[笔记][Learning Python][8. 高级主题]

Python

37. Unicode 和字节串

37.4 编写 Unicode 字符串

转换编码

`Python 3.X` 的文件对象（用 `open` 内置函数创建的）在读取文本字符串的时候自动地编码它们，并且在写入文本字符串的时候自动解码它们。

> 注：这里编码和解码采用的字符集是随着用户平台而变化的。
> 可以用如下方式查看。

```
>>> import locale
>>> locale.getpreferredencoding(False)
'cp936'
>>> help(locale.getpreferredencoding)
Help on function getpreferredencoding in module locale:

getpreferredencoding(do_setlocale=True)
    Return the charset that the user is likely using.
```

> 至于为什么要加一个 `False` 参数，详见这个链接。
> https://stackoverflow.com/questions/23743160/locale-getpreferredencoding-why-does-this-reset-string-letters

---

38. 被管理的属性

38.2 描述符

基础知识

```
class Descriptor:
    "docstring goes here"
    def __get__(self, instance, owner): ...   # Return attr value
    def __set__(self, instance, value): ...   # Return nothing (None)
    def __delete__(self, instance): ...       # Return nothing (None)
```

带有 `__get__`、`__set__`、`__delete__` 这些特殊方法的类都可以看作描述符，并且当它们的一个实例被赋值给另一个类的属性的时候，当访问属性的时候，就会自动调用这些方法。

> 个人记忆：
>
> - 这些方法都要有 `self`，指的是**描述符对象**
> - 这些方法第二个参数都是 `instance`，指的是**被描述对象**
> - `get` 还有一个 `owner`，指的是**被描述类**
> - `set` 还有一个 `value`，指的是要**设置的值**

38.3 `__getattr__` 和 `__getattribute__`

基础知识
可以用来实现 *委托设计模式*。

```
class Wrapper:
    def __init__(self, obj):
        self.wrapped = obj

    def __getattr__(self, attrname):
        print('Trace: ' + attrname)
        return getattr(self.wrapped, attrname)

x = Wrapper([1, 2, 3])
x.append(4)
print(x.wrapped)
"""
Trace: append
[1, 2, 3, 4]
"""
```

38.4 示例：属性验证

使用 `property` 验证
`validate_properties.py`

```
class CardHolder(object):
    acctlen = 8
    retireage = 59.5

    def __init__(self, acct, name, age, addr):
        self.acct = acct
        self.name = name
        self.age = age
        self.addr = addr

    def getName(self):
        return self.__name
    def setName(self):
        value = value.lower().replace(' ', '_')
        self.__name = value
    name = property(getName, setName)

    def getAge(self):
        return self.__age
    def setAge(self, value):
        if value < 0 or value > 150:
```

```
            raise ValueError('invalid age')
        else:
            self.__age = value
    age = property(getAge, setAge)

    def getAcct(self):
        return self.__acct[:-3] + '***'
    def setAcct(self, value):
        value = value.replace('-', '')
        if len(value) != self.acctlen:
            raise TypeError('invalid acct nubmer')
        else:
            self.__acct = value
    acct = property(getAcct, setAcct)

    def remainGet(self):
        return self.retireage - self.age
    remain = property(remainGet)
```

测试代码

传入类所在模块的名称（不带后缀）作为命令行的唯一参数。

`validate_tester.py`

```
from __future__ import print_function

def loadclass():
    import sys, importlib
    modulename = sys.argv[1]
    module = importlib.import_module(modulename)
    print('[Using: %s]' % module.CardHolder)
    return module.CardHolder

def printholder(who):
    print(who.acct, who.name, who.age, who.remain, who.addr, sep=' / ')

if __name__ == '__main__':
    CardHolder = loadclass()
    bob = CardHolder('1234-5678', 'Bob Smith', 40, '123 main st')
    printholder(bob)
    bob.name = 'Bob Q. Smith'
    bob.age = 50
    bob.acct = '23-45-67-89'
    printholder(bob)

    sue = CardHolder('5678-12-34', 'Sue Jones', 35, '124 main st')
    printholder(sue)
    try:
        sue.age = 200
    except:
```

```
            print('Bad age for Sue')

        try:
            sue.remain = 5
        except:
            print("Can't set sue.remain")

        try:
            sue.acct = '1234567'
        except:
            print('Bad acct for Sue')
```

运行结果：

```
PS D:\code\learningpython\C38> py validate_tester.py validate_properties
[Using: <class 'validate_properties.CardHolder'>]
12345*** / bob_smith / 40 / 19.5 / 123 main st
23456*** / bob_q._smith / 50 / 9.5 / 123 main st
56781*** / sue_jones / 35 / 24.5 / 124 main st
Bad age for Sue
Can't set sue.remain
Bad acct for Sue
```

使用描述符验证

`property` 基本上是描述符的一种受限制的形式。

选项一：使用共享的描述符实例状态的验证。

`validate_descriptors1.py`

```
class CardHolder(object):
    acctlen = 8
    retireage = 59.5

    def __init__(self, acct, name, age, addr):
        self.acct = acct
        self.name = name
        self.age = age
        self.addr = addr

    class Name(object):
        def __get__(self, instance, owner):
            return self.name
        def __set__(self, instance, value):
            value = value.lower().replace(' ', '_')
            self.name = value
    name = Name()

    class Age(object):
```

```
        def __get__(self, instance, owner):
            return self.age
        def __set__(self, instance, value):
            if value < 0 or value > 150:
                raise ValueError('invalid age')
            else:
                self.age = value
    age = Age()

    class Acct(object):
        def __get__(self, instance, owner):
            return self.acct[:-3] + '***'
        def __set__(self, instance, value):
            value = value.replace('-', '')
            if len(value) != instance.acctlen:
                raise TypeError('invalid acct number')
            else:
                self.acct = value
    acct = Acct()

    class Remain(object):
        def __get__(self, instance, owner):
            return instance.retireage - instance.age
        def __set__(self, instance, value):
            raise TypeError('cannot set remain')
    remain = Remain()
```

测试结果

```
PS D:\code\learningpython\C38> py validate_tester.py validate_descriptors
1
[Using: <class 'validate_descriptors1.CardHolder'>]
12345*** / bob_smith / 40 / 19.5 / 123 main st
23456*** / bob_q._smith / 50 / 9.5 / 123 main st
56781*** / sue_jones / 35 / 24.5 / 124 main st
Bad age for Sue
Can't set sue.remain
Bad acct for Sue
```

在这个例子中实际的 `name` 值附加到了描述符对象，而不是客户类实例。

`validate_tester2.py`

```
from __future__ import print_function

from validate_tester import loadclass
CardHolder = loadclass()

bob = CardHolder('1234-5678', 'Bob Smith', 40, '123 main st')
```

```
print('bob:', bob.name, bob.acct, bob.age, bob.addr)

sue = CardHolder('5678-12-34', 'Sue Jones', 35, '124 main st')
print('sue:', sue.name, sue.acct, sue.age, sue.addr)
print('bob:', bob.name, bob.acct, bob.age, bob.addr)
```

测试结果：

```
PS D:\code\learningpython\C38> py validate_tester2.py validate_descriptor
s1
[Using: <class 'validate_descriptors1.CardHolder'>]
bob: bob_smith 12345*** 40 123 main st
sue: sue_jones 56781*** 35 124 main st
bob: sue_jones 56781*** 35 123 main st
```

选项二：使用基于客户实例状态的验证

`validate_descriptors2.py`

```
class CardHolder(object):
    acctlen = 8
    retireage = 59.5

    def __init__(self, acct, name, age, addr):
        self.acct = acct
        self.name = name
        self.age = age
        self.addr = addr

    class Name(object):
        def __get__(self, instance, owner):
            return instance.__name

        def __set__(self, instance, value):
            value = value.lower().replace(' ', '_')
            instance.__name = value
    name = Name()

    class Age(object):
        def __get__(self, instance, owner):
            return instance.__age

        def __set__(self, instance, value):
            if value < 0 or value > 150:
                raise ValueError('invalid age')
            else:
                instance.__age = value
    age = Age()
```

```
    class Acct(object):
        def __get__(self, instance, owner):
            return instance.__acct[:-3] + '***'

        def __set__(self, instance, value):
            value = value.replace('-', '')
            if len(value) != instance.acctlen:
                raise TypeError
            else:
                instance.__acct = value
    acct = Acct()

    class Remain(object):
        def __get__(self, instance, owner):
            return instance.retireage - instance.age

        def __set__(self, instance, value):
            raise TypeError('cannot set remain')
    remain = Remain()
```

测试结果如下：

```
PS D:\code\learningpython\C38> py validate_tester.py validate_descriptors
2
[Using: <class 'validate_descriptors2.CardHolder'>]
12345*** / bob_smith / 40 / 19.5 / 123 main st
23456*** / bob_q._smith / 50 / 9.5 / 123 main st
56781*** / sue_jones / 35 / 24.5 / 124 main st
Bad age for Sue
Can't set sue.remain
Bad acct for Sue
PS D:\code\learningpython\C38> py validate_tester2.py validate_descriptor
s2
[Using: <class 'validate_descriptors2.CardHolder'>]
bob: bob_smith 12345*** 40 123 main st
sue: sue_jones 56781*** 35 124 main st
bob: bob_smith 12345*** 40 123 main st
```

注意：这一版本不支持通过类的描述符访问，只支持通过客户实例来访问。

```
>>> from validate_descriptors1 import CardHolder
>>> bob = CardHolder('1234-5678', 'Bob Smith', 40, '123 main st')
>>> bob.name
'bob_smith'
>>> CardHolder.name
'bob_smith'

>>> from validate_descriptors2 import CardHolder
```

```
>>> bob = CardHolder('1234-5678', 'Bob Smith', 40, '123 main st')
>>> bob.name
'bob_smith'
>>> CardHolder.name
AttributeError: 'NoneType' object has no attribute '_Name__name'
```

使用 `__getattr__` 验证

`validate_getattr.py`

> 注意这个版本是 `2` 和 `3` 通用的，所以不用写成新式类。
> `__getattr__` 很早就有了。

```
class CardHolder:
    acctlen = 8
    retireage = 59.5

    def __init__(self, acct, name, age, addr):
        self.acct = acct
        self.name = name
        self.age = age
        self.addr = addr

    def __getattr__(self, name):
        if name == 'acct':
            return self._acct[:-3] + '***'
        elif name == 'remain':
            return self.retireage - self.age
        else:
            raise AttributeError(name)

    def __setattr__(self, name, value):
        if name == 'name':
            value = value.lower().replace(' ', '_')
        elif name == 'age':
            if value < 0 or value > 150:
                raise ValueError('invalid age')
        elif name == 'acct':
            name = '_acct'
            value = value.replace('-', '')
            if len(value) != self.acctlen:
                raise TypeError('invalid acct number')
        elif name == 'remain':
            raise TypeError('cannot set remain')
        self.__dict__[name] = value
```

测试结果：

```
PS D:\code\learningpython\C38> py validate_tester.py validate_getattr
```

```
[Using: <class 'validate_getattr.CardHolder'>]
12345*** / bob_smith / 40 / 19.5 / 123 main st
23456*** / bob_q._smith / 50 / 9.5 / 123 main st
56781*** / sue_jones / 35 / 24.5 / 124 main st
Bad age for Sue
Can't set sue.remain
Bad acct for Sue
PS D:\code\learningpython\C38> py validate_tester2.py validate_getattr
[Using: <class 'validate_getattr.CardHolder'>]
bob: bob_smith 12345*** 40 123 main st
sue: sue_jones 56781*** 35 124 main st
bob: bob_smith 12345*** 40 123 main st
```

使用 `__getattribute__` 验证

`validate_getattribute.py`

```
class CardHolder(object):
    acctlen = 8
    retireage = 59.5

    def __init__(self, acct, name, age, addr):
        self.acct = acct
        self.name = name
        self.age = age
        self.addr = addr

    def __getattribute__(self, name):
        superget = object.__getattribute__
        if name == 'acct':
            return superget(self, 'acct')[:-3] + '***'
        elif name == 'remain':
            return superget(self, 'retireage') - superget(self, 'age')
        else:
            return superget(self, name)

    def __setattr__(self, name, value):
        if name == 'name':
            value = value.lower().replace(' ', '_')
        elif name == 'age':
            if value < 0 or value > 150:
                raise ValueError('invalid age')
        elif name == 'acct':
            value = value.replace('-', '')
            if len(value) != self.acctlen:
                raise TypeError('invalid acct number')
        elif name == 'remain':
            raise TypeError('cannot set remain')
        self.__dict__[name] = value
        # object.__setattr__(self, name, value)
```

测试结果：

```
PS D:\code\learningpython\C38> py validate_tester.py validate_getattribut
e
[Using: <class 'validate_getattribute.CardHolder'>]
12345*** / bob_smith / 40 / 19.5 / 123 main st
23456*** / bob_q._smith / 50 / 9.5 / 123 main st
56781*** / sue_jones / 35 / 24.5 / 124 main st
Bad age for Sue
Can't set sue.remain
Bad acct for Sue

PS D:\code\learningpython\C38> py validate_tester2.py validate_getattribu
te
[Using: <class 'validate_getattribute.CardHolder'>]
bob: bob_smith 12345*** 40 123 main st
sue: sue_jones 56781*** 35 124 main st
bob: bob_smith 12345*** 40 123 main st
```

---

39. 装饰器

39.1 什么是装饰器

39.5 直接管理函数和类

它们也可以用来管理函数和类对象自身，而不只是管理对它们随后的调用。

39.6 示例：“私有” 和 “公有” 属性

`access1.py`

```
"""
File access1.py (3.X + 2.X)
Privacy for attributes fetched from class instances.
See self-test code at end of file for a usage example.
Decorator same as: Doubler = Private('data', 'size')(Doubler).
Private returns onDecorator, onDecorator returns onInstance,
and each onInstance instance embeds a Doubler instance.
```

```
"""

traceMe = False


def trace(*args):
    if traceMe:
        print('[' + ' '.join(map(str, args)) + ']')


# Private 实际上是个带参装饰器，所以有三层
# 1. Private 函数，返回一个装饰器
# 2. onDecorator 函数，返回一个类
# 3. onInstance 类，被装饰的类实例化的时候实际上运行的就是它
# privates 是一个元组，里面放着所有私有属性
def Private(*privates):          # privates in enclosing scope
    def onDecorator(aClass):     # aClass in enclosing scope
        class onInstance:        # wrapped in instance attribute
            def __init__(self, *args, **kargs):
                # 把 Doubler 类的实例包装在 onInstance 实例属性 wrapped 上
                self.wrapped = aClass(*args, **kargs)

            # 这里会拦截类外面取属性的操作
            # 比如 label，data
            def __getattr__(self, attr):    # My attrs don't call getattr
                trace('get:', attr)         # Others assumed in wrapped
                if attr in privates:  # 如果该属性在私有属性元组中
                    # 抛出异常
                    raise TypeError('private attribute fetch: ' + attr)
                else:  # 如果该属性没有在私有属性元组，直接从被包装的对象中取属性
                    return getattr(self.wrapped, attr)

            # 这里会拦截类外面设置属性的操作
            def __setattr__(self, attr, value):    # Outside accesses
                trace('set:', attr, value)          # Others run normally
                # 如果设置的是 wrapped
                # 说明是初始化的时候，onInstance 实例需要设置这个属性
                # 那么不要拦截，通过属性字典设置
                if attr == 'wrapped':               # Allow my attrs
                    self.__dict__[attr] = value     # Avoid looping
                # 如果设置的属性在私有属性元组中
                elif attr in privates:
                    # 抛出异常
                    raise TypeError('private attribute change: ' + attr)
                else:
                    # 否则的话，不是私有属性，把设置操作路由（分发）给 self.wrapp
ed
                    # 即 Doubler 实例
                    setattr(self.wrapped, attr, value)  # Wrapped obj att
rs
        return onInstance  # Or use __dict__
    return onDecorator
```

```
if __name__ == '__main__':
    traceMe = True

    # 这里设置了两个私有属性，data 和 size
    @Private('data', 'size')     # Doubler = Private(...)(Doubler)
    class Doubler:
        # 类内部的属性操作不会被拦截
        # 因为类内部的 self 就是 Doubler 实例，而不是 onInstance 实例
        # Doubler 所有属性的赋值与取值都是开放的
        def __init__(self, label, start):
            self.label = label   # Accesses inside the subject class
            self.data = start    # Not intercepted: run normally

        def size(self):
            return len(self.data)  # Methods run with no checking

        def double(self):          # Because privacy not inherited
            for i in range(self.size()):
                self.data[i] = self.data[i] * 2

        def display(self):
            print('%s => %s' % (self.label, self.data))

    # 实际上是调用了 onInstance 类进行创建
    # 创建过程中调用了 onInstance 类的初始化方法 __init__
    # 设置 self.wrapped 的时候触发了 onInstance 类的 __setattr__ 方法
    X = Doubler('X is', [1, 2, 3])  # label 被设置成 X is，data 被设置成 [1,
 2, 3]
    Y = Doubler('Y is', [-10, -20, -30])  # label 被设置成 Y is，data 被设置
成 [-10, -20, -30]
    print()
    """
    [set: wrapped <__main__.Doubler object at 0x000001726FA74278>]
    [set: wrapped <__main__.Doubler object at 0x000001726FA742E8>]
    """

    # The following all succeed
    # 外部访问，触发 onInstance 的 __getattr__
    # 不是私有属性，正常输出
    print(X.label)  # Accesses outside subject class
    print()
    """
    [get: label]
    X is
    """

    X.display(); X.double(); X.display()  # Intercepted: validated, deleg
ated
    print()
    """
```

```
    [get: label]
    X is
    [get: display]
    X is => [1, 2, 3]
    [get: double]
    [get: display]
    X is => [2, 4, 6]
    """

    print(Y.label)
    print()
    """
    [get: label]
    Y is
    """

    Y.display()
    Y.double()
    Y.label = 'Spam'
    Y.display()
    """
    [get: display]
    Y is => [-10, -20, -30]
    [get: double]
    [set: label Spam]
    [get: display]
    Spam => [-20, -40, -60]
    """

    # The following all fail properly
    """
    print(X.size())  # prints "TypeError: private attribute fetch: size"
    print(X.data)
    X.data = [1, 1, 1]
    X.size = lambda S: 0
    print(Y.data)
    print(Y.size())
    """
```

本代码有个小 `bug` ：

```
# 在外部所有对属性的访问都会被拦截（除了在外部直接对 wrapped 属性的访问）
# 所以这里其实是有个 bug 的
# 比如我可以通过 wrapped 来访问私有属性 size 和 data
print(X.wrapped.size())
```

39.7 示例：验证函数参数

> 下面是我自己写的版本

`rangetest.py`

```
def rangetest(*, percent):
    def decorator(func):
        def wrapper(self, given_percent):
            if given_percent < percent[0] or given_percent > percent[1]:
                raise ValueError('invalid percentage')
            res = func(self, given_percent)
            return res
        return wrapper
    return decorator


class Person:
    @rangetest(percent=(0.0, 1.0))
    def giveRaise(self, percent):
        self.pay = int(self.pay * (1 + percent))


if __name__ == '__main__':
    p = Person()
    p.pay = 100
    p.giveRaise(1.1)
    print(p.pay)

"""
Traceback (most recent call last):
  File "rangetest.py", line 21, in <module>
    p.giveRaise(1.1)
  File "rangetest.py", line 5, in wrapper
    raise ValueError('invalid percentage')
ValueError: invalid percentage
"""
```

关键字和默认参数的推广

> 这里用到了我不太会的东西。

`__code__` 属性，存储着一个函数的代码对象 `code object`。
代码对象 `code object` 是一小段可运行的 `Python` 代码在 `CPython` 内部的表示，它可以是函数、模块、类或者生成器表达式。

`co_nlocals`：函数的局部变量个数，等价于 `len(code_obj.co_varnames)`。
`co_argcount`：函数接收的参数个数，不包括 `*args`、强制关键字参数和 `**kwargs`。
`co_varnames`：它是存放着函数所有局部变量名的元组，包括参数。它首先包含普通参数，然后是 `*args` 和 `**kwargs` 的名字，然后是按照使用顺序排列的其他局部变量的名字。

```
"""
File rangetest.py: function decorator that performs range-test
validation for arguments passed to any function or method.
Arguments are specified by keyword to the decorator. In the actual
call, arguments may be passed by position or keyword, and defaults
may be omitted. See rangetest_test.py for example use cases.
"""
trace = True

# 调用装饰器的时候，只接收关键字参数
def rangetest(**argchecks):
    def onDecorator(func):
        if not __debug__:
            return func
        # 其实这个 else 可以省略，然后减少缩进层级
        else:
            code = func.__code__
            # co_varnames 元组里面是函数所有局部变量的名称
            # 不包括 *args，强制关键字参数和 **kwargs
            # 被装饰函数和类没有定义 *args、强制关键字参数以及 **kwargs
            # 所以这里取到的就是所有局部变量的名称
            # 存放顺序首先是可调用对象的参数，然后是其他的局部变量
            # co_argcount 属性是参数个数
            # 所以这么取出来的 allargs 就是所有参数
            allargs = code.co_varnames[:code.co_argcount]
            funcname = func.__name__

            def onCall(*pargs, **kargs):
                expected = list(allargs)  # 把切片出来的元组转成列表
                # 用 pargs 的长度切片，剔除默认参数和关键字参数
                # 得到所有位置参数
                positionals = expected[:len(pargs)]

                # **argchecks，其中 argchecks 是一个字典
                # items 得到 (key, value) 这样的一个键值对
                for (argname, (low, high)) in argchecks.items():
                    # 先看这个参数名是不是以关键字形式指定的
                    if argname in kargs:
                        if kargs[argname] < low or kargs[argname] > high:
                            errmsg = '{0} argument "{1}" not in {2}..{3}'
                            errmsg = errmsg.format(funcname, argname, lo
w, high)
                            raise TypeError(errmsg)

                    # 如果不是以关键字形式指定的
                    elif argname in positionals:
```

```
                        position = positionals.index(argname)
                        if pargs[position] < low or pargs[position] > hig
h:
                            # 这里其实可以提取冗余代码封装成函数
                            errmsg = '{0} argument "{1}" not in {2}..{3}'
                            errmsg = errmsg.format(funcname, argname, lo
w, high)
                            raise TypeError(errmsg)
                    else:
                        if trace:
                            print('Argument "{0}" defaulted'.format(argna
me))
                # onCall 就是装饰器运行后实际被调用的东西
                # 首先进行了参数检查
                # 检查没问题之后返回原来函数调用
                return func(*pargs, **kargs)
            # 这里要返回包装器 Wrapper
            # 也就是最终函数会变成这个
            return onCall
    # 最外层返回一个装饰器
    return onDecorator
```

---

40. 元类

40.2 元类模型

> `type` 和 `object` 互为实例。但是它们都是 `type` 类派生出来的？

```
>>> isinstance(type, object)
True
>>> isinstance(object, type)
True
>>> type.__class__
<class 'type'>
>>> object.__class__
<class 'type'>
```

40.5 继承与实例

`metainstance.py`

```
class MetaOne(type):
    def __new__(meta, classname, supers, classdict):
        print('In MetaOne.new:', classname)
        return type.__new__(meta, classname, supers, classdict)

    def toast(self):
        return 'toast'


class Super(metaclass=MetaOne):
    def spam(self):
        return 'spam'


class Sub(Super):
    def eggs(self):
        return 'eggs'


if __name__ == '__main__':
    X = Sub()
    print()
    """
    In MetaOne.new: Super
    In MetaOne.new: Sub
    """

    print(Sub.toast)
    print(Sub.spam)
    # 这里书上写的是 Sub.spam，应该是版本更新有改动
    # spam 方法的前缀现在是定义它的类!
    print(X.spam)
    """
    <bound method MetaOne.toast of <class '__main__.Sub'>>
    <function Super.spam at 0x0000021CD06861E0>
    <bound method Super.spam of <__main__.Sub object at 0x0000021CD068AD6
8>>
    """
```

描述符特例

```
class D:
    def __get__(self, instance, owner): print('__get__')

    def __set__(self, instance, value): print('__set__')
```

```
class C:
    d = D()


I = C()
I.d
print()
"""
__get__
"""

I.d = 1
print()
"""
__set__
"""

I.__dict__['d'] = 'spam'
I.d
print()
"""
__get__
"""

print(I.__dict__)
"""
{'d': 'spam'}
"""
```

---

41. 一切美好的事物

`massive tome`：大部头
`tome`：卷，册，大而重的书