Python3 错误和异常 Python3 标准库概览

Python3 面向对象

Python从设计之初就已经是一门面向对象的语言，正因为如此，在Python中创建一个类和对象是很容易的。本章节我们将详细介绍Python的面向对象编程。

如果你以前没有接触过面向对象的编程语言，那你可能需要先了解一些面向对象语言的一些基本特征，在头脑里头形成一个基本的面向对象的概念，这样有助于你更容易的学习Python的面向对象编程。

接下来我们先来简单的了解下面向对象的一些基本特征。

面向对象技术简介

- **类(Class):** 用来描述具有相同的属性和方法的对象的集合。它定义了该集合中每个对象所共有的属性和方法。对象是类的实例。
- **方法：**类中定义的函数。
- **类变量：**类变量在整个实例化的对象中是公用的。类变量定义在类中且在函数体之外。类变量通常不作为实例变量使用。
- **数据成员：**类变量或者实例变量用于处理类及其实例对象的相关的数据。
- **方法重写：**如果从父类继承的方法不能满足子类的需求，可以对其进行改写，这个过程叫方法的覆盖（override），也称为方法的重写。
- **局部变量：**定义在方法中的变量，只作用于当前实例的类。
- **实例变量：**在类的声明中，属性是用变量来表示的。这种变量就称为实例变量，是在类声明的内部但是在类的其他成员方法之外声明的。
- **继承：**即一个派生类（derived class）继承基类（base class）的字段和方法。继承也允许把一个派生类的对象作为一个基类对象对待。例如，有这样一个设计：一个Dog类型的对象派生自Animal类，这是模拟"是一个（is-a）"关系（例图，Dog是一个Animal）。
- **实例化：**创建一个类的实例，类的具体对象。
- **对象：**通过类定义的数据结构实例。对象包括两个数据成员（类变量和实例变量）和方法。

和其它编程语言相比，Python 在尽可能不增加新的语法和语义的情况下加入了类机制。

Python中的类提供了面向对象编程的所有基本功能：类的继承机制允许多个基类，派生类可以覆盖基类中的任何方法，方法中可以调用基类中的同名方法。

对象可以包含任意数量和类型的数据。

类定义

语法格式如下：

```
class ClassName:
<statement-1>
.
.
```

```
.
<statement-N>
```

类实例化后，可以使用其属性，实际上，创建一个类之后，可以通过类名访问其属性。

类对象

类对象支持两种操作：属性引用和实例化。

属性引用使用和 Python 中所有的属性引用一样的标准语法：**obj.name。**

类对象创建后，类命名空间中所有的命名都是有效属性名。所以如果类定义是这样:

实例(Python 3.0+)

```
#!/usr/bin/python3
class MyClass:
"""一个简单的类实例"""
i = 12345
def f(self):
return 'hello world'
# 实例化类
x = MyClass()
# 访问类的属性和方法
print("MyClass 类的属性 i 为: ", x.i)
print("MyClass 类的方法 f 输出为: ", x.f())
```

以上创建了一个新的类实例并将该对象赋给局部变量 x，x 为空的对象。

执行以上程序输出结果为：

```
MyClass 类的属性 i 为:  12345
MyClass 类的方法 f 输出为:  hello world
```

类有一个名为 __init__() 的特殊方法（**构造方法**），该方法在类实例化时会自动调用，像下面这样：

```
def __init__(self):
self.data = []
```

类定义了 __init__() 方法，类的实例化操作会自动调用 __init__() 方法。如下实例化类 MyClass，对应的 __init__() 方法就会被调用:

```
x = MyClass()
```

当然， __init__() 方法可以有参数，参数通过 __init__() 传递到类的实例化操作上。例如:

实例(Python 3.0+)

```
#!/usr/bin/python3
class Complex:
def __init__(self, realpart, imagpart):
self.r = realpart
self.i = imagpart
```

```
x = Complex(3.0, -4.5)
print(x.r, x.i) # 输出结果: 3.0 -4.5
```

self代表类的实例，而非类

类的方法与普通的函数只有一个特别的区别——它们必须有一个额外的**第一个参数名称**, 按照惯例它的名称是 self。

```
class Test:
def prt(self):
print(self)
print(self.__class__)
t = Test()
t.prt()
```

以上实例执行结果为：

```
<__main__.Test instance at 0x100771878>
__main__.Test
```

从执行结果可以很明显的看出，self 代表的是类的实例，代表当前对象的地址，而 self.class 则指向类。

self 不是 python 关键字，我们把他换成 runoob 也是可以正常执行的:

```
class Test:
def prt(runoob):
print(runoob)
print(runoob.__class__)
t = Test()
t.prt()
```

以上实例执行结果为：

```
<__main__.Test instance at 0x100771878>
__main__.Test
```

类的方法

在类的内部，使用 `def` 关键字来定义一个方法，与一般函数定义不同，类方法必须包含参数 self, 且为第一个参数，self 代表的是类的实例。

实例(Python 3.0+)

```
#!/usr/bin/python3
#类定义
class people:
#定义基本属性
name = ''
age = 0
#定义私有属性,私有属性在类外部无法直接进行访问
__weight = 0
#定义构造方法
```

```
def __init__(self,n,a,w):
self.name = n
self.age = a
self.__weight = w
def speak(self):
print("%s 说: 我 %d 岁。" %(self.name,self.age))
# 实例化类
p = people('runoob',10,30)
p.speak()
```

执行以上程序输出结果为：

```
runoob 说: 我 10 岁。
```

继承

Python 同样支持类的继承，如果一种语言不支持继承，类就没有什么意义。派生类的定义如下所示:

```
class DerivedClassName(BaseClassName1):
<statement-1>
.
.
.
<statement-N>
```

需要注意圆括号中基类的顺序，若是基类中有相同的方法名，而在子类使用时未指定，python从左至右搜索 即方法在子类中未找到时，从左到右查找基类中是否包含方法。

BaseClassName（示例中的基类名）必须与派生类定义在一个作用域内。除了类，还可以用表达式，基类定义在另一个模块中时这一点非常有用:

```
class DerivedClassName(modname.BaseClassName):
```

实例(Python 3.0+)

```
#!/usr/bin/python3
#类定义
class people:
#定义基本属性
name = ''
age = 0
#定义私有属性,私有属性在类外部无法直接进行访问
__weight = 0
#定义构造方法
def __init__(self,n,a,w):
self.name = n
self.age = a
self.__weight = w
def speak(self):
print("%s 说: 我 %d 岁。" %(self.name,self.age))
```

```
#单继承示例
class student(people):
grade = ''
def __init__(self,n,a,w,g):
#调用父类的构函
people.__init__(self,n,a,w)
self.grade = g
#覆写父类的方法
def speak(self):
print("%s 说: 我 %d 岁了，我在读 %d 年级"%(self.name,self.age,self.grade))
s = student('ken',10,60,3)
s.speak()
```

执行以上程序输出结果为：

```
ken 说: 我 10 岁了，我在读 3 年级
```

多继承

Python同样有限的支持多继承形式。多继承的类定义形如下例:

```
class DerivedClassName(Base1, Base2, Base3):
<statement-1>
.
.
.
<statement-N>
```

需要注意圆括号中父类的顺序，若是父类中有相同的方法名，而在子类使用时未指定，python从左至右搜索 即方法在子类中未找到时，从左到右查找父类中是否包含方法。

实例(Python 3.0+)

```
#!/usr/bin/python3
#类定义
class people:
#定义基本属性
name = ''
age = 0
#定义私有属性,私有属性在类外部无法直接进行访问
__weight = 0
#定义构造方法
def __init__(self,n,a,w):
self.name = n
self.age = a
self.__weight = w
def speak(self):
print("%s 说: 我 %d 岁。" %(self.name,self.age))
#单继承示例
class student(people):
grade = ''
def __init__(self,n,a,w,g):
```

```
#调用父类的构函
people.__init__(self,n,a,w)
self.grade = g
#覆写父类的方法
def speak(self):
print("%s 说: 我 %d 岁了，我在读 %d 年级"%(self.name,self.age,self.grade))
#另一个类，多重继承之前的准备
class speaker():
topic = ''
name = ''
def __init__(self,n,t):
self.name = n
self.topic = t
def speak(self):
print("我叫 %s，我是一个演说家，我演讲的主题是 %s"%(self.name,self.topic))
#多重继承
class sample(speaker,student):
a =''
def __init__(self,n,a,w,g,t):
student.__init__(self,n,a,w,g)
speaker.__init__(self,n,t)
test = sample("Tim",25,80,4,"Python")
test.speak() #方法名同，默认调用的是在括号中排前地父类的方法
```

执行以上程序输出结果为：

```
我叫 Tim，我是一个演说家，我演讲的主题是 Python
```

方法重写

如果你的父类方法的功能不能满足你的需求，你可以在子类重写你父类的方法，实例如下：

实例(Python 3.0+)

```
#!/usr/bin/python3
class Parent: # 定义父类
def myMethod(self):
print ('调用父类方法')
class Child(Parent): # 定义子类
def myMethod(self):
print ('调用子类方法')
c = Child() # 子类实例
c.myMethod() # 子类调用重写方法
super(Child,c).myMethod() #用子类对象调用父类已被覆盖的方法
```

super() 函数是用于调用父类(超类)的一个方法。

执行以上程序输出结果为：

```
调用子类方法
调用父类方法
```

更多文档：

Python 子类继承父类构造函数说明

类属性与方法

类的私有属性

__private_attrs：两个下划线开头，声明该属性为私有，不能在类的外部被使用或直接访问。在类内部的方法中使用时 **self.__private_attrs。**

类的方法

在类的内部，使用 def 关键字来定义一个方法，与一般函数定义不同，类方法必须包含参数 `self`，且为第一个参数，`self` 代表的是类的实例。

`self` 的名字并不是规定死的，也可以使用 `this`，但是最好还是按照约定是用 `self`。

类的私有方法

__private_method：两个下划线开头，声明该方法为私有方法，只能在类的内部调用 ，不能在类的外部调用。**self.__private_methods。**

实例

类的私有属性实例如下：

实例(Python 3.0+)

```
#!/usr/bin/python3
class JustCounter:
__secretCount = 0 # 私有变量
publicCount = 0 # 公开变量
def count(self):
self.__secretCount += 1
self.publicCount += 1
print (self.__secretCount)
counter = JustCounter()
counter.count()
counter.count()
print (counter.publicCount)
print (counter.__secretCount) # 报错，实例不能访问私有变量
```

执行以上程序输出结果为：

```
1
2
2
Traceback (most recent call last):
  File "test.py", line 16, in <module>
    print (counter.__secretCount)  # 报错，实例不能访问私有变量
AttributeError: 'JustCounter' object has no attribute '__secretCount'
```

类的私有方法实例如下：

实例(Python 3.0+)

```
#!/usr/bin/python3
class Site:
def __init__(self, name, url):
self.name = name # public
self.__url = url # private
def who(self):
print('name : ', self.name)
print('url : ', self.__url)
def __foo(self): # 私有方法
print('这是私有方法')
def foo(self): # 公共方法
print('这是公共方法')
self.__foo()
x = Site('菜鸟教程', 'www.runoob.com')
x.who() # 正常输出
x.foo() # 正常输出
x.__foo() # 报错
```

以上实例执行结果：

```
root@iZ23mtq8bs1Z:~/test# python3 test.py
name  :  菜鸟教程
url :  www.runoob.com
这是公共方法
这是私有方法
Traceback (most recent call last):
  File "test.py", line 22, in <module>
    x.__foo()
AttributeError: 'Site' object has no attribute '__foo'
```

外部不能调用私有方法

类的专有方法：

- **__init__** : 构造函数，在生成对象时调用
- **__del__** : 析构函数，释放对象时使用
- **__repr__** : 打印，转换
- **__setitem__** : 按照索引赋值
- **__getitem__**: 按照索引获取值
- **__len__**: 获得长度
- **__cmp__**: 比较运算
- **__call__**: 函数调用
- **__add__**: 加运算
- **__sub__**: 减运算
- **__mul__**: 乘运算
- **__truediv__**: 除运算

- **__mod__**: 求余运算
- **__pow__**: 乘方

运算符重载

Python同样支持运算符重载，我们可以对类的专有方法进行重载，实例如下：

实例(Python 3.0+)

```
#!/usr/bin/python3
class Vector:
def __init__(self, a, b):
self.a = a
self.b = b
def __str__(self):
return 'Vector (%d, %d)' % (self.a, self.b)
def __add__(self,other):
return Vector(self.a + other.a, self.b + other.b)
v1 = Vector(2,10)
v2 = Vector(5,-2)
print (v1 + v2)
```

以上代码执行结果如下所示:

```
Vector(7,8)
```

← Python3 错误和异常 Python3 标准库概览 →