马哥教育

主讲：马永亮(马哥)
QQ:113228115
客服QQ：2813150558，1661815153
http://www.magedu.com
http://mageedu.blog.51cto.com

马哥教育

Python类与面向对象

主讲：马永亮(马哥)
QQ:113228115
客服QQ：2813150558，1661815153
http://www.magedu.com
http://mageedu.blog.51cto.com

面向对象编程(OOP)

- 程序 = 指令+数据
  - 代码可以选择以指令为核心或以数据为核心进行编写
- 两种范型
  - 以指令为核心：围绕"正在发生什么"进行编写
    - 面向过程编程：程序具有一系列线性步骤；主体思想是代码作用于数据
  - 以数据为核心：围绕"将影响谁"进行编写
    - 面向对象编程(OOP)：围绕数据及为数据严格定义的接口来组织程序，用数据控制对代码的访问

面向对象的核心概念

- 所有编程语言的最终目的都是提供一种抽象方法
  - 在机器模型("解空间"或"方案空间")与实际解决的问题模型("问题空间")之间，程序员必须建立一种联系
    - 面向过程：程序 = 算法+数据结构
    - 面向对象：将问题空间中的元素以及它们在解空间中的表示物抽象为对象，并允许通过问题来描述问题而不是方案
      - 可以把实例想象成一种新型变量，它保存着数据，但可以对自身的数据执行操作

面向对象的核心概念

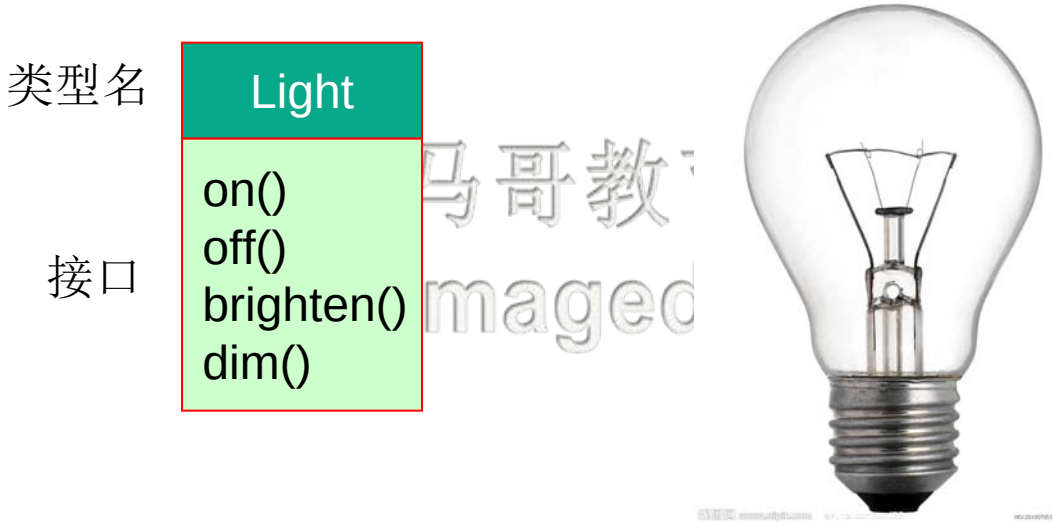
- 类型由状态集合(数据)和转换这些状态的操作集合组成
  - 类抽象
    - 类：定义了被多个同一类型对象共享的结构和行为(数据和代码)
    - 类的数据和代码：即类的成员
      - 数据：成员变量或实例变量
      - 成员方法：简称为方法，是操作数据的代码，用于定义如何使用成员变量；因此一个类的行为和接口是通过方法来定义的
    - 方法和变量：
      - 私有：内部使用
      - 公共：外部可见

面向对象的程序设计方法

- 所有东西都是对象
- 程序是一大堆对象的组合
  - 通过消息传递，各对象知道自己该做什么
  - 消息：即调用请求，它调用的是从属于目标对象的一个方法
- 每个对象都有自己的存储空间，并可容纳其它对象
  - 通过封装现有对象，可以制作成新型对象
- 每个对象都属于某一类型
  - 类型，也即类
  - 对象是类的实例
  - 类的一个重要特性为"能发什么样的消息给它"
- 同一个类的所有对象都能接收相同的消息

对象的接口

- 定义一个类后，可以根据需要实例化出多个对象
- 如何利用对象完成真正有用的工作？
  - 必须有一种办法能向对象发出请求，令其做一些事情
  - 每个对象仅能接受特定的请求
    - 能向对象发送的请求由其"接口"进行定义
    - 对象的"类型"或"类"则规定了它的接口形式



类间关系

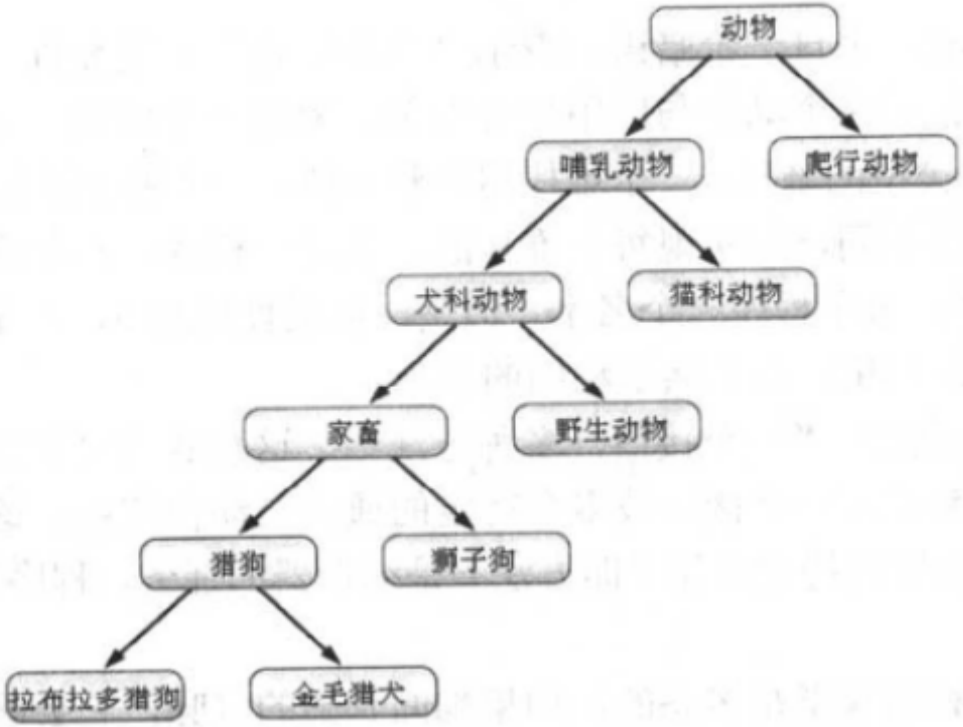
- 依赖("uses-a")
  - 一个类的方法操纵另一个类的对象
- 聚合("has-a")
  - 类A的对象包含类B的对象
- 继承("is-a")
  - 描述特殊与一般关系

面向对象编程的原则

- 面向对象的模型机制有3个原则：封装、继承及多态
- 封装(Encapsulation)
  - 隐藏实现方案细节
  - 将代码及其处理的数据绑定在一起的一种编程机制，用于保证程序和数据不受外部干扰且不会被误用

面向对象编程的原则

- 继承(Inheritance)
  - 一个对象获得另一个对象属性的过程；用于实现按层分类的概念
  - 一个深度继承的子类继承了类层次中它的每个祖先的所有属性
  - 超类、基类、父类
  - 子类、派生类

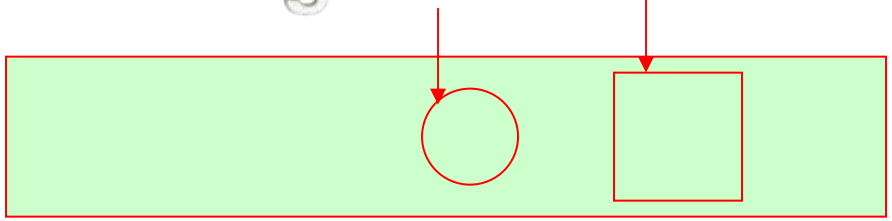


面向对象编程的原则

- 多态性(Polymorphism)
  - 允许一个接口被多个通用的类动作使用的特性，具体使用哪个动作与应用场合相关
  - “一个接口，多个方法”
    - 用于为一组相关的动作设计一个通用的接口，以降低程序复杂性

Python类和实例

- 类是一种数据结构，可用于创建实例
  - 一般情况下，类封装了数据和可用于该数据的方法
- Python类是一个可调用对象，即类对象
- Python2.2之后，类是一种自定义类型，而实例则是声明某个自定义类型的变量
- 实例初始化
  - 通过调用类来创建实例
    - instance = ClassName(args...)
  - 类在实例化时可以使用__init__和__del__两个特殊的方法



Python中创建类

- Python使用class关键字创建类，语法格式如下
  - class ClassName(bases):
    - 'class documentation string'
    - class_suite
  - 超类是一个或多个用于继承的父类的集合
  - 类体可以包含：声明语句、类成员定义、数据属性、方法
  - 注意：
    - 如果不存在继承关系，ClassName后面的"(bases)"可以不提供
    - 类文档为可选
- class语句的一般形式
  - class ClassName(bases):
    - data = value ← 定义数据属性
    - def method(self,...): ← 定义方法属性
      - self.member = value

例子

- Python中，class语句类似def，是可执行代码；直到运行class语句后类才会存在

- class语句内，任何赋值语句都会创建类属性
- 每个实例对象都会继承类的属性并获得自己的名称空间

Python类方法及调用

- 实例(对象)通常包含属性
  - 可调用的属性：方法
    - object.method()
  - 数据属性
- 在OOP中，实例就像是带有"数据"的记录，而类是处理这些记录的"程序"
  - 通过实例调用方法相当于调用所属类的方法来处理当前实例
    - 类似instance.method(args...)会被自动转换为class.method(instance,args...)
      - 如前面的例子，x.display()会被自动转换为FirstClass.display(x)，即调用类的方法来处理实例x
  - 因此，类中每个方法必须具有self参数，它隐含当前实例之意
  - 在方法内对self属性做赋值运算会产生每个实例自己的属性
  - Python规定，没有实例，方法不允许被调用，此即为"绑定"

Python类和实例的属性

- class语句中的赋值语句会创建类属性，如前面例子中的spam
- 在类方法中对传给方法的特殊参数self进行赋值会创建实例属性

```
>>> class MyClass():
        gender = 'Male'
        def setName(self, who):
            self.name = who
```

通过爬树搜索，gender属性会从MyClass类中获取到

```
>>> x = MyClass()
>>> y = MyClass()
>>> x.gender
'Male'
>>> x.name
Traceback (most recent call last):
  File "<pyshell#22>", line 1, in <module>
    x.name
AttributeError: MyClass instance has no attribute 'name'
>>> x.setName('tom')
>>> x.name
'tom'
>>> y.setName('jerry')
>>> y.gender, y.name
('Male', 'jerry')
>>> x.gender, x.name
('Male', 'tom')
```

在setName方法调用之前，MyClass类不会把name属性附加到实例x上，当然也可以重载__init__创建构造器直接为实例提供

Python构造器

- 创建实例时，Python会自动调用类中的__init__方法，以隐性地为实例提供属性
  - __init__方法被称为构造器
  - 如果类中没有定义__init__方法，实例创建之初仅是一个简单的名称空间

```
>>> class MyClass():
        gender = 'Male'
        def __init__(self, who):
            self.name = who


>>> x = MyClass('tom')
>>> y = MyClass('jerry')
>>> x.gender, x.name
('Male', 'tom')
>>> y.gender, y.name
('Male', 'jerry')
```

第二个例子

数据属性(成员变量)

```
>>> class Animal:
        name = 'Someone'
        def __init__(self,voice='HI'):
            self.voice = voice
        def __del__(self):
            pass
        def saySomething(self):
            print self.voice
```

重载构造函数

重载析构函数

方法属性(成员函数)

```
>>> tom = Animal()
>>> tom.saySomething()
HI
>>> jerry = Animal('Hello!')
>>> jerry.saySomething()
Hello!
```

创建实例

调用实例方法

类的特殊属性

- 可以使用类的__dict__字典属性或Python内置的dir()函数来获取类的属性

```
>>> dir(MyClass)
['__doc__', '__init__', '__module__', 'gender']
>>> MyClass.__dict__
{'gender': 'Male', '__module__': '__main__', '__doc__': None, '__init__': <funct
ion __init__ at 0x0000000002B7C3C8>}
```

C.__name__	类C的名字（字符串）
C.__doc__	类C的文档字符串
C.__bases__	类C的所有父类构成的元组
C.__dict__	类C的属性
C.__module__	类C定义所在的模块（1.5 版本新增）
C.__class__	实例C对应的类（仅新式类中）

实例属性

- 实例仅拥有数据属性(严格意义上来说，方法是类属性)
  - 通常通过构造器"__init__"为实例提供属性
  - 这些数据属性独立于其它实例或类
  - 实例释放时，其属性也将被清除
- 内建函数dir()或实例的特殊属性__dict__可用于查看实例属性

```
>>> dir(x)
['__doc__', '__init__', '__module__', 'gender', 'name']
>>> x.__dict__
{'name': 'tom'}
```


- 实例的特殊属性

I.__class__	实例化 I 的类
I.__dict__	I 的属性

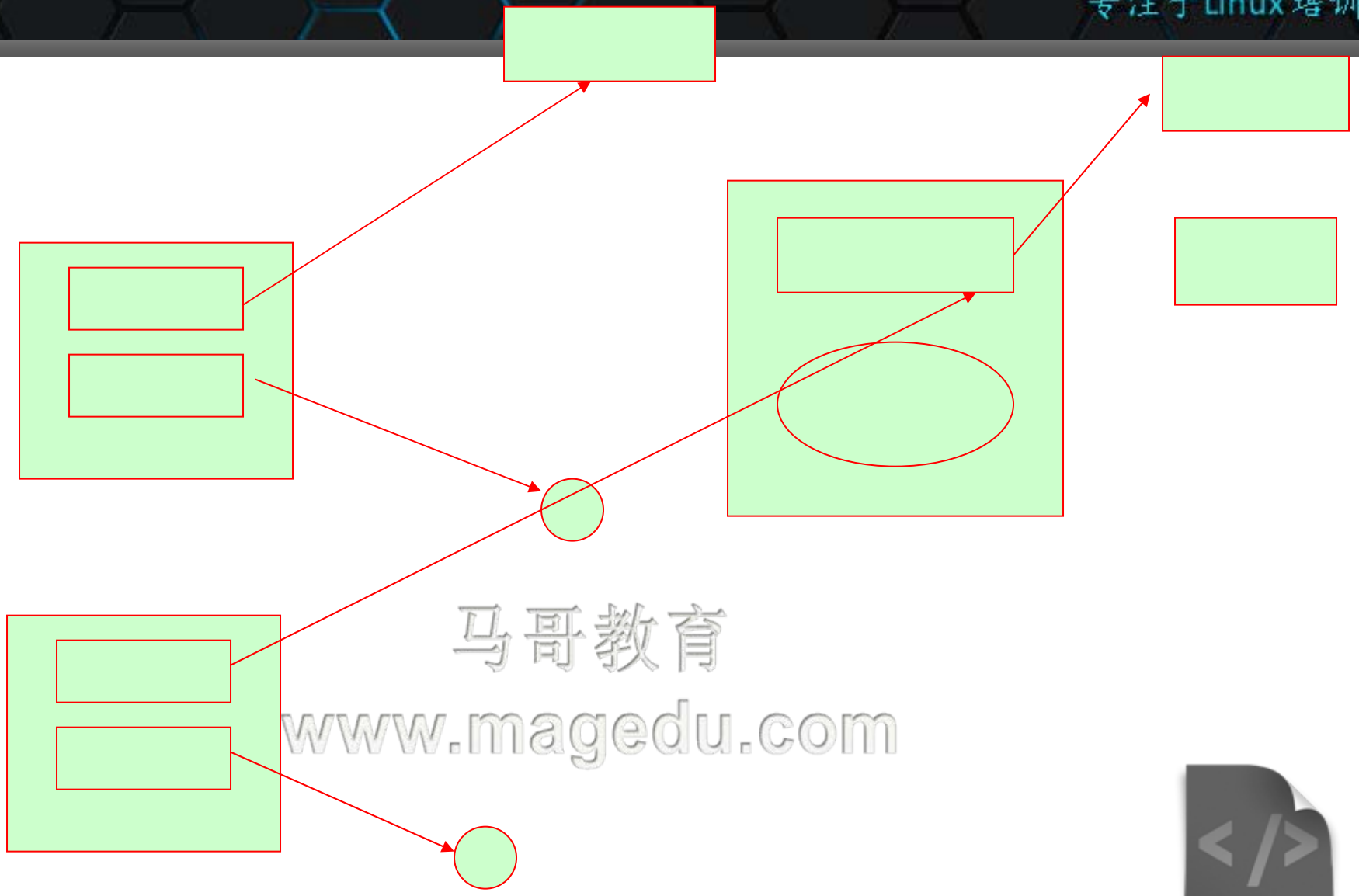
Python类方法中可用的变量

- 方法的可用变量
  - 实例变量:指定变量名称及实例自身进行引用
    - self.变量名
  - 局部变量：方法内部创建的变量，可直接使用
  - 类变量(也称静态变量)：通过指定变量名与类名进行引用
    - 类名.变量名
  - 全局变量：直接使用

为实例继承的属性重新赋值

```
>>> x = FirstClass()
>>> y = FirstClass()
>>> x.spam, y.spam
(30, 30)
>>> x.spam = 40
>>> x.spam, y.spam
(40, 30)
>>> FirstClass.spam = 56
>>> x.spam, y.spam
(40, 56)
```

数字为不可变类型，因此x.spam已经指向其它对象，但y.spam已经随类的重新赋值而改变



继承

- 继承描述了基类的属性如何"遗传"给派生类
  - 子类可以继承它的基类的任何属性，包括数据属性和方法
  - 一个未指定基类的类，其默认有一个名为object的基类
  - Python允许多重继承
- 创建子类
  - 创建子类时，只需要在类名后跟一个或从其中派生的父类
  - class SubClassName(ParentClass1[, ParentClass2, ...])
    - 'optional class documentation string'
    - class_suite

Python类继承的例子

- 子类可以继承它的基类的任何属性，包括数据属性和方法

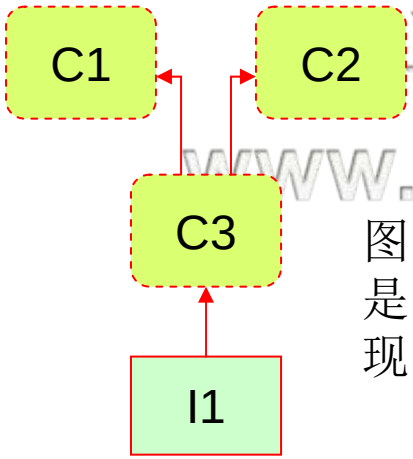
```
>>> class ParentClass(object):
        'Parent Class'
        gender = 'Male'
        def setName(self, who):
                self.name = who


>>> class ChildClass(ParentClass):
        'Child Class'
        def displayInfo(self):
                print self.gender, self.name


>>> x = ChildClass()
>>> x.setName('tom')
>>> x.displayInfo()
Male tom
>>> dir(ParentClass)
['__class__', '__delattr__', '__dict__', '__doc__', '__format__', '__getattribut
e__', '__hash__', '__init__', '__module__', '__new__', '__reduce__', '__reduce_e
x__', '__repr__', '__setattr__', '__sizeof__', '__str__', '__subclasshook__', '_
_weakref__', 'gender', 'setName']
>>> dir(ChildClass)
['__class__', '__delattr__', '__dict__', '__doc__', '__format__', '__getattribut
e__', '__hash__', '__init__', '__module__', '__new__', '__reduce__', '__reduce_e
x__', '__repr__', '__setattr__', '__sizeof__', '__str__', '__subclasshook__', '_
_weakref__', 'displayInfo', 'gender', 'setName']
```

Python类的继承和属性搜索

- Python中几乎所有属性的获取都可以使用"object.attribute"的格式
  - 不过，此表达式会在Python中启动搜索——搜索连续的树
- class语句会产生一个类对象，对class的调用会创建实例，实例自动连结至创建了此实例的类
  - 类连结至其超类的方式
    - 将超类列在类头部的括号内，其从左至右的顺序会决定树中的次序
    - 由下至上，由左至右



图中所有的对象都是名称空间，而继承就是由下而上搜索此树来寻找属性名称所出现的最低的位置

继承方法专用化

- 继承会先在子类寻找变量名，然后才查找超类，因此，子类可以对超类的属性重新定义来取代继承而来的行为
  - 子类可以完全取代从超类继承而来的属性
  - 也可以通过已覆盖的方法回调超类来扩展超类的方法

```
>>> class ParClass(object):
        def setInfo(self, sex='Male'):
                self.gender = sex


>>> class ChiClass(ParClass):
        def setInfo(self, who):
                self.name = who


>>> x = ChiClass()
>>> x.setInfo('tom')
>>> x.name
'tom'
>>> x.gender

Traceback (most recent call last):
  File "<pyshell#102>", line 1, in <module>
    x.gender
AttributeError: 'ChiClass' object has no attribute 'gender'
```

```
>>> class ParClass(object):
        def setInfo(self, sex='Male'):
                self.gender = sex


>>> class ChiClass(ParClass):
        def setInfo(self, who):
                self.name = who
                ParClass.setInfo(self)


>>> x = ChiClass()
>>> x.setInfo('tom')
>>> x.name
'tom'
>>> x.gender
'Male'
```

类、实例和其它对象的内建函数

- issubclass()
  - 布尔函数，判断一个类是否由另一个类派生，语法：
    - issubclass(sub, sup)
- isinstance()
  - 布尔函数，判断一个对象是否是给定类的实例，语法：
    - isinstance(obj1, class_obj2)
- hasattr()
  - 布尔函数，判断一个对象是否拥有指定的属性，语法：
    - hasattr(obj, 'attr')
  - 同类的函数还有getattr()、setattr()和delattr()
- super()
  - 在子类中找出其父类以便于调用其属性
  - 一般情况下仅能采用非绑定方式调用祖先类方法
  - 而super()可用于传入实例或类型对象，语法
    - super(type[, obj])

运算符重载

- 运算符重载是指在方法中拦截内置的操作——当类的实例出现在内置操作中，Python会自动调用自定义的方法，并且返回自定义方法的操作结果
  - 运算符重载让类拦截常规的Python运算
    - 类可重载所有Python表达式运算符
    - 类也可重载打印、函数调用、属性点号运算等内置运算
  - 重载使类实例的行为像内置类型
  - 重载通过提供特殊名称的类方法实现

- 运算符重载并非必需，并且通常也不是默认的

基于特殊的方法定制类

- 除了__init__和__del__之外，Python类支持使用许多的特殊方法
  - 特殊方法都以双下划线开头和结尾，有些特殊方法有默认行为，没有默认行为的是为了留到需要的时候再实现
  - 这些特殊方法是Python中用来扩充类的强大工具，它们可以实现
    - 模拟标准类型
    - 重载操作符
  - 特殊方法允许类通过重载标准操作符+，*，甚至包括分段下标及映射操作[]来模拟标准类型

常见的运算符重载方法

方法	重载	调用
__init__	构造函数	对象建立：X = Class(args)
__del__	析构函数	X对象收回
__add__	运算符+	如果没有_iadd_，X + Y，X += Y
__or__	运算符|（位OR）	如果没有_ior_，X | Y，X |= Y
__repr__ ,__str__	打印、转换	print (X)、 repr(X)、 str(X)
__call__	函数调用	X(*args,**kargs)
__getattr__	点号运算	X.undefined
__setattr__	属性赋值语句	X.any = value
__delattr__	属性删除	del X.any
__getattribute__	属性获取	X.any
__getitem__	索引运算	X[key],X[i:j]，没__iter__时的for循环和其他迭代器
__setitem__	索引赋值语句	X[key] = value，X[i:j] = sequence
__delitem__	索引和分片删除	del X[key]，del X[i:j]

常见的运算符重载方法

__len__	长度	len(X)，如果没有__bool__，真值测试
__bool__	布尔测试	bool(X)，真测试（在Python 2.6中叫做__nonzero__）
__lt__,__gt__, __le__,__ge__, __eq__,__ne__	特定的比较	X < Y, X > Y, X <= Y, X >=Y, X== Y, X != Y(或者在Python 2.6中只有__cmp__)
__radd__	右侧加法	Other + X
__iadd__	实地（增强的）加法	X += Y (or else __add__)
__iter__, __next__	迭代环境	I=iter(X), next(I); for loops, in if no __contains__, all comprehensions, map(F,X), 其他(__next__在Python 2.6中称为next)
__contains__	成员关系测试	item in X (任何可迭代的)
__index__	整数值	hex(X), bin(X), oct(X), O[X], O[X:]（替代Python 2中的__oct__、__hex__）
__enter__, __exit__	环境管理器	with obj as var:
__get__, __set__, __delete__	描述符属性	X.attr, X.attr = value, del X.attr
__new__	创建	在__init__之前创建对象

关于马哥教育

- 博客：http://mageedu.blog.51cto.com
- 主页：http://www.magedu.com
- QQ：2813150558，1661815153，113228115
- QQ群：203585050，279599283

马哥教育

Thank You!