School of Economics ang Management, Beihang University

现代程序设计技术

赵吉昌
jichang@buaa.edu.cn

本周内容

- 面向对象编程
  - 观察者模式
  - 抽象类
  - 接口
  - 泛函数

观察者模式

- 亦称
  - 发布（publish ）-订阅（Subscribe）模式
  - 模型-视图（View）模式
  - 源-收听者(Listener)模式
  - 从属者模式
- 要义
  - 一个目标对象管理所有依赖于它的观察者对象，并且在它本身的状态改变时主动发出通知
  - 观察者模式完美地将观察者和被观察的对象分离开

观察者模式

- 优点
  - 观察者与被观察者之间抽象耦合
  - 可以触发多个符合单一职责的模块
  - 可以很方便地实现广播
- 场景
  - 消息交换，如消息队列；
  - 多级触发，如一个中断即会引发一连串反应
- 缺点
  - 效率不一定高

观察者模式

- Demo 1
  - ps1.py
- Demo 2
  - ps2.py

抽象类

- type()函数
  - 并非仅仅返回对象的类型
  - Python使用type()函数创建类对象
    - 函数和类不是编译时定义的,而是在运行时动态创建的
  - type()函数依次传入3个参数
    - 类名称
    - 继承的父类集合(tuple)
    - 属性（数据或方法）字典
  - Demo: mcl.py

抽象类

- 元类
  - metaclass
  - 控制类的创建行为
    - 先定义metaclass，然后创建类
    - 先定义metaclass，就可以创建类，最后创建实例
    - 类可以看成metaclass创建的“实例”
  - 元类的定义
    - metaclass的类名总是以Metaclass结尾
    - metaclass是类的模板，所以必须从type类型派生
  - 元类的使用
    - 通过关键字参数metaclass来指定元类来创建类
    - Demo: listm.py

抽象类

- 元类
  - 为什么要动态修改？
    - 可以直接在继承list的类里添加add方法
  - 但有时需要动态定义类
    - Object Relational Mapping(ORM)
    - 关系数据库的一行映射为一个实例对象，也就是一个类对应一个表
    - “透明化”数据库相关的操作
    - 类需要动态定义
    - Demo: orm.py

抽象类

- 抽象类
  - abstract class
  - 特殊的类，只能被继承，不能被实例
  - 从不同的类中抽取相同的属性和行为
- 抽象类与普通类的区别
  - 抽象类中有抽象方法
  - 不能被实例化，只能被继承
  - 子类必须实现抽象方法

抽象类

- 抽象类的实现
  - 借助abc模块实现
  - `import abc`
  - `class Fruit(metaclass=abc.ABCMeta):`
  - `@abc.abstractmethod`
  - `def harvest(self):`
  - `pass`
  - `@abc.abstractmethod`
  - `def grow(self):`
  - `pass`
  - 也可以写成：`class Fruit(abc.ABC)`

抽象类

- 抽象类的实现
  - 继承抽象类
    - `class Watermelon(Fruit):`
    - `    def harvest(self):`
    - `        print("从地里摘")`
    - `    def grow(self):`
    - `        print("用种子种")`
  - 注册抽象类
    - `@Fruit.register #Fruit.register(Orange)`
    - `class Orange:`
    - `    def harvest(self):`
    - `        print("从树上摘")`
    - `    def grow(self):`
    - `        print("种橘子树")`

抽象类

- 抽象类的实现
  - Demo: ac.py
  - 继承与注册的差别
  - `w=Watermelon()`
  - `o=Orange()`
  - `isinstance(w,Fruit)`
  - `isintance(o,Fruit)`
  - `issubclass(Watermelon,Fruit)`
  - `issubclass(Orange,Fruit)`
  - `print([sc.__name__ for sc in Fruit.__subclasses__()])`
    - 不会包含注册的子类

接口

- 接口
  - Interface
  - Python中没有专门的支持
  - 但可以约定任何方法都只是一种规范，具体的功能需要子类实现
  - 与抽象类的区别
    - 抽象类中可以对一些抽象方法做出基础实现
    - 接口中所有方法只是规范，不做任何实现
  - 使用原则
    - 继承抽象类应该尽量避免多继承
    - 继承接口鼓励多继承

泛函数

- 泛函数
  - 函数对不同的参数类型会提供不同的处理方式
  - 通过装饰器来实现
  - 类似于重载机制
    - `from functools import singledispatch`
    - `@singledispatch`
    - `def func():`
      - `pass`
    - `@func.register(int)#注册到int的处理`
    - `def _(num):`
      - `pass`
    - `Demo: olf.py`