionary

Dictionary representation of source to destination aliasing. Has the effect of allowing the user to key something with `source` when its `destination` that is meant.

Argument

⟨schema⟩ — dictionary

Dictionary of schema elements, used to define the validation rules for each entry.

#direction(..⟨args⟩) ⟶ schema

Generates a schema that accepts only directions as valid.

#either(..⟨schema⟩, ..⟨args⟩) ⟶ schema

Argument

..⟨schema⟩ — dictionary

Positional arguments of validation schemes in order or preference that an input value should satisfy.

`#function(..⟨args⟩)` → `schema`
Generates a schema that accepts only functions as valid.

`#fraction(..⟨args⟩)` → `schema`
Generates a schema that accepts only fractions as valid.

`#gradient(..⟨args⟩)` → `schema`
Generates a schema that accepts only gradient objects as valid.

`#label(..⟨args⟩)` → `schema`
Generates a schema that accepts only labels as valid.

`#length(..⟨args⟩)` → `schema`
Generates a schema that accepts only lengths as valid.

`#location(..⟨args⟩)` → `schema`
Generates a schema that accepts only locations as valid.

`#number(⟨min⟩: none, ⟨max⟩: none, ..⟨args⟩)` → `schema`
Generates a schema that accepts only numbers as valid.

`#plugin(..⟨args⟩)` → `schema`
Generates a schema that accepts only plugins as valid.

`#ratio(..⟨args⟩)` → `schema`
Generates a schema that accepts only ratios as valid.

`#relative(..⟨args⟩)` → `schema`
Generates a schema that accepts only relative types, lengths, or ratios as valid.

`#regex(..⟨args⟩)` → `schema`
Generates a schema that accepts only regex expressions as valid.

`#selector(..⟨args⟩)` → `schema`
Generates a schema that accepts only selectors as valid.

`#string(⟨min⟩: none, ⟨max⟩: none, ..⟨args⟩)` → `schema`
Generates a schema that accepts only strings as valid.

`#stroke(..⟨args⟩)` → `schema`
Generates a schema that accepts only stroke objects as valid.

`#symbol(..⟨args⟩)` → `schema`
Generates a schema that accepts only symbol types as valid.

`#tuple(..⟨schema⟩, ..⟨args⟩)` → `schema`

Argument

..⟨schema⟩ `schema`

Positional arguments of validation schemes representing a tuple.

`#version(..⟨args⟩)` → `schema`
Generates a schema that accepts only version objects as valid.

#sink(⟨positional⟩: none, ⟨named⟩: none, ..⟨args⟩) ⟶ schema

Argument

⟨positional⟩ — schema | none

Schema that `args.pos()` must satisfy. If `none`, no positional arguments may be present

Argument

⟨named⟩ — schema | none

Schema that `args.named()` must satisfy. If `none`, no named arguments may be present

#choice(⟨choices⟩, ..⟨args⟩) ⟶ schema

Argument

⟨choices⟩ — array

Array of valid inputs

II.6. z.coerce

```
#array()                    #date()
#content()                  #dictionary()
```

#dictionary(⟨fn⟩)

If the tested value is not already of dictionary type, the function provided as argument is expected to return a dictionary type with a shape that passes validation.

```
#let schema = z.dictionary(
  pre-transform: z.coerce.dictionary((it)=>(name: it)),
  (name: z.string())
)
#z.parse("Hello", schema) \
#z.parse((name: "Hello"), schema)
```

```
(name: "Hello")
(name: "Hello")
```

Argument

⟨fn⟩ `function`

Transformation function that the tested value and returns a dictionary that has a shape that passes validation.

#array(⟨self⟩, ⟨it⟩)

If the tested value is not already of array type, it is transformed into an array of size 1

```
#let schema = z.array(
  pre-transform: z.coerce.array,
  z.string()
)
#z.parse("Hello", schema) \
#z.parse(("Hello", "world"), schema)
```

```
("Hello",)
("Hello", "world")
```

#content(⟨self⟩, ⟨it⟩)

Tested value is forceably converted to content type

```
#let schema = z.content(
  pre-transform: z.coerce.content
)
#type(z.parse("Hello", schema)) \
#type(z.parse(123456, schema))
```

```
content
content
```

#date(⟨self⟩, ⟨it⟩)

An attempt is made to convert string, numeric, or dictionary inputs into datetime objects

```
#let schema = z.date(
  pre-transform: z.coerce.date
)
#z.parse(2020, schema) \
#z.parse("2020-03-15", schema) \
#z.parse("2020/03/15", schema) \
#z.parse((year: 2020, month: 3, day: 15), schema) \
```

```
datetime(year: 2020, month: 1, day: 1)
datetime(year: 2020, month: 3, day: 15)
datetime(year: 2020, month: 3, day: 15)
datetime(year: 2020, month: 3, day: 15)
```

```
#array()                    #date()
#content()                  #dictionary()
```

#dictionary(⟨fn⟩)

If the tested value is not already of dictionary type, the function provided as argument is expected to return a dictionary type with a shape that passes validation.

```
#let schema = z.dictionary(
  pre-transform: z.coerce.dictionary((it)=>(name: it)),
  (name: z.string())
)
#z.parse("Hello", schema) \
#z.parse((name: "Hello"), schema)
```

```
(name: "Hello")
(name: "Hello")
```

Argument

⟨fn⟩ `function`

Transformation function that the tested value and returns a dictionary that has a shape that passes validation.

#array(⟨self⟩, ⟨it⟩)

If the tested value is not already of array type, it is transformed into an array of size 1

```
#let schema = z.array(
  pre-transform: z.coerce.array,
  z.string()
)
#z.parse("Hello", schema) \
#z.parse(("Hello", "world"), schema)
```

```
("Hello",)
("Hello", "world")
```

#content(⟨self⟩, ⟨it⟩)

Tested value is forceably converted to content type

```
#let schema = z.content(
  pre-transform: z.coerce.content
)
#type(z.parse("Hello", schema)) \
#type(z.parse(123456, schema))
```

```
content
content
```

#date(⟨self⟩, ⟨it⟩)

An attempt is made to convert string, numeric, or dictionary inputs into datetime objects

```
#let schema = z.date(
  pre-transform: z.coerce.date
)
#z.parse(2020, schema) \
#z.parse("2020-03-15", schema) \
#z.parse("2020/03/15", schema) \
#z.parse((year: 2020, month: 3, day: 15), schema) \
```

```
datetime(year: 2020, month: 1, day: 1)
datetime(year: 2020, month: 3, day: 15)
datetime(year: 2020, month: 3, day: 15)
datetime(year: 2020, month: 3, day: 15)
```

II.7. z.assert

```
#ends-with()          #max()          #starts-with()
#eq()                 #min()
#matches()            #one-of()
```

#one-of(⟨list⟩)

Asserts that the given value is contained within the provided list. Useful for complicated enumeration types.

Argument

⟨list⟩ `array`

An array of inputs considered valid.

#min(⟨rhs⟩)

Asserts that tested value is greater than or equal to argument

#max(⟨rhs⟩)

Asserts that tested value is less than or equal to argument

#eq(⟨rhs⟩)

Asserts that tested value is exactly equal to argument

#starts-with(⟨value⟩)

Asserts that a tested value starts with the argument (string)

#ends-with(⟨value⟩)

Asserts that a tested value ends with the argument (string)

#matches(⟨needle⟩, ⟨message⟩: (...) => ...)

Asserts that a tested value matches with the needle argument (string)

II.8. z.assert.length

```
#equals()             #max()          #min()
```

#min(⟨rhs⟩)

Asserts that tested value's length is greater than or equal to argument

#max(⟨rhs⟩)

Asserts that tested value's length is less than or equal to argument

#equals(⟨rhs⟩)

Asserts that tested value's length is exactly equal to argument

Part III.
Advanced Documentation

III.1. Validation heuristic

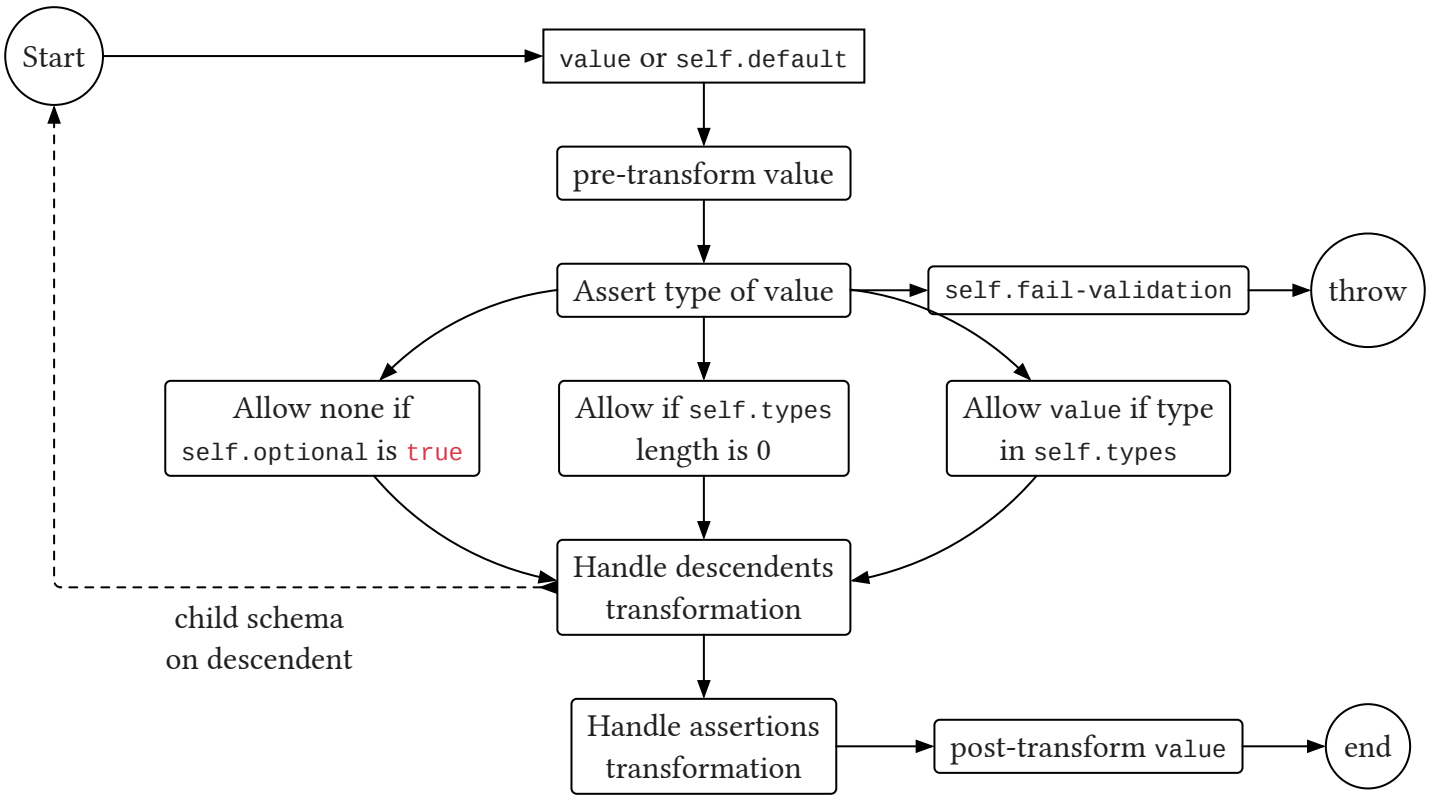

Figure 1: Flow diagram representation of parsing heuristic when validating a value against a schema.

Part IV.
Index