魔法属性

笔记本：	Python提高-2		
创建时间：	2018/5/8 20:11	**更新时间：**	2018/5/8 20:57
作者：	ly		
标签：	__init__ 双下划线		

Python中存在一些具有特殊含义的属性

- **__doc__　　表示类的描述信息**　(doc为world的文件后缀）

```
class Foo:
    """描述类信息"""
    def func(self):
        pass


print(foo.__doc__)      # "描述类信息"          输出类的描述信息
```

- **__module__ ，表示当前操作的是哪个对应的模块　　__class__表示当前操作的的对象的类是哪个**

test.py文件里面的代码：

```
class Person(object):
    def __init__(self):
        self.name = "ly"
```

main.py文件里面的代码：

```
from test import Person

obj = Person()
print(obj.__module__)      # 输出  test    即：输出模块名
print(obi.__class__)        # 输出 test.Person    即：输出类
```

- **__dict__ 类或对象中的所有属性 （即：详细信息）**

```
class Province(object):
    def __init__(self, name, count):
        self.name = name
        self.count = count

    def func(self, *args, **kwargs):
        print("func")


print(Province.__dict__)      # 获取类的属性，即类属性、类方法


obj1 = Province("山东", 1000)   # 创建对象
print(obj1.__dict__)        # 获取对象obj1的属性
```

```
# 输出：{'count':1000, 'name':'山东'}
```

- **__init__　初始化方法，通过类创建对象时，自动触发执行**

```
class Person:
    def __init__(self, name):
        self.name = name
        self.age = 18

obj = Person("ly")      # 自动执行类中的__init__方法
```

- **__del__　当对象在内存中要被释放时，自动触发执行。**

注：此方法一般无须定义，因为Python是一门高级语言，程序员在使用时无需关心内存的分配和释放，因为此工作都是交给Python解释器来执行，所以，__del__的调用是由解释器在进行垃圾回收时自动触发执行的。

```
class foo:
    def __del__(self):
        print("我要走咯")      # 当类对象要释放内存的时候就会执行
```

- **__str__　类中定义了__str__方法，打印 对象 时，默认输出该方法的返回值。**

```
class Foo:
    def __str__(self):
        return "hello"


obj = Foo()
print(obj)      # 输出：hello
```

- **__call__　对象后面加括号，触发执行**

与__init__区别：
_init__方法的执行是由创建对象触发的，即：对象 = 类名() ；
__call__ 方法的执行是由对象后加括号触发的，即：对象() 或者 类()()

```
class Foo:
    def __init__(self):
        pass

    def __call__(self, *args, **kwargs):
        print("__call__")


obj = Foo()   # 执行__init__
obj()         # 执行__call__
```

- **__getitem__、__setitem__、__delitem__(了解)**

用于索引操作，如字典。以上分别表示获取、设置、删除数据

```
class Foo(object):

    def __getitem__(self, key):
        print('__getitem__', key)

    def __setitem__(self, key, value):
        print('__setitem__', key, value)

    def __delitem__(self, key):
        print('__delitem__', key)


obj = Foo()

result = obj['k1']      # 自动触发执行 __getitem__
obj['k2'] = 'laowang'   # 自动触发执行 __setitem__
del obj['k1']           # 自动触发执行 __delitem__
```

- **__getslice__、__setslice__、__delslice__(了解)**

该三个方法用于分片操作，如：列表

```
class Foo(object):

    def __getslice__(self, i, j):
        print('__getslice__', i, j)

    def __setslice__(self, i, j, sequence):
        print('__setslice__', i, j)

    def __delslice__(self, i, j):
        print('__delslice__', i, j)

obj = Foo()

obj[-1:1]                   # 自动触发执行 __getslice__
obj[0:1] = [11,22,33,44]    # 自动触发执行 __setslice__
del obj[0:2]                # 自动触发执行 __delslice__
```