魔法方法

Python 里有一种方法，叫做魔法方法。Python 的类里提供的，两个下划线开始，两个下划线结束的方法，就是魔法方法，魔法方法在恰当的时候就会被激活，自动执行。 魔法方法的两个特点：

- 两侧各有两个下划线；
- "咒语"名字已经由 Python 官方定义好，我们不能乱写。

1.__init__方法

`__init__()` 方法，在创建一个对象时默认被调用，不需要手动调用。在开发中，如果希望在创建对象的同时，就设置对象的属性，可以对 `__init__` 方法进行改造。

```
class Cat:
    """这是一个猫类"""
    def __init__(self,name):  # 重写了 __init__ 魔法方法
        self.name = name

    def eat(self):
        return "%s爱吃鱼"%self.name
    def drink(self):
        return '%s爱喝水'%self.name

    """
        tom = Cat()
        TypeError: __init__() missing 1 required positional argument: 'name'
        这种写法在运行时会直接报错！因为 __init__ 方法里要求在创建对象时，必须要传递 name 属性，如果不传入会直接报错！
    """

tom = Cat("Tom")  # 创建对象时，必须要指定name属性的值
tom.eat()   # tom爱吃鱼
```


注意：

1. `__init__()` 方法在创建对象时，会默认被调用，不需要手动的调用这个方法。
2. `__init__()` 方法里的self参数，在创建对象时不需要传递参数，python解释器会把创建好的对象引用直接赋值给self
3. 在类的内部，可以使用self来使用属性和调用方法；在类的外部，需要使用对象名来使用属性和调用方法。
4. 如果有多个对象，每个对象的属性是各自保存的，都有各自独立的地址。
5. 方法是所有对象共享的，只占用一份内存空间，方法被调用时会通过self来判断是哪个对象调用了实例方法。

2.__del__方法

创建对象后，python解释器默认调用 `__init__()` 方法；

而当删除对象时，python解释器也会默认调用一个方法，这个方法为 `__del__()` 方法。

```
class Student:
    def __init__(self,name,score):
        print('__init__方法被调用了')
        self.name = name
        self.score = score

    def __del__(self):
        print('__del__方法被调用了')
s = Student('lisi',95)
del s
input('请输入内容')
```

3.__str__方法

`__str__` 方法返回对象的描述信息，使用 `print()` 函数打印对象时，其实调用的就是这个对象的 `__str__` 方法。

```
class Cat:
    def __init__(self,name,color):
        self.name = name
        self.color = color

tom = Cat('Tom','white')

# 使用 print 方法打印对象时，会调用对象的 __str__ 方法，默认会打印类名和对象的地址名
print(tom)   # <__main__.Cat object at 0x0000021BE3B9C940>
```

如果想要修改对象的输出的结果，可以重写 `__str__` 方法。

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

    def __str__(self):
        return '哈哈'

p  = Person('张三',18)
print(p)   # 哈哈  打印对象时，会自动调用对象的 __str__ 方法
```

一般情况下，我们在打印一个对象时，可能需要列出这个对象的所有属性。

```
class Student:
    def __init__(self,name,score):
        self.name = name
        self.score = score
    def __str__(self):
        return '姓名是:{},成绩是{}分'.format(self.name,self.score)

s = Student('lisi',95)
print(s)   # 姓名是:lisi,成绩是95分
```

4. __repr__方法

`__repr__` 方法和 `__str__` 方法功能类似，都是用来修改一个对象的默认打印内容。在打印一个对象时，如果没有重写 `__str__` 方法，它会自动来查找 `__repr__` 方法。如果这两个方法都没有，会直接打印这个对象的内存地址。

```
class Student:
    def __init__(self, name, score):
        self.name = name
        self.score = score

    def __repr__(self):
        return 'helllo'


class Person:
    def __repr__(self):
        return 'hi'

    def __str__(self):
        return 'good'


s = Student('lisi', 95)
print(s)  # hello

p = Person()
print(p)  # good
```


5. __call__方法

对象后面加括号，触发执行。

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

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


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

总结

1. 当创建一个对象时，会自动调用 `__init__` 方法，当删除一个对象时，会自动调用 `__del__` 方法。
2. 使用 `__str__` 和 `__repr__` 方法，都会修改一个对象转换成为字符串的结果。一般来说， `__str__` 方法的结果更加在意可读性，而 `__repr__` 方法的结果更加在意正确性(例如:datetime模块里的datetime类)