Python面向对象编程-继承

在本节中我们构建一组类来做一些具体的事情，将会创建两个类 Person Manager.

In [1]:
```
#示例代码1.创建一个Person类

class Person:
    def __init__(self, name, age, price):
        self.name = name
        self.age = age
        self.price = price

    def say(self):
        print "My name is %s and age is %s"%(
            self.name, self.age
        )
    def raisePrice(self, percent):
        self.price = self.price * (1.1 + percent)

    def __str__(self):
        return 'Name: %s Price: %s' %(self.name,self.price)

lily = Person("lily", '20', 1000)
lily.raisePrice(0.2)
print lily
```

```
Name: lily Price: 1300.0
```

在类中编写一个运算符重载的方法， 这个方法在实例上运行的时候, 该方法截获并处理内置操作——打印操作, 运算符重载 **str**(可能是第二常用的运算符重载方法)

一. 通过子类来定制行为

例: 定义一个子类manager当其实例要涨工资的时候, 默认加多10%的额外奖金, 因为Manager类相对于类树处于底层, 所以他可以覆盖父类中的方法.

实现该需求代码有两种写法.

In [3]:
```
#示例代码2.不好的写法
class Manger(Person):
    def raisePrice(self, percent, other=0.1):
        self.price = self.price * (1.1 + percent + other)

tom = Manger("tom", 21, 1000)
tom.raisePrice(0.2)
print tom
```

```
Name: tom Price: 1400.0
```

不好的原因:

任何时候当你复制代码的时候, 几乎都会使未来的维护工作倍增, 假如用这个方式, 如果一旦改变了工资增加的方式(如不再是 1.1 + percent), 此时我们将修改两个地方代码.

In [4]:
```
#示例代码3.好的写法

class Manager(Person):
    def raisePrice(self, percent, other=0.1):
        Person.raisePrice(self, percent + other)

tom = Manager("tom", 21, 1000)
tom.raisePrice(0.2)
print tom
```

```
Name: tom Price: 1400.0
```

二. 如果不使用继承会怎样?

1. 我们可以从头编写一个Manager类, 它是全新并且独立的代码, 但是这样会导致代码的冗余, 增加未来对代码维护的工作.
2. 可以在原处修改Person类变成Manager, 但是这样可能会使原来需要Person行为的地方无法满足需求.

三.构造函数的继承

In [5]:
```
#示例代码4.

#父类A
class A(object):
    def __init__(self, name):
        self.name=name
        print "name:", self.name
    def getName(self):
        return 'A ' + self.name

#子类不重写__init__, 实例化子类时, 会自动调用父类定义的__init__

class B(A):
    def getName(self):
        return 'B '+ self.name

if __name__=='__main__':
    b=B('hello')
    print b.getName()
```

```
name: hello
B hello
```

In [6]:
```
#子类如果重写了__init__时, 实例化子类, 就不会调用父类已经定义的__init__

class B(A):
    def __init__(self, name):
        print "hi"
        self.name =  name
    def getName(self):
        return 'B '+ self.name

if __name__=='__main__':
    b=B('hello')
    print b.getName()
```

```
hi
B hello
```

结论:

1. 子类不重写**init**, 实例化子类时, 会自动调用父类定义的**init**.
2. 但重写了子类**init**时, 实例化子类, 就不会调用父类已经定义的**init**.
3. 为了能使用或扩展父类的行为，最好显示调用父类的**init**方法

In [9]:
```
#示例代码5.子类重写父类的构造函数

class c1(object):
    def __init__(self, name, age):
        self.name = name
        self.age = age
    def __str__(self):
        return 'c1 my name is %s my age is %s' %(self.name, self.age)

#第一种方式：通过c1.__init__ 来重定义构造函数，如果在构造的时候要运行更高的__init__方法，需要通过这种方式将参数传递给超类的构造函数

class c2(c1):
    def __init__(self, name, age, job):
        c1.__init__(self, name, age)
        self.job = job
    def __str__(self):
        return 'c2 name is %s age is %s job is %s' %(self.name, self.age, self.job)

x = c2('lily', '20', 'hr')
print x
```

```
c2 name is lily age is 20 job is hr
```

In [10]:
```
#第二种方式：

class c3(c1):
    def __init__(self, name, age, job):
        super(c3, self).__init__(name, age)  ##要求父类必须是新式类
        self.job = job
    def __str__(self):
        return 'c3 my name is %s age is %s job is %s' %(self.name, self.age, self.job)

x = c3("lily", '20', 'hr')
print x
```

```
c3 my name is lily age is 20 job is hr
```

In [ ]:
```

```