设计模式之python

软件设计模式：

又称设计模式（Design pattern），是一套被反复使用、多数人知晓的、经过分类编目的、代码设计经验 的总结。使用设计模式是为了可重用代码、让代码更容易被他人理解、保证代码可靠性、程序的重用性。它 不限于一种特定的语言，它是一种解决问题的思想和方法。

设计模式的意义：

1. 设计模式以一种标准的方式供广大开发人员使用，为开发者的沟通提供了一套机制，帮助开发者更好地明白和更清晰地描述一段被给出的代码。
2. 设计模式可以使人们更加方便简单复用成功的设计模式和结构。
3. 设计模式可以使人们深入理解面向对象的设计思想，提高软件的开发效率，节约设计成本。

设计模式6大原则

1.开闭原则（Open Close Principle）

开闭原则就是说对扩展开放，对修改关闭。在程序需要进行拓展的时候，不能去修改原有的代码，实现一个热插拔的效果。所以一句话概括就是：为了使程序的扩展性好，易于维护和升级。想要达到这样的效果，我们需要使用接口和抽象类，后面的具体设计中我们会提到这点。

2.里氏代换原则（Liskov Substitution Principle）

里氏代换原则(Liskov Substitution Principle LSP)面向对象设计的基本原则之一。 里氏代换原则中说，任何基类可以出现的地方，子类一定可以出现。 LSP是继承复用的基石，只有当衍生类可以替换掉基类，软件单位的功能不受到影响时，基类才能真正被复用，而衍生类也能够在基类的基础上增加新的行为。里氏代换原则是对“开-闭”原则的补充。实现“开-闭”原则的关键步骤就是抽象化。而基类与子类的继承关系就是抽象化的具体实现，所以里氏代换原则是对实现抽象化的具体步骤的规范。—— From Baidu 百科

3.依赖倒转原则（Dependence Inversion Principle）

这个是开闭原则的基础，具体内容：是对接口编程，依赖于抽象而不依赖于具体。

4.接口隔离原则（Interface Segregation Principle）

这个原则的意思是：使用多个隔离的接口，比使用单个接口要好。还是一个降低类之间的耦合度的意思，从这儿我们看出，其实设计模式就是一个软件的设计思想，从大型软件架构出发，为了升级和维护方便。所以上文中多次出现：降低依赖，降低耦合。

5.迪米特法则（最少知道原则）（Demeter Principle）

为什么叫最少知道原则，就是说：一个实体应当尽量少的与其他实体之间发生相互作用，使得系统功能模块相对独立。

6.合成复用原则（Composite Reuse Principle）

原则是尽量使用合成/聚合的方式，而不是使用继承。

设计模式分类

1.创建型模式：

理论：现阶段，社会化的分工越来越细，自然在软件设计方面也是如此，因此对象的创建和对象 的使用分开也就成为了必然趋势。因为对象的创建会消耗掉系统的很多资源，所以单独对对象的 创建进行研究，从而能够高效地创建对象就是创建型模式要探讨的问题。这里有6个具体的创建型 模式可供研究，它们分别是：

- 1.工厂方法模式（Factory Method ）
- 2.抽象工厂模式（Abstract Factory）
- 3.创建者模式（Builder）
- 4.原型模式（Prototype）
- 5.单例模式（Singleton）

2.结构型模式：

理论：在解决了对象的创建问题之后，对象的组成以及对象之间的依赖关系就成了开发人员关注的焦点，因为如何设计对象的结构、继承和依赖关系会影响到后续程序的维护性、代码的健壮性、耦合性等。对象结构的设计很容易体现出设计人员水平的高低，这里有7个具体的结构型模式可供研究，它们分别是：

- 6.适配器模式（Adapter）
- 7.代理模式（proxy）
- 8.装饰模式 （Decorator）
- 9.桥接模式（Bridge,多维度）
- 10.组合模式（Composite）
- 11.外观模式 （Facade）
- 12.享元模式（Flyweight）

3.行为型模式：

理论：在对象的结构和对象的创建问题都解决了之后，就剩下对象的行为问题了，如果对象的行为设计的好，那么对象的行为就会更清晰，它们之间的协作效率就会提高，这里有11个具体的行为型模式可供研究，它们分别是：

- 13.解释器模式(Interpreter)
- 14.模板方法（Template Method）
- 15.责任链模式（Chain of Responsibility）
- 16.命令模式（Command）
- 17.迭代器模式（Iterator）
- 18.中介者模式（Mediator）
- 19.备忘录模式（Memento）
- 20.观察者（Observer）
- 21.状态模式（State）
- 22.策略模式（Strategy）
- 23.访问者模式（Visitor）

实战实例

创建型实例

1.工厂模式

```
# -*- coding: utf-8 -*-
class Burger():
    name=""
    price=0.0
    def getPrice(self):
        return self.price
    def setPrice(self,price):
        self.price=price
    def getName(self):
        return self.name
class cheeseBurger(Burger):
    def __init__(self):
        self.name="cheese burger"
        self.price=10.0
class spicyChickenBurger(Burger):
    def __init__(self):
        self.name="spicy chicken burger"
        self.price=15.0

class Snack():
    name = ""
    price = 0.0
    type = "SNACK"
    def getPrice(self):
        return self.price
    def setPrice(self, price):
        self.price = price
    def getName(self):
        return self.name


class chips(Snack):
    def __init__(self):
        self.name = "chips"
        self.price = 6.0


class chickenWings(Snack):
    def __init__(self):
        self.name = "chicken wings"
        self.price = 12.0

class Beverage():
    name = ""
    price = 0.0
    type = "BEVERAGE"
    def getPrice(self):
        return self.price
    def setPrice(self, price):
        self.price = price
    def getName(self):
```

```
        return self.name


class coke(Beverage):
    def __init__(self):
        self.name = "coke"
        self.price = 4.0


class milk(Beverage):
    def __init__(self):
        self.name = "milk"
        self.price = 5.0


class foodFactory():
    type=""
    def createFood(self,foodClass):
        print (self.type," factory produce a instance.")
        foodIns=foodClass()
        return foodIns
class burgerFactory(foodFactory):
    def __init__(self):
        self.type="BURGER"
class snackFactory(foodFactory):
    def __init__(self):
        self.type="SNACK"
class beverageFactory(foodFactory):
    def __init__(self):
        self.type="BEVERAGE"


if  __name__=="__main__":
    burger_factory=burgerFactory()
    snack_factorry=snackFactory()
    beverage_factory=beverageFactory()
    cheese_burger=burger_factory.createFood(cheeseBurger)
    print (cheese_burger.getName(),cheese_burger.getPrice())
    chicken_wings=snack_factorry.createFood(chickenWings)
    print (chicken_wings.getName(),chicken_wings.getPrice())
    coke_drink=beverage_factory.createFood(coke)
    print (coke_drink.getName(),coke_drink.getPrice())
```

2.抽象工厂模式

```
#-*- coding:utf-8 -*-
'''
Abstract Factory
'''
```

```
class AbstractFactory(object):
    computer_name = ''
    def createCpu(self):
        pass
    def createMainboard(self):
        pass

class IntelFactory(AbstractFactory):
    computer_name = 'Intel I7-series computer '
    def createCpu(self):
        return IntelCpu('I7-6500')

    def createMainboard(self):
        return IntelMainBoard('Intel-6000')

class AmdFactory(AbstractFactory):
    computer_name = 'Amd 4 computer '

    def createCpu(self):
        return AmdCpu('amd444')

    def createMainboard(self):
        return AmdMainBoard('AMD-4000')

class AbstractCpu(object):
    series_name = ''
    instructions = ''
    arch=''

class IntelCpu(AbstractCpu):
    def __init__(self,series):
        self.series_name = series

class AmdCpu(AbstractCpu):
    def __init__(self,series):
        self.series_name = series

class AbstractMainboard(object):
    series_name = ''

class IntelMainBoard(AbstractMainboard):
    def __init__(self,series):
        self.series_name = series

class AmdMainBoard(AbstractMainboard):
    def __init__(self,series):
        self.series_name = series

class ComputerEngineer(object):

    def makeComputer(self,factory_obj):
```

```
        self.prepareHardwares(factory_obj)

    def prepareHardwares(self,factory_obj):
        self.cpu = factory_obj.createCpu()
        self.mainboard = factory_obj.createMainboard()

        info = '''------- computer [%s] info:
    cpu: %s
    mainboard: %s
 -------- End --------
        '''% (factory_obj.computer_name,self.cpu.series_name,self.mainboard.series_name)
        print(info)
if __name__ == "__main__":
    engineer = ComputerEngineer()   #装机工程师

    intel_factory = IntelFactory()    #intel工厂
    engineer.makeComputer(intel_factory)

    amd_factory = AmdFactory()      #adm工厂
    engineer.makeComputer(amd_factory)
```

3.创建者模式

```
# -*- coding: utf-8 -*-

class Burger():
    name=""
    price=0.0
    def getPrice(self):
        return self.price
    def setPrice(self,price):
        self.price=price
    def getName(self):
        return self.name
class cheeseBurger(Burger):
    def __init__(self):
        self.name="cheese burger"
        self.price=10.0
class spicyChickenBurger(Burger):
    def __init__(self):
        self.name="spicy chicken burger"
        self.price=15.0


class Snack():
    name = ""
    price = 0.0
    type = "SNACK"
    def getPrice(self):
        return self.price
    def setPrice(self, price):
```

```
        self.price = price
    def getName(self):
        return self.name


class chips(Snack):
    def __init__(self):
        self.name = "chips"
        self.price = 6.0


class chickenWings(Snack):
    def __init__(self):
        self.name = "chicken wings"
        self.price = 12.0

class Beverage():
    name = ""
    price = 0.0
    type = "BEVERAGE"
    def getPrice(self):
        return self.price
    def setPrice(self, price):
        self.price = price
    def getName(self):
        return self.name


class coke(Beverage):
    def __init__(self):
        self.name = "coke"
        self.price = 4.0


class milk(Beverage):
    def __init__(self):
        self.name = "milk"
        self.price = 5.0

class order():
    burger=""
    snack=""
    beverage=""
    def __init__(self,orderBuilder):
        self.burger=orderBuilder.bBurger
        self.snack=orderBuilder.bSnack
        self.beverage=orderBuilder.bBeverage
    def show(self):
        print ("Burger:%s"%self.burger.getName())
        print("Snack:%s"%self.snack.getName())
        print("Beverage:%s"%self.beverage.getName())

class orderBuilder():
```

```
    bBurger=""
    bSnack=""
    bBeverage=""
    def addBurger(self,xBurger):
        self.bBurger=xBurger
    def addSnack(self,xSnack):
        self.bSnack=xSnack
    def addBeverage(self,xBeverage):
        self.bBeverage=xBeverage
    def build(self):
        return order(self)



if  __name__=="__main__":
    order_builder=orderBuilder()
    order_builder.addBurger(spicyChickenBurger())
    order_builder.addSnack(chips())
    order_builder.addBeverage(milk())
    order_1=order_builder.build()
    order_1.show()
```

4.原型模式

```
# -*- coding: utf-8 -*-

class simpleLayer:
    background=[0,0,0,0]
    content="blank"
    def getContent(self):
        return self.content
    def getBackgroud(self):
        return self.background
    def paint(self,painting):
        self.content=painting
    def setParent(self,p):
        self.background[3]=p
    def fillBackground(self,back):
        self.background=back

if  __name__=="__main__":
    dog_layer=simpleLayer()
    dog_layer.paint("Dog")
    dog_layer.fillBackground([0,0,255,0])
    print ("Background:",dog_layer.getBackgroud())
    print ("Painting:",dog_layer.getContent())


#拷贝图层
from copy import copy, deepcopy
class simpleLayer:
```

```
    background=[0,0,0,0]
    content="blank"
    def getContent(self):
        return self.content
    def getBackgroud(self):
        return self.background
    def paint(self,painting):
        self.content=painting
    def setParent(self,p):
        self.background[3]=p
    def fillBackground(self,back):
        self.background=back
    def clone(self):
        return copy(self)
    def deep_clone(self):
        return deepcopy(self)
if  __name__=="__main__":
    dog_layer=simpleLayer()
    dog_layer.paint("Dog")
    dog_layer.fillBackground([0,0,255,0])
    print ("Background:",dog_layer.getBackgroud())
    print ("Painting:",dog_layer.getContent())
    another_dog_layer=dog_layer.clone()
    print ("Background:", another_dog_layer.getBackgroud())
    print ("Painting:", another_dog_layer.getContent())
```

5.单例模式

```
#encoding=utf8
import threading
import time
#这里使用方法__new__来实现单例模式
class Singleton(object):#抽象单例
    def __new__(cls, *args, **kw):
        if not hasattr(cls, '_instance'):
            orig = super(Singleton, cls)
            cls._instance = orig.__new__(cls, *args, **kw)
        return cls._instance
#总线
class Bus(Singleton):
    lock = threading.RLock()
    def sendData(self,data):
        self.lock.acquire()
        time.sleep(3)
        print ("Sending Signal Data...",data)
        self.lock.release()
#线程对象，为更加说明单例的含义，这里将Bus对象实例化写在了run里
class VisitEntity(threading.Thread):
    my_bus=""
    name=""
```

```
    def getName(self):
        return self.name
    def setName(self, name):
        self.name=name
    def run(self):
        self.my_bus=Bus()
        self.my_bus.sendData(self.name)

if  __name__=="__main__":
    for i in range(3):
        print ("Entity %d begin to run..."%i)
        my_entity=VisitEntity()
        my_entity.setName("Entity_"+str(i))
        my_entity.start()
```

结构型实例

6.适配器模式

```
#定义两个公司的员工类
class ACpnStaff:
    name=""
    id=""
    phone=""
    def __init__(self,id):
        self.id=id
    def getName(self):
        print ("A protocol getName method...id:%s"%self.id)
        return self.name
    def setName(self,name):
        print ("A protocol setName method...id:%s"%self.id)
        self.name=name
    def getPhone(self):
        print ("A protocol getPhone method...id:%s"%self.id)
        return self.phone
    def setPhone(self,phone):
        print ("A protocol setPhone method...id:%s"%self.id)
        self.phone=phone
class BCpnStaff:
    name=""
    id=""
    telephone=""
    def __init__(self,id):
        self.id=id
    def get_name(self):
```

```
        print ("B protocol get_name method...id:%s"%self.id)
        return self.name
    def set_name(self,name):
        print( "B protocol set_name method...id:%s"%self.id)
        self.name=name
    def get_telephone(self):
        print ("B protocol get_telephone method...id:%s"%self.id)
        return self.telephone
    def set_telephone(self,telephone):
        print ("B protocol get_name method...id:%s"%self.id)
        self.telephone=telephone
#适配器构造
class CpnStaffAdapter:
    b_cpn=""
    def __init__(self,id):
        self.b_cpn=BCpnStaff(id)
    def getName(self):
        return self.b_cpn.get_name()
    def getPhone(self):
        return self.b_cpn.get_telephone()
    def setName(self,name):
        self.b_cpn.set_name(name)
    def setPhone(self,phone):
        self.b_cpn.set_telephone(phone)
if __name__=="__main__":
    acpn_staff=ACpnStaff("123")
    acpn_staff.setName("X-A")
    acpn_staff.setPhone("10012345678")
    print ("A Staff Name:%s"%acpn_staff.getName())
    print ("A Staff Phone:%s"%acpn_staff.getPhone())
    bcpn_staff=CpnStaffAdapter("456")
    bcpn_staff.setName("Y-B")
    bcpn_staff.setPhone("99987654321")
    print ("B Staff Name:%s"%bcpn_staff.getName())
    print ("B Staff Phone:%s"%bcpn_staff.getPhone())
```

7.代理模式

```
#该服务器接受如下格式数据，addr代表地址，content代表接收的信息内容
info_struct=dict()
info_struct["addr"]=10000
info_struct["content"]=""
class Server:
    content=""
    def recv(self,info):
        pass
    def send(self,info):
        pass
    def show(self):
        pass
class infoServer(Server):
```

```
    def recv(self,info):
        self.content=info
        return "recv OK!"
    def send(self,info):
        pass
    def show(self):
        print ("SHOW:%s"%self.content)


class serverProxy:
    pass
class infoServerProxy(serverProxy):
    server=""
    def __init__(self,server):
        self.server=server
    def recv(self,info):
        return self.server.recv(info)
    def show(self):
        self.server.show()

class whiteInfoServerProxy(infoServerProxy):
    white_list=[]
    def recv(self,info):
        try:
            assert type(info)==dict
        except:
            return "info structure is not correct"
        addr=info.get("addr",0)
        if not addr in self.white_list:
            return "Your address is not in the white list."
        else:
            content=info.get("content","")
            return self.server.recv(content)
    def addWhite(self,addr):
        self.white_list.append(addr)
    def rmvWhite(self,addr):
        self.white_list.remove(addr)
    def clearWhite(self):
        self.white_list=[]

if  __name__=="__main__":
    info_struct = dict()
    info_struct["addr"] = 10010
    info_struct["content"] = "Hello World!"
    info_server = infoServer()
    info_server_proxy = whiteInfoServerProxy(info_server)
    print (info_server_proxy.recv(info_struct))
    info_server_proxy.show()
    info_server_proxy.addWhite(10010)
    print (info_server_proxy.recv(info_struct))
    info_server_proxy.show()
```

8.装饰模式

```
class Beverage():
    name = ""
    price = 0.0
    type = "BEVERAGE"
    def getPrice(self):
        return self.price
    def setPrice(self, price):
        self.price = price
    def getName(self):
        return self.name


class coke(Beverage):
    def __init__(self):
        self.name = "coke"
        self.price = 4.0


class milk(Beverage):
    def __init__(self):
        self.name = "milk"
        self.price = 5.0

#定义装饰器类
class drinkDecorator():
    def getName(self):
        pass

    def getPrice(self):
        pass


class iceDecorator(drinkDecorator):
    def __init__(self, beverage):
        self.beverage = beverage

    def getName(self):
        return self.beverage.getName() + " +ice"

    def getPrice(self):
        return self.beverage.getPrice() + 0.3


class sugarDecorator(drinkDecorator):
    def __init__(self, beverage):
        self.beverage = beverage

    def getName(self):
        return self.beverage.getName() + " +sugar"

    def getPrice(self):
```

```
        return self.beverage.getPrice() + 0.5

if  __name__=="__main__":
    coke_cola=coke()
    print ("Name:%s"%coke_cola.getName())
    print ("Price:%s"%coke_cola.getPrice())
    ice_coke=iceDecorator(coke_cola)
    print ("Name:%s" % ice_coke.getName())
    print ("Price:%s" % ice_coke.getPrice())
```

9.桥接模式

```
class Shape:
    name=""
    param=""
    def __init__(self,*param):
        pass
    def getName(self):
        return self.name
    def getParam(self):
        return self.name,self.param

class Pen:
    shape=""
    type=""
    def __init__(self,shape):
        self.shape=shape
    def draw(self):
        pass

#构造形状：矩形和原圆形
class Rectangle(Shape):
    def __init__(self,long,width):
        self.name="Rectangle"
        self.param="Long:%s Width:%s"%(long,width)
        print ("Create a rectangle:%s"%self.param)
class Circle(Shape):
    def __init__(self,radius):
        self.name="Circle"
        self.param="Radius:%s"%radius
        print ("Create a circle:%s"%self.param)

#构造画笔：普通画笔和画刷
class NormalPen(Pen):
    def __init__(self,shape):
        Pen.__init__(self,shape)
        self.type="Normal Line"
    def draw(self):
        print ("DRAWING %s:%s----PARAMS:%s"%
(self.type,self.shape.getName(),self.shape.getParam()))
class BrushPen(Pen):
```

```
    def __init__(self,shape):
        Pen.__init__(self,shape)
        self.type="Brush Line"
    def draw(self):
        print ("DRAWING %s:%s----PARAMS:%s" % (self.type,self.shape.getName(),
self.shape.getParam()))

if __name__ == "__main__":
    normal_pen = NormalPen(Rectangle("20cm", "10cm"))
    brush_pen = BrushPen(Circle("15cm"))
    normal_pen.draw()
    brush_pen.draw()
```

10.组合模式

```
class Company:
    name = ''
    def __init__(self, name):
        self.name = name
    def add(self, company):
        pass
    def remove(self, company):
        pass
    def display(self, depth):
        pass
    def listDuty(self):
        pass

#混凝土公司
class ConcreteCompany(Company):
    childrenCompany = None
    def __init__(self, name):
        Company.__init__(self,name)
        self.childrenCompany = []
    def add(self, company):
        self.childrenCompany.append(company)
    def remove(self, company):
        self.childrenCompany.remove(company)
    def display(self, depth):
        print('-'*depth + self.name)
        for component in self.childrenCompany:
            component.display(depth+1)
    def listDuty(self):
        for component in self.childrenCompany:
            component.listDuty()
#人力资源部
class HRDepartment(Company):
    def __init__(self, name):
         Company.__init__(self,name)
    def display(self, depth):
        print ('-'*depth + self.name)
```

```
    def listDuty(self): #履行职责(注册与转移管理)
        print ('%s\t Enrolling & Transfering management.' % self.name)
#财务部
class FinanceDepartment(Company):
    def __init__(self, name):
        Company.__init__(self,name)
    def display(self, depth):
        print ("-" * depth + self.name)
    def listDuty(self): #履行职责
        print ('%s\tFinance Management.'%self.name)
#研发部门
class RdDepartment(Company):
    def __init__(self,name):
        Company.__init__(self,name)
    def display(self, depth):
        print ("-"*depth+self.name)
    def listDuty(self):
        print ("%s\tResearch & Development."% self.name)

'''假设总公司下设东边的分公司一个，东边的分公司下设东北公司和东南' \
公司，显示公司层级，并罗列这些的公司中各部门的职责，可以构建如下业务场景：'''
if __name__=="__main__":
    root = ConcreteCompany('HeadQuarter')#总部
    root.add(HRDepartment('HQ HR'))
    root.add(FinanceDepartment('HQ Finance'))
    root.add(RdDepartment("HQ R&D"))

    comp = ConcreteCompany('East Branch')
    comp.add(HRDepartment('East.Br HR'))
    comp.add(FinanceDepartment('East.Br Finance'))
    comp.add(RdDepartment("East.Br R&D"))
    root.add(comp)

    comp1 = ConcreteCompany('Northast Branch')
    comp1.add(HRDepartment('Northeast.Br HR'))
    comp1.add(FinanceDepartment('Northeast.Br Finance'))
    comp1.add(RdDepartment("Northeast.Br R&D"))
    comp.add(comp1)

    comp2 = ConcreteCompany('Southeast Branch')
    comp2.add(HRDepartment('Southeast.Br HR'))
    comp2.add(FinanceDepartment('Southeast.Br Finance'))
    comp2.add(RdDepartment("Southeast.Br R&D"))
    comp.add(comp2)

    root.display(1)

    root.listDuty()
```

11.外观模式

```
class AlarmSensor:
    def run(self):
        print ("Alarm Ring...")
class WaterSprinker:
    def run(self):
        print ("Spray Water...")
class EmergencyDialer:
    def run(self):
        print ("Dial 119...")

 #可以如下操作
# if __name__=="__main__":
#     alarm_sensor=AlarmSensor()
#     water_sprinker=WaterSprinker()
#     emergency_dialer=EmergencyDialer()
#     alarm_sensor.run()
#     water_sprinker.run()
#     emergency_dialer.run()

#门面构建
class EmergencyFacade:
    def __init__(self):
        self.alarm_sensor=AlarmSensor()
        self.water_sprinker=WaterSprinker()
        self.emergency_dialer=EmergencyDialer()
    def runAll(self):
        self.alarm_sensor.run()
        self.water_sprinker.run()
        self.emergency_dialer.run()

if __name__=="__main__":
    emergency_facade=EmergencyFacade()
    emergency_facade.runAll()
```

12.享元模式

```
class Coffee:
    name = ''
    price =0
    def __init__(self,name):
        self.name = name
        self.price = len(name)#在实际业务中，咖啡价格应该是由配置表进行配置，或者调用接口获取等方式得到，此处为说明享元模式，将咖啡价格定为名称长度，只是一种简化
    def show(self):
        print ("Coffee Name:%s Price:%s"%(self.name,self.price))
#顾客类
class Customer:
    name=""
    def __init__(self,name):
        self.name=name
    def order(self,coffee_name):
```

```
        print ("%s ordered a cup of coffee:%s"%(self.name,coffee_name))
        return Coffee(coffee_name)
#控制实例化的类：咖啡工厂
class CoffeeFactory():
    coffee_dict = {}
    def getCoffee(self, name):
        if self.coffee_dict.has_key(name) == False:
            self.coffee_dict[name] = Coffee(name)
        return self.coffee_dict[name]
    def getCoffeeCount(self):
        return len(self.coffee_dict)

#重写Customer类
class Customer:
    coffee_factory=""
    name=""
    def __init__(self,name,coffee_factory):
        self.name=name
        self.coffee_factory=coffee_factory
    def order(self,coffee_name):
        print ("%s ordered a cup of coffee:%s"%(self.name,coffee_name))
        return self.coffee_factory.getCoffee(coffee_name)
#假设业务中短时间内有多人订了咖啡，业务模拟如下：
if __name__=="__main__":
    coffee_factory=CoffeeFactory()
    customer_1=Customer("A Client",coffee_factory)
    customer_2=Customer("B Client",coffee_factory)
    customer_3=Customer("C Client",coffee_factory)
    c1_capp=customer_1.order("cappuccino") #c1_capp="cappuccino"
    c1_capp.show()
    c2_mocha=customer_2.order("mocha")
    c2_mocha.show()
    c3_capp=customer_3.order("cappuccino")
    c3_capp.show()
    print ("Num of Coffee Instance:%s"%coffee_factory.getCoffeeCount())
```

行为型实例

13.解释器模式

```
#模拟吉他
class PlayContext():
    play_text = None

class Expression():
```

```
    def interpret(self, context):
        if len(context.play_text) == 0:
            return
        else:
            play_segs=context.play_text.split(" ")
            for play_seg in play_segs:
                pos=0
                for ele in play_seg:
                    if ele.isalpha():
                        pos+=1
                        continue
                    break
                play_chord = play_seg[0:pos]
                play_value = play_seg[pos:]
                self.execute(play_chord,play_value)
    def execute(self,play_key,play_value):
        pass

class NormGuitar(Expression):
    def execute(self, key, value):
        print ("Normal Guitar Playing--Chord:%s Play Tune:%s"%(key,value))

if __name__=="__main__":
    context = PlayContext()
    context.play_text = "C53231323 Em43231323 F43231323 G63231323"
    guitar=NormGuitar()
    guitar.interpret(context)
```

14.模板方法

```
#虚拟股票查询器
# class StockQueryDevice():
#     stock_code="0"
#     stock_price=0.0
#     def login(self,usr,pwd):
#         pass
#     def setCode(self,code):
#         self.stock_code=code
#     def queryPrice(self):
#         pass
#     def showPrice(self):
#         pass
class StockQueryDevice():
    stock_code="0"
    stock_price=0.0
    def login(self,usr,pwd):
        pass
    def setCode(self,code):
        self.stock_code=code
    def queryPrice(self):
        pass
```

```
    def showPrice(self):
        pass

    def operateQuery(self, usr, pwd, code):
        if not self.login(usr, pwd):
            return False
        self.setCode(code)
        self.queryPrice()
        self.showPrice()
        return True
#WebA和WebB的查询器
class WebAStockQueryDevice(StockQueryDevice):
    def login(self,usr,pwd):
        if usr=="myStockA" and pwd=="myPwdA":
            print ("Web A:Login OK... user:%s pwd:%s"%(usr,pwd))
            return True
        else:
            print ("Web A:Login ERROR... user:%s pwd:%s"%(usr,pwd))
            return False
    def queryPrice(self):
        print ("Web A Querying...code:%s "%self.stock_code)
        self.stock_price=20.00
    def showPrice(self):
        print ("Web A Stock Price...code:%s price:%s"%(self.stock_code,self.stock_price))
class WebBStockQueryDevice(StockQueryDevice):
    def login(self,usr,pwd):
        if usr=="myStockB" and pwd=="myPwdB":
            print ("Web B:Login OK... user:%s pwd:%s"%(usr,pwd))
            return True
        else:
            print ("Web B:Login ERROR... user:%s pwd:%s"%(usr,pwd))
            return False
    def queryPrice(self):
        print ("Web B Querying...code:%s "%self.stock_code)
        self.stock_price=30.00
    def showPrice(self):
        print ("Web B Stock Price...code:%s price:%s"%(self.stock_code,self.stock_price))
# #查询A股票
# if  __name__=="__main__":
#     web_a_query_dev=WebAStockQueryDevice()
#     web_a_query_dev.login("myStockA","myPwdA")
#     web_a_query_dev.setCode("12345")
#     web_a_query_dev.queryPrice()
#     web_a_query_dev.showPrice()


if  __name__=="__main__":
    web_a_query_dev=WebAStockQueryDevice()
    web_a_query_dev.operateQuery("myStockA","myPwdA","12345")
```

15.责任链模式

```
class manager():
    successor = None
    name = ''
    def __init__(self, name):
        self.name = name
    def setSuccessor(self, successor):
        self.successor = successor
    def handleRequest(self, request):
        pass
class lineManager(manager):
    def handleRequest(self, request):
        if request.requestType == 'DaysOff' and request.number <= 3:
            print ('%s:%s Num:%d Accepted OVER' % (self.name, request.requestContent,
request.number))
        else:
            print ('%s:%s Num:%d Accepted CONTINUE' % (self.name, request.requestContent,
request.number))
            if self.successor != None:
                self.successor.handleRequest(request)
class departmentManager(manager):
    def handleRequest(self, request):
        if request.requestType == 'DaysOff' and request.number <= 7:
            print ('%s:%s Num:%d Accepted OVER' % (self.name, request.requestContent,
request.number))
        else:
            print ('%s:%s Num:%d Accepted CONTINUE' % (self.name, request.requestContent,
request.number))
            if self.successor != None:
                self.successor.handleRequest(request)
class generalManager(manager):
    def handleRequest(self, request):
        if request.requestType == 'DaysOff':
            print ('%s:%s Num:%d Accepted OVER' % (self.name, request.requestContent,
request.number))
class request():
    requestType = ''
    requestContent = ''
    number = 0


if  __name__=="__main__":
    line_manager = lineManager('LINE MANAGER')
    department_manager = departmentManager('DEPARTMENT MANAGER')
    general_manager = generalManager('GENERAL MANAGER')

    line_manager.setSuccessor(department_manager)
    department_manager.setSuccessor(general_manager)

    req = request()
    req.requestType = 'DaysOff'
    req.requestContent = 'Ask 1 day off'
    req.number = 1
    line_manager.handleRequest(req)
```

```
    req.requestType = 'DaysOff'
    req.requestContent = 'Ask 5 days off'
    req.number = 5
    line_manager.handleRequest(req)

    req.requestType = 'DaysOff'
    req.requestContent = 'Ask 10 days off'
    req.number = 10
    line_manager.handleRequest(req)
```

16.命令模式

```
#后台系统
class backSys():
    def cook(self,dish):
        pass
class mainFoodSys(backSys):
    def cook(self,dish):
        print ("MAINFOOD:Cook %s"%dish)
class coolDishSys(backSys):
    def cook(self,dish):
        print ("COOLDISH:Cook %s"%dish)
class hotDishSys(backSys):
    def cook(self,dish):
        print ("HOTDISH:Cook %s"%dish)

#前台系统
class waiterSys():
    menu_map=dict()
    commandList=[]
    def setOrder(self,command):
        print ("WAITER:Add dish")
        self.commandList.append(command)

    def cancelOrder(self,command):
        print ("WAITER:Cancel order...")
        self.commandList.remove(command)

    def notify(self):
        print ("WAITER:Nofify...")
        for command in self.commandList:
            command.execute()
#前台系统中的notify接口直接调用命令中的execute接口，执行命令
class Command():
    receiver = None
    def __init__(self, receiver):
        self.receiver = receiver
    def execute(self):
        pass
class foodCommand(Command):
```

```
    dish=""
    def __init__(self,receiver,dish):
        self.receiver=receiver
        self.dish=dish
    def execute(self):
        self.receiver.cook(self.dish)

class mainFoodCommand(foodCommand):
    pass
class coolDishCommand(foodCommand):
    pass
class hotDishCommand(foodCommand):
    pass

#菜单类
class menuAll:
    menu_map=dict()
    def loadMenu(self):#加载菜单，这里直接写死
        self.menu_map["hot"] = ["Yu-Shiang Shredded Pork", "Sauteed Tofu, Home Style",
"Sauteed Snow Peas"]
        self.menu_map["cool"] = ["Cucumber", "Preserved egg"]
        self.menu_map["main"] = ["Rice", "Pie"]
    def isHot(self,dish):
        if dish in self.menu_map["hot"]:
            return True
        return False
    def isCool(self,dish):
        if dish in self.menu_map["cool"]:
            return True
        return False
    def isMain(self,dish):
        if dish in self.menu_map["main"]:
            return True
        return False

if  __name__=="__main__":
    dish_list=["Yu-Shiang Shredded Pork","Sauteed Tofu, Home Style","Cucumber","Rice"]#顾客
点的菜
    waiter_sys=waiterSys()
    main_food_sys=mainFoodSys()
    cool_dish_sys=coolDishSys()
    hot_dish_sys=hotDishSys()
    menu=menuAll()
    menu.loadMenu()
    for dish in dish_list:
        if menu.isCool(dish):
            cmd=coolDishCommand(cool_dish_sys,dish)
        elif menu.isHot(dish):
            cmd=hotDishCommand(hot_dish_sys,dish)
        elif menu.isMain(dish):
            cmd=mainFoodCommand(main_food_sys,dish)
        else:
            continue
```

```
        waiter_sys.setOrder(cmd)
    waiter_sys.notify()
```

17.迭代器模式

```
# if __name__=="__main__":
#     lst=["hello Alice","hello Bob","hello Eve"]
#     lst_iter=iter(lst)
#     for i in lst_iter:
#         print (i)
# #   print (lst_iter.next())

#使用迭代器模式构造可迭代对象
class MyIter(object):
    def __init__(self, n):
        self.index = 0
        self.n = n
    def __iter__(self):
        return  MyIter(self.n)
    def __next__(self):
        if self.index < self.n:
            value = self.index**2
            self.index += 1
            return value
        else:
            raise StopIteration()
if __name__=="__main__":
    x_square=MyIter(10)
    for x in x_square:
        print (x)
    for x in x_square: #__iter__方法中的返回值，由于直接返回了self，因而该迭代器是无法重复迭代的
        print (x)
    #解决方法：
    # def __iter__(self):
    #     return MyIter(self.n)

#使用生成器来迭代
def MyGenerater(n):
    index=0
    while index<n:
        yield index**2
        index+=1
if __name__=="__main__":
    x_square=MyGenerater(10)
    for x in x_square:
        print('=======')
        print (x)
```

18.中介者模式

```
#构造三个子系统
class colleague():#中介者模式中称为同事
    mediator = None
    def __init__(self,mediator):
        self.mediator = mediator
class purchaseColleague(colleague):#购置类
    def buyStuff(self,num):
        print ("PURCHASE:Bought %s"%num)
        self.mediator.execute("buy",num)
    def getNotice(self,content):
        print ("PURCHASE:Get Notice--%s"%content)
class warehouseColleague(colleague):#仓库类
    total=0
    threshold=100
    def setThreshold(self,threshold):
        self.threshold=threshold
    def isEnough(self):
        if self.total<self.threshold:
            print ("WAREHOUSE:Warning...Stock is low... ")
            self.mediator.execute("warning",self.total)
            return False
        else:
            return True
    def inc(self,num):
        self.total+=num
        print ("WAREHOUSE:Increase %s"%num)
        self.mediator.execute("increase",num)
        self.isEnough()
    def dec(self,num):
        if num>self.total:
            print ("WAREHOUSE:Error...Stock is not enough")
        else:
            self.total-=num
            print ("WAREHOUSE:Decrease %s"%num)
            self.mediator.execute("decrease",num)
        self.isEnough()
class salesColleague(colleague):#销售类
    def sellStuff(self,num):
        print ("SALES:Sell %s"%num)
        self.mediator.execute("sell",num)
    def getNotice(self, content):
        print ("SALES:Get Notice--%s" % content)
#构造中介者，协调各个类的操作
class abstractMediator():
    purchase=""
    sales=""
    warehouse=""
    def setPurchase(self,purchase):
        self.purchase=purchase
    def setWarehouse(self,warehouse):
        self.warehouse=warehouse
    def setSales(self,sales):
        self.sales=sales
```

```
    def execute(self,content,num):
        pass
class stockMediator(abstractMediator):
    def execute(self,content,num):
        print ("MEDIATOR:Get Info--%s"%content)
        if  content=="buy":
            self.warehouse.inc(num)
            self.sales.getNotice("Bought %s"%num)
        elif content=="increase":
            self.sales.getNotice("Inc %s"%num)
            self.purchase.getNotice("Inc %s"%num)
        elif content=="decrease":
            self.sales.getNotice("Dec %s"%num)
            self.purchase.getNotice("Dec %s"%num)
        elif content=="warning":
            self.sales.getNotice("Stock is low.%s Left."%num)
            self.purchase.getNotice("Stock is low. Please Buy More!!! %s Left"%num)
        elif content=="sell":
            self.warehouse.dec(num)
            self.purchase.getNotice("Sold %s"%num)
        else:
            pass

if  __name__=="__main__":
    mobile_mediator=stockMediator()#先配置
    mobile_purchase=purchaseColleague(mobile_mediator)
    mobile_warehouse=warehouseColleague(mobile_mediator)
    mobile_sales=salesColleague(mobile_mediator)
    mobile_mediator.setPurchase(mobile_purchase)
    mobile_mediator.setWarehouse(mobile_warehouse)
    mobile_mediator.setSales(mobile_sales)

    mobile_warehouse.setThreshold(200)
    mobile_purchase.buyStuff(300)
    mobile_sales.sellStuff(120)
```

19.备忘录模式

```
#游戏进度保存
import random
class GameCharacter():
    vitality = 0#生命值
    attack = 0#攻击力
    defense = 0#防御值
    def displayState(self):
        print ('Current Values:')
        print ('Life:%d' % self.vitality)
        print ('Attack:%d' % self.attack)
        print ('Defence:%d' % self.defense)
    def initState(self,vitality,attack,defense):
        self.vitality = vitality
```

```
        self.attack = attack
        self.defense = defense
    #保存状态
    def saveState(self):
        return Memento(self.vitality, self.attack, self.defense)
    #恢复状态
    def recoverState(self, memento):
        self.vitality = memento.vitality
        self.attack = memento.attack
        self.defense = memento.defense
class FightCharactor(GameCharacter):
    def fight(self):
        self.vitality -= random.randint(1,10)

#备忘录
class Memento:
    vitality = 0
    attack = 0
    defense = 0
    def __init__(self, vitality, attack, defense):
        self.vitality = vitality
        self.attack = attack
        self.defense = defense

if __name__=="__main__":
    game_chrctr = FightCharactor()
    game_chrctr.initState(100,79,60)
    game_chrctr.displayState()
    memento = game_chrctr.saveState()
    game_chrctr.fight()
    game_chrctr.displayState()
    game_chrctr.recoverState(memento)
    game_chrctr.displayState()

#游戏进度保存
import random
class GameCharacter():
    vitality = 0#生命值
    attack = 0#攻击力
    defense = 0#防御值
    def displayState(self):
        print ('Current Values:')
        print ('Life:%d' % self.vitality)
        print ('Attack:%d' % self.attack)
        print ('Defence:%d' % self.defense)
    def initState(self,vitality,attack,defense):
        self.vitality = vitality
        self.attack = attack
        self.defense = defense
    #保存状态
    def saveState(self):
        return Memento(self.vitality, self.attack, self.defense)
    #恢复状态
```

```
    def recoverState(self, memento):
        self.vitality = memento.vitality
        self.attack = memento.attack
        self.defense = memento.defense
class FightCharactor(GameCharacter):
    def fight(self):
        self.vitality -= random.randint(1,10)

#备忘录
class Memento:
    vitality = 0
    attack = 0
    defense = 0
    def __init__(self, vitality, attack, defense):
        self.vitality = vitality
        self.attack = attack
        self.defense = defense

if __name__=="__main__":
    game_chrctr = FightCharactor()
    game_chrctr.initState(100,79,60)
    game_chrctr.displayState()
    memento = game_chrctr.saveState()
    game_chrctr.fight()
    game_chrctr.displayState()
    game_chrctr.recoverState(memento)
    game_chrctr.displayState()
```

20.观察者模式

```
#火警报警器
#观察者
class Observer:
    def update(self):
        pass
class AlarmSensor(Observer):
    def update(self,action):
        print ("Alarm Got: %s" % action)
        self.runAlarm()
    def runAlarm(self):
        print( "Alarm Ring...")
class WaterSprinker(Observer):
    def update(self,action):
        print ("Sprinker Got: %s" % action)
        self.runSprinker()
    def runSprinker(self):
        print ("Spray Water...")
class EmergencyDialer(Observer):
    def update(self,action):
        print ("Dialer Got: %s"%action)
        self.runDialer()
```

```
    def runDialer(self):
        print ("Dial 119...")
#被观察者
class Observed:
    observers=[]
    action=""
    def addObserver(self,observer):
        self.observers.append(observer)
    def notifyAll(self):
        for obs in self.observers:
            obs.update(self.action)
class smokeSensor(Observed):
    def setAction(self,action):
        self.action=action
    def isFire(self):
        return True

if __name__=="__main__":
    alarm=AlarmSensor()
    sprinker=WaterSprinker()
    dialer=EmergencyDialer()

    smoke_sensor=smokeSensor()
    smoke_sensor.addObserver(alarm)
    smoke_sensor.addObserver(sprinker)
    smoke_sensor.addObserver(dialer)


    if smoke_sensor.isFire():
        smoke_sensor.setAction("On Fire!")
        smoke_sensor.notifyAll()
```

21.状态模式

```
#电梯控制器
class LiftState:
    def open(self):
        pass
    def close(self):
        pass
    def run(self):
        pass
    def stop(self):
        pass
#开门状态
class OpenState(LiftState):
    def open(self):
        print( "OPEN:The door is opened...")
        return self
    def close(self):
        print ("OPEN:The door start to close...")
```

```
        print ("OPEN:The door is closed")
        return StopState()
    def run(self):
        print ("OPEN:Run Forbidden.")
        return self
    def stop(self):
        print ("OPEN:Stop Forbidden.")
        return self
class RunState(LiftState):
    def open(self):
        print ("RUN:Open Forbidden.")
        return self
    def close(self):
        print ("RUN:Close Forbidden.")
        return self
    def run(self):
        print ("RUN:The lift is running...")
        return self
    def stop(self):
        print ("RUN:The lift start to stop...")
        print ("RUN:The lift stopped...")
        return StopState()
class StopState(LiftState):
    def open(self):
        print ("STOP:The door is opening...")
        print ("STOP:The door is opened...")
        return OpenState()
    def close(self):
        print ("STOP:Close Forbidden")
        return self
    def run(self):
        print ("STOP:The lift start to run...")
        return RunState()
    def stop(self):
        print ("STOP:The lift is stopped.")
        return self
#上下文类记录在业务中调度状态的转移
class Context:
    lift_state=""
    def getState(self):
        return self.lift_state
    def setState(self,lift_state):
        self.lift_state=lift_state
    def open(self):
        self.setState(self.lift_state.open())
    def close(self):
        self.setState(self.lift_state.close())
    def run(self):
        self.setState(self.lift_state.run())
    def stop(self):
        self.setState(self.lift_state.stop())

if __name__=="__main__":
```

```
    ctx = Context()
    ctx.setState(StopState())
    ctx.open()
    ctx.run()
    ctx.close()
    ctx.run()
    ctx.stop()
```

22.策略模式

```
#构造客户类，包括常用的联系方式和基本信息以及要发送的内容
class customer:
    customer_name=""
    snd_way=""
    info=""
    phone=""
    email=""
    def setPhone(self,phone):
        self.phone=phone
    def setEmail(self,mail):
        self.email=mail
    def getPhone(self):
        return self.phone
    def getEmail(self):
        return self.email
    def setInfo(self,info):
        self.info=info
    def setName(self,name):
        self.customer_name=name
    def setBrdWay(self,snd_way):
        self.snd_way=snd_way
    def sndMsg(self):
        self.snd_way.send(self.info)

#发送方式
class msgSender:
    dst_code=""
    def setCode(self,code):
        self.dst_code=code
    def send(self,info):
        pass
class emailSender(msgSender):
    def send(self,info):
        print ("EMAIL_ADDRESS:%s EMAIL:%s"%(self.dst_code,info))
class textSender(msgSender):
    def send(self,info):
        print ("PHONE_NUMBER:%s TEXT:%s"%(self.dst_code,info))

if  __name__=="__main__":
    customer_x=customer()
    customer_x.setName("CUSTOMER_X")
```

```
    customer_x.setPhone("10023456789")
    customer_x.setEmail("customer_x@xmail.com")
    customer_x.setInfo("Welcome to our new party!")
    text_sender=textSender()
    text_sender.setCode(customer_x.getPhone())
    customer_x.setBrdWay(text_sender)
    customer_x.sndMsg()
    mail_sender=emailSender()
    mail_sender.setCode(customer_x.getEmail())
    customer_x.setBrdWay(mail_sender)
    customer_x.sndMsg()
```

23.访问者模式

```
#药房业务系统
#构造药品类和工作人员类
class Medicine:
    name=""
    price=0.0
    def __init__(self,name,price):
        self.name=name
        self.price=price
    def getName(self):
        return self.name
    def setName(self,name):
        self.name=name
    def getPrice(self):
        return self.price
    def setPrice(self,price):
        self.price=price
    def accept(self,visitor):
        pass
#抗生素
class Antibiotic(Medicine):
    def accept(self,visitor):
        visitor.visit(self)
#感冒药
class Coldrex(Medicine):
    def accept(self,visitor):
        visitor.visit(self)
class Visitor:
    name=""
    def setName(self,name):
        self.name=name
    def visit(self,medicine):
        pass
class Charger(Visitor):#划价员
    def visit(self,medicine):
        print ("CHARGE: %s lists the Medicine %s. Price:%s " %
(self.name,medicine.getName(),medicine.getPrice()))
class Pharmacy(Visitor):#药房管理员
```

```
    def visit(self,medicine):
        print ("PHARMACY:%s offers the Medicine %s. Price:%s" %
(self.name,medicine.getName(),medicine.getPrice()))

class ObjectStructure:
    pass
#构建处方
class Prescription(ObjectStructure):
    medicines=[]
    def addMedicine(self,medicine):
        self.medicines.append(medicine)
    def rmvMedicine(self,medicine):
        self.medicines.append(medicine)
    def visit(self,visitor):
        for medc in self.medicines:
            medc.accept(visitor)

if __name__=="__main__":
    yinqiao_pill=Coldrex("Yinqiao Pill",2.0)
    penicillin=Antibiotic("Penicillin",3.0)
    doctor_prsrp=Prescription()
    doctor_prsrp.addMedicine(yinqiao_pill)
    doctor_prsrp.addMedicine(penicillin)
charger=Charger()
charger.setName("Doctor Strange")
pharmacy=Pharmacy()
pharmacy.setName("Doctor Wei")
doctor_prsrp.visit(charger)
doctor_prsrp.visit(pharmacy)
```