第1编 Python语法基础

中国人民大学
RENMIN UNIVERSITY OF CHINA

第4讲 类与对象

余力

buaayuli@ruc.edu.cn

中國人民大學
RENMIN UNIVERSITY OF CHINA

01. 基本概念

类

- 类别：定义新的对象
  - 内建对象：数值, 字符串, 序列, 字典, tuple
  - 继承(inheritance)、组合(composition)
- 与模块的比较
  - 多个实体对象
    - 每个新的对象都拥有独立的名称空间
    - 从同一个类别出来的对象都可以存取该类别的属性
  - 个别化(customization)
    - 类别支持继承, 又可以覆盖

类定义

- 书写格式

```
class <name>(superclass, ...):    #指定给name
    data = value                  #共享类别数据
    def method(self, ...):        #成员函数
        self.member = value       #实体对象数据
```

```
class Subclass:          #定义子类别
    data = "little"      #指定类别属性
    def __init__(self, value):
        self.data = value
    def display(self):
        print (self.data, Subclass.data)
```

```
x = Subclass(1)
y = Subclass(2)
x.display()
    1 little
y.display()
    2 little
    y. __init_()_
```

类别成员函数

- 与函数其实很像
  - 差别在于成员函数的第一个自变量
  - 会自行把实体对象的成员函数调用转换成类别对象的函数调用
  - **instance.method(args…) → class.method(instance, args…)**

```
class NextClass:                                    #定义类别
        def printer(self, text):                    #定义成员函数
            print (text)
x = NextClass()                                     #做出实体
x.printer("Hello python!")                          #调用成员函数
      Hello python!
NextClass.printer(x, "Hello python!")               #类别成员函数
      Hello python!
```

成员函数的绑定

```
class larc:
    def ok(self, message):
        print (message)
object1 = larc()
x = object1.ok        #有绑定的成员函数
x("hello")            #隐含了实体
t = larc.ok           #无绑定的成员函数
t(object1,"hello")    #必须传递实体
```

```
class Foo(object):
    def foo():
        print "call foo"
    def foo_one(self):
        print "call foo_one"
```

```
>>> Foo.foo()
call foo_
```

```
>>> Foo().foo()
TypeError: foo() takes 0 positional arguments but 1 was given
```

```
>>> Foo.foo_one()
TypeError: foo_one() missing 1 required positional argument: 'self'
```

```
>>> Foo().foo_one()
call foo_one
```

类属性和类实例的属性

```
class AAA():
    aaa = 10

# 情形1
obj1=AAA()
obj2=AAA()
print (obj1.aaa,obj2.aaa,AAA.aaa)

# 情形2
obj1.aaa+=2
print (obj1.aaa,obj2.aaa,AAA.aaa)

# 情形3
AAA.aaa+=3
print (obj1.aaa,obj2.aaa,AAA.aaa)
```

obj1.aaa经过一次+=操作后，实际上与AAA.aaa脱离了关系，而obj2.aaa没有经过任何的属性操作，因此其只会从其所属的类AAA中去获得aaa

基本概念-多个实体对象

- 实体对象从类别而来
  - 调用类别对象会生成新的实体对象
  - 每个实体对象都会继承类别的属性并得到独立的名称空间
  - 传递给self的自变量会改变实体对象的属性
    - 类别的成员函数第一个自变量会指向正在处理的实体对象
    - 传递self的自变量会建立或改变实体对象的数据

```
class FirstClass:                    #定义类别对象
    def setdata(self, value):        #定义类别成员函数
        self.data = value            #self是实体
    def display(self):
        print (self.data)
```

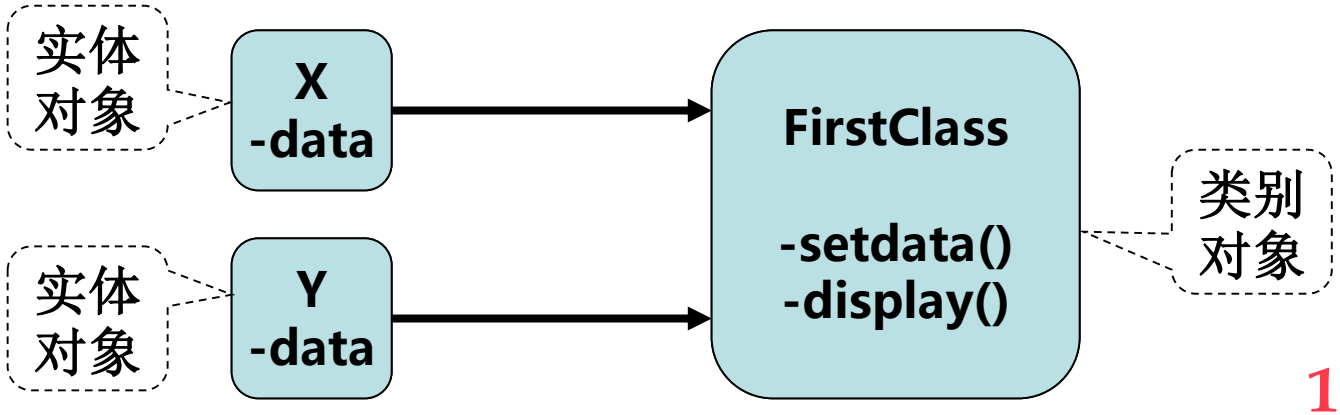
基本概念-多个实体对象

```
x = FirstClass()                #两个实体对象
y = FirstClass()                #每一个都是新的名称空间
x.setdata("King Arthur")        #调用成员函数, self = x
y.setdata(3.14159)              #执行FirstClass.setdata(y, 3.14159)
x.display(), y.display()        #self.data两个都不同
```

- ("King Arthur", 3.14159)

```
x.data = "New value"            #也可以使用名称评定的方式设定
x.display()
```

- New value



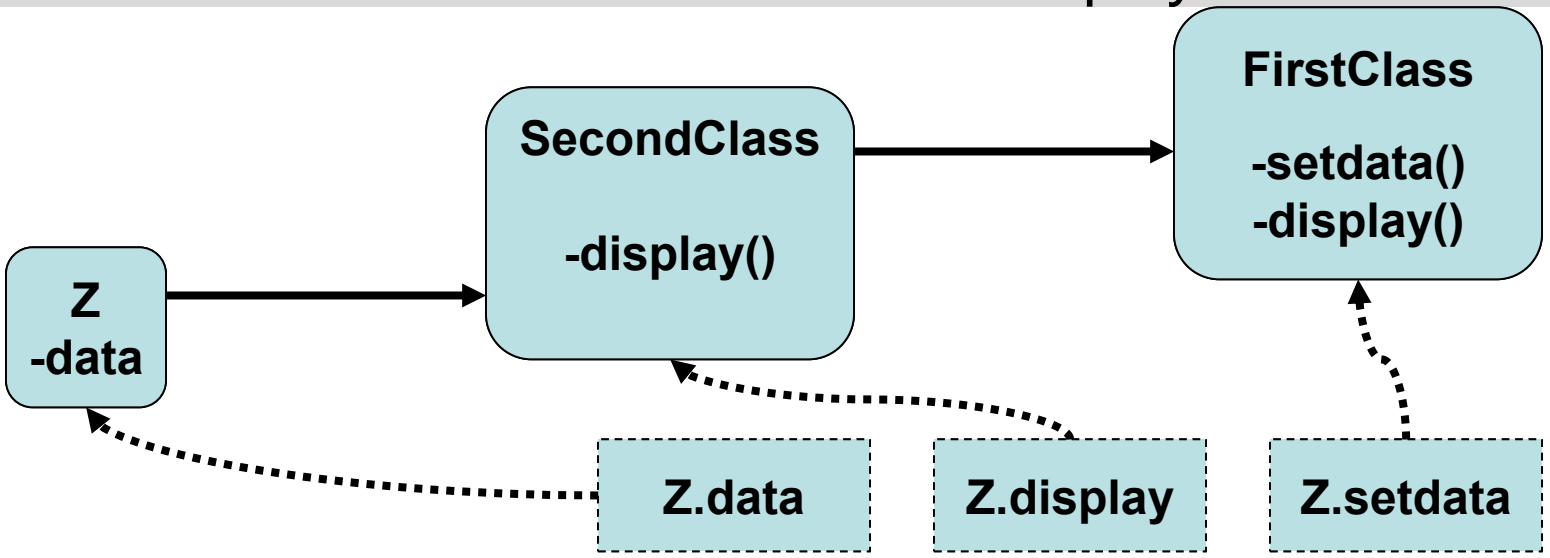
基本概念-个别化与继承(1/2)

- 类别会从母类别继承属性
  - 如果无法在子类别找到, 会自动往母类别上找
- 实例对象继承的属性不限于产出它的类别
- 如果子类别里有重新定义母类别的名称, 子类别会覆盖掉原先继承下来的属性

```
class SecondClass(FirstClass):  #继承setdata
    def display(self):                    #变更display
        print ("Current value = %s" % self.data)
```

基本概念-个别化与继承(2/2)

- z = SecondClass()
- z.setdata(42)　　#setdata在FirstClass找到
- z.display()　　#display在SecondClass被替代掉
  - Current value = "42"
- x.display()
  - New value　　#FirstClass里的display并没有被影响



中国人民大学
RENMIN UNIVERSITY OF CHINA

02. 运算符重载

运算符重载

- 写成__X__形式的成员函数, 即可拦截
- 类别也可以重载对象的运算:
  - 输出, 调用, 评定用法等等
- 重载使得类别实体更能像内建型态般运作
- 重载的实作方法
  - 透过提供特殊成员函数名称而达成
  - 皆以__X__的方式存在

运算符重载-常见的成员函数

成员函数	重载	用法
__init__	建构子	建立对象 class()
__del__	解构子	释放对象
__add__	运算符"+"	X + Y
__or__	运算符"|"	X | Y
__repr__	打印,转换型态	Print X, `X`
__getattr__	名称评定用法	X.未定义
__getitem__	索引值参考	X[key], for loops, in tests
__setitem__	索引值指定运算	X[key] = value
__getslice__	切片运算	X[low:high]
__len__	长度	len(X), truth tests
__cmp__	比较	X == Y, X < Y

基本概念-运算符重载(1/2)

- Python使用运算符时, 就会自动调用
- 当对象出现某个+运算, __add__就会被调用

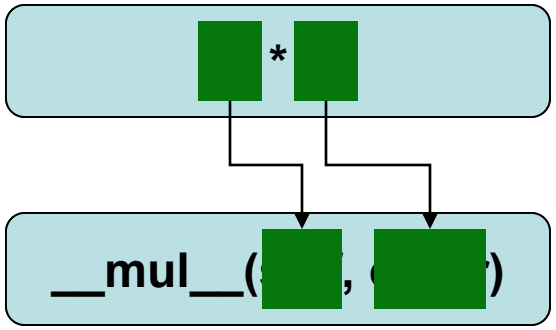
```
class ThirdClass(SecondClass):    # 继承 SecondClass
    def __init__(self,  value):            # ThirdClass(Value)
        self.data = value
    def __add__(self, value):           # self + value
        return ThirdClass(self.data +value)
    def __mul__(self, value):           # self * value
        self.data = self.data * value
```

基本概念-运算符重载(2/2)

- a = ThirdClass("abc")
- a.display()
  - Current value = "abc"
- b = a +"xyz"
- b.display()
  - Current value = "abcxyz"

a * 3

- a.display()
  - Current value = "abcabcabc"




```
def __mul__(self, value):
        self.data = self.data * value
```

运算符重载-__getitem__

- __getitem__会拦截实体对象索引值参考的运算

```
class indexer:
    def __getitem__(self, index):
        return index ** 2
X = indexer()
for index in range(5):
    print (X[index])          #调用__getitem__(X, index)
...
    0 1 4 9 16
```

运算符重载-__getitem__

- 由于for循环的运作方式(使用索引值0~N)使得__getitem__会被调用

```
class stepper:
    def __getitem__(self, i):
        return self.data[i]
X = stepper()
X.data = "larc"
for item in X:
        print (item)            # for的索引值是 0...3

    l a r c
>>> "a" in X        # in运算符也会调用__getitem__
    1
```

运算符重载-__getattr__

- __getattr__会拦截未定义的属性
  - 若以评定用法调用一个不存在的属性名称, 就会被调用
  - 且该属性名称将会以字符串形式传递给__getattr__
  - 如果在继承树的搜寻程序内找的到属性名称, 就不会被调用

```
class empty:
    def __getattr__(self, attr):
        if attr =="age":
            return 36
        else:
            return "undefined value"
```

```
X = empty()
X.age
    36
X.name
    "undefined value"
```

减法重载-__sub__

```
class  New_Number:
    def  __init__(self, start):
        self.data = start
    def  __sub__(self, other):
        return  self.data – 2*other
num = New_Number(20)
y = num - 5 -5 # invoke __sub__ method
print (y)
```

输出重载-__repr__

- # 重构__repr__方法后，不管直接输出对象还是通过print打印的信息，都按__repr__方法中定义的格式进行显示了

```
class adder:
    def __init__(self, value=10):                  #预设value值
        self.data = 2+value                          #建构设定
    def __add__(self, other):
        self.data = self.data +2* other             #加上other
        print (self.data)
    def __repr__(self):
        return "self.data"
X = adder(1)    #__init__
X + 2
X + 2               #__add__
X                   #__repr__
```

```
3
5
Out[1]:
self.data
```

03. 迭代器

迭代器

```
class Fib:
    def __init__(self):
        self.a = 0
        self.b = 1
    def __iter__(self):
        return self
    def __next__(self):
        self.a, self.b = self.b,  self.b+self.a
        return self.a

fibs = Fib()
for f in fibs:
    if f>1000:
        print (f)
        break
```

__iter__会返回一个迭代器

所谓迭代器：具有next方法

（这个方法调用时不需要任何参数）的对象

迭代器

```
class Squares:
    def __init__(self,start,stop):
        self.value = start - 1
        self.stop = stop
    def __iter__(self):
        return self
    def __next__(self):
        if self.value == self.stop:
            raise StopIteration
        self.value += 1
        return self.value ** 2
```

```
for i in Squares(1,5):
    print(i,end = ' ')
```

1 4 9 16 25

```
X=Squares(1,5)
for i in X:
    print(i,end = ' ')
```

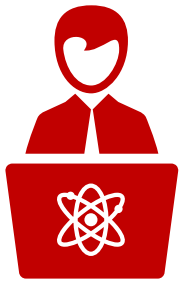
1 4 9 16 25

生成器

```
def range(max):
    n,a,b =0,0,1
    while n < max:
        yield n
        n = n+1
    return 'done'
for i in fib(6):
    print(i)
```

```
def fib(max):
    n,a,b=0,0,1
    while n < max:
        yield b
        a,b =b,a+b
        n = n+1
    return 'done'
g = fib(6)
while True:
    try:
        x = next(g)
        print('generator: ',x)
    except StopIteration as e:
        print("生成器返回值： ",e.value)
        break
```

中国人民大学
RENMIN UNIVERSITY OF CHINA



谢谢大家！