魔法方法

1. 被两条下划线所包含的方法叫做魔法方法
        这些方法在进行特定的操作时会自动调用，他们是Python面向对象的结晶
        __init__(self):
        __call__(self):
2. 魔法方法是Python的一切
        底层

- __init__()

1. __init__(self):
        初始化，构造方法
2. 返回值只能是None

- __new__()

__new__(cls,[参数，参数2])
1. 第一次被执行的方法
2. 创建对象时自动被调用一次
        比init方法先执行
3. 作用
        创建对象
4. new方法会将参数进一步传递给init
        d=Dog(name,age,color)
        1. __new__(d,name,age,color)
        2. __init__(name,age,color)
5. new方法一般情况不要修改
6. new方法执行，init不一定并执行

```
# 测试的是否执行了new方法
    class Mystr(str):
        def __new__(cls,string):
            #将string转成大写
            string=string.upper()
            print(string)


    m=Mystr('xiao')
# 测试线执行new后执行init
    class Mystr:

        def __init__(self):
            print('调用了init')

        def __new__(cls):
```

```
            print('调用了new')


            return object.__new__(cls)


    m=Mystr()
-------------------------------------
# init方法不一定实现
    class Mystr:

        def __init__(self):
            print('调用了init')

        def __new__(cls):
            print('调用了new')
            print(cls)

            return object.__new__(A)


    m=Mystr()
```

- __del__()

```
析构函数
1. 定义类里面，如果不写Python后台会提供默认的析构函数
2. 在对象的生命周期结束时，被调用
3. 可以释放内存空间

class TestClass:
    def __init__(self):
        print('init')


    def __del__(self):
        print(1111111111)



t=TestClass()
a=t
```

- __str__()

```
当str()被调用时会执行__str__()
__str__实际上是被print函数默认调用的，当print(实例对象)，默认调用str

class A:
```

```
    def __init__(self,name):
        self.name=name


class Mystr:  #str类
    def __init__(self,name):
        self.name=name

    def __str__(self):
        return self.name

    def __repr__(self):
        return self.name

a=A('Tom')
print(a)
b=Mystr('Jake')   #str('jake') ---- Mystr('jake')
print(b)
```

- __getattr__(self , item)

```
当调用一个不存在的属性时会调用，如果属性存在则不调用
class A:
    def __init__(self,name):
        self.name=name
    def __getattr__(self,item):
        print('执行了')
        return '属性不存在'
```

- __setattr__(self , name,value)

```
所有的属性设置都会调用这个方法，并且拥有这个魔法方法时才可以设置属性，注意防止发生递归

class A:
    def __init__(self,name):
        self.name=name
    def __setattr__(self,name,value):
        print('111111')
        object.__setattr__(self,name,value)



a=A('Tom')
print(a.name)
```

- getattribute(self.item)

```
访问属性时会调用（存在，或不存在都调用）
class A:
    def __init__(self,name):
        self.name=name
    def __setattr__(self,name,value):
        print('111111')
        object.__setattr__(self,name,value)

    def __getattribute__(self,item):
        print('222222222222')

a=A('Tom')
print(a.name)
```

算数相关的魔法方法

- list() tuple() str()

```
1. python2.2版本之前有类和类型的概念
        a=10  int ---类型
2. Python2.2之后，没有了类型的概念
        a=10  int----int对象
  1+1  ---数据   --二进制10   int类无关
  a=10   int(10)   为了使数据变成对象进行操作---万物皆对象

  a+1=11   对象+数据 魔法方法： __add__()
```

```
1. __add__()  --  +
2. mul   ---  *
3. sub   ---  -
4. truedicv --- /
5. floordiv ---  //
6. mod  --- %
7. pow  --- **
8. lshift ---<<
9. rshift --->>
10. and  ---  &
11. xor --- ^
12. or  ---
        |
```

```
class Myint(int):
    def __add__(self,other):
        print('执行了')
        return int.__add__(self,other)


a=Myint(100)
b=Myint(200)

print(a+b)
```

- 反运算

```
指更改方法的调用主动性
a+b===主动性在a
b+a===主动性在b
1+a===主动性不在1，在a

__r*__(self,other)


class Myint(int):
    def __add__(self,other):
        print('执行了')
        return int.__add__(self,other)

    def __radd__(self,other):
        print('radd执行了')
        return int.__radd__(self,other)
a=Myint(100)
b=Myint(200)

print(1+a)
```