



# Python语言

## 第八章 类和对象

8.1 理解面向对象

8.2 类的定义与使用

8.3 类的特点

8.4 实验

8.5 小结

8.6 习题

70年代，出现信息建模方法；

以信息实体为构造块，以数据结构为中心来开发软件；  
对功能的处理较弱。

80年代，出现面向对象方法（OOM）。

对信息建模方法进行了发展；

软件模型是问题域的完整和直接映射；

分析、设计和编程采用一致的概念和表示。

是一种运用“对象”、“类”、“继承”、“封装”、“聚合”、“关联”、“消息”、“多态性”等概念和原则来构造系统的软件开发方法。

### OOM的基本思想

- ◆ 客观世界中的事物都是对象，对象间存在一定关系，复杂对象由简单对象组成；
- ◆ 具有相同属性和操作的对象属于一个类，对象是类的一个实例；
- ◆ 类之间可以有层次结构，子类继承父类的全部属性和操作，且可有自己的属性和操作；
- ◆ 类有封装性，隐藏或公开自己的属性或操作；对象只能通过消息请求其他对象或自己的操作；
- ◆ 强调运用人类日常思维方法。

OO主要基本概念	OO主要基本原则
对象	抽象
属性	分类
操作	封装
类	消息通信
继承	多态性
聚合	
关联	

## 对象 (Object)

是系统中用来描述客观事物的一个实体，  
它是构成系统的基本单位，  
由一组属性和施加于这组属性的一组操作构成。

## 对象中的属性 (Attribute)

用来描述对象**静态**特征的一个数据项。

## 对象中的操作 (Operation)

用来描述对象**动态**特征（行为）的一个动作序列。

## 类 (class)

相同属性和操作的一组对象属于同一个类，  
它为属于该类的全部对象提供了统一的**抽象描述**，  
由一个类名、一组属性和一组操作构成，  
同一个类的对象之间，属性值可不同，操作完全相同。

## 类的作用

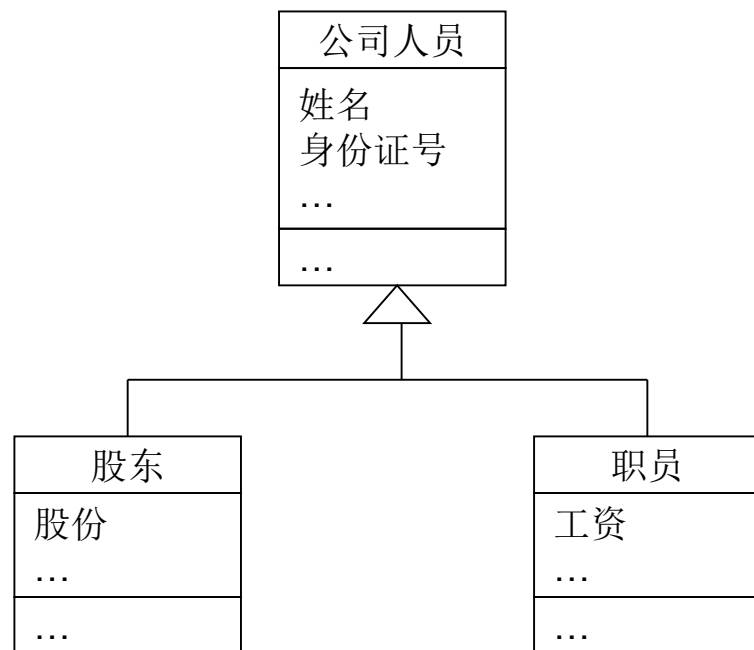
用于创建对象，对象是类的实例。

类名
属性...
方法...

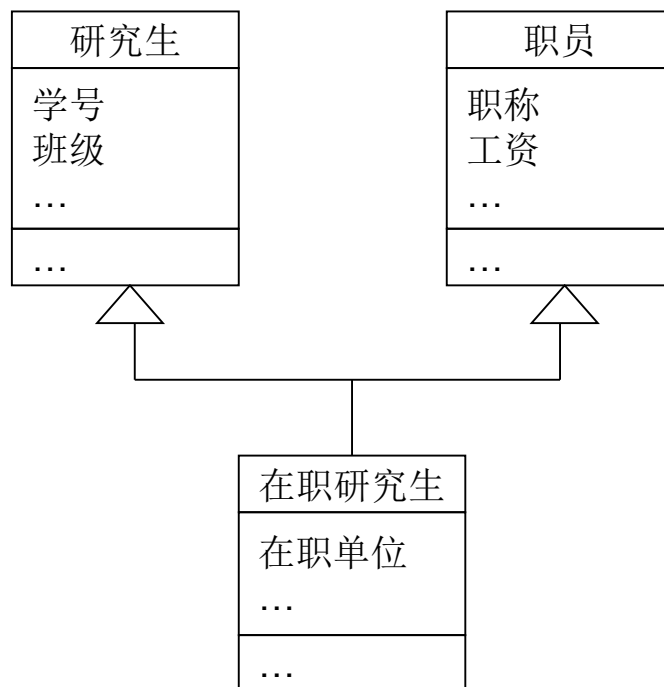
对象名: 类名
...

- 继承 (inheritance)

- 特殊类自动拥有其一般类的全部属性和操作，称为特殊类对一般类的继承，也称一般类对特殊类的泛化。



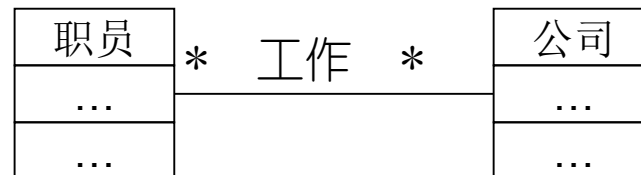
单继承



多继承

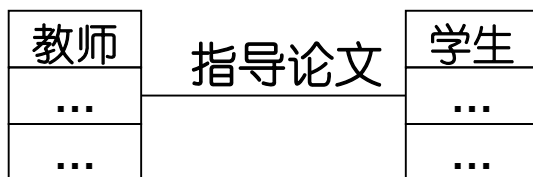
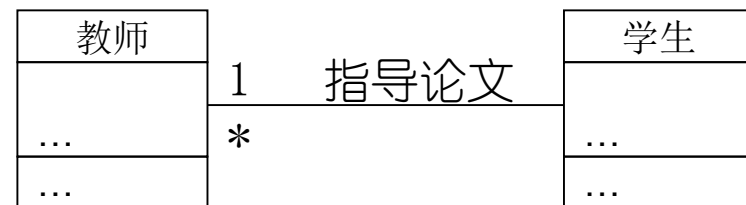
- 关联 (association) (类之间的连接)

- 类之间的静态关系;
- 类实例为对象, 对象之间用链连接。
  - 链是关联的实例。



- 链 (Link 对象之间的连接)

- 是关联的实例, 是对象之间的语义连接。
- 两个对象间有链, 则一个对象可访问另一个对象。



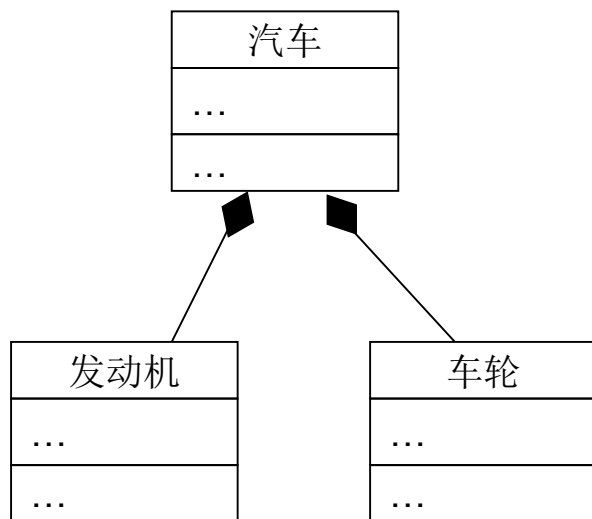
类之间的关联



对象之间的链



- 聚合（aggregation）
  - 一个（较复杂）对象由其他若干（较简单）对象作为其构成成分，这种对象间关系称为聚合；
  - 是关联的一种。



- 面向对象的基本原则

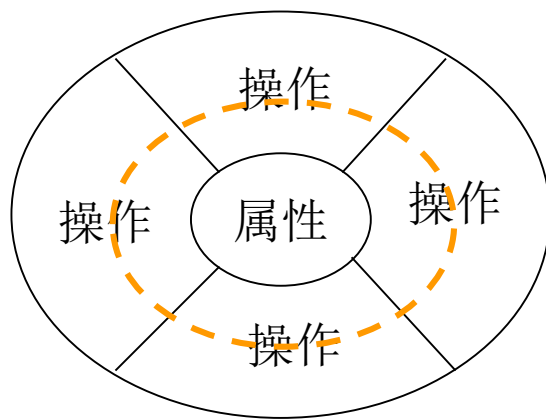
- 抽象

- 从事物中舍弃个别的、非本质的特征，保留共同的、本质特征的做法。

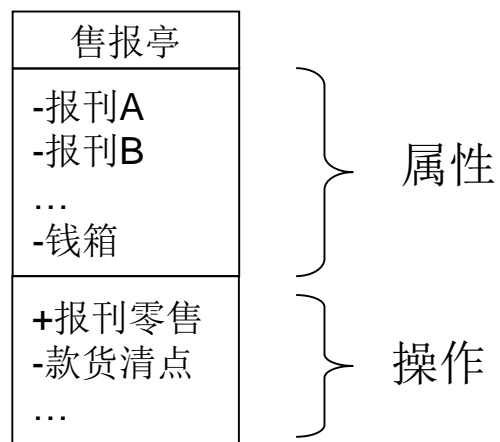
- 分类

- 较多地注意事物之间的差别，把具有相同属性和操作的对象划分一类。

- 封装 (encapsulation)
  - 用对象把属性和操作包装起来，形成一个独立的实体单位，并尽量对外隐蔽内部细节。

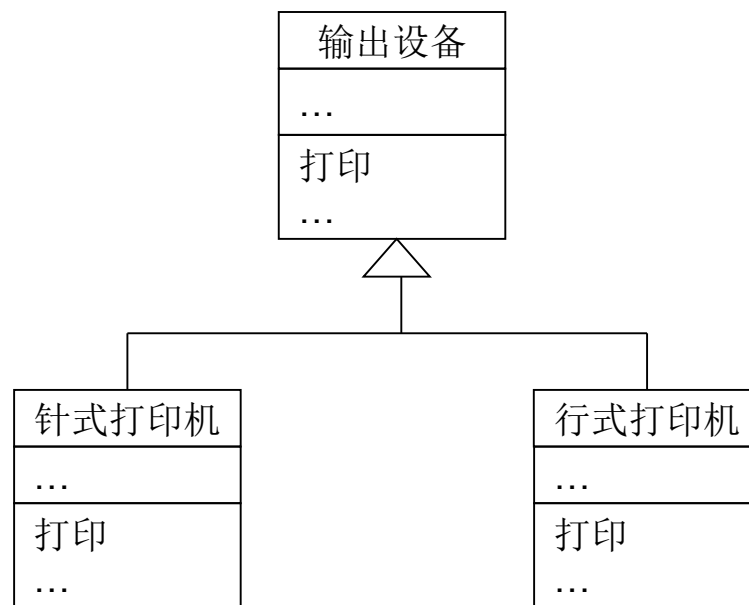


封装与信息隐蔽



- 消息 (message)
  - 对象通过它提供的操作在系统中发挥作用;
  - 其他对象向这个对象发请求, 该对象响应并执行指定操作;
  - 向一个对象发送的操作请求, 称为消息;
  - 对象之间通过消息进行通信, 实现对象之间的动态联系。
  - 在c++中是函数调用, 在Java中是方法激活。

- 多态性 (polymorphism)
  - 特殊类可定义同名的属性或操作，来代替继承来的属性或操作；
  - 一般类中的属性或操作，在不同特殊类中可有不同的定义。



### 8.1.1 什么是面向对象编程

面向对象编程(Object Oriented Programming), 简称OOP, 是一种程序设计思想, 是以建立模型体现出来的抽象思维过程和面向对象的方法。模型是用来反映现实世界中事物特征的, 是对事物特征和变化规律的抽象, 是更普遍、更集中、更深刻地描述客体的特征。OOP把对象作为程序的基本单元, 一个对象包含了数据和操作数据的函数。

类名
属性...
方法...

对象名:类名
...

### 8.1.2 面向对象术语简介

面向对象常用的术语如下：

类：是创建对象的代码段，描述了对对象的特征、属性、要实现的功能以及采用的方法等。

类名
属性...
方法...

属性：描述了对对象的静态特征。

方法：描述了对对象的动态动作。

对象：对象是类的一个实例，就是模拟真实事件，把数据和代码都集合到一起，即属性、方法的集合。

实例：就是类的实体。

### 8.1.2 面向对象术语简介

实例化：创建类的一个实例的过程。

封装：把对象的属性、方法、事件集中到一个统一的类中，并对调用者屏蔽其中的细节。

继承：一个类共享另一个类的数据结构和方法的机制称为继承。起始类称为基类、超类、父类，而继承类称为派生类、子类。继承类是对被继承类的扩展。

多态：一个同样的函数对于不同的对象可以具有不同的实现，就称为多态。

接口：定义了方法、属性的结构，为其成员提供规约，不提供实现。不能直接从接口创建对象，必须首先创建一个类来实现接口所定义的内容。



### 8.1.2 面向对象术语简介

重载：一个方法可以具有许多不同的接口，但方法的名称是相同的。

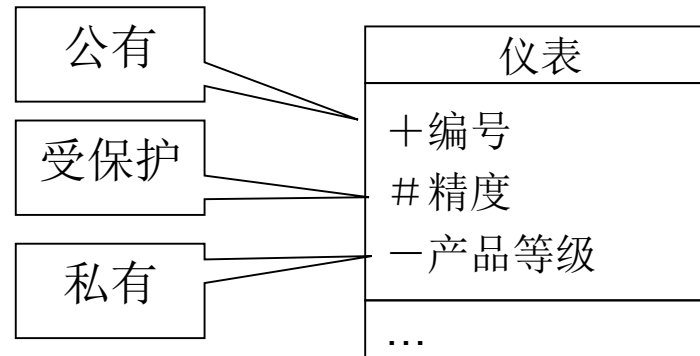
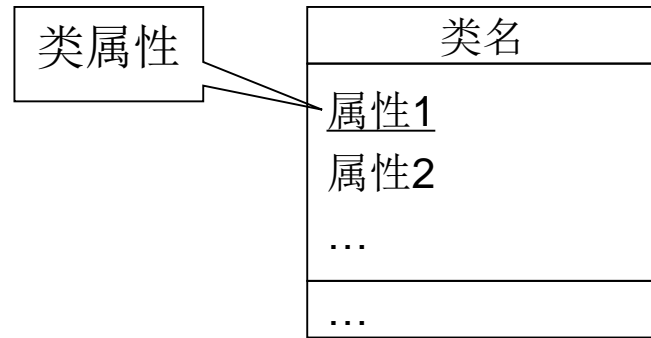
事件：事件是由某个外部行为所引发的对象方法。

重写：在派生类中，对基类某个方法的程序代码进行重新编写，使其实现不同的功能，我们把这个过程称为重写。

构造函数：是创建对象所调用的特殊方法。

析构函数：是释放对象时所调用的特殊方法。

### 8.1.2 面向对象术语简介



## 第八章 类和对象

8.1 理解面向对象

8.2 类的定义与使用

8.3 类的特点

8.4 实验

8.5 小结

8.6 习题

### 8.2.1 类的定义

- 概念
- 类（class）
  - 相同属性和操作的一组对象属于同一个类，
  - 它为属于该类的全部对象提供了统一的抽象描述，
  - 由一个类名、一组属性和一组操作构成，
  - 同一个类的对象之间，属性值可不同，操作完全相同。
- 类的作用
  - 用于创建对象，对象是类的实例。

### 8.2.1 类的定义

类就是对象的属性和方法的封装，静态的特征称为属性，动态的动作称为方法。

类通常的语法格式如下：

**class ClassName:**

**#属性**

**[属性定义体]**

**#方法**

**[方法定义体]**

### 8.2.1 类的定义

类的定义以关键字class开始，类名必须以大写字母开头，类名后面紧跟冒号“:”。

类的定义示例如下：

# Python的类名约定以大写字母开头

从上例可以看出，属性就是变量，静态的特征，方法就是一个一个的函数，通过这些函数来描述动作行为。

```
class Person:
    # 属性
    skincolor = "yellow"
    high = 168
    weight = 65

    # 方法
    def goroad(self):
        print("人走路动作的测试.....")

    def sleep(self):
        print("睡觉，晚安！")
```

### 8.2.1 类的定义

在定义类属性的时候一般用**名词**,

定义类方法的时候一般用**动词**,

类名约定以**大写字母**开头,

函数约定以**小写字母**开头。

```
class Point(): #object):
    _x:int
    _y:int

    def __init__(self, x=0, y=0):
        self._x = x
        self._y = y

    def __str__(self):
        return ' [%d,%d]' % (self._x, self._y)

p = Point(20, 20)
print('Point((20,20)) => ', p)
```

### 8.2.2 类的使用

类定义好之后，就可将类实例化为对象。类实例化对象的语法格式如下：

对象名 = 类名 ()

实例化对象的操作符是：等号“=”，在类实例化对象的时候，类名后面要添加一个括号“()”。

类实例化示例：

```
>>> p = Person()
```

上例将类Person实例化为对象p。



### 8.2.3 类的构造方法及专有方法

类的构造方法是：\_\_init\_\_(self)。

只要实例化一个对象的时候，这个方法就会在对象被创建的时候自动调用。

实例化对象的时候是可以传入参数的，这些参数会自动传入\_\_init\_\_(self,param1,param2,...)方法中，我们可以通过重写这个方法来自定义对象的初始化操作。如下例所示代码：

```
class Bear:
    def __init__(self, name):
        self.name = name

    def kill(self):
        print("%s,是保护动物, 不能杀..." % self.name)

a = Bear('狗熊')
a.kill()
狗熊,是保护动物, 不能杀...
```

### 8.2.3 类的构造方法及专有方法

在例中，我们重写了\_\_init\_\_(self)方法，如果没有重写，它的默认调用就是\_\_init\_\_(self),没有任何参数或只有一个self参数，所以在实例化的时候，参数是空的。在例子中，我们给了它一个参数name，就成了\_\_init\_\_(self,name),在实例化的时候就可以传参数了，因为第一个参数self是默认的，就把“狗熊”传给“name”，运行程序后得到了我们期望的结果，这样使用起来就非常方便。

另外，我们还可以把传入的参数设置为默认参数，我们在实例化的时候不传入参数系统也不会报错，如下例所示代码

```
class Bear:
    def __init__(self, name):
        self.name = name

    def kill(self):
        print("%s,是保护动物，不能杀..." % self.name)

a = Bear('狗熊')
a.kill()
狗熊,是保护动物，不能杀...
```

### 8.2.3 类的构造方法及专有方法

是默认的，就把“狗熊”传给“name”，运行程序后得到了我们期望的结果，这样使用起来就非常方便。

另外，我们还可以把传入的参数设置为默认参数，我们在实例化的时候不传入参数系统也不会报错，如下例所示代码：

在例中，我们把构造函数的参数name设置为默认值：“默认的熊”

```
class Bear:
    def __init__(self, name='默认的熊'):
        self.name = name

    def kill(self):
        print("%s,是保护动物, 不能杀..." % self.name)

a = Bear()
a.kill()
默认的熊,是保护动物, 不能杀...
c = Bear('替代熊')
a.kill()
替代熊,是保护动物, 不能杀...
```

### 8.2.3 类的构造方法及专有方法

对象实例化的时候没有传值给参数name，程序运行正确，输出了期望的结果：“默认的熊,是保护动物，不能杀…”，当在对象实例化的时候给参数name传值为：“替代熊”，当对象c调用方法kill()时，输出了正确的结果：“替代熊,是保护动物，不能杀…”。

### 8.2.4 类的访问权限

大家都知道，在C++/java中，是通过关键字public、private来表明访问的权限是公有、私有的。然而在Python中，默认情况下对象的属性和方法都是公开的，公有的，通过点(.)操作符来访问，如下例所示代码：

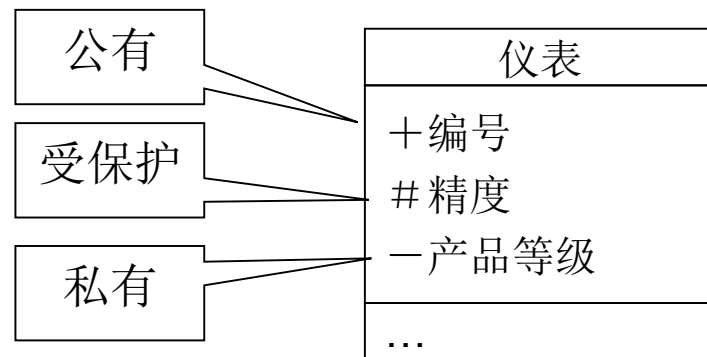
```
class Company:  
    name = "云创科技"
```

运行结果如下：

```
>>> c = Company()
```

```
>>> c.name
```

```
'云创科技'
```



在8.2.4类的访问权限中，我们直接通过点(.)来访问了类Company的变量name，运行得到了结果：云创科技。

为了实现类似于私有变量的特征，Python内部采用了**name mangling**的技术（名字重整或名字改变），在变量名或函数名前加上两个下划线（“\_\_”），这个函数或变量就变为私有了。

为了访问类中的私有变量，有一个折衷的处理办法，如下例所示代码：

```
class Company:
    __name = "云创科技"

    def getname(self):
        return self.__name
```

## 8.2.4 类的访问权限

运行结果如下：

```
>>> c = Company()
```

```
>>> c.getname()
```

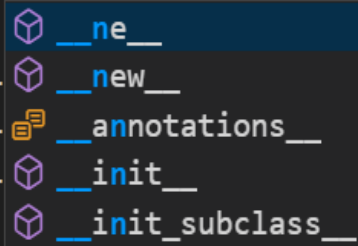
'云创科技'

```
class Company:
    __name = "云创科技"

    def getname(self):
        return self.__name
```

```
if __name__ == '__main__':
    comp = Company()
    comp.__n

p1 = Poi
p2 = Poi
p3 = Poi
p = Point(20, 20)
```



在上例中，我们在类Company内部重新定义一个方法getname(self)，在程序外部通过访问对象的getname()方法来访问了类Company中的私有变量：\_\_name。

### 8.2.4 类的访问权限

实际上，Python把双下划线“\_\_”开头的变量名，改为了：单下划线“\_”类名+双下划线“\_\_”变量名，即【\_类名\_\_变量名】，因此我们可以通过以下的访问方式来访问类的私有变量，如下例所示代码：

运行结果如下：

```
>>> c = Company()
```

```
>>> c.getname()
```

```
'云创科技'
```

```
>>> c._Company__name
```

```
'云创科技'
```

```
class Company:
    __name = "云创科技"
    def getname(self):
        return self.__name
```



### 8.2.4 类的访问权限

在上例中，我们访问了对象c的`_Company_name`，运行得到了期望的结果。

从上述可见，就目前而言，**Python的私有机制是伪私有**，Python的类是没有权限控制的，变量是可以被外部调用的。

### 8.2.5 获取对象信息

类实例化对象之后，对象就可以调用类的属性和方法，语法格式如下：

**对象名.属性名**

**对象名.方法名**

对象调用类的属性或方法的操作符是：点“.”。

对象调用属性或方法的示例：

```
>>> p.goroad()
```

人走路动作的测试……

```
>>> p.skincolor
```

```
class Person:
    # 属性
    skincolor = "yellow"
    high = 168
    weight = 65

    # 方法
    def goroad(self):
        print("人走路动作的测试.....")

    def sleep(self):
        print("睡觉, 晚安! ")
```

### 8.2.5 获取对象信息

'yellow'

>>> p.sleep()

睡觉，晚安！

>>> p.high

168

>>> p.weight

65

在上例中，对象p分别调用了类Person的方法goroad、属性skincolor、方法sleep、属性high、属性weight。

```
class Person:
    # 属性
    skincolor = "yellow"
    high = 168
    weight = 65

    # 方法
    def goroad(self):
        print("人走路动作的测试.....")

    def sleep(self):
        print("睡觉，晚安！ ")
```

## 第八章 类和对象

8.1 理解面向对象

8.2 类的定义与使用

8.3 类的特点

8.4 实验

8.5 小结

8.6 习题

### 8.3.1 封装

从形式上看，对象封装了属性就是变量，而方法和函数是独立性很强的模块，封装就是一种信息掩蔽技术，使数据更加安全。

例如，列表(list)是Python的一个序列对象，我们要对列表进行调整，如下所示代码：

```
>>> list1 = ['K','J','L','Q','M']
```

```
>>> list1.sort()
```

```
>>> list1
```

```
['J', 'K', 'L', 'M', 'Q']
```

在上例中，我们调用了排序函数sort()对无序的列表进行了正序排序。

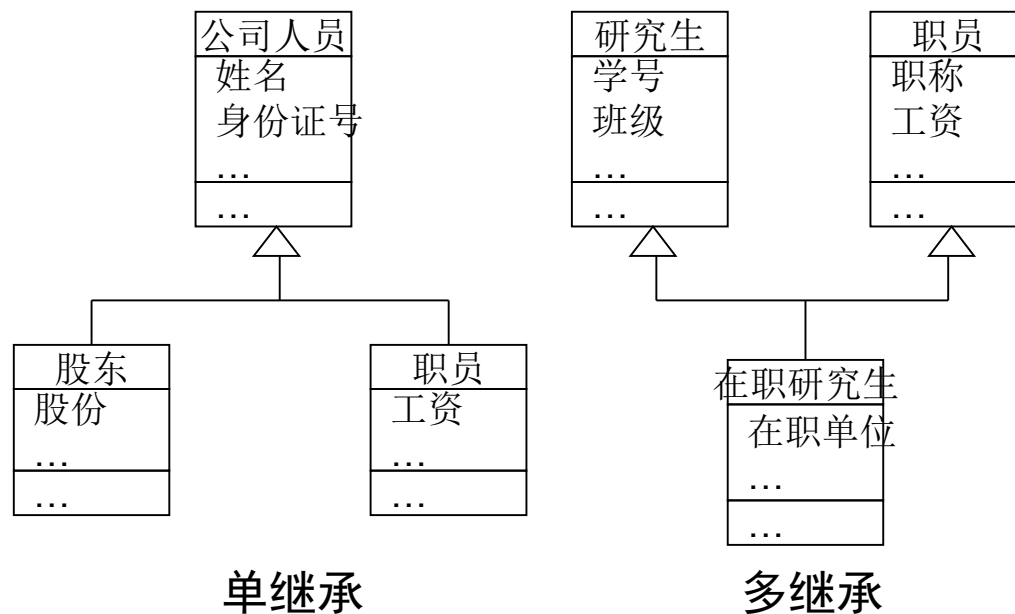
### 8.3.1 封装

由此可见，Python的列表就是对象，它提供了若干种方法来供我们根据需求调整整个列表，**但是我们不知道列表对象里面的方法是如何实现的**，也不知道列表对象里面有哪些变量，这就是**封装**。它封装起来，只给我们需要的方法的名字，然后我们调用这个名字，知道它可以实现就可以了，然而它没有具体告诉我们是怎样实现的。

## 8.3.3 继承

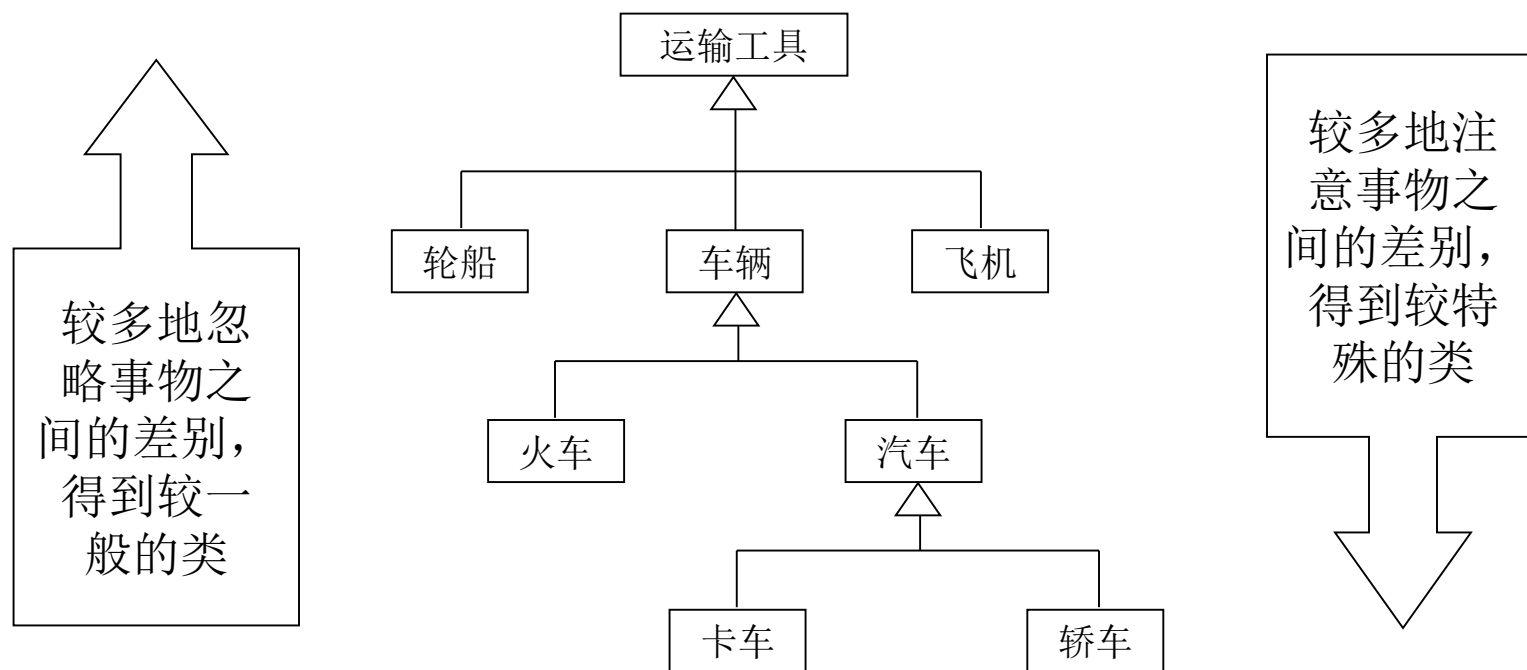
- 继承 (inheritance)
  - 特殊类自动拥有其一般类的全部属性和操作，称为特殊类对一般类的继承，也称一般类对特殊类的泛化。

在OOP程序设计中，当我们定义一个class的时候，可以从某个现有的class继承，新的class称为子类 (Subclass)，而被继承的class称为基类、父类或超类 (Base class、Super class)。



## 8.3.3 继承

- 继承的层次与抽象、分类原则的运用



继承有传递性, 简化了人们对事物的认识和描述, 有益于软件复用。



### 8.3.3 继承

在使用类继承机制的时候，有几点需要注意：

- (1)如果子类中定义与父类同名的方法或属性，则会自动覆盖父类对应的方法或属性。
- (2)子类重写了父类的方法就会把父类的同名方法覆盖，如果被重写的子类同名的方法里面没有引入父类同名的方法，实例化对象要调用父类的同名方法的时候，程序就会报错。

要解决上述问题，就要在子类里面重写父类同名方法的时候，先引入父类的同名方法。

要实现这个继承的目的，有2种技术可采用：

一是调用父类方法，

二是使用super函数。

### 8.3.3 继承

“调用父类方法”的技术，它的语法格式如下：

**paraname.func(self)**

语法各项解释如下：

Paraname——父类的名称；

.——点操作符；

func——子类要重写的父类的同名方法名称；

self——子类的实例对象，注意这里不是父类的实例对象。

## 8.3.3 继承

```
import random as r

class parents:
    def __init__(self):
        self.x = r.randint(0, 10)
        self.y = r.randint(0, 10)

    def move(self):
        self.x -= 1
        self.y -= 1
        print("我现在的位置是：", self.x, self.y)

class child(parents):
    def __init__(self):
        self.test = True

    def test(self):
        if self.test:
            print("调用子类，if")
            self.test = False
        else:
            print("调用父类，else")
```

```
class Child(Parents):
    def __init__(self):
        Parents.__init__(self)
        self.test = True
```

```
>>> a = parents()
>>> a.move
<bound method parents.move of <__main__.parents object at 0x03336050>>
>>> a = parents()
>>> a.move()
我现在的位置是： 7 8
>>> b = child()
>>> b.move()
Traceback (most recent call last):
  File "<pyshell#5>", line 1, in <module>
    b.move()
  File "C:\Users\Administrator\Desktop\1.py", line 9, in move
    self.x -= 1
AttributeError: 'child' object has no attribute 'x'
```

```
>>> a = Child()
>>> a.move()
我现在的位置是： 2 1
>>>
```

### 8.3.3 继承

“使用super函数”的技术，super函数可以自动找到基类的方法和传入self参数，它的语法格式如下：

```
super().func([parameter])
```

语法各项解释如下：

super()——super函数；

.——点操作符

func——子类要重写的父类的同名方法名称；

parameter——可选参数，如果参数是self可以省略。

使用super函数的方便之处在于不用写任何基类的名称，直接写重写的方法就可以了，这样Python会自动到基类去寻找，尤其是在多重继承中，或者子类有多个祖先类的时候，super函数会自动到多种层级关系里面去寻找同名的方法。

## 8.3.3 继承

```
import random as r

class parents:
    def __init__(self):
        self.x = r.randint(0,10)
        self.y = r.randint(0,10)

    def move(self):
        self.x -= 1
        self.y -= 1
        print("我现在的位置是：", self.x, self.y)

class child(parents):
    def __init__(self):
        self.test = True

    def test(self):
        if self.test:
            print("调用子类，if")
            self.test = False
        else:
            print("调用父类，else")
```

```
>>> a = parents()
>>> a.move
<bound method parents.move of <__main__.parents object at 0x03336050>>
>>> a = parents()
>>> a.move()
我现在的位置是： 7 8
>>> b = child()
>>> b.move()
Traceback (most recent call last):
  File "<pyshell#5>", line 1, in <module>
    b.move()
  File "C:\Users\Administrator\Desktop\1.py", line 9, in move
    self.x -= 1
AttributeError: 'child' object has no attribute 'x'
```

使用super函数（超赞，有点很多，推荐使用）

```
class Child(Parents):
    def __init__(self):
        super().__init__()
        self.test = True
```

```
===
>>> a = Child()
>>> a.move()
我现在的位置是： -1 8
>>>
```

### 8.3.4 多重继承

可以同时继承多个父类的属性和方法，称为多重继承。语法格式如下：

```
Class ClassName(Base1, Base2, Base3):
```

```
.....
```

ClassName ——子类的名字；

Base1, Base2, Base3——基类1的名字，基类2的名字，基类3的名字；有多少个基类，名字依次写入即可。

虽然多重继承的机制可以让子类继承多个基类的属性和方法使用起来很方便，但很容易导致代码混乱，有时候会引起不可预见的Bug，对程序而言不可预见的Bug几乎就是致命的。因此，当我们不确定必须要使用“多重继承”语法的时候，尽量避免使用它。

### 8.5 多态

多态性 (polymorphism) 一般性描述

特殊类 (子类) 可定义同名的属性或操作, 来代替继承来的属性或操作;

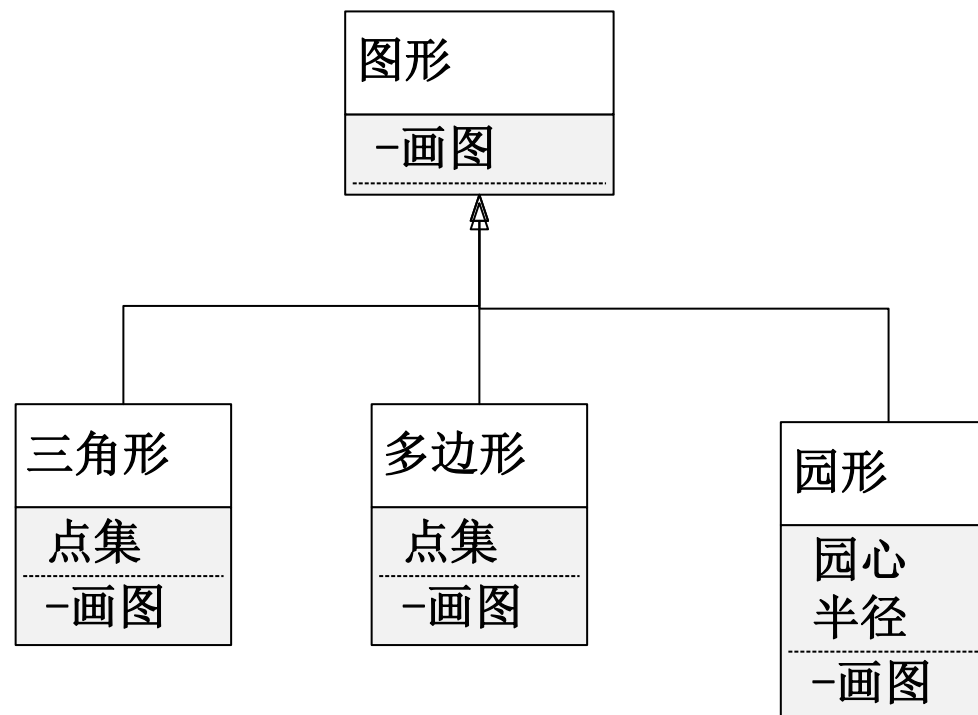
一般类 (父类) 中的属性或操作, 在不同特殊类中可有不同的定义

Python中多态的作用让具有不同功能的函数可以使用相同的函数名, 这样就可以用一个函数名调用不同内容(功能)的函数。

Python中多态的特点

- 1、只关心对象的实例**方法是否同名**, 不关心对象所属的类型;
- 2、**对象所属的类之间, 继承关系可有可无**;
- 3、多态的好处可以增加代码的外部调用灵活度, 让代码更加通用, 兼容性比较强;
- 4、多态是调用方法的技巧, 不会影响到类的内部设计。

### 8.5 多态





## 8.3 类的特点

### 8.5 多态

#### 多态的应用场景

#### 1. 对象所属的类之间没有继承关系

调用同一个函数`fly()`，  
传入不同的参数（对象），  
可以达成不同的功能

```
duck = Duck()
fly(duck)

swan = Swan()
fly(swan)

plane = Plane()
fly(plane)
```

```
class Duck(object):    # 鸭子类
    def fly(self):
        print("鸭子沿着地面飞起来了")
```

```
class Swan(object):    # 天鹅类
    def fly(self):
        print("天鹅在空中翱翔")
```

```
class Plane(object):   # 飞机类
    def fly(self):
        print("飞机隆隆地起飞了")
```

```
def fly(obj):          # 实现飞的功能函数
    obj.fly()
```

运行结果：  
鸭子沿着地面飞起来了  
天鹅在空中翱翔  
飞机隆隆地起飞了

## 8.3 类的特点

### 8.5 多态

2. 对象所属的类之间有继承关系（应用更广）

这里的多态性体现是向同一个函数，

传递不同参数后，可以实现不同功能。

#定义一个函数，函数调用类中的p()方法

```
def fc(obj):
```

```
    obj.p()
```

```
gradapa1 = gradapa(3000)
```

```
father1 = father(2000,"工人")
```

```
mother1 = mother(1000,"老师")
```

```
fc(gradapa1)
```

```
fc(father1)
```

```
print(fc(mother1))
```

```
===运行结果: =====
```

```
this is gradapa
```

```
this is father,我重写了父类的方法
```

```
this is mother,我重写了父类的方法
```

```
100
```

```
class gradapa(object):
```

```
    def __init__(self,money):
```

```
        self.money = money
```

```
    def p(self):
```

```
        print("this is gradapa")
```

```
class father(gradapa):
```

```
    def __init__(self,money,job):
```

```
        super().__init__(money)
```

```
        self.job = job
```

```
    def p(self):
```

```
        print("this is father,我重写了父类的方法")
```

```
class mother(gradapa):
```

```
    def __init__(self, money, job):
```

```
        super().__init__(money)
```

```
        self.job = job
```

```
    def p(self):
```

```
        print("this is mother,我重写了父类的方法")
```

```
        return 100
```

## 第八章 类和对象

8.1 理解面向对象

8.2 类的定义与使用

8.3 类的特点

8.4 实验

8.5 小结

8.6 习题

8.4.1 声明类

8.4.2 类的继承和多态

8.4.3 复制对象

## 8.4 所有类的基类 Object

```
class Ground():
    """
    场地 高 宽
    """
    #私有变量
    __width = 800
    __height = 600

    #用函数方式访问取值
    def getWidth(self):
        return self.__width
    def setWidth(self,value):
        self.__width = value

    def getHeight(self):
        return self.__height
    def setHeight(self,value):
        self.__height = value
```

```
if __name__ == '__main__':
    ground = Ground()

    #用函数方式取值
    print (ground.getWidth(),ground.getHeight())
    #修改
    ground.setWidth(600)
    ground.setHeight(400)
    print (ground.getWidth(),ground.getHeight())
```

```
800 600
600 400
```

## 8.4

```
class Ground():
    """
    场地 高 宽
    """
    #私有变量
    __width = 800
    __height = 600

    #用装饰器的方式访问
    #装饰器@property、@setter
    # @property定义只读属性, @setter定义可读可写属性
    @property
    def Width(self):
        return self.__width
    @Width.setter
    def Width(self,value):
        self.__width = value

    @property
    def Height(self):
        return self.__height
    @Height.setter
    def Height(self,value):
        self.__height = value
```

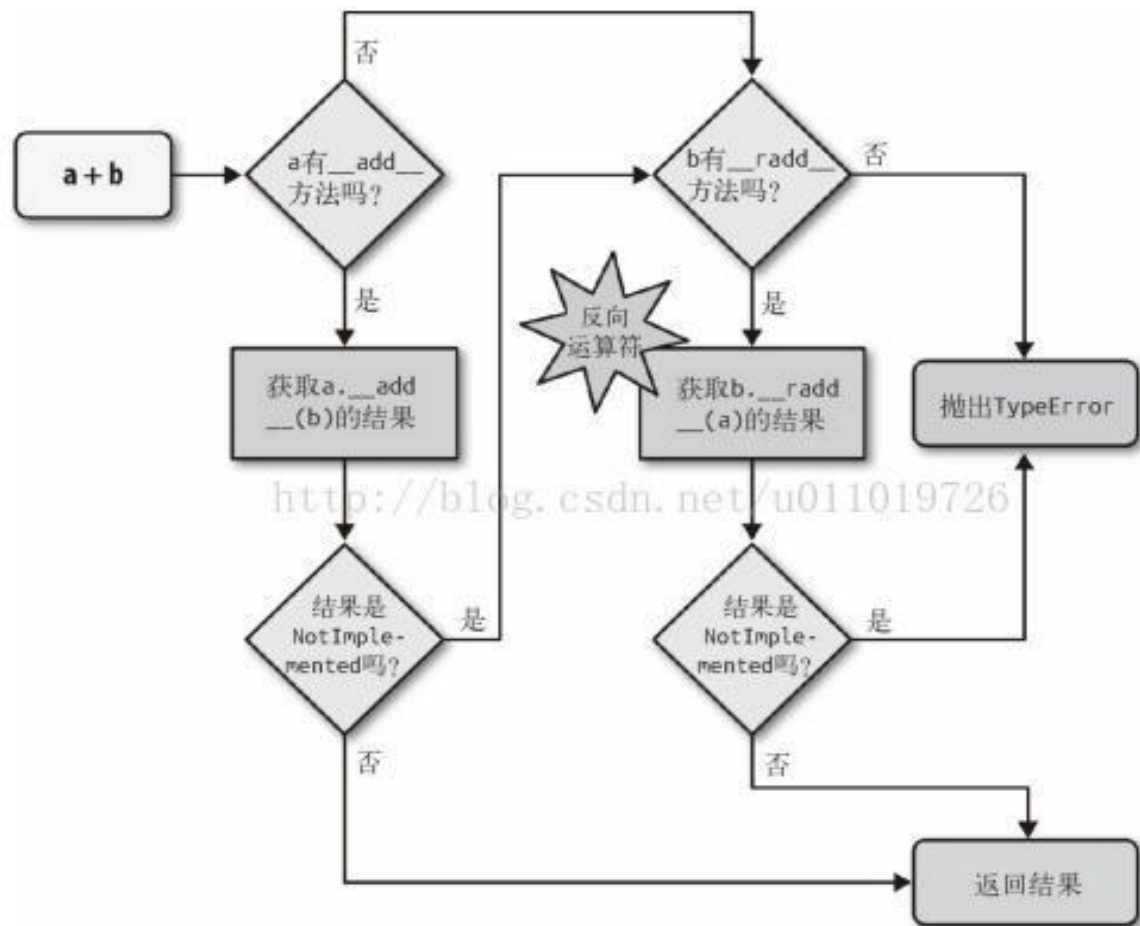
```
if __name__ == '__main__':
    ground = Ground()

    #用函数方式取值
    print (ground.getWidth(),ground.getHeight())
    #修改
    ground.Width = 500
    ground.Height= 300
    print (ground.Width,ground.Height)
```

```
800 600
500 300
```

## 所有类的基类 Object + - \* / 重载

定义class的时候，默认继承就是object了



```
class A: #== class A(object)
    def __add__(self, other):
        print("A __add__")
```

```
    def __radd__(self, other):
        print("A __radd__")
```

```
class B:
    pass
```

```
>>> a = A()
```

```
>>> b = B()
```

```
>>> a+b
```

```
A __add__
```

```
>>> b+a
```

```
A __radd__
```

```
>>> c = B()
```

```
>>> b + c
```

```
Traceback (most recent call last):
```

```
File "<stdin>", line 1, in <module>
```

```
TypeError: unsupported operand type(s) for +: 'instance'
and 'instance'
```

## 所有类的基类 Object + - \*/ 重载

定义class的时候，默认继承就是object了

### 运算符重载

通过运算符重载实现对象之间的运算。

常用有的运算符与函数方法的对应关系：

<code>__add__():+</code>	<code>__lt__():&lt;</code>	<code>__or__():或</code>
<code>__sub__():-</code>	<code>__eq__():=</code>	
<code>__mul__():*</code>	<code>__len__():长度</code>	
<code>__div__():/</code>	<code><b>__str__():</b>print</code>	

```
class A: #== class A(object)
    def __add__(self, other):
        print("A __add__")

    def __radd__(self, other):
        print("A __radd__")

class B:
    pass
```

```
>>> a = A()
>>> b = B()
>>> a+b
A __add__
>>> b+a
A __radd__
>>> c = B()
>>> b + c
Traceback (most recent call last):
  File "<stdin>", line 1, in <module>
TypeError: unsupported operand type(s) for +: 'instance'
and 'instance'
```



```
class MyClass:

    def __init__(self, height, weight):
        self.height = height
        self.weight = weight

    # 两个对象相加, 返回一个新的类
    def __add__(self, others):
        return MyClass(self.height +
others.height, self.weight + others.weight)

    # 两个对象相减, 返回一个新的类
    def __sub__(self, others):
        return MyClass(self.height -
others.height, self.weight - others.weight)

    # 说一下自己的参数
    def intro(self):
        print("高为", self.height, " 重为",
self.weight)
```

```
def main():
    a = MyClass(height=10, weight=5)
    a.intro()

    b = MyClass(height=20, weight=10)
    b.intro()

    c = b - a
    c.intro()

    d = a + b
    d.intro()

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

高为 10 重为 5  
高为 20 重为 10  
高为 10 重为 5  
高为 30 重为 15

Process finished with exit code 0

```

class Point(object):
    _x: int
    _y: int

    def __init__(self, x=0, y=0):
        self._x = x
        self._y = y

    def __str__(self):
        return '%d,%d' % (self._x, self._y)

    # 重载加法 + 加 Point、list 或 tuple 三个类型
    def __add__(self, other):
        '''
        重载加法 + 加 Point、list 或 tuple 三个类型
        '''
        if isinstance(other, tuple) | isinstance(other, list):
            print(type(other))
            p = Point()
            p._x = self._x + other[0]
            p._y = self._y + other[1]
            return p
        elif isinstance(other, Point):
            print('重载加法 + ' + str(type(other)))
            print(type(other))
            p = Point()
            p._x = self._x + other._x
            p._y = self._y + other._y
            return p

```

```

    def __radd__(self, other):
        return self.__add__(other)

    # 重载乘法 乘数值
    def __mul__(self, other):
        '''
        重载乘法 乘数值
        '''
        if isinstance(other, int) | isinstance(other, float):
            print(type(other))
            p = Point()
            p._x = other * self._x
            p._y = other * self._y
            return p

    # 重载减法 -
    # def __sub__(self, other):
    # 重载除法 /
    # def __truediv__(self, other):

    # def __radd__(self, other):
    #     return self.__add__(other)

```

## 第八章 类和对象

8.1 理解面向对象

8.2 类的定义与使用

8.3 类的特点

8.4 实验

8.5 小结

8.6 习题

我们发现想要设计一门出色的语言，就要从现实世界里面去寻找，学习，并归纳抽象出真理包含到其中。

继承可以把父类的所有功能都直接拿过来，这样就不必重零做起，子类只需要新增自己特有的方法，也可以把父类不适合的方法覆盖重写。

继承可以一级一级地继承下来。而任何类，最终都可以追溯到根类。

有了继承，才能有多态。

## 第八章 类和对象

8.1 理解面向对象

8.2 类的定义与使用

8.3 类的特点

8.4 实验

8.5 小结

8.6 习题

## 习题：

1. 面向对象程序设计的特点分别为？
2. 假设c为类C的对象且包含一个私有数据成员“\_\_name”，那么在类的外部通过对象c直接将其私有数据成员“\_\_name”的值设置为kate的语句可以写作什么？

感谢聆听

