

Practical Metaclasses and Decorators

Sim Zacks

Principal Engineer, Red Hat

2-May-2016

Agenda

- Metaclasses
- Decorators
- Practical Usage



Metaclasses

Metaclasses

```
>>> class MyClass(object):
...     def __init__(self):
...         self.x = 5
...
...     def add(self):
...         self.x += 1
...
>>> mc = MyClass()
>>> type(mc)
<class '__main__.MyClass'>
>>> type(MyClass)
<type 'type'>
```

Metaclasses

```
>>> type(5)
<type 'int'>
```

```
>>> type(type(5))
<type 'type'>
```

```
>>> type([])
<type 'list'>
```

```
>>> type(type([]))
<type 'type'>
```

```
>>> type(type)
<type 'type'>
```

Metaclasses

```
def __init__(self):  
    self.x = 5
```

```
def add(self):  
    self.x += 1
```

```
NewClass = type(  
    "NewClass", (), {"__init__": __init__, "add": add})
```

```
>>> nc = NewClass()
```

```
>>> nc.x
```

```
5
```

```
>>> nc.add()
```

```
>>> nc.x
```

```
6
```

Metaclasses

```
class NewType(type):  
    def __new__(cls, name, bases, attrs):  
        print "%s was created" % name  
        return super(NewType, cls). \  
            __new__(cls, name, bases, attrs)
```

```
>>> NewClass2 = NewType("NewClass2", (),  
    {"__init__": __init__, "add": add})
```

```
NewClass2 was created
```

```
>>> nc2 = NewClass2()
```

```
>>> nc2.x
```

```
5
```

```
>>> nc2.add()
```

```
>>> nc2.x
```

```
6
```

Metaclasses

```
class MyClass(object):  
    __metaclass__ = NewType  
    def __init__(self):  
        self.x = 5  
  
    def add(self):  
        self.x += 1
```


Decorators

Decorators

```
from django.views.decorators.http \
    import require_GET

@require_GET
def my_view(request):
    # Only GET requests make it this far.
    # POSTs will fail
    pass
```

Decorators

```
def require_GET(func):
    @wraps(func)
    def inner(request, *args, **kwargs):
        if request.method != "GET":
            return HttpResponseNotAllowed \
                (request_method_list)
        return func(request, *args, **kwargs)
    return inner
```

Decorators

```
def my_dec(func):  
    def inner(*args, **kwargs):  
        ...  
        return func(*args, **kwargs)  
    return inner
```

```
@my_dec  
def a(a, b, c):  
    pass  
  
>>> a.__name__  
inner
```

Decorators

```
from functools import wraps
def my_dec(func):
    @wraps(func)
    def inner(*args, **kwargs):
        print "func %s" % func.__name__
        res = func(*args, **kwargs)
    return inner
```

```
>>> a.__name__
a
```

Decorators

```
>>> def f(a, b, c):  
...     d = 1  
...     e = 2  
  
...  
>>> f.func_code.co_varnames  
( 'a', 'b', 'c', 'd', 'e' )  
>>> f.func_code.co_varnames \  
...     [:f.func_code.co_argcount]  
( 'a', 'b', 'c' )
```

Decorators

```
@my_dec  
def a(a, b, c):  
    pass
```

```
>>> "b" in a.func_code.co_varnames \  
      [:a.func_code.co_argcount]
```

```
False
```

```
>>> a.func_code.co_varnames  
( 'args', 'kwargs' )
```

Decorators

```
try:
    from decorator import decorate
except ImportError:
    raise ImportError("decorator>=4.0.9 is
                       required")

def inner(func, *args, **kwargs):
    ...

def my_dec(func):
    decorate(func, inner)
```


Decorators

```
>>> "b" in a.func_code.co_varnames \
..     [:a.func_code.co_argcount]
True

>>> a.__name__
a
```

Practical Usage

Practical Usage

Requirement

- Send all usage/exception data to logstash server

Environment

- Library with > 50 classes and $> 10,000$ lines of code
- All classes are descendants of a single base class

Practical Usage

```
import logstash
_logger = logging.getLogger('python-logstash-logger')
_logger.setLevel(logging.INFO)
_logger.addHandler(
    logstash.LogstashHandler(
        logstash_url,
        logstash_port,
        version=1))
data = {}
data["XYZ"] = class_name
...
_logger.info(SYSTEM_NAME, extra=data)
```

Practical Usage

```
class BaseClass()
    def send_usage_data(self, *args, **kwargs):
        ...

class A(BaseClass)
    def yadayadayada():
        ...
        self.send_usage_data(
            user=self.username, ...)
        ...
```

Practical Usage

```
def send_usage_data(func):  
    def inner(*arg, **kwargs):  
        send_data(...)  
        try:  
            return func(...)  
        except:  
            send_error_data  
            raise
```

Practical Usage

```
class A(BaseClass)
    @send_usage_data
    def yadayadayada():
        ...
```

Practical Usage

```
class LoggingMeta(type):
    def __new__(cls, name, bases, attrs):
        logstash_parms = {}
        for item, attr_val in attrs.items():
```


Practical Usage

```
if isinstance(attr_val, types.FunctionType):
    attrs[item] = log_wrapper(attr_val)

elif isinstance(attr_val, classmethod):
    attrs[item] = classmethod(
        log_wrapper(attr_val.
            __get__(object).__func__))

elif item.startswith("_logstash"):
    logstash_parms[item] = attr_val
```

Practical Usage

```
if logstash_parms:
    init_logger(**logstash_parms)
return super(LoggingMeta, cls). \
    __new__(cls, name, bases, attrs)
```

Practical Usage

```
def my_logger(func, *args, **kwargs):  
    ...  
    _logger.info("SYSTEM", extra=data)  
    try:  
        return func(*args, **kwargs)  
    except Exception, e:  
        data["v_errormsg"] = e  
        _logger.error("SYSTEM", extra=data)  
        raise  
  
def log_wrapper(func):  
    return decorate(func, my_logger)
```

Practical Usage

```
Class BaseClass(object):  
    __metaclass__ = LoggingMeta  
    _logstash_url = ...  
    ...
```

Practical Usage

Metaclass Example Code

https://github.com/simzacks/random_code/blob/master/logstasher.py



THANK YOU



plus.google.com/+RedHat



facebook.com/redhatinc



linkedin.com/company/red-hat



twitter.com/RedHatNews



youtube.com/user/RedHatVideos

Q&A



redhat.

We're Looking for you!

<https://www.redhat.com/en/jobs>

Contact me:

sim@redhat.com