scrapy-spider-metadata

Release 0.0.0

Zyte Group Ltd

Mar 22, 2024

CONTENTS

`scrapy-spider-metadata` is a Python 3.8+ library of utilities to extend Scrapy spiders with usable metadata.

In particular, it provides a nice way for Scrapy spiders to define, validate, document and pre-process their arguments as pydantic models.

CHAPTER

ONE

SPIDER METADATA

This library allows retrieving spider metadata defined in spider classes.

If a spider class defines *spider parameters*, their schema will also be included in the retrieved metadata.

1.1 Defining spider metadata

You can declare arbitrary metadata in your spider classes as a dictionary attribute named `metadata`:

```
from scrapy import Spider

class MySpider(Spider):
    name = "my_spider"
    metadata = {
        "description": "This is my spider.",
        "category": "My basic spiders",
    }
```

As this attribute is shared between instances of the class and of its subclasses, be careful not to modify it in place. Here is a simple way to add or change some values in a subclass:

```
from scrapy import Spider

class BaseSpider(Spider):
    metadata = {
        "description": "Base spider.",
        "category": "Base spiders",
    }

class BaseNewsSpider(BaseSpider):
    metadata = {
        **BaseSpider.metadata,
        "description": "Base news spider.",
    }

class CNNSpider(BaseNewsSpider):
    metadata = {
        **BaseNewsSpider.metadata,
        "description": "CNN spider.",
        "category": "Concrete spiders",
```

(continues on next page)

(continued from previous page)

```
        "website": "CNN",
    }
```

1.2 Getting spider metadata

scrapy-spider-metadata provides the following function for retrieving the metadata for a specific spider class:

scrapy_spider_metadata.**get_spider_metadata**(*spider_cls: Type[Spider]*, *, *normalize: bool = False*) → Dict[str, Any]

Return the metadata for the spider class.

Return a copy of the `metadata` dict. If the spider class defines *spider parameters*, the returned dict will have an additional `param_schema` key which value is the *JSON Schema* for the parameters.

Parameters

- **spider_cls** – The spider class.
- **normalize** – Normalize the returned schema.

Returns

The complete spider metadata.

CHAPTER

TWO

PARAMETERS

Spider arguments have some limitations:

- There is no standard way for spiders to indicate which parameters they expect or support.
- Since arguments can come from the command line, and those are always strings, you have to write code to convert arguments to a different type when needed.
- If you want argument validation, such as making sure that arguments are of the right type, or that required arguments are present, you must implement validation yourself.

scrapy-spider-metadata allows overcoming those limitations.

2.1 Defining supported parameters

To define spider parameters, define a subclass of pydantic.BaseModel, and then make your spider also inherit from *Args* with your parameter specification class as its parameter:

```
from pydantic import BaseModel
from scrapy import Request, Spider
from scrapy_spider_metadata import Args

class MyParams(BaseModel):
    pages: int

class MySpider(Args[MyParams], Spider):
    name = "my_spider"

    def start_requests(self):
        for index in range(1, self.args.pages + 1):
            yield Request(f"https://books.toscrape.com/catalogue/page-{index}.html")
```

To learn how to define parameters in your pydantic.BaseModel subclass, see the Pydantic usage documentation.

Defined parameters make your spider:

- Halt with an exception if there are missing arguments or any provided argument does not match the defined parameter validation rules.

  Note: By default extra arguments are silently ignored, but you can change that.

- Expose an instance of your parameter specification class, that contains the parsed version of your spider arguments, e.g. converted to their expected type.

For example, if you run the spider in the *example above* with the `pages` parameter set to the string `"42"`, `self.args.pages` is the `int` value `42` (`self.pages` remains `"42"`).

Also, if you do not pass a value for `pages` at all, the spider will not start, because `pages` is a required parameter. All parameters without a default value are considered required parameters.

2.2 Getting the parameter specification as JSON Schema

Given a spider class with *defined parameters*, you can get a JSON Schema representation of the parameter specification of that spider using the *`get_param_schema()`* class function:

```
>>> MySpider.get_param_schema()
{'properties': {'pages': {'title': 'Pages', 'type': 'integer'}}, 'required': ['pages'],
↪'title': 'MyParams', 'type': 'object'}
```

scrapy-spider-metadata uses Pydantic to generate the JSON Schema, so your version of pydantic can affect the resulting output.

2.3 Parameters API

class scrapy_spider_metadata.**Args**(**args*, ***kwargs*)

Validates and type-converts spider arguments into the `args` instance attribute according to the *spider parameter specification*.

classmethod get_param_schema(*normalize: bool = False*) → Dict[Any, Any]

Return a `dict` with the *parameter definition* as JSON Schema.

If *normalize* is `True`, the returned schema will be the same regardless of whether you are using Pydantic 1.x or Pydantic 2.x. The normalized schema may not match the output of any Pydantic version, but it will be functionally equivalent where possible.

CHAPTER

THREE

CHANGES

3.1 0.1.2 (2023-10-09)

- Fixed the params schema output for optional fields and enums.

3.2 0.1.1 (2023-10-09)

- Added “normalize” parameter to `get_spider_metadata` function.

3.3 0.1.0 (2023-09-29)

Initial version.

INDEX

A

G