7.结构型模式-适配器模式

概述：
　　结构型模式主要用于处理系统中不同实体间的关系，如话题与评论这类场景．今天介绍其中的适配器模式：适配器模式，通俗的说就是设计接口/api，以保证程序符合开放/封闭原则，保持新老代码间的兼容性．适配器模式在实际开发中的使用非常频繁，通常在系统的不同层次间，以及为系统添加扩展功能等场景时使用．因为通常情况下，原系统的代码要么无法获取—如库等、要么难以冒险重构—如运行5年以上的老旧系统牵一发而动全身，在设计中使用适配器模式，可以保证在不修改原系统代码的前提下，实现新需求与原系统的对接．

示例代码：存在一套旧系统，里面包含Human和Synthesizer类

In [6]:
```
class Human:
    def __init__(self, name):
        self.name = name
    def __str__(self):
        print("My name is %s" %self.name)
    def speak(self):
        print("This is %s speaking.." %self.name)

class Synthesizer:
    def __init__(self, name):
        self.name = name
    def __str__(self):
        print("This is synth %s" %self.name)
    def play(self):
        print("I %s like play" %self.name)
```

现在新增 Computer 类如下：

In [ ]:
```
class Computer:
    def __init__(self, name):
        self.name = name
    def __str__(self):
        print("this is Computer %s.." %self.name)
    def execute(self):
        print("I can execute programs")
```

　　并且对于扩展系统来说，所有动作函数均使用Obj.execute()来执行．即对于调用者来说，原系统的Synthesizer.play()和 Human.speak()是不存在的，必须像调用Computer.execute()一样使用Synthesizer.execute()和Human.execute()来调用原系统中对象的执行函数，由于程序的设计需符合开放/封闭原则：

　　开放：对扩展开放，意味着有新的需求或变化时，可以对现有代码进行扩展，以适应新的情况．
　　封闭：对修改封闭，意味着类一旦设计完成，就可以独立完成其工作，而不要对类进行任何修改．

所以我们无法修改原系统函数，此时新系统就可以采用适配器模式进行设计．我们可以创建一个Adapter类专门用于统一接口，代码如下：

In [5]:
```
class Adapter:
    def __init__(self, obj, kwargs={}):
        self.obj = obj
        self.__dict__.update(kwargs)

    def __str__(self):
        print("this is Adapter..")
```

　　简单来说，这里是使用了Python的一个特殊语法特性class.__dict__属性，即类的内部字典．这个特殊的属性是一个字典，存放了这个类所包含的所有属性和方法，所以这里将传入的类进行处理，将需要被适配器处理的方法添加到内部字典中，生成一个属于这个新适配器对象的方法．

In [7]:
```
def main():
    c = Computer("mac book pro")
    h = Human("harry")
    s = Synthesizer("synth")

    a2 = Adapter(h, {"execute": h.speak})
    a3 = Adapter(s, {"execute": s.play})

    for j in [c, a2, a3]:
        j.execute()

if __name__ == '__main__':
    main()
```

```
I can execute programs
This is harry speaking..
I synth like play
```