
parameter Documentation

Release 0.0.2

Gray King

Apr 28, 2017

Contents

1	Benefits	3
2	Example with tornado	5
2.1	Normal pattern	5
2.2	Parameter pattern	5
3	Indices and tables	15
	Python Module Index	17

`Parameter` is using to get and check HTTP parameters like use ORM.

CHAPTER 1

Benefits

- Less code to check arguments.
- Pass http arguments to other function with a single object.
- IDE friendly, IDE can easily detect the complation.
- Easy to linter, the linter can easily detect attribute error.

Example with tornado

Normal pattern

```
from tornado import web

class DemoHandler(web.RequestHandler):
    def get(self):
        action = self.get_argument("action", None)
        arg1 = self.get_argument("arg1", None)
        arg2 = self.get_argument("arg2", None)

        # ...

        if action:
            pass

        if arg1:
            pass

        # ...

        do(action, arg1, arg2, ...)
```

Parameter pattern

```
from tornado import web

from parameter import Model, Argument
from parameter import types
from parameter.adapter import TornadoAdapter
```

```
class DemoEntity(Model):
    action = Argument("action", types.String, required=False,
                      miss_message="Please choose action",
                      invalid_message="Invalid action")
    arg1 = Argument("arg1", types.Integer)
    arg2 = Argument("arg2", types.Double)
    # ...

class DemoHandler(web.RequestHandler):
    def get(self):
        demo = DemoEntity(TornadoAdapter(self))
        do(demo)
```

Contents:

Guide

Installation

Use pip to install parameter.

```
$ pip install -U parameter
```

Define model

Inherit from `Model` and use `Argument` to define a model.

```
from parameter import Model, Argument
from parameter import types

class Person(Model):
    name = Argument(types.String)
    age = Argument(types.Integer)
```

After model defined, you can use an adapter to create a instance.

```
from parameter.adapter import JSONAdapter

person = Person(JSONAdapter({"name": "Gray", "age": 18}))

print(person.name)      # output: Gray
print(person.age)       # output: 18
```

Alias

If a parameter's name is not same with the attribute name, we can use `alias`.

```
from parameter import Model, Argument
from parameter import types
```

```
class Person(Model):
    children = Argument(types.String, alias="child", multiple=True)
```

The above code will map `child` argument to the `children`.

List

If a parameter have multiple arguments, just set `multiple=True` in *Argument*.

```
from parameter import Model, Argument
from parameter import types

class Person(Model):
    name = Argument(types.String)
    age = Argument(types.Integer)
    children = Argument(types.String, alias="child", multiple=True)

# Assume the request is: /person?name=Gray&age=18&child=Tom&child=Jim
person = Person(DemoAdapter(request))

print(person.name)      # output: Gray
print(person.age)       # output: 18
print(person.children)  # maybe output: ["Tom", "Jim"]
```

Nested

parameter support nested by *parameter.types.Nested*.

```
from parameter import Model, Argument
from parameter import types
from parameter.adapter import JSONAdapter

class Person(Model):
    name = Argument(types.String)
    age = Argument(types.Integer)

class Computer(Model):
    arch = Argument(types.String)
    belong = Argument(types.Nested(Person))

computer = Computer(JSONAdapter({"arch": "x86", "belong": {"name": "Gray", "age": 10}}
↪))

assert computer.arch == "x86"
assert isinstance(computer.person, Person)
assert computer.person.name == "Gray"
assert computer.person.age == 18
```

List nested

parameter nested also can be a list with multiple argument.

```
from parameter import Model, Argument
from parameter import types
from parameter.adapter import JSONAdapter

class Computer(Model):
    arch = Argument(types.String)

class Person(Model):
    name = Argument(types.String)
    age = Argument(types.Integer)
    computers = Argument(types.Nested(Computer), multiple=True)

person = Person(JSONAdapter({"name": "Gray", "age": 10, "computers": [
    {"arch": "x86"},
    {"arch": "x86_64"},
]}))

assert person.name == "Gray"
assert person.age == 18
assert isinstance(person.computers, list)
assert len(person.computers) == 2
assert isinstance(person.computers[0], Computer)
assert isinstance(person.computers[1], Computer)

assert person.computers[0].arch == "x86"
assert person.computers[1].arch == "x86_64"
```

Handling exception

While creating model, there two exceptions that user must to care:

- `parameter.exception.ArgumentMissError`: Raising when argument is missing
- `parameter.exception.ArgumentInvalidError`: Raising when argument is invalid

Argument

```
class parameter.model.Argument (type_, default=[], alias=None, multiple=False,
                                miss_message=None, invalid_message=None)
```

Represents a parameter in HTTP request.

```
__init__(type_, default=[], alias=None, multiple=False, miss_message=None, in-
         valid_message=None)
```

Initialize

Parameters

- **type** (`parameter.types.BaseType`.) – The parameter's type, indicated using an instance which subclasses `parameter.types.BaseType`.
- **default** – The default value.
- **alias** – The alias name of this argument as represented in the HTTP request.
- **multiple** – This argument have multiple values.

- **miss_message** – The message of *ArgumentMissError*
- **invalid_message** – The message of *ArgumentInvalidError*

convert (*value*)

Check and convert the value to the specified type.

Raises *ArgumentMissError*

Raises *ArgumentInvalidError*

classmethod is_init_default (*value*)

Returns True if the value is the initial default.

Types

Add custom type

If you want add your own type, you need inherit from *BaseType* and override the abstract method `convert`. It used to convert an raw value from request to the current type.

Here is an example:

```
from parameter.types import BaseType

class CVSList(BaseType):
    def convert(self, val):
        return val.split(",")
```

The above type receive a string value, and returns a list that split by comma.

Then you can use the type you have defined.

```
from parameter import Model, Argument

class DemoEntity(Model):
    names = Argument("names", CVSList)
```

If you want some custom options, you can define the constructor method.

```
from parameter.types import BaseType

class CVSList(BaseType):
    def __init__(self, separator=","):
        self.separator = separator

    def convert(self, val):
        return val.split(self.separator)
```

The you can define a different separator.

```
from parameter import Model, Argument

class DemoEntity(Model):
    names = Argument("names", CVSList(separator="|"))
```

class `parameter.types.BaseType`

Base class of the types.

convert (*val*)

Convert a value to this type.

Raises `parameter.exception.MismatchError`

class `parameter.types.Date` (*format=u'%Y-%m-%d'*)

convert (*val*)

class `parameter.types.Datetime` (*format=u'%Y-%m-%d %H:%M:%S'*)

convert (*val*)

class `parameter.types.Decimal` (*context=None*)

convert (*val*)

class `parameter.types.Double`

convert (*val*)

class `parameter.types.Integer`

Integer type.

convert (*val*)

class `parameter.types.Nested` (*model_cls*)

convert (*adapter*)

Returns an instance which subclasses `Model`

Parameters **adapter** (*BaseAdapter*) – Adapter of the hosted model.

class `parameter.types.String` (*max_len=None, encoding=u'utf8'*)

String type. This is `str` in Python2 and `bytes` in Python3.

convert (*val*)

class `parameter.types.Unicode` (*max_len=None, encoding=u'utf8'*)

Unicode type. This is `unicode` in Python2 and `str` in Python3.

convert (*val*)

Adapters

Tornado adapter

class `parameter.adapter.TornadoAdapter` (*handler*)

Tornado adapter.

Usage:

```
from tornado import web

from parameter import Model, Argument, types
```

```

from parameter.adapter import TornadoAdapter

class UserEntity(Model):
    username = Argument("username", types.String, max_len=100)
    password = Argument("password", types.String, max_len=64)
    name = Argument("name", types.Unicode, max_len=50)
    age = Argument("age", types.Integer, default=18)

class DemoHandler(web.RequestHandler):
    def get(self):
        entity = UserEntity(TornadoAdapter(self))

        self.write({
            "name": entity.name,
            "age": entity.age,
        })

```

JSON adapter

class `parameter.adapter.JSONAdapter` (*data*)
 JSON adapter to get arguments from a JSON object.

Usage:

```

from parameter import Model, Argument, types
from parameter.adapter import JSONAdapter

data = {"a": 1, "b": 2}

class DataEntity(Model):
    a = Argument("a", types.Integer)
    b = Argument("b", types.Integer)

adapter = JSONAdapter(data)
entity = DataEntity(adapter)

print(entity.a)           # 1
print(entity.b)           # 2

```

Nested:

```

from parameter import Model, Argument, types
from parameter.adapter import JSONAdapter

data = {"a": 1, "b": 2, "person": {"age": 18, "name": "Gray"}}

class PersonEntity(Model):
    age = Argument("age", types.Integer)
    name = Argument("name", types.Unicode)

class DataEntity(Model):
    a = Argument("a", types.Integer)
    b = Argument("b", types.Integer)
    person = Argument("person", types.Nested(PersonEntity))

```

```
adapter = JSONAdapter(data)
entity = DataEntity(adapter)

print(entity.a)           # 1
print(entity.b)           # 2
print(entity.person.age)  # 18
print(entity.person.name) # Gray
```

Add custom adapter

class `parameter.model.BaseAdapter`

To implement your own adapter, you need inherit from *BaseAdapter*.

There two methods must be overwriten:

- `get_argument`: Returns a single value
- `get_arguments`: Returns a sequence of values.

Example:

```
from parameter.model import BaseAdapter

class DemoAdapter(BaseAdapter):
    "demo adapter"
    def __init__(self, arguments):
        self.arguments = arguments

    def get_argument(self, name, default):
        return self.arguments.get(name, default)

    def get_arguments(self, name):
        return self.arguments.getlist(name)
```

If you want your adapter to support nested, you need to override the `spawn` method, this method use the given value to return an new instance of the current adapter.

```
from parameter.model import BaseAdapter

class DemoAdapter(BaseAdapter):
    # see above
    def spawn(self, arguments):
        return DemoAdapter(arguments)
```

get_argument (*name, default, *args, **kwargs*)

Returns the argument's value via name.

Parameters

- **name** – The name of the argument.
- **default** – The default value.

Raises *ArgumentMissError*

Raises *ArgumentInvalidError*

get_arguments (*name*, **args*, ***kwargs*)

Returns the argument's values via *name*.

Parameters *name* – The name of the argument.

Raises *ArgumentMissError*

Raises *ArgumentInvalidError*

static spawn (*val*)

Use the new value to spawn an new adapter of this adapter.

Exceptions

Exceptions of this package.

exception `parameter.exception.ArgumentError` (*message*, *name*)

Argument base Exception

exception `parameter.exception.ArgumentInvalidError` (*message*, *name*, *source*)

exception `parameter.exception.ArgumentMissError` (*message*, *name*)

exception `parameter.exception.ConvertError`

exception `parameter.exception.MaxlenExceedError`

exception `parameter.exception.MismatchError`

Type mismatch.

exception `parameter.exception.ParameterException`

Base exception of this package.

CHAPTER 3

Indices and tables

- `genindex`
- `modindex`
- `search`

p

`parameter.exception`, [13](#)
`parameter.types`, [9](#)

Symbols

`__init__()` (parameter.model.Argument method), 8

A

Argument (class in parameter.model), 8

ArgumentError, 13

ArgumentInvalidError, 13

ArgumentMissError, 13

B

BaseAdapter (class in parameter.model), 12

BaseType (class in parameter.types), 9

C

`convert()` (parameter.model.Argument method), 9

`convert()` (parameter.types.BaseType method), 10

`convert()` (parameter.types.Date method), 10

`convert()` (parameter.types.Datetime method), 10

`convert()` (parameter.types.Decimal method), 10

`convert()` (parameter.types.Double method), 10

`convert()` (parameter.types.Integer method), 10

`convert()` (parameter.types.Nested method), 10

`convert()` (parameter.types.String method), 10

`convert()` (parameter.types.Unicode method), 10

ConvertError, 13

D

Date (class in parameter.types), 10

Datetime (class in parameter.types), 10

Decimal (class in parameter.types), 10

Double (class in parameter.types), 10

G

`get_argument()` (parameter.model.BaseAdapter method),
12

`get_arguments()` (parameter.model.BaseAdapter method),
12

I

Integer (class in parameter.types), 10

`is_init_default()` (parameter.model.Argument class
method), 9

J

JSONAdapter (class in parameter.adapter), 11

M

MaxlenExceedError, 13

MismatchError, 13

N

Nested (class in parameter.types), 10

P

parameter.exception (module), 13

parameter.types (module), 9

ParameterException, 13

S

`spawn()` (parameter.model.BaseAdapter static method),
13

String (class in parameter.types), 10

T

TornadoAdapter (class in parameter.adapter), 10

U

Unicode (class in parameter.types), 10