← Python OS 文件/目录方法

Python 面向对象 →

Python super() 函数

Python 内置函数

描述

super() 函数是用于调用父类(超类)的一个方法。

super 是用来解决多重继承问题的，直接用类名调用父类方法在使用单继承的时候没问题，但是如果使用多继承，会涉及到查找顺序（MRO）、重复调用（钻石继承）等种种问题。

MRO 就是类的方法解析顺序表, 其实也就是继承父类方法时的顺序表。

语法

以下是 super() 方法的语法:

```
super(type[, object-or-type])
```

参数

- type -- 类。
- object-or-type -- 类，一般是 self

Python3.x 和 Python2.x 的一个区别是: Python 3 可以使用直接使用 **super().xxx** 代替 **super(Class, self).xxx** :

Python3.x 实例:

```
class A:
    def add(self, x):
        y = x+1
        print(y)
class B(A):
    def add(self, x):
        super().add(x)
b = B()
b.add(2)  # 3
```

Python2.x 实例:

分类导航

HTML / CSS
JavaScript
服务端
数据库
移动端
XML 教程
ASP.NET
Web Service
开发工具
网站建设

Python 高级教程


```
#!/usr/bin/python
# -*- coding: UTF-8 -*-

class A(object):  # Python2.x 记得继承 object
  def add(self, x):
     y = x+1
     print(y)
class B(A):
  def add(self, x):
    super(B, self).add(x)
b = B()
b.add(2)  # 3
```

返回值

无。

实例

以下展示了使用 super 函数的实例：

实例

```
#!/usr/bin/python
# -*- coding: UTF-8 -*-

class FooParent(object):
  def __init__(self):
    self.parent = 'I\'m the parent.'
    print ('Parent')

  def bar(self,message):
    print ("%s from Parent" % message)

class FooChild(FooParent):
  def __init__(self):
    # super(FooChild,self) 首先找到 FooChild 的父类（就是类 FooParent），然后把类 FooChild 的对象转换为类 FooParent 的对象
    super(FooChild,self).__init__()
    print ('Child')

  def bar(self,message):
    super(FooChild, self).bar(message)
    print ('Child bar fuction')
    print (self.parent)

if __name__ == '__main__':
  fooChild = FooChild()
  fooChild.bar('HelloWorld')
```

执行结果：

```
Parent
Child
```

```
HelloWorld from Parent
Child bar fuction
I'm the parent.
```

Python 内置函数

Python OS 文件/目录方法 Python 面向对象

2 篇笔记 写笔记

直接用类名调用父类方法在使用单继承的时候没问题，但是如果使用多继承，会涉及到查找顺序（MRO）。一开始看到这句话，不太理解。

48

看了这个文章后，明白了。

我们在学习 Python 类的时候，总会碰见书上的类中有 __init__() 这样一个函数，很多同学百思不得其解，其实它就是 Python 的构造方法。

构造方法类似于类似 init() 这种初始化方法，来初始化新创建对象的状态，在一个对象呗创建以后会立即调用，比如像实例化一个类：

```
f = FooBar()
f.init()
```

使用构造方法就能让它简化成如下形式：

```
f = FooBar()
```

你可能还没理解到底什么是构造方法，什么是初始化，下面我们再来举个例子：

```
class FooBar:
    def __init__(self):
        self.somevar = 42

>>>f = FooBar()
>>>f.somevar
```

我们会发现在初始化 FooBar 中的 somevar 的值为 42 之后，实例化直接就能够调用 somevar 的值；如果说你没有用构造方法初始化值得话，就不能够调用，明白了吗？

在明白了构造方法之后，我们来点进阶的问题，那就是构造方法中的初始值无法继承的问题。

例子：

```
class Bird:
    def __init__(self):
```

```
        self.hungry = True
    def eat(self):
        if self.hungry:
            print 'Ahahahah'
        else:
            print 'No thanks!'

class SongBird(Bird):
    def __init__(self):
        self.sound = 'Squawk'
    def sing(self):
        print self.song()

sb = SongBird()
sb.sing()  # 能正常输出
sb.eat()  # 报错，因为 songgird 中没有 hungry 特性
```

那解决这个问题的办法有两种：

1、调用未绑定的超类构造方法（多用于旧版 python 阵营）

```
class SongBird(Bird):
    def __init__(self):
        Bird.__init__(self)
        self.sound = 'Squawk'
    def sing(self):
        print self.song()
```

原理：在调用了一个实例的方法时，该方法的self参数会自动绑定到实例上（称为绑定方法）；如果直接调用类的方法（比如Bird.__init__），那么就没有实例会被绑定，可以自由提供需要的self参数（未绑定方法）。

2、使用super函数（只在新式类中有用）

```
class SongBird(Bird):
    def __init__(self):
        super(SongBird,self).__init__()
        self.sound = 'Squawk'
    def sing(self):
        print self.song()
```

原理：它会查找所有的超类，以及超类的超类，直到找到所需的特性为止。

sme000 7个月前 (04-23)

43

经典的菱形继承案例，BC 继承 A，然后 D 继承 BC，创造一个 D 的对象。

```
    ---> B ---
A --|       |--> D
    ---> C ---
```

使用 super() 可以很好地避免构造函数被调用两次。

```
# 思考题正确答案


class A():
  def __init__(self):
    print('enter A')
    print('leave A')


class B(A):
  def __init__(self):
    print('enter B')
    super().__init__()
    print('leave B')


class C(A):
  def __init__(self):
    print('enter C')
    super().__init__()
    print('leave C')


class D(B, C):
  def __init__(self):
    print('enter D')
    super().__init__()
    print('leave D')


d = D()
```

执行结果是：

```
enter D
enter B
enter C
enter A
leave A
leave C
leave B
leave D
```

yyYanis 5个月前 (06-11)