Python 元类

wtforms源码

单例模式

用模块、框架实现业务功能，作为扩展的知识点。

创建类

```python
# 定义类
class Foo(object):
    def __init__(self, name):
        self.name = name
    def __new__(cls, *args, **kwargs):
        return object.__new__(cls)

# 根据类创建对象
# 1 执行类的new方法，创建空对象  【构造方法】 {}
# 2 执行类的init方法，初始化对象 【初始化方法】{name:"luffy"}
obj = Foo("luffy")
```

对象是基础类创建的。

问题：类是谁创建的?

答案：类是由type创建。

```python
# 传统方式创建类
class Foo(object):
    v1 = 123
    def func(self):
        return 666
# 非传统方式创建类
Foo = type("Foo", (object,),{"v1": 123, "func":lambda self:666})
# 非传统方式创建对象
obj = Foo()
# 非传统方式调用v1的变量
print(obj.v1)
```

```python
# 传统方式创建类(直观)
"""
class Foo(object):
    v1 = 123

    def func(self):
        return 666
print(Foo)
"""
# 非传统方式（一行）
# 1 创建类型
#   - 类名
#   - 继承类
#   - 成员
Fa = type("Foo", (object,), {"v1":123, "func": lambda self:666, "do":do})
# 2 根据类创建对象
obj = Fa()
# 3 调用对象中的v1变量
print(obj.v1)
# 4 执行对象中的func方法
result = obj.func()
```

类默认是以type创建，怎么让伊特类的创建改成其他的东西（元类）。

```python
# type 创建Foo类
class Foo(object):
    pass
# 其他的东西创建类
class Foo(object, metaclass=其他的东西)
    pass
```

```python
class MyType(type):
    pass

class Foo(object, metaclass=MyType):
    pass
# Foo类由MyType创建
```

```python
class MyType(type):
    def __init__(self, *args, **kwargs):
        super().__init__(*args, **kwargs)

    def __new__(cls, *args, **kwargs):
        new_cls = super().__new__(cls, *args, **kwargs)
        return new_cls

    def __call__(self, *args, **kwargs):
        # 调用自己的那个类 __new__ 方法去创建对象
        empty_object = self.__new__(self)
        # 调用你自己的__init__ 方法取初始化
        self.__init__(empty_object, *args, **kwargs)
        return empty_object

class Foo(object, metaclass=MyType):
    def __init__(self, name):
        self.name = name

# 假设Foo是一个对象 由MyType创建
# Foo其实是MyType的一个对象
# Foo() -> MyType对象()
v1 = Foo("alex")
print(v1)
print(v1.name)
```

wtforms源码

```python
from wtforms import Form
from wtforms.fields import simple
class LoginForm(Form) :
    name = simple.StringField(label='用户名', render_kw={'class': 'form-control'})
    pwd = simple.PasswordField(label= '密码', render_kw={'class':'form-control'})

form = LoginForm( )
print(form.name) #类变量
print(form.pwd) #类变量
```

Python 复制代码

```python
from wtforms import Form
from wtforms.fields import simple

class FormMeta(type):
    def __init__(cls, name, bases, attrs):
        type.__init__(cls, name, bases, attrs)
        cls._unbound_fields = None
        cls._Wtforms_meta = None

    def __call__(cls, *args, **kwargs):
        """
        Construct a new Form instance .
        Creates the unbound_ fields list and the internal wtforms_ meta
        subclass of the class Meta in order to allow a proper inher itance
        hierarchy.
        """
        if cls._unbound_fields is None:
            fields = []
            for name in dir(cls):
                if not name.startswith('_'):
                    unbound_field = getattr(cls, name)
                    if hasattr (unbound_field, '_formfield'):
                        fields. append( (name, unbound_field))
            # We keep the name as the. second element of the sort
            # to ensure a stable sort.
            fields .sort (key=lambda x:(x[1].creation_ounter, x[0]))
            cls._unbound_fields = fields
        # Create a subclass of the 'class Meta' using all the ancestors .
        if cls._wtforms_meta is None:
            bases=[]
            for mro_class in cls.__mro__:
                if 'Meta' in mro_class.__dict__:
                    bases.append(mro_class.Meta)
            cls._wtforms_meta = type('Meta' , tuple(bases), {})
        return type.__call__(cls, *args, **kwargs)

def with_metaclass(meta, base=object):
    # FormMeta("NewBase". (BaseForm,), {} )
    # type( "NewBase", ( BaseForm,), {} )
    return meta("NewBase", (base,), {})
"""
class NewBase ( BaseForm, metaclass=FormMeta):
    pass
class Form( NewBase):
"""
```

```python

class Form(with_metaclass(FormMeta, BaseForm)):
    pass
# LoginForm其实是由FormMeta 创建的。
# 1.创建类时，会执行FormMeta 的__new__ 和__init__，内部在类中添加了两个类变量 _unbound_fields 和_wtforms_meta
class LoginForm(Form):
    name = simple.StringField(label='用户名', render_kw={'class': ' form-control' })
    pwd = simple.PasswordField(labe1='密码', render_kw={'class': 'form-control'})
# 2.根据LoginForm类去创建对象。FormMeta._ call___ 方法 -> LoginForm中的new去创建对象，init去初始化对象。
form = LoginForm( )
print( form.name) # 类变量
print(form.pwd) # 类变量
#  问题1:此时LoginForm是由 type or FormMeta创建?
"""
类中metaclass,自己类由于metaclass定义的类来创建。
类继承某个类，父类metaclass, 自己类由于metaclass定义的类来创建。
"""
```

在学习元类之后，在：

- 类创建，自定义功能
- 对象的创建前后，自定义功能

单例模式

元类的方式

Python | 复制代码

```python
class MyType(type):
    def __init__(self, name, bases, attrs):
        super().__init__(name, bases, attrs)
        self.instance = None

    def __call__(self, *args, **kwargs):
        # 1判断是否有对象，  有不穿件
        if not self.instance:
            self.__init__(self.instance, *args, **kwargs)
        return self.instance


class Singleton(object, metaclass=MyType):
    pass


class Foo1(Singleton, metaclass=MyType):
    pass


class Foo2(Singleton):
    pass

v1 = Foo1()
v2 = Foo1()

print(v1)
print(v2)
```