python的面向对象编程

基本介绍

类与实例相互关联着：类是对象的定义，而实例是"真正的实物"，它存放了类中所定义的对象的具体信息。

新式类和经典类声明的最大不同在于，所有新式类必须继承至少一个父类，参数bases可以是一个（单继承）或多个（多重继承）用于继承的父类。

object 是“所有类之母”。如果你的类没有继承任何其他父类，object 将作为默认的父类。它位于所有类继承结构的最上层。如果你没有直接或间接的子类化一个对象，那么你就定义了一个经典类。

如果你没有指定一个父类，或者如果所子类化的基本类没有父类，你这样就是创建了一个经典类。

类的实例化，不用new

```
    myFirstObject = MyNewObjectType()
```

如果没有把一个实例保存到一个变量中，它会被自动垃圾收集器回收，因为任何引用指向这个实例。刚刚所做的就是为那个实例分配了一块内存，然后释放了它。

类仅用作名称空间（namespaces）。这意味着你把数据保存在变量中，对他们按名称空间进行分组，使得他们处于同样的关系空间中-----所谓的关系是使用标Python句点属性标识。

改进类的方式之一就是给类添加功能，也就是加入一些新的方法。

self参数代表对象本身，当用实例调用方法时，由解释器悄悄传给对象的。假如有一个带有两个参数的方法，所有的调用只需要传递第二个参数。

init类似于类构造器，在类被实例化时被调用。

靠继承来进行子类化是创建和定制新类类型的一种方式，新的类将保持已存在类所有的特性，而不会改动原来类的定义（指对新类的改动不会影响到原来的类）

```
    class EmplAddrBookEntry(AddrBookEntry):'Employee Address Book Entry class'
```

重点：命名类 属性 方法

类名通常由大写字母打头。这是标准惯例，可以帮助你识别类，特别是在实例化过程中(有时看起来像函数调用)。还有，数据属性（译者注：变量或常量）听起来应当是数据值的名字，方法名应当指出对应对象或值的行为。另一种表达方式是：数据值应该使用名词作为名字，方法使用谓词（动词加对象）。数据项是操作的对象、方法应当表明程序员想要在对象进行什么操作。在上面我们定义的类中，遵循了这样的方针，数据值像“name”，“phone”和“email”，行为如“updatePhone”，“updateEmail”。这就是常说的“混合记法(mixedCase)”或“骆驼记法(camelCase)”。Python 规范推荐使用骆驼记法的下划线方式，比如，“update_phone”，“update_email”。类也要细致命名，像“AddrBookEntry”，“RepairShop”等等就是很好的名字。

面向对象编程

面向对象编程与面向对象设计的关系：

面向对象设计（OOD）不会特别要求面向对象编程语言。事实上，OOD 可以由纯结构化语言来实现，比如C，但如果想要构造具备对象性质和特点的数据类型，就需要在程序上作更多的努力。当一门语言内建OO 特性，OO 编程开发就会更加方便高效。

另一方面，一门面向对象的语言不一定会强制你写OO 方面的程序。例如C++可以被认为“更好的C”；而Java，则要求万物皆类，此外还规定，一个源文件对应一个类定义。然而，在Python 中，类和OOP 都不是日常编程所必需的。尽管它从一开始设计就是面向对象的，并且结构上支持OOP，但Python 没有限定或要求你在你的应用中写OO 的代码。OOP 是一门强大的工具，不管你是准备进入，学习，过渡，或是转向OOP，都可以任意支配。

常用术语

- 抽象/实现
- 封装/接口
- 合成
- 派生/继承/继承结构
- 泛化/特化
- 多态
- 自省/反射

类

为什么是术语“class”？这个术语很可能起源于使用类来识别和归类特定生物所属的生物种族，类还可以派生出相似但有差异的子类。

在Python 中，类声明与函数声明很相似，头一行用一个相应的关键字，接下来是一个作为它的定义的代码体。 二者都允许你在他们的声明中创建函数，闭包或者内部函数（即函数内的函数），还有在类中定义的方法。

基类是一个或多个用于继承的父类的集合；类体由所有声明语句，类成员定义，数据属性和函数组成。类通常在一个模块的顶层进行定义，以便类实例能够在类所定义的源代码文件中的任何地方被创建。

Python 并不支持纯虚函数（像C++）或者抽象方法（如在JAVA 中），这些都强制程序员在子类中定义方法。作为替代方法，你可以简单地在基类方法中引发NotImplementedError 异常，这样可以获得类似的效果。

什么是属性呢？

属性就是属于另一个对象的数据或者函数元素,可以通过我们熟悉的句点属性标识法来访问。

在Python 中，方法是如何实现及调用的。通常，Python中的所有方法都有一个限制：在调用前，需要创建一个实例。

数据属性仅仅是所定义的类的变量。它们表示这些数据是与它们所属的类对象绑定的，不依赖于任何类实例。

绑定（绑定及非绑定方法）

为与OOP 惯例保持一致，Python 严格要求，没有实例，方法是不能被调用的。这种限制即Python所描述的绑定概念(binding)，在此，方法必须绑定（到一个实例）才能直接被调用。非绑定的方法可能可以被调用，但实例对象一定要明确给出，才能确保调用成功。然而，不管是否绑定，方法都是它所在的类的固有属性，即使它们几乎总是通过实例来调用的。

要知道一个类有哪些属性，有两种方法。

1. 使用dir()内建函数
2. 通过访问类的字典属性**dict**

特殊类属性

name	content
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.***__module__***	类C定义所在的模块
C.***__dict__***	类C的属性
C.***__class__***	实例C对应的类

前述的**dict**属性包含一个字典，由类的数据属性组成。访问一个类属性的时候，Python解释器将会搜索字典以得到需要的属性。如果在**dict**中没有找到，将会在基类的字典中进行搜索，采用“深度优先搜索”顺序。基类集的搜索是按顺序的，从左到右，按其在类定义时，定义父类参数时的顺序。对类的修改会仅影响到此类的字典；基类的**dict**属性不会被改动的。

Python 支持模块间的类继承。

由于类型和类的统一性，当访问任何类的**class**属性时，你将发现它就是一个类型对象的实例。换句话说，一个类已是一种类型了。因为经典类并不认同这种等价性（一个经典类是一个类对象，一个类型是一个类型对象），对这些对象来说，这个属性并未定义。

init new del的注意点

- 不要忘记首先调用父类的**del**()。
- 调用 del x 不表示调用了x.**del**() -----前面也看到，它仅仅是减少x 的引用计数。
- 如果你有一个循环引用或其它的原因，让一个实例的引用逗留不去，该对象的**del**()可能永远不会被执行。
- **del**()未捕获的异常会被忽略掉（因为一些在**del**()用到的变量或许已经被删除了）。
- 不要在**del**()中干与实例没任何关系的事情。
- 除非你知道你正在干什么，否则不要去实现**del**()。
- 如果你定义了**del**，并且实例是某个循环的一部分，垃圾回收器将不会终止这个循环——你需要自己显式调用del。

跟踪实例

Python 没有提供任何内部机制来跟踪一个类有多少个实例被创建了，或者记录这些实例是些什么东西。如果需要这些功能，你可以显式加入一些代码到类定义或者**init**()和**del**()中去。最好的方式是使用一个静态成员来记录实例的个数。靠保存它们的引用来跟踪实例对象是很危险的，因为你必须合理管理这些引用，不然，你的引用可能没办法释放（因为还有其它的引用）！

实例仅拥有数据属性（方法严格来说是类属性），后者只是与某个类的实例相关联的数据值，并且可以通过句点属性标识法来访问。这些值独立于其它实例或类。当一个实例被释放后，它的属性同时也被清除了。

实例属性

能够在“运行时”创建实例属性，是Python 类的优秀特性之一，从C++或Java 转过来的人会被小小的震惊一下，因为C++或Java 中所有属性在使用前都必须明确定义/声明。Python 不仅是动态类型，而且在运行时，允许这些对象属性的动态创建。这种特性让人爱不释手。当然，我们必须提醒读者，创建这样的属性时，必须谨慎。一个缺陷是，属性在条件语句中创建，如果该条件语句块并未被执行，属性也就不存在，而你在后面的代码中试着去访问这些属性，就会有错误发生。故事的精髓是告诉我们，Python 让你体验从未用过的特性，但如果你使用它了，你还是要小心为好。