Python 进阶

目录

*args 和 **kwargs

当我们重写一个方法

```
class Model(ModelBase):
    ....
    ....
    def save(self, ob, force_insert=False, force_update=False, using=None,
             update_fields=None):
        ....
        ....
```

```
class DjangoModel(models.Model):
    ....
    ....
    def save(self, ob, force_insert=False, force_update=False, using=None,
             update_fields=None):
        do_something()   # 内容被修改之前之前做一些事情
        return super(DjangoModel, self).delete(ob, force_insert=False,
            force_update=False, using=None, update_fields=None)
```

最好的方式

```python
class DjangoModel(models.Model):
    ....
    ....
    def save(self, *args, **kwargs):
        do_something()   # 内容被修改之前之前做一些事情
        return super(DjangoModel, self).delete(*args, **kwargs)
```

闭包&装饰器

闭包

```
def _wrap_with(code):

    def inner(text, bold=False):
        c = code
        if bold:
            c = "1;%s" % c
        return "\033[%sm%s\033[0m" % (c, text)
    return inner

red = _wrap_with('31')
green = _wrap_with('32')
yellow = _wrap_with('33')

# 首先得有一个嵌套的函数，即函数中定义了另一个函数
# 闭包则是一个集合，它包括了外部函数的局部变量
# 这些局部变量在外部函数返回后也继续存在，并能被内部函数引用。
```

Decorator(装饰器)就是*闭 包 实 现 的*

Decorator(装饰器)

```python
from colors import red

def warn_output(function):
    """警告消息装饰器"""

    def deco(*args, **kwargs):
        output = function(*args, **kwargs)
        return red(output)

    return deco


@warn_output
def disk_avail():
    return "剩余: 10%"
```

带参数的装饰器

```
def with_color_message(color_function):
    def wrapper(function):
        """警告消息装饰器"""

        def deco(*args, **kwargs):
            output = function(*args, **kwargs)
            return color_function(output)

        return deco
    return wrapper


@with_color_message(red)
def disk_avail_warn():
    return "10%"

@with_color_message(green)
def disk_avail_info():
    return "80%
```

嵌套装饰器

```
from colors import red, green

def with_color_message(color_function):
    def wrapper(function):
        def deco(*args, **kwargs):
            output = function(*args, **kwargs)
            return color_function(output)
        return deco
    return wrapper

def with_auto_set_level(function):
    def deco(*args, **kwargs):
        avail = function(*args, **kwargs)
        if avail < 60:
            return "%s%% OK!" % avail
        else:
            return "%s%% WARN!" % avail
    return deco

@with_color_message(red)
@with_auto_set_level
def disk_avail_warn():
    return 10
```

更多有用的装饰器

@retry(重试装饰器)

@trace(跟踪函数处理DEBUG)

@timeout(超时装饰器)

decorator_useful.py

Lambda

Lambda 匿名函数

```
def mul(x):
    return x * x


mul2 = lambda x: x * x


句式

lambda 参数,参数: 表达式
```

Map & Filter

Map

```
def mul(x):
    return x * x

# 第一种方式 - 小学生
results = []
for n in xrange(100):
    results.append(mul(n))


# 第二种方式 - 普通
results = [mul(x) for x in xrange(100)]

# 第三种方式 - 专家
results = map(mul, xrange(100))
```

```
# 第四种方式 - 神
results = map(lambda x: x*x, xrange(100))
```

Filter

```
odds = filter(lambda x: x % 2 == 0, xrange(100))

# [0, 2, 4, 6, 8, 10, 12, 14, 16, 18, 20,
# 22, 24, 26, 28, 30, 32, 34, 36, 38, 40,
# 42, 44, 46, 48, 50, 52, 54, 56, 58, 60,
# 62, 64, 66, 68, 70, 72, 74, 76, 78, 80,
# 82, 84, 86, 88, 90, 92, 94, 96, 98]
```

三元运算符

简单的判断

```
#如果条件为真，返回真 否则返回假
condition_is_true if condition else condition_is_false


is_fat = True
state = "fat" if is_fat else "not fat"


# 晦涩一点的用法比较少见，它使用了元组，请继续看：

(if_test_is_false, if_test_is_true)[test]

fat = True
fitness = ("skinny", "fat")[fat]
print("Ali is ", fitness)
#输出: Ali is fat
```

容器(Collections)

Collections

- defaultdict
- counter
- deque
- namedtuple
- enum.Enum(python 3.4)

Defaultdict

```python
import json
from collections import defaultdict

colours = (
    ('张三', 'Yellow'),
    ('李四', 'Blue'),
    ('王五', 'Green'),
    ('李四', 'Black'),
    ('张三', 'Red'),
    ('铁柱', 'Silver'),
    )

favourite_colours = defaultdict(list)

for name, colour in colours:
    favourite_colours[name].append(colour)
```

```
{
  "张三": [
    "Yellow",
    "Red"
  ],
  "李四": [
    "Blue",
    "Black"
  ],
  "王五": [
    "Green"
  ],
  "铁柱": [
    "Silver"
  ]
}


# 当Key重复的时候， 直接对values形成list
```

Defaultdict - 案例

```
some_dict = {}
some_dict['colours']['favourite'] = "yellow"

## 异常输出: KeyError: 'colours'
Traceback (most recent call last):
  File "<stdin>", line 1, in <module>
KeyError: 'colours'
```

```
import json
from collections import defaultdict

tree = lambda: defaultdict(tree)
some_dict = tree()
# 这里不会有KeyError
some_dict['colours']['favourite'] = "yellow"
some_dict['person']['name'] = "伊丽莎白 - 铁柱"
# 可以使用json.dumps 来打印some_dict
print(json.dumps(some_dict, ensure_ascii=False))
# {"colours": {"favourite": "yellow"}}
```

Counter

```
>>> from collections import Counter
>>>
>>> colours = (
...     ('Yasoob', 'Yellow'),
...     ('Ali', 'Blue'),
...     ('Arham', 'Green'),
...     ('Ali', 'Black'),
...     ('Yasoob', 'Red'),
...     ('Ahmed', 'Silver'),
... )
>>>
>>> favs = Counter(name for name, colour in colours)
>>> print(favs)
Counter({'Yasoob': 2, 'Ali': 2, 'Arham': 1, 'Ahmed': 1})
```

Counter 是个计数器

它可以帮助我们针对某项数据进行统计

deque

```
'''
deque提供了一个双端队列，你可以从头/尾两端添加或删除元素。
要想使用它，首先我们要从collections中导入deque模块
'''
>>> from collections import deque
>>> d = deque()
>>> d.append('1')
>>> d.append('2')
>>> d.append('3')
>>> print(len(d))
3
>>> print(d[0])
1
>>> d = deque(range(5))
>>> print(len(d))
5
>>> d.popleft()
0
>>> d.pop()
4
```

deque - 案例

```
# 当队列元素超过maxlen先前追加的元素将会被挤掉.
>>> d = deque(maxlen=30)
>>> [ d.append(i) for i in range(50) ]
>>> d
deque([20,
 21, 22, 23, 24, 25, 26, 27, 28, 29, 30,
 31, 32, 33, 34, 35, 36, 37, 38, 39, 40,
 41, 42, 43, 44, 45, 46, 47, 48, 49], maxlen=30)

# 你也可以很方便的从两端扩展元素
>>> d = deque([1,2,3,4,5])
>>> d.extendleft([0])
>>> d.extend([6,7,8])
>>> print(d)
deque([0, 1, 2, 3, 4, 5, 6, 7, 8])
```

namedtuple

您可能已经熟悉元组。
一个元组是一个不可变的列表，你可以存储一个数据的序列，
它和命名元组(namedtuples)非常像，但有几个关键的不同。
主要相似点是都不像列表，你不能修改元组中的数据。
为了获取元组中的数据，你需要使用整数作为索引

```
man = ('Ali', 30)
print(man[0])
## 输出: Ali
```

嗯，那namedtuples是什么呢？它把元组变成一个针对简单任务的容器。
你不必使用整数索引来访问一个namedtuples的数据。
你可以像字典(dict)一样访问namedtuples，但namedtuples是不可变的。

```
from collections import namedtuple
Animal = namedtuple('Animal', 'name age type')
perry = Animal(name="perry", age=31, type="cat")
print(perry)
## 输出: Animal(name='perry', age=31, type='cat')
print(perry.name)
## 输出: 'perry'
print(perry._asdict())
## OrderedDict([('name', 'perry'), ('age', 11), ('type', 'cat')])
```

对象自省

dir

```
my_list = [1, 2, 3]
dir(my_list)
# Output: ['__add__', '__class__', '__contains__', '__delattr__', '__delitem__',
# '__delslice__', '__doc__', '__eq__', '__format__', '__ge__', '__getattribute__',
# '__getitem__', '__getslice__', '__gt__', '__hash__', '__iadd__', '__imul__',
# '__init__', '__iter__', '__le__', '__len__', '__lt__', '__mul__', '__ne__',
# '__new__', '__reduce__', '__reduce_ex__', '__repr__', '__reversed__', '__rmul__',
# '__setattr__', '__setitem__', '__setslice__', '__sizeof__', '__str__',
# '__subclasshook__', 'append', 'count', 'extend', 'index', 'insert', 'pop',
# 'remove', 'reverse', 'sort']
```

type & id

```
# type() 函数返回一个对象的类型
>>> print(type(''))
<type 'str'>
>>> print(type([]))
<type 'list'>
>>> print(type({}))
<type 'dict'>
>>> print(type(dict))
<type 'type'>
>>> print(type(3))
<type 'int'>

# id()函数返回任意不同种类对象的唯一ID
>>> name = "Yasoob"
>>> print(id(name))
4525331200
```

inspect 模块

```
# inspect模块也提供了许多有用的函数，来获取活跃对象的信息。比方说，你可以查看一个对象的成员
>>> import inspect
>>> print(inspect.getmembers(perry))
[('__add__', <method-wrapper '__add__' of Animal object at 0x10dc194c8>),
('__class__', <class '__main__.Animal'>),
('__contains__', <method-wrapper '__contains__' of Animal object at 0x10dc194c8>),
('__delattr__', <method-wrapper '__delattr__' of Animal object at 0x10dc194c8>),
('__dict__', OrderedDict([('name', 'perry'), ('age', 11), ('type', 'cat')])),
('__doc__', 'Animal(name, age, type)'),
('__eq__', <method-wrapper '__eq__' of Animal object at 0x10dc194c8>),
('__format__', <built-in method __format__ of Animal object at 0x10dc194c8>),
('__ge__', <method-wrapper '__ge__' of Animal object at 0x10dc194c8>),
('__getattribute__', <method-wrapper '__getattribute__' of Animal object at 0x10dc194c8>),
('__getitem__', <method-wrapper '__getitem__' of Animal object at 0x10dc194c8>),
....
```

一行式

简单的web服务器

```
# Python 2
$ python -m SimpleHTTPServer

# Python 3
$ python -m http.server
```

漂亮的JSON

```
$ cat file.json | python -m json.tool
```

更多模块: http://pythonwise.blogspot.com/2015/01/python-m.html

CSV转JSON

```
# CSV转换为json

python -c "import csv,json;print json.dumps(list(csv.reader(open('csv_file.csv'))))"

# 确保csv_file.csv 是需要转换的csv文件
```

列表辗平

```
a_list = [[1, 2], [3, 4], [5, 6]]
print(list(itertools.chain.from_iterable(a_list)))
# Output: [1, 2, 3, 4, 5, 6]

# or
print(list(itertools.chain(*a_list)))
# Output: [1, 2, 3, 4, 5, 6]
```

上下文管理器

Context Managers

```
with open('some_file', 'w') as opened_file:
    opened_file.write('Hello!')

# 等同于

file = open('some_file', 'w')
try:
    file.write('Hello!')
finally:
    file.close()
```

创建一个上下文类

```
# 一个上下文管理器的类，最起码要定义__enter__和__exit__方法。

class File(object):
    def __init__(self, file_name, method):
        self.file_obj = open(file_name, method)
    def __enter__(self):
        return self.file_obj
    def __exit__(self, type, value, traceback):
        self.file_obj.close()


# 使用

with File('demo.txt', 'w') as opened_file:
    opened_file.write('Hello!')
```

推荐学习网站

寻找库

- Google
- Github(代码托管网站)
- PyPI(The python packages index)

文档站

- stackoverflow(程序设计领域问答网站)
- segmentfault(国内的stackoverflow)
- gitbook.com
- readthedocs.org

本教程脚本案例

http://192.168.2.34/junqi.gao/python-intro/tree/master

Q&A

The End，Thanks!