dict_deserializer Documentation

Release 0.0.6

Rolf van Kleef

May 16, 2021

Contents

CHAPTER 1

dict_deserializer.annotations

class dict_deserializer.annotations.**Discriminator**

Bases: abc.ABC

Base class for all discriminator.

check (*d: dict*)

Returns true or false, depending on whether the discriminator matches the provided class.

Parameters **d** – the data

Returns True when this class is valid, False otherwise.

class dict_deserializer.annotations.**FunctionDiscriminator** (*matcher*)

Bases: *dict_deserializer.annotations.Discriminator*

Discriminates according to a custom function.

check (*d: dict*)

Returns true or false, depending on whether the discriminator matches the provided class.

Parameters **d** – the data

Returns True when this class is valid, False otherwise.

class dict_deserializer.annotations.**KeyValueDiscriminator** (*key*, *value*, *has_value=True*)

Bases: *dict_deserializer.annotations.Discriminator*

Discriminates on key and optionally a value.

check (*d: dict*)

Returns true or false, depending on whether the discriminator matches the provided class.

Parameters **d** – the data

Returns True when this class is valid, False otherwise.

dict_deserializer.annotations.**abstract** (*cls*)

Declares that this class cannot be instanced. Only subclasses that are not declared abstract could be instanced.

This is equivalent to setting the class property `_abstract=True`.

Parameters **`cls`** – The class that should be abstract

Returns The same class

`dict_deserializer.annotations.`**`discriminate`**(*key=None*, *value=<object object>*, *matcher=None*)

Class level annotation to specify requirements of the raw datastructure in order to be allowed to deserialize into this class (or its subclasses).

Parameters

- **`key`** – The key to discriminate against
- **`value`** – (Optionally) the value that the property designated by `key` should hold
- **`matcher`** – (Optionally) a custom function to discriminate with.

Returns A function that should wrap the class to be discriminated.

`dict_deserializer.annotations.`**`validated`**(*default=None*)

Used to decorate a validator function. Can be used if one would want to constrain the value of a property. Of course, `@property` may be used as well. Throw a TypeError when validation fails.

Parameters **`default`** – The default (initial) value of this property

Returns the wrapper function.

CHAPTER 2

dict_deserializer.deserializer

class dict_deserializer.deserializer.**Deserializable**(***kwargs*)
Bases: `object`

Base class for all automagically deserializing classes.

classmethod get_attrs() → Dict[str, dict_deserializer.deserializer.Rule]
Returns a list of all type rules for the given class.

Returns a dict from property to type rule.

class dict_deserializer.deserializer.**DeserializableMeta**
Bases: `type`

Metaclass for all Deserializable

class dict_deserializer.deserializer.**Rule**(*type*, *default=None*)
Bases: `object`

This class is primarily used as a container to store type information.

error_string()

static to_rule(*tpe*) → dict_deserializer.deserializer.Rule
Ensures type is a rule. Otherwise, it will be converted into a rule.

Parameters **tpe** – The type/rule.

Returns a Rule

validate(*key: str*, *value*)
Returns the original value, a default value, or throws. :param key: The key of this field. :param value: Which value to validate. :return: value, default.

dict_deserializer.deserializer.**deserialize**(*rule: dict_deserializer.deserializer.Rule*, *data*, *try_all: bool = True*, *key: str = '[root]'*)
Converts the passed in data into a type that is compatible with rule.

Parameters

- **rule** –

- **`data`** –
- **`try_all`** – Whether to attempt other subtypes when a TypeError has occurred. This is useful when automatically deriving discriminators.
- **`key`** – Used for exceptions and error reporting. Preferrably the full path to the current value.

Returns An instance matching Rule.

`dict_deserializer.deserializer.`**`get_deserialization_classes`**(*t*, *d*, *try_all=True*) → List[type]

Find all candidates that are a (sub)type of t, matching d.

Parameters

- **`t`** – The type to match from.
- **`d`** – The dict to match onto.
- **`try_all`** – Whether to support automatic discrimination.

Returns an ordered list of candidate classes to deserialize into.

CHAPTER 3

Dictionary deserializer

Dictionary deserializer is a project built to convert dictionaries into composite classes in an intuitive way. Special attention was also paid to being friendly to static type-checkers and IDE autocompletes.

It is expected that this library is used together with a JSON-to-dict deserializer like `json.loads`.

3.1 Installation

In order to use it, simply add the dependency `dictionary-deserializer` to your requirements file:

3.1.1 Pipenv

```
pipenv install dictionary-deserializer
```

3.1.2 Pip

```
pip install dictionary-deserializer
pip freeze > requirements.txt
```

3.2 Design

This project was originally meant as a proof of concept, to be used to find other projects that would be able to replace this, with the required feature set. That project was not found, and therefore, this project was expanded.

3.2.1 Requirements

- Use type hints for type validation
- Allow polymorphism
  - Through `typing.Unions`
  - Through subclassing
- Support a large part of the `typing` module's types
- Allow validations on values
- Be able to validate and deserialize *any* compliant JSON structure
- Be compatible with static type checkers and IDE hinting
- Have a small impact on existing code starting to use this library

3.3 Examples

None of this code is actually useful if you don't understand how to use it. It is very simple. Here are some examples:

3.3.1 Specifying a structure

```python
from typing import Optional

from dict_deserializer.deserializer import Deserializable

class User(Deserializable):
    email: str                  # Type must be a string
    username: str               # Type must be a string
    password: Optional[str]     # Type must either be a string or a None
```

3.3.2 Deserialization

```python
from dict_deserializer.deserializer import deserialize, Rule

# Successful
deserialize(Rule(User), {
    'email': 'pypi@rolfvankleef.nl',
    'username': 'rkleef',
})

# Fails because optional type is wrong
deserialize(Rule(User), {
    'email': 'pypi@rolfvankleef.nl',
    'username': 'rkleef',
    'password': 9.78,
})
```

3.3.3 Polymorphic structures

```python
from typing import Optional, Any, List

from dict_deserializer.deserializer import Deserializable
from dict_deserializer.annotations import abstract

@abstract
class DirectoryObject(Deserializable):
    name: str
    meta: Any

class User(DirectoryObject):
    full_name: str
    first_name: Optional[str]

class Group(DirectoryObject):
    members: List[DirectoryObject]
```

If you deserialize into `Rule(DirectoryObject)`, the matching class will automatically be selected. If none of the subclasses match, an error is thrown since the DirectoryObject is declared abstract.

If you want to discriminate not by field names or types, but by their values, one can choose to define a `@discriminator` annotation.

3.3.4 Value validations

The syntax for validating the value of a key is currently a bit weird. It is incompatible with existing syntax for defaults, but the type syntax is the same.

Example:

```python
from typing import Optional

from dict_deserializer.deserializer import Deserializable
from dict_deserializer.annotations import validated

class Test(Deserializable):
    name: Optional[str]

    @validated(default='Unknown')
    def name(self, value):
        if len(value) > 20:
            raise TypeError('Name may not be longer than 20 characters.')
```

3.4 Limitations

This library uses the `typing` module extensively. It does, however, only support some of its types. This is a list of verified composite types:

- `Union` (Including `Optional`)
- `Dict`
- `List`

- `Tuple`
- `Any`
- `dict_deserializer.deserializer.Deserializable`
- `dict`
- `list`

It supports these types as terminal types:

- `int`
- `float`
- `str`
- `NoneType`
- `bool`

3.5 Planned features

- NamedTuples
  - The anonymous namedtuple and the class-namedtuples with (optionally) type annotations.
- Dataclasses
- A way to allow deserializing into a class not extending `Deserializable`
- Enums
- Sets
  - From lists

Python Module Index

Index