python_常用的特殊属性和方法

1.__slots__属性
用于限定一个类所拥有的属性，以及能否动态添加或删除属性
注意：父类有的属性，子类继承一定会有(即使子类的__slots__中没有限制)
使用方法：
```
class A(object):
    __slots__=['属性1'，'属性2'，'__dict__']
```
Note:如果不加'__dict__'那么A就只能有属性1和属性2，不能动态添加属性

2.dir非常重要的解析属性方法
可以将一些对象的可用属性方法全部列出来，然后可以试着调用其中的一些魔法方法

```
C:\Users\guoqingpei>python
Python 3.4.3 (v3.4.3:9b73f1c3e601, Feb 24 2015, 22:43:06) [MSC v.1600 32 bit (Intel)] on win32
Type "help", "copyright", "credits" or "license" for more information.
>>> a =1
>>> dir(a)
['__abs__', '__add__', '__and__', '__bool__', '__ceil__', '__class__', '__delattr__', '__dir__', '__divmod__', '__doc__'
, '__eq__', '__float__', '__floor__', '__floordiv__', '__format__', '__ge__', '__getattribute__', '__getnewargs__', '__g
t__', '__hash__', '__index__', '__init__', '__int__', '__invert__', '__le__', '__lshift__', '__lt__', '__mod__', '__mul_
_', '__ne__', '__neg__', '__new__', '__or__', '__pos__', '__pow__', '__radd__', '__rand__', '__rdivmod__', '__reduce__',
 '__reduce_ex__', '__repr__', '__rfloordiv__', '__rlshift__', '__rmod__', '__rmul__', '__ror__', '__round__', '__rpow__'
, '__rrshift__', '__rshift__', '__rsub__', '__rtruediv__', '__rxor__', '__setattr__', '__sizeof__', '__str__', '__sub__'
, '__subclasshook__', '__truediv__', '__trunc__', '__xor__', 'bit_length', 'conjugate', 'denominator', 'from_bytes', 'im
ag', 'numerator', 'real', 'to_bytes']
>>> a = -1
>>> a.__abs__
<method-wrapper '__abs__' of int object at 0x5BF6D1B0>
>>> a.__abs__()
1
```

dir的运行过程是:
如果当前对象的类定义了__slots__，没有定义__dict__,那么就查当前类的父类中的__dict__,依次类推获取当前对象所有的可用值

3.python对象的__dict__属性
__dict__属性用来动态添加属性和方法，所以__slots__中如果添加了__dict__就具备了动态添加属性和方法的能力
如果一个类的__slots__定义中没有__dict__,那么这个类的对象中是没有__dict__方法的
如果一个类定义了__dir__方法，那么这个类的对象dir列出来的始终是这个类定义__dir__方法的返回值
但是dir()始终是可以列出的

4.__str__方法
用来定义一个对象被print调用时以怎么样的形式体现
__repr__=__str__(__repr__方法是在命令行交互界面时，输入A()显示的内容)
使用示例：
```
class A(object):
    def __str__(self):
        return "Object A"
    __repr__=__str__
```

5.__iter__方法
可以将一个对象在迭代器中使用：
只需要在__iter__方法中返回一个可迭代的对象，这个对象含有__next__方法
```
class Fibs(object):
    def __init__(self,count =0):
        self.a = 0
        self.b = 1
        self.num = 0

    def __iter__(self):
        return self

    def __next__(self):
        if self.a<10:
            self.a ,self.b = self.b,self.a+self.b
            return self.a
        else:
            raise StopIteration
//添加此方法支持Fibs()[i]取值
    def __getitem__(self,i):
        self.num = 0
```

```
        print(isinstance(i,int))
        if(isinstance(i,int)):
            while(self.num<=i ):
                self.a ,self.b = self.b,self.a+self.b
                self.num+=1
            return self.a
        if(isinstance(i,slice)): //支持Fibs()[start:end]列表切片
            sliceList = []
            start = i.start
            end = i.stop
            while(self.num<=end):
                self.a ,self.b = self.b,self.a+self.b
                if(self.num>=start and self.num<end):
                    sliceList.append(self.a)
                self.num+=1
        return sliceList

fibs = Fibs()
for fibNum in fibs:
    print(fibNum)
fibs[0]
fibs[1]
```

6.__getitem__,__setitem__

__getitem__:当想用下标访问某个对象的元素时(如Object[index])，会自动调用__getitem(self,i)__方法

__setitem__:当想通过下标给对象的元素赋值时(如Object[index] = obj2),会自动调用__setitem__(self,key,val)方法

实例：

```
class Dict(dict):
    def __init__(self,*args,**kwargs):
        super(Dict,self).__init__(*args,**kwargs)

    def __getitem__(self,key):
        if key not in self.keys():
            print("Does not exist key:{key}".format_map(locals()))
            super(Dict,self).__setitem__(key,-1)
        return  super(Dict,self).__getitem__(key) ##特别注意：在__getitem__的内部不能有Object[index]的形式存在，否则会递归调用__getitem__

    def __setitem__(self,key,val):
        if isinstance(key,str):
            return
        super(Dict,self).__setitem__(key,val)##特别注意：在__setitem__的内部不能有Object[key]=val的形式存在，否则会递归调用__setitem__
```

7.__getattr__

当想用.的方式(Object.attr)来获取一个属性时就会自动调用对象的__getattr__方法

可以用于动态添加属性，下面的示例用于动态生成一个Rest API的路径

Note:一般在没有找到属性时，

注意，只有在没有找到属性的情况下，才调用__getattr__，已有的属性，比如name，不会在__getattr__中查找。

会抛出raise AttributeError("this attribute does not exist")异常

```
class Chain(object):

    def __init__(self,path=''):
        self.path = path

    def __getattr__(self,path):
        return Chain('%s/%s'%(self.path,path))

    def __str__(self):
        return self.path

    def __call__(self):
        print("this is a Chain Object")

if __name__=="__main__":
    chain = Chain()
    print(chain.str.a.b.root)
```

8.__call__
可以将一个类的实例(对象)作为一个函数调用
在上面的Chain类中，我们添加了一个__call__方法，于是就可以在调用时采用:Chain()()来打印
注意：__call__方法不是类方法，只能被类的实例(对象调用)，第一个括号Chain()用来生成类的实例，第二个()用来调用__call__方法

9.__contains__
判断一个值是否在该类实例中：会被in调用

```
class A(object):

    def __init__(self):
        pass

    def __contains__(self,val): #被in调用
        if val:
            return True
        return False

if __name__=="__main__":
    print("function Name:{}".format(0 in A()))
```

10.Python中的上下文情景管理器
with：进入时调用__enter__ 返回该情景所需要的变量, 退出时(包括异常退出时)调用__exit__(self,exc_type,exc_value,traceback)
实例：用上下文情景构建文件读写上下文管理(事实上:python语言内部实现就是这么做的)

```
class FileOpen:

    def __init__(self, filename, mode):
        self.filename = filename
        self.mode = mode

    def __enter__(self):
        self.openedFile = open(self.filename, self.mode)
        return self.openedFile

    def __exit__(self, *unused):
        self.openedFile.close()

with FileOpen('testFile.txt', 'w+') as f:
    f.write("Hello World from our new Context Manager!")
```

参考文档：
http://www.jb51.net/article/55734.htm
http://blog.csdn.net/business122/article/details/7568446