metaframe Documentation

Release 1.0.1

Daniel Rodriguez

October 25, 2015

Contents

Contents:

CHAPTER 1

Introduction

`metaframe` is a MetaClass infrastructure to intercept instance creation/initialization enabling modification of args/kwargs and instance.

1.1 Features:

- `MetaFrame` metaclass to apply to any object - With embedded staticmethod with_metaclass to enable inheritance
- `MetaFrameBase` class from which classes can inherit
- 3 hooks (classmethods)
  - `_new_pre`: called before object creation
  - `_new_do`: called for object creation
  - `_init_pre`: called after object creation / before object initialization
  - `_init_do`: called fo object initialization
  - `_init_post`: called after object initialization

1.2 Installation

`metaframe` is self-contained with no external dependencies

From pypi:

```
pip install metaframe
```

From source:

- Place the *metaframe* directory found in the sources inside your project

CHAPTER 2

Usage

`metaframe` allows placing hooks into the creation/initializaion of objects, enabling use cases like:

- Modification of args/kwargs on the fly
- Instance scanning/modification

2.1 Direct Inheritance

The package offers an already **metaclassed** base class supporting the infrastructure.

- `MetaFrameBase`

2.1.1 Intercepting Object Creation

An example from one of the tests included in the sources.

```
import metaframe as mf


class FrameTest(mf.MetaFrameBase):
    _KEY = 'ft'
    _VAL = True

    def __init__(self, *args, **kwargs):
        self._val = kwargs.get(self._KEY, False)

    def check_val(self):
        return self._val == self._VAL

    @classmethod
    def _new_pre(cls, *args, **kwargs):
        # Insert a kwarg
        kwargs[cls._KEY] = cls._VAL
        return cls, args, kwargs
```

Doing something with it:

```
ft = FrameTest()
print('ft.check_val:', ft.check_val())
```

Yields the following output:

```
ft.check_val: True
```

From the example:

- No kwargs were passed to `FrameTest` for instantiation
- During init `self._KEY` ('ft') was extracted from kwargs and assigned to `self._val`
- The `kwargs` were actually modified in the `classmethod()` where `self._VAL` was added with key ''self._KEY''

  And the modified `kwargs` were returned to be fed to object creation/initialization
- Hence `check_val()` returning `True`

2.1.2 Before initialization

The previous example can be extended to undo the effect achieved during object creation.

Let's add a hook before init

```
    @classmethod
    def _init_pre(cls, obj, *args, **kwargs):
        # Remove the kwarg
        kwargs.pop(cls._KEY)
        return obj, args, kwargs
```

Doing something with it:

```
ft = FrameTest()
print('ft.check_val:', ft.check_val())
```

Yields the following output:

```
ft.check_val: False
```

The new code in `_init_pre()` removes the key `self._KEY` from the passed `kwargs` and returns them for object initialization.

2.1.3 After initialization

Redoing the effect by directly operating on the instance can be done after initialization.

The hook after __init__

```
    @classmethod
    def _init_post(cls, obj, *args, **kwargs):
        # change self._val ... to the expected value
        obj._val = obj._VAL
        return obj, args, kwargs
```

Repeating execution:

```
ft = FrameTest()
print('ft.check_val:', ft.check_val())
```

Yields the following output:

```
ft.check_val: True
```

In this case the post initialization hook has directly changed the value of attribute `_val` after object init.

2.2 Applying the metaclass

Instead of inheriting from `MetaFrameBase` a derived metaclass for your class can be created:

```
import metaframe as mf

class MyMetaClass(mf.MetaFrame):
    def _new_pre(cls, *args, **kwargs):
        # Insert a kwarg
        kwargs[cls._KEY] = cls._VAL
        return cls, args, kwargs

    def _init_pre(cls, obj, *args, **kwargs):
        # Remove the kwarg
        kwargs.pop(cls._KEY)
        return obj, args, kwargs

    def _init_post(cls, obj, *args, **kwargs):
        # change self._val ... to the expected value
        obj._val = obj._VAL
        return obj, args, kwargs
```

Now there is no need to declare the 3 hoods as `classmethods` because they are already being declared in the MetaClass.

The `FrameTest` class would now look like this:

```
class FrameTest(MyMetaClass.as_metaclass(object)):
    _KEY = 'ft'
    _VAL = True

    def __init__(self, *args, **kwargs):
        self._val = kwargs.get(self._KEY, False)

    def check_val(self):
        return self._val == self._VAL
```

The execution examples remain unchanged.

Alternatively, you can directly MetaFrame-enable a class applying `MetaFrame` as metaclass and defining the methods in the class as `@classmethods`:

```
class FrameTest(mf.MetaFrame.as_metaclass(object)):
    _KEY = 'ft'
    _VAL = True

  @classmethod
  def _new_pre(cls, *args, **kwargs):
      # Insert a kwarg
      kwargs[cls._KEY] = cls._VAL
      return cls, args, kwargs

  @classmethod
  def _init_pre(cls, obj, *args, **kwargs):
      # Remove the kwarg
      kwargs.pop(cls._KEY)
      return obj, args, kwargs

  @classmethod
```

```
  def _init_post(cls, obj, *args, **kwargs):
      # change self._val ... to the expected value
      obj._val = obj._VAL
      return obj, args, kwargs

    def __init__(self, *args, **kwargs):
        self._val = kwargs.get(self._KEY, False)

    def check_val(self):
        return self._val == self._VAL
```

CHAPTER 3

Reference

class metaframe.**MetaFrame**

This Metaclass intercepts instance creation/initialization enabling use cases like modification of args, kwargs and/or scanning of the object post init

_new_pre(**args*, ***kwargs*)

Called before the object is created.

Parameters

- **cls** (*automatic*) – The class which is going to be instantiated
- **args** – To be passed to __new__ for class instantiation
- **kwargs** – To be passed to __new__ for class instantiation

Returns cls, args, kwargs as a tuple

The return values need not be the same that were passed

_new_do(**args*, ***kwargs*)

Called for object creation

Parameters

- **cls** (*automatic*) – The class which is going to be instantiated
- **args** – To be passed to __new__ for class instantiation
- **kwargs** – To be passed to __new__ for class instantiation

Returns obj, args, kwargs as a tuple

Note that in this method the 1st return value is no the 1st passed argument (unlike in the rest of methods) It is the created instance and not the passed class

The return values need not be the same that were passed

_init_pre(*obj*, **args*, ***kwargs*)

Called after object creation and before the object is init'ed

Parameters

- **cls** (-) – The class which has been instantiated
- **obj** (-) – The class instance which has been created
- **args** (-) – To be passed to __init__ for object initialization
- **kwargs** (-) – To be passed to __init__ for object initialization

Returns obj, args, kwargs as a tuple

The return values need not be the same that were passed

_init_do(*obj*, **args*, ***kwargs*)
Called for object initialization

Parameters

- **cls** (-) – The class which has been instantiated
- **obj** (-) – The class instance which has been created
- **args** (-) – To be passed to `__init__` for object initialization
- **kwargs** (-) – To be passed to `__init__` for object initialization

Returns obj, args, kwargs as a tuple

The return values need not be the same that were passed

_init_post(*obj*, **args*, ***kwargs*)
Called after object initialization

Parameters

- **cls** (-) – The class which has been instantiated
- **obj** (-) – The class instance which has been created
- **args** (-) – Which were passed to `__init__` for object initialization
- **kwargs** (-) – Which were passed to `__init__` for object initialization

Returns obj, args, kwargs as a tuple

The return values need not be the same that were passed. But modifying `args` and/or `kwargs` no longer plays a role because the object has already been created and initialized

__call__(**args*, ***kwargs*)
Creates an initializes an instance of cls calling the pre-new, pre-init/post-init hooks with the passed/returned `args` / `kwargs`

classmethod as_metaclass(*meta*, **bases*)
Create a base class with "this metaclass" as metaclass

Meant to be used in the definition of classes for Py2/3 syntax equality

Parameters **bases** – a list of base classes to apply (object if none given)

class `metaframe`.**MetaFrameBase**
Enables a class to MetaFrame-enabled through inheritance without having to specify/declare a metaclass

CHAPTER 4

Indices and tables

- genindex
- modindex
- search

Symbols

A

M