Python类代码编写基础

一. 类对象和实例对象
1. 类对象提供默认的行为, 它是实例对象的工厂, 来自于语句:

```
class c1:

 pass
```

2. 实例对象是程序处理的实际对象, 各自都有独立的命名空间; 来自于调用

```
x = c1()
```

二. Python类的主要特点
1. class语句创建类对象并将其赋值给变量
2. class内中的赋值语句创建了类的属性
3. 类属性提供了对象的状态和行为

三. Python类实例的特点
1. 像函数那样调用类对象会创建新的实例对象
2. 每个实例对象继承了类的属性并获取自己的命名空间
3. 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
   在类的方法函数内, 第一个参数self会引用正处理的实例对象, 对self的属性做赋值运算, 会创建或修改实例内的数据而不是类的数据.

In [4]:
```
#示例代码1.
class FirstClass:
    def setName(self, value):
        self.name = value
    def getName(self):
        print self.name
    def display(self):
        print self.name

x = FirstClass()
y = FirstClass()
x.setName('python-training')
x.getName()
y.setName('cisco')
y.getName()
```

```
python-training
cisco
```

第二个例子(继承)

除了作为工厂来生成多个实例对象之外, 类也可以引入子类来进行扩展, 而不对当前的父类进行修改, 这就是Python类的层次结构, 在阶层较低的地方覆盖现有的属性, 让行为特定化, 实际上向层次的下端越深入, 软件就会变的越特定.

In [5]:
```
#示例代码2.

class SecondClass(FirstClass):
    def display(self):
        print 'currnet value = %s' %self.name

z = SecondClass()
z.setName('second-class-python-training')
z.display()
```

```
currnet value = second-class-python-training
```

在上面的例子中SecondClass创建了其实例对象, setName依然是执行FirstClass中的版本, 但是display使用的是SecondClass中的版本, 两个类中的display方法打印的内容不相同, 可见我们不是修改FirstClass而是对它进行了定制.

In [7]:
```
#示例代码3.在模块中导入类

from demo11 import *
i1 = demo11()
print i1.x
i1.foo()
```

```
demo11 x
this is demo11 foo
```

四. 运算符重载初步

什么是运算符重载?

它可以让类截获并响应在内置类型上的运算, 如数学运算 切片 打印和点号运算等.

1. 以双下划线命名的方法 **x**, 这种特殊的命名方法, 用来拦截运算, 每种运算和特殊的命名方法之间, 存在一个固定不变的映射关系.
2. 当实例出现内置运算时（如 + -）这种类方法会自动调用; 如果实例对象继承了一个**add**方法, 当实例使用运算符+时, 这个**add**方法将会被调用, 这个方法的返回值就变成了相应表达式的结果.
3. 运算符重载几乎可以截获并实现内置类型的所有运算.
4. 最常用的运算符重载方法是 **init**.

In [9]:
```
#示例代码4.运算符重载
class ThirdClass(SecondClass):
    def __init__(self, value):
        self.name=value
    def __add__(self, other):
        print self.name + other
    def __str__(self):
        return self.name

x = ThirdClass('third-python-training')
x.display()
x + '-cisco'
```

```
currnet value = third-python-training
third-python-training-cisco
```

五. 类的命名空间

1. Python实例有自己的**dict**, 它对应的类也有自己的**dict**, 用来记录其详细属性信息.
2. 打印类的**dict**属性时, 列出了所包含的属性, 包括一些类内置属性和类变量

In [10]:
```
class cls:
    clsvar = 1
    def __init__(self):
        self.invar = 2
ins1 = cls()
ins2 = cls()

print cls.__dict__  #在类的__dict__中列出了类cls所包含的属性, 包括一些内置属性 类变量 以及构造方法.
print ins1.__dict__ #实例变量包含在对象ins1的__dict__属性中
```

```
{'clsvar': 1, '__module__': '__main__', '__doc__': None, '__init__': <function __init__ at 0x7f59080ff5f0>}
{'invar': 2}
```

一个对象的属性查找, 遵循先找实例对象自己, 然后是类, 然后是父类.

In [12]:
```
ins1.clsvar = 20
print cls.clsvar
print ins1.clsvar
print ins2.clsvar
print cls.__dict__
print ins1.__dict__
print ins2.__dict__
```

```
1
20
1
{'clsvar': 1, '__module__': '__main__', '__doc__': None, '__init__': <function __init__ at 0x7f59080ff5f0>}
{'invar': 2, 'clsvar': 20}
{'invar': 2}
```

In [14]:
```
ins1.x = 11
print ins1.x
print ins1.__dict__
```

```
11
{'invar': 2, 'x': 11, 'clsvar': 20}
```

In [16]:
```
cls.m = 21
print cls.m
print cls.__dict__
print ins1.m
print ins2.m
```

```
21
{'clsvar': 1, '__module__': '__main__', 'm': 21, '__doc__': None, '__init__': <function __init__ at 0x7f59080ff5f0>}
21
21
```

In [17]:
```
ins3 = cls()
print ins3.__dict__
print ins3.invar
print ins3.clsvar
print ins3.m
```

```
{'invar': 2}
2
1
21
```

In [20]:
```
#代码优化思路.

directory = {'a': 'apple', 'b': 'banana', 'c': 'cat'}

class c1:
    def __init__(self, directory):
        self.a = directory.get('a')
        self.b = directory.get('b')
        self.c = directory.get('c')

#优化后的代码

class c2:
    def __init__(self, directory):
        self.__dict__.update(directory)

x = c1(directory)
y = c2(directory)

print x.__dict__
print y.__dict__
```

```
{'a': 'apple', 'c': 'cat', 'b': 'banana'}
{'a': 'apple', 'c': 'cat', 'b': 'banana'}
```

In [ ]: