Python面向对象语法精讲

嵩 天

Python类的高级话题

嵩 天

Python类的高级话题

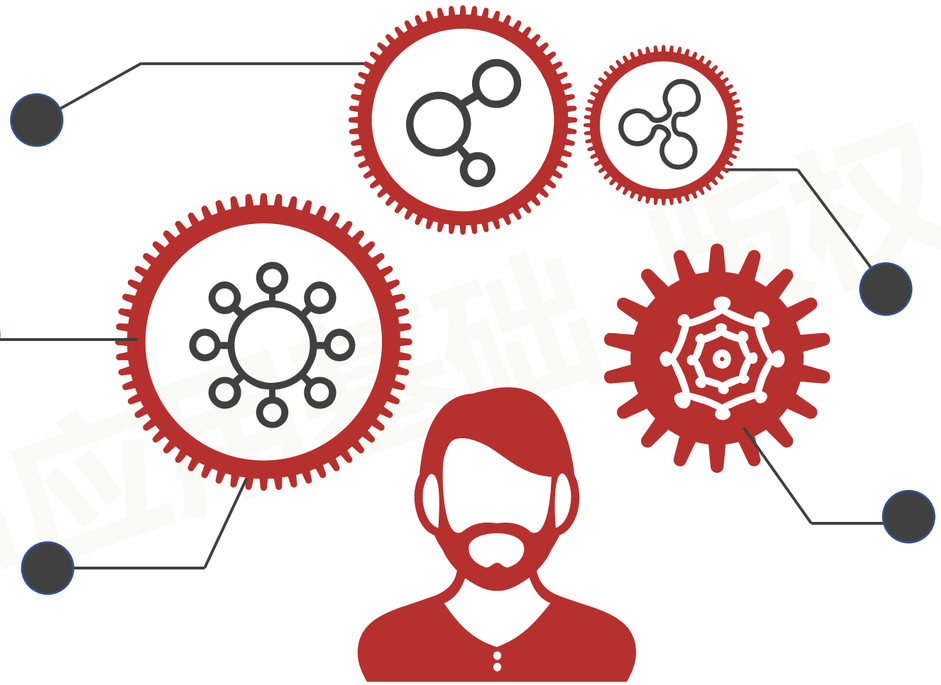
单元开篇

单元开篇

(3) 自定义的异常类型

(2) 类的特性装饰器

(1) 命名空间的理解



(4) 类的名称修饰

(5) Python最小空类

Python类的高级话题

Python类的高级话题

命名空间的理解

命名空间的理解

命名空间 Namespace：从名字到对象的一种映射

- 作用域：全局变量名在模块命名空间，局部变量名在函数命名空间
- 属性和方法在类命名空间，名字全称：<命名空间>.<变量/函数名>
- 命名空间底层由一个dict实现，变量名是键，变量引用的对象是值

命名空间的理解

命名空间 Namespace：从名字到对象的一种映射

- 复数z，z.real和z.imag是对象z命名空间的两个属性
- 对象d，d.name和d.printName()是对象d命名空间的属性和方法
- global和nonlocal是两个声明命名空间的保留字

命名空间的理解

```
count = 0

def getCounting(a):
    count = 0
    if a != "":
        def doCounting():
            nonlocal count
            count += 1
        doCounting()
    return count

print(getCounting("1"), count)
print(getCounting("2"), count)
print(getCounting("3"), count)
```

模块的命名空间

第一层函数的命名空间

第二层函数的命名空间

命名空间的理解

```
count = 0

def getCounting(a):
    count = 0
    if a != "":
        def doCounting():
            nonlocal count
            count += 1
        doCounting()
    return count

print(getCounting("1"), count)
print(getCounting("2"), count)
print(getCounting("3"), count)
```

nonlocal声明变量不在当前命名空间

变量在上层命名空间，但不是全局

命名空间的理解

```
count = 0

def getCounting(a):
    count = 0
    if a != "":
        def doCounting():
            nonlocal count
            count += 1
        doCounting()
    return count

print(getCounting("1"), count)
print(getCounting("2"), count)
print(getCounting("3"), count)
```

```
1 0
1 0
1 0
```

命名空间的理解

```
count = 0

def getCounting(a):
    count = 0
    if a != "":
        def doCounting():
            global count
            count += 1
        doCounting()
    return count

print(getCounting("1"), count)
print(getCounting("2"), count)
print(getCounting("3"), count)
```

global声明变量在全局命名空间

命名空间的理解

```
count = 0

def getCounting(a):
    count = 0
    if a != "":
        def doCounting():
            global count
            count += 1
        doCounting()
    return count

print(getCounting("1"), count)
print(getCounting("2"), count)
print(getCounting("3"), count)
```

```
0 1
0 2
0 3
```

Python类的高级话题

类的特性装饰器

类的特性装饰器

```
d = DemoClass("老李")
d.age = -99
```

- d.age 如果作为年龄，其值有误，但属性不能检测异常值
- 如何为属性增加异常检测呢?

类的特性装饰器

@property：类的特性装饰器

- 使用@property把类中的方法变成对外可见的“属性”
- 类内部：表现为方法
- 类外部：表现为属性

类的特性装饰器

```python
class DemoClass:
    def __init__(self, name):
        self.name = name

    @property
    def age(self):
        return self._age

    @age.setter
    def age(self, value):
        if value < 0 or value > 100:
            value = 30
        self._age = value


dc1 = DemoClass("老李")
dc1.age = -100
print(dc1.age)
```

@property 用于转换方法为属性

@<方法名>.setter

用于设定属性的赋值操作

类的特性装饰器

```python
class DemoClass:
    def __init__(self, name):
        self.name = name

    @property
    def age(self):
        return self._age

    @age.setter
    def age(self, value):
        if value < 0 or value > 100:
            value = 30
        self._age = value


dc1 = DemoClass("老李")
dc1.age = -100
print(dc1.age)
```

返回一个属性值

类的特性装饰器

```python
class DemoClass:
    def __init__(self, name):
        self.name = name

    @property
    def age(self):
        return self._age

    @age.setter
    def age(self, value):
        if value < 0 or value > 100:
            value = 30
        self._age = value


dc1 = DemoClass("老李")
dc1.age = -100
print(dc1.age)
```

对同名属性值的赋值操作进行处理

类的特性装饰器

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

    @property
    def age(self):
        return self._age

    @age.setter
    def age(self, value):
        if value < 0 or value > 100:
            value = 30
        self._age = value


dc1 = DemoClass("老李")
dc1.age = -100
print(dc1.age)
```

```
30
```

Python类的高级话题

自定义的异常类型

自定义的异常类型

异常 Exception 也是一种Python类

- try-except捕捉自定义的异常
- 继承Exception类，可以给出自定义的异常类
- 自定义异常类是类继承的正常应用过程

自定义的异常类型

```python
class DemoException(Exception):
    pass

try:
    raise DemoException()
except DemoException:
    print("捕获DemoException异常")
```

自定义一个异常

捕捉这个异常

自定义的异常类型

```
class DemoException(Exception):
    pass


try:
    raise DemoException()
except DemoException:
    print("捕获DemoException异常")
```

raise保留字产生异常

自定义的异常类型

```python
class DemoException(Exception):
    def __init__(self, name, msg = "自定义异常"):
        self.name = name
        self.msg = msg

try:
    raise DemoException("脚本错误")
except DemoException as e:
    print("{}异常的报警是{}".format(e.name, e.msg))
```

自定义一个异常类型

捕捉这个异常及异常对象e

自定义的异常类型

```
class DemoException(Exception):
    def __init__(self, name, msg = "自定义异常"):
        self.name = name
        self.msg = msg

try:
    raise DemoException("脚本错误")
except DemoException as e:
    print("{}异常的报警是{}".format(e.name, e.msg))
```

raise保留字产生异常

Python类的高级话题

类的名称修饰

类的名称修饰

名称修饰 Name Mangling：类中名称的变换约定

- Python通过名称修饰完成一些重要功能
- 采用下划线(_)进行名称修饰，分为5种情况
- _X、X_、__X、 __X__、_

类的名称修饰

_X：单下划线开头的名称修饰

- 单下划线开头属性或方法为类内部使用　PEP8
- 只是约定，仍然可以通过<对象名>.<属性名>方式访问
- 功能：from XX import *时不会导入单下划线开头的属性或方法

类的名称修饰

```
class DemoClass:
    def __init__(self, name):
        self.name = name
        self._nick = name + "同志"

    def getNick(self):
        return self._nick

dc1 = DemoClass("老李")
print(dc1.getNick())
print(dc1._nick)
```

约定内部使用

仍然可以外部调用

类的名称修饰

```
class DemoClass:
    def __init__(self, name):
        self.name = name
        self._nick = name + "同志"

    def getNick(self):
        return self._nick

dc1 = DemoClass("老李")
print(dc1.getNick())
print(dc1._nick)
```

```
老李同志
老李同志
```

类的名称修饰

X_：单下划线结尾的名称修饰

- 单下划线结尾属性或方法为避免与保留字或已有命名冲突　PEP8
- 只是约定，无任何功能性对应

类的名称修饰

```
class DemoClass:
    def __init__(self, name):
        self.name = name
        self.class_ = name + "同志"

    def getNick(self):
        return self.class_

dc1 = DemoClass("老李")
print(dc1.getNick())
print(dc1.class_)
```

仅是为了避免重名

类的名称修饰

__X：双下划线开头的名称修饰

- 双下划线开头属性或方法将被解释器修改名称，避免命名冲突
- 不是约定，而是功能性，实现私有属性、私有方法
- __X会被修改为：_<类名>__X

类的名称修饰

```
class DemoClass:
    def __init__(self, name):
        self.name = name
        self.__nick = name + "同志"

    def getNick(self):
        return self.__nick

dc1 = DemoClass("老李")
print(dc1.getNick())
print(dc1._DemoClass__nick)
print(dc1.__nick)
```

私有属性

类的名称修饰

```
class DemoClass:
    def __init__(self, name):
        self.name = name
        self.__nick = name + "同志"

    def getNick(self):
        return self.__nick

dc1 = DemoClass("老李")
print(dc1.getNick())
print(dc1._DemoClass__nick)
print(dc1.__nick)
```

```
老李同志
老李同志
AttributeError: 'DemoClass'
    object has no attribute
    '__nick'
```

类的名称修饰

__X__：双下划线开头和结尾的名称修饰

- 双下划线开头和结尾的属性或方法无任何特殊功能，名字不被修改
- 部分名称是保留属性或保留方法

类的名称修饰

```
class DemoClass:
    def __init__(self, name):
        self.name = name
        self.__nick__ = name + "同志"

    def getNick(self):
        return self.__nick__

dc1 = DemoClass("老李")
print(dc1.getNick())
print(dc1.__nick__)
```

正常的属性名称

类的名称修饰

```
class DemoClass:
    def __init__(self, name):
        self.name = name
        self.__nick__ = name + "同志"

    def getNick(self):
        return self.__nick__

dc1 = DemoClass("老李")
print(dc1.getNick())
print(dc1.__nick__)
```

```
老李同志
老李同志
```

类的名称修饰

_：单下划线

- 单下划线是一个无关紧要的名字，无特殊功能

类的名称修饰

```
class DemoClass:
    def __init__(self, name):
        self.name = name
        _ = "同志"
        self.__nick__ = name + _

    def getNick(self):
        return self.__nick__

dc1 = DemoClass("老李")
print(dc1.getNick())
print(dc1.__nick__)
```

无关紧要的名字

老李同志

老李同志

小结：类的名称修饰

- _X：约定内部使用，仅在import *时不被引用
- X_：避免与保留字冲突，无特殊功能
- __X：不被子类继承、可用于设定私有、改变为：_<类名>__X
- __X__：无特殊功能，部分用于保留属性和保留方法
- _：无特殊功能，不重要的命名

Python类的高级话题

Python最小空类

Python最小空类

Python最小空类

```
class <类名>():
    pass
```

Python最小空类

Python最小空类的作用

- 类是一个命名空间，最小空类可以当作命名空间使用
- 最小空类可以辅助数据存储和使用
- 动态增加属性是Python类的一个特点

Python最小空类

```
class EmptyClass:
    pass


a = EmptyClass()
a.name = "老李"
a.age  = 50
a.family = {"儿子": "小李", "女儿": "囡李"}
print(a.family)
print(a.__dict__)
```

建立一个最小空类

通过增加属性实现对数据被保存

Python最小空类

```
class EmptyClass:
    pass


a = EmptyClass()
a.name = "老李"
a.age  = 50
a.family = {"儿子": "小李", "女儿": "囡李"}
print(a.family)
print(a.__dict__)
```

```
{'儿子': '小李', '女儿': '囡李'}
{'name': '老李', 'age': 50,
 'family': {'儿子': '小李',
            '女儿': '囡李'}}
```

Python类的高级话题

单元小结

单元小结

(3) 自定义的异常类型

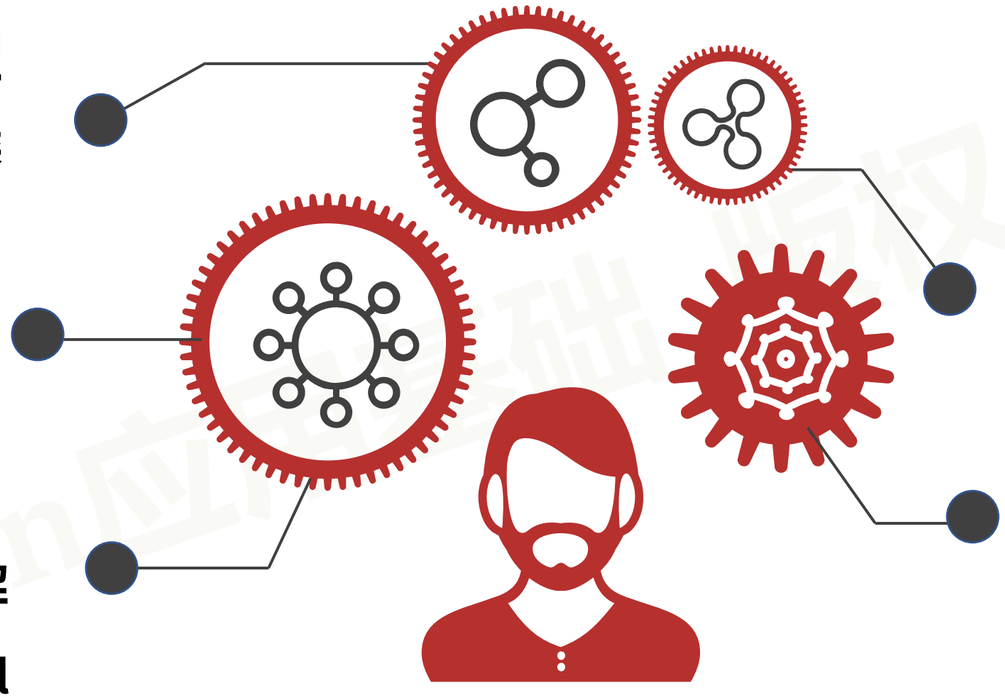
Exception类

(2) 类的特性装饰器

@property、@.setter

(1) 命名空间的理解

命名作用域、nonlocal、global



(4) 类的名称修饰

类中下划线命名的5种情况

(5) Python最小空类

属性的动态增加及作用

Python类的高级话题


Python 123
Thank you