第 3 章

类 与 模 块

3.1 类 和 对 象


视频讲解

Python 采用了面向对象程序设计的思想，以类和对象为基础，将数据和操作封装成一个类，通过类的对象进行数据操作。

3.1.1 类的格式与创建对象

1. 类的一般形式

类由类声明和类体组成，而类体又由成员变量和成员方法组成，其一般形式如下：

```
class  类名:                          ← 类声明
    成员变量                          } 类体
    def  成员方法名(self)             }
```

在类声明中，class 是声明类的关键字，表示类声明的开始，类声明后面跟着类名，按习惯类名要用大写字母开头，并且类名不能用阿拉伯数字开头。

在类体中定义的成员方法与在类外定义的函数一般形式是相同的。也就是说，通常把定义在类体中的函数称为方法。

类中的 self 在调用时代表类的实例，与 C++或 Java 中的 this 作用类似。

2. 创建类的对象

类在使用时，必须创建类的对象，再通过类的对象来操作类中的成员变量和成员方法。

创建类对象的格式为：

```
对象名 = 类名()
```

3. 调用类的成员方法

调用类的成员方法时，需要通过类对象调用，其调用格式如下：

```
对象名.方法名(self)
```

【例 3.1】 编写一个计算两数之和的类。

```
class Myclass:
    def sum(self, x, y):
        self.x = x
        self.y = y
        return self.x + self.y

obj = Myclass()
s = obj.sum(3, 5)
print('s =',s)
```

定义类 Myclass

创建类对象，并通过对象调用类的成员方法

在程序的类定义中，方法 sum(self,x,y)的参数 self 代表类对象自身，self.x = x 即把赋值语句右边的参数 x 值赋值给左边类成员变量 x。为了区分参数及成员变量，在成员变量 x 前面添加 self。

程序运行结果如下：

```
s = 8
```

4. 类的公有成员和私有成员

在 Python 程序中定义的成员变量和方法默认都是公有成员，类之外的任何代码都可以随意访问这些成员。如果在成员变量和方法名之前加上两个下画线“__”作前缀，则该变量或方法就是类的私有成员。私有成员只能在类的内部调用，类外的任何代码都无法访问这些成员。

【例 3.2】 私有成员示例。

```
class testPrivate:
    def _ _init_ _(self, x, y):
        self._ _x = x
        self._ _y = y
    def add(self):
        self._ _s = self._ _x + self._ _y
        return self._ _s
    def printData(self):
        print (self._ _s)

t = testPrivate(3, 5)   # 创建类对象
s = t.add()
t.printData()
print('s = ',s)
```

定义私有方法 _ _init_ _( )

定义类中的普通成员方法

定义类中的普通成员方法

在类的外部调用公有方法

程序运行结果如下：

```
8
s = 8
```

5. 类的构造方法

在 Python 中，类的构造方法为 _ _init_ _( )，其中方法名开始和结束的下画线是双下

画线。构造方法属于对象，每个对象都有自己的构造方法。

如果一个类在程序中没有定义_ _init_ _( )方法，则系统会自动建立一个方法体为空的_ _init_ _( )方法。

如果一个类的构造方法带有参数，则在创建类对象时需要赋实参给对象。

在程序运行时，构造方法在创建对象时由系统自动调用，不需要用调用方法的语句显式调用。

【例 3.3】 设计一个员工类 Person。该类有 Name（姓名）和 Age（年龄）两个变量，可以从键盘输入员工姓名、年龄等信息。

程序代码如下：

```
class Person:
    def _ _init_ _(self, Name, Age):
        self.name = Name
        self.age = Age
    def main(self):
        print(self.name)
        print(self.age)

name = input('input  name:')
age = input('input age:')
p = Person(name，age)
p.main()   # 调用类体中的方法
```

定义构造方法，进行初始化，该构造方法带有参数

定义 main()方法，输出信息

创建对象时，也要带实参。此时系统自动调用构造方法

程序运行结果如下：

```
input name:  sundy
input age:  22
sundy
22
```

6. 析构方法

在 Python 中，析构方法为 _ _del_ _ ( )，其中开始和结束的下画线是双下画线。析构方法用于释放对象所占用的资源，在 Python 系统销毁对象之前自动执行。析构方法属于对象，每个对象都有自己的析构方法。如果类中没有定义_ _del_ _( )方法，则系统会自动提供默认的析构方法。

【例 3.4】 析构方法示例。

程序代码如下：

```
class Mood:
    def _ _init_ _(self,x):
        self.x=x
        print('产生对象',x)

    def _ _del_ _(self):
        print('销毁对象', self.x)
```

定义构造方法，创建对象时触发

定义析构方法，释放对象时触发

```
f1 = Mood(1)
f2 = Mood(2)
del f1
del f2
```

程序运行结果如下：

```
产生对象1
产生对象2
销毁对象1
销毁对象2
```

3.1.2 类的继承

类的继承是为代码复用而设计的，是面向对象程序设计的重要特征之一。当设计一个新类时，如果可以继承一个已有的类，无疑会大幅度减少开发工作量。

在继承关系中，已有的类称为父类或基类，新设计的类称为子类或派生类。派生类可以继承父类的公有成员，但不能继承其私有成员。

在继承中，父类的构造方法_ _init_ _( )不会自动调用，如果在子类中需要调用父类的方法，可以使用内置函数 super()或通过“父类名.方法名( )”的方式实现。

1. 类的单继承

类的单继承的一般形式为：

```
class 子类名(父类名):
   子类的类体语句
```

【例 3.5】 定义一个父类 Person，再定义一个子类 Sunny 继承 Person，并在子类中调用父类的方法。

程序代码如下：

```
class Person:
    def __init__(self, Name, Age):
        self.name = Name
        self.age = Age
    def main(self):
        print('姓名:', self.name)
        print('年龄:', self.age)

class Sunny(Person):
    def __init__(self, name, age, score):
     super(sunny, self).__init__(name, age)
        self.score = score
def prn(self):
        Person.main(self)
        print('成绩:', self.score)
```

定义父类，构造方法带有参数

调用父类的构造方法

定义子类

```
name = input('请输入姓名: ')
age = input('请输入年龄: ')
score = input('请输入成绩: ')
s = Sunny(name,age, score)        # 实例化子类对象
s.prn()                           # 调用子类的方法
```

程序运行结果如下：

```
请输入姓名： 张大山
请输入年龄： 22
请输入成绩： 88
姓名： 张大山
年龄： 22
成绩： 88
```

2. 类的多继承

Python 支持多继承，多继承的一般形式为：

```
class  子类名(父类名1,父类名2, …, 父类名n):
   子类的类体语句
```

Python 在多继承时，如果这些父类中有相同的方法名，而在子类中使用时没有指定父类名，则 Python 系统将从左往右按顺序搜索。

【例 3.6】 多继承示例。

程序代码如下：

```
class A:
    def __init__(self):
        self.one="第一个父类"
class B:
    def __init__(self):
        self.two="第二个父类"

class C(A,B):
    def __init__(self):
        A.__init__(self)
        B.__init__(self)
    def prn(self):
        print(self.one, '\n', self.two)

subc=C()
subc.prn()
```

定义 A 父类

定义 B 父类

定义 A、B 的子类

程序运行结果如下：

```
第一个父类
第二个父类
```

3.1.3 运算符重载

Python 语言提供了运算符重载功能，大大增强了语言的灵活性。运算符重载就是重新定义运算法则。在 Python 中，重载加法运算使用_ _add_ _()方法定义运算法则，重载减法运算使用_ _sub_ _()方法运算法则。

【例 3.7】 设有两个二维元组：（7，10）和（5，−2），它们的加法运算法则为对应元素相加。它们的减法运算法则为对应元素相减。编写程序，计算这两个元组相加、相减的值。

程序代码如下：

```
class Vector:
   def __init__(self, a, b):
      self.a = a
      self.b = b

   def __str__(self):
      return 'Vector (%d, %d)' % (self.a, self.b)

   def __add__(self,other):
      return Vector(self.a + other.a, self.b + other.b)

   def __sub__(self,other):
      return Vector(self.a - other.a, self.b - other.b)

v1 = Vector(7,10)
v2 = Vector(5,-2)
print(v1 + v2)
print(v1 - v2)
```

定义构造方法

定义元组显示方式

定义加法运算法则

定义减法运算法则

程序运行结果如下：

```
Vector(12, 8)
Vector(2, 12)
```

视频讲解

3.2 模　　块

一个较大型的程序通常都是由许多功能函数或类组成的，为了方便程序开发团队分工协作，可以将所建的函数或类保存为模块（module）形式的独立文件，未来可以供其他程序调用。模块的扩展名为 py。

3.2.1 函数模块及函数模块的导入

1. 建立函数模块

在 Python 中，每个包含有函数的 Python 文件都可以作为一个模块来使用，其模块名

就是文件名。

【例 3.8】 创建函数模块示例。

设有 Python 文件 hello.py，其中包含 hh()函数，代码如下：

```
def hh():
        str="你好, Python! "
        return str
```

这样，就建立了一个名为 hello 的模块，其中的 hh()函数可以供其他程序调用。

2. 模块的导入

在 Python 中用关键字 import 来导入某个模块，其导入模块的形式有两种。

1）用 import 形式导入模块中的所有函数

用 import 导入模块的一般形式为：

```
import  模块名
```

比如要引用例 3.8 中的模块 hello，就可以在文件最开始的地方用

```
import hello
```

语句来导入。

在调用 import 导入模块的函数时，必须使用以下形式来调用：

```
模块名.函数名
```

例如，调用 hello 模块中的 hh()函数。

```
import hello
s = hello.hh()
print(s)
```

导入 math 模块

调用 math 模块中的 hh()函数

2）用 from…import…形式导入模块中指定的函数或变量

用 from…import…导入模块的一般形式为：

```
from  模块名 import  函数名或变量名
```

比如要引用模块 math 中的 sqrt()函数，可以用

```
from  math  import sqrt()
```

语句来导入。

在调用 from…import…导入模块的函数时，直接使用函数名来调用模块中的函数，而不需要在函数的前面加上模块名。

【例 3.9】 编写一个计算两数之和的模块，再在另一个程序中调用该模块。

（1）编写模块代码，其中包含有计算两数之和的函数 sum()，保存为 ex3_9_1.py。

```
def sum(n1, n2):
    s = n1 + n2
```

```
    return s
```

（2）编写调用模块程序 ex3_9_2.py，其代码如下：

```
import ex3_9_1
ss = ex3_9_1.sum(3, 5)
print("3 + 5 =",ss)
```

程序 ex3_9_2.py 的运行结果为：

```
3 + 5 = 8
```

3.2.2 类模块

类模块与函数模块类似，将类保存为独立的 Python 文件，其他程序则通过导入模块语句，调用模块中的类。

1. 用 from…import…形式导入类模块

导入类模块的语法格式与导入函数模块的语法格式相同，它的语法如下：

```
from  模块名 import  类名
```

【例 3.10】 设有一个包含银行类的模块，编写程序调用该模块。

（1）编写模块代码，其中包含有银行储户信息类，保存为 ex3_10_Banks.py。

```
# 包含银行储户信息类的模块
class Banks():
    def __init__(self, cname):      # 初始化
        self.__name = cname         # 储户姓名
        self.__amount = 0           # 储户总金额

    def save_money(self, money):    # 存款操作
        self.__amount += money      # 增加总金额
        print("存款", money, "元，目前余额：", self.__amount) # 显示存款信息

    def get_money(self, money):     # 取款操作
        self.__amount -= money      # 减少总金额
        print("取款", money, "元，目前余额：", self.__amount) # 显示存款信息
```

（2）编写调用模块程序 ex3_10_2.py，其代码如下：

```
from ex3_10_banks import Banks      # 导入模块中的 Banks 类

zdsbank = Banks("张大山")           # 声明 Banks 类对象
zdsbank.save_money(8000)            # 存款为 8000 元
```

程序 ex3_10_2.py 的运行结果为：

```
存款 8000 元，目前余额： 8000
```

2. 用 import 形式导入类模块

与用 import 导入函数模块的形式相同，也可以使用如下语法格式导入类模块：

```
import  模块名
```

比如要引用例 3.10 的类模块 ex3_10_banks，就可以在文件最开始的地方用

```
import ex3_10_banks
```

语句来导入。

在调用 import 导入模块的类时，必须使用以下形式来调用：

```
模块名.类名
```

例如，编写用 import 导入类模块 ex3_10_banks 的程序 ex3_10_3.py，其代码如下：

```
import ex3_10_banks                          # 导入 ex3_10_banks 模块

zdsbank = ex3_10_banks.Banks("张大山")       # 声明 Banks 类对象
zdsbank.save_money(8000)                     # 存款为 8000 元
```

程序 ex3_10_3.py 的运行结果为：

```
存款 8000 元，目前余额： 8000
```

3.2.3　常用标准库模块及导入模块的顺序

1. 常用标准库模块

Python 系统的标准库中定义了很多模块，从 Python 语言自身特定的类型和声明，到一些只用于少数程序的不著名的模块，林林总总有 200 多个。

表 3.1 列出了一些常用的标准库模块。

表 3.1　一些常用的标准库模块

模块分类	模块名称	说　明
核心模块	os 模块	os 模块中的大部分函数通过对应平台相关模块实现,其常用方法有 open()、file()、listdir()、system()等函数
	sys 模块	sys 模块用于处理 Python 运行时环境。例如，退出系统时，使用命令：sys.exit(1)
	time 模块	math 模块实现了许多对浮点数的数学运算函数。例如，使用 math 模块的 sqrt()函数进行开平方根的运算
线程与进程模块	threading 模块	threading 模块为线程提供了一个高级接口，只需要继承 Thread 类，定义好 run()方法，就可以创建一个新的线程
	queue 模块	queue 模块提供了一个线程安全的队列 (queue) 实现。通过它可以在多个线程里安全地访问同一个对象

续表

模块分类	模块名称	说　明
网络协议模块	socket 模块	socket 模块实现了网络数据传输层的接口，使用该模块可以创建客户端或是服务器的套接字 Socket 通信
	socketserver 模块	socketserver 为各种基于 Socket 套接字的服务器提供了一个框架，该模块提供了大量的类对象，可以用它们来创建不同的服务器
	urllib 模块	urllib 模块为 HTTP、FTP 以及 Gopher 提供了一个统一的客户端接口，它会自动地根据 URL 选择合适的协议处理器
	httplib 模块	httplib 模块提供了一个 HTTP 客户端接口
	webbrowser 模块	webbrowser 模块提供了一个到系统标准 Web 浏览器的接口。它提供了一个 open()方法, 接收文件名或 URL 作为参数, 然后在浏览器中打开它

2. 导入模块的顺序

当设计的程序需要导入多个模块时，应按照下面的顺序依次导入模块：

（1）导入 Python 系统的标准库模块，如 os、sys 等；

（2）导入第三方扩展库模块，如 pygame、mp3play 等；

（3）导入自己定义和开发的本地模块。

3.2.4 使用 pip 安装和管理扩展模块

1. 安装 pip

Python 安装第三方的模块，大多使用包管理工具 pip 进行安装。Python 包管理工具 pip 提供了对 Python 包的查找、下载、安装、卸载的功能。

pip 下载地址为 https://pypi.python.org/pypi/pip#downloads。选择 pip-9.0.1.tar.gz 文件进行下载。

下载完成后，将解压的文件保存到一个文件夹，使用控制台命令窗口进入解压目录，输入安装命令：

```
python setup.py install
```

pip 安装完后还需要配置环境变量，这样 pip 指令才能生效。找到 Python 安装路径下的 scripts 目录，复制该路径。例如：

```
C:\Users\pandap\AppData\Local\Programs\Python\Python36-32\Scripts
```

将其添加到系统环境变量 path 中。

2. 通过 pip 安装扩展模块

当前，pip 已经成为管理 Python 扩展模块的主要方式。常用 pip 命令如表 3.2 所示。

表 3.2　常用 pip 命令

pip 命令	说明	pip 命令	说明
install	安装模块	list	列出已安装模块
download	下载模块	show	显示模块详细信息.
uninstall	卸载模块	search	搜索模块
freeze	按着一定格式输出已安装模块列表	help	帮助

例如：

（1）安装 MySQL 数据库管理模块：

```
pip install pymysql
```

（2）安装图形处理库模块：

```
pip install pillow
```

（3）安装 SomePackage 模块：

```
pip install SomePackage
```

（4）卸载 SomePackage 模块：

```
pip uninstall SomePackage
```

（5）查看当前已经安装的模块：

```
pip list
```

查看当前已经安装的模块命令运行结果如图 3.1 所示。

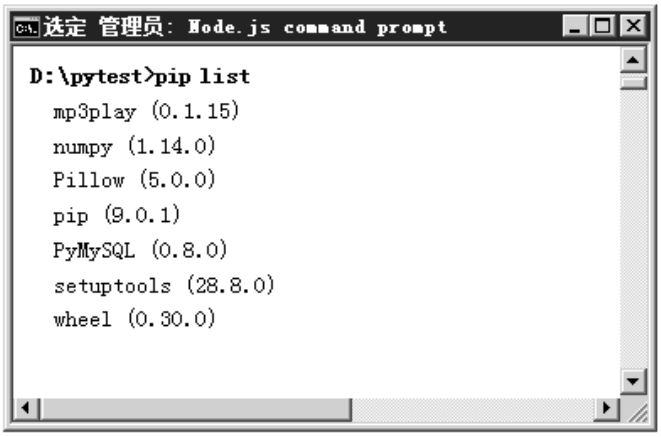

图 3.1　查看已经安装的模块

3.2.5　使用 Anaconda 安装和管理扩展模块

Anaconda 是由 Python 管理的开源数据科学平台，其中包含了 180 个科学计算模块及其依赖项，可以用它来安装和管理扩展模块。

1. 下载和安装 Anaconda

1）下载 Anaconda 安装包

Anaconda 安装包的下载地址为 https://www.anaconda.com，该网站提供了 Python 2.7 和 Python 3.7 两个版本的安装程序，这里选择安装 Python 3.7 版。

2）安装 Anaconda

Anaconda 的安装比较简单，双击安装包后，按安装向导的指引即可以完成安装。

3）配置环境变量

完成 Anaconda 的安装后还需要配置环境变量，Anaconda 才能有效使用。其配置方法与 pip 的配置方法相同，在系统的环境变量 path 中，添加 Anaconda 安装路径下的 scripts 目录。

2. 通过 Anaconda 安装扩展模块

运行 Anaconda，打开 Anaconda Navigator 窗口，在主界面左边的导航栏中选择 Environments 选项，在右边下拉列表框中选择 All 选项，搜索并勾选相应的安装包，然后单击 Apply 按钮安装模块，如图 3.2 所示。

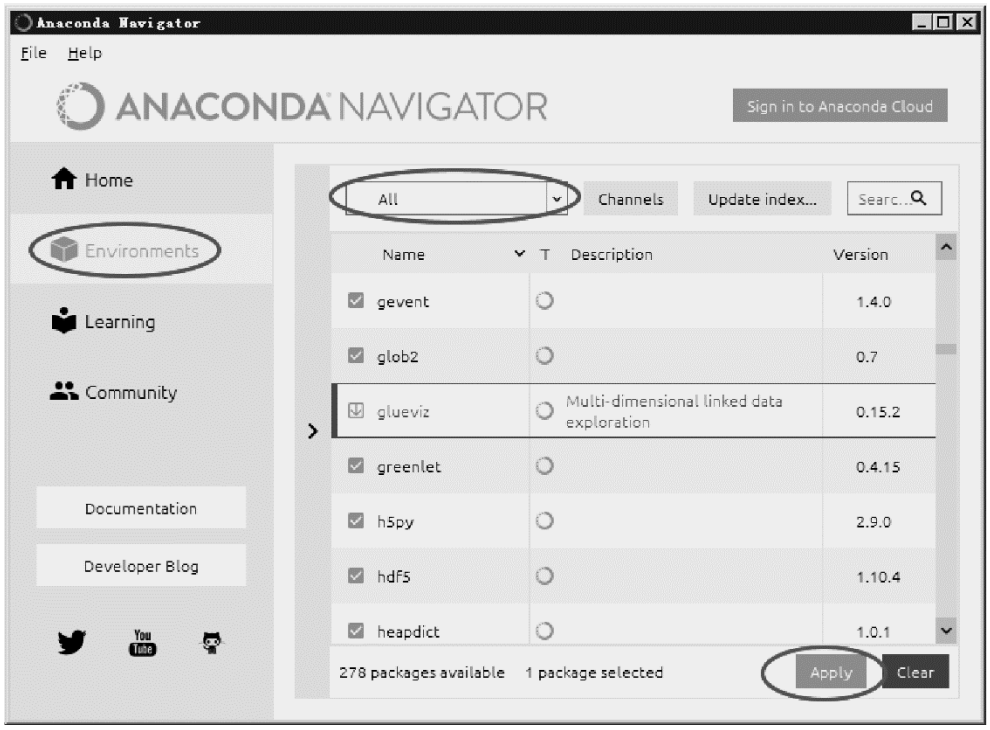

图 3.2　通过 Anaconda 安装模块

3.3　案 例 精 选

视频讲解

【例 3.11】 设计一个学生类。这个学生类中包含学生的学号、姓名和成绩。计算 3 名学生的平均成绩。

程序代码如下：

```
class Student:

    def __init__(self, sid, name, scro):
        self.sid = sid
        self.name = name
        self.scro = scro

    def cot(self):
        return self.scro

    def prnid(self):
        print('学号:',self.sid, '姓名:',self.name, '成绩:',self.scro)


stu1 = Student('a1001','张大山',92)
stu2 = Student('a1002','李晓丽',82)
stu3 = Student('a1003','赵志勇',97)

stu1.prnid()
stu2.prnid()
stu3.prnid()
s = stu1.cot() + stu2.cot() + stu3.cot()
print('平均成绩：',int(s/3))
```

程序运行结果如下：

```
学号：a1001 姓名：张大山 成绩：92
学号：a1002 姓名：李晓丽 成绩：82
学号：a1003 姓名：赵志勇 成绩：97
平均成绩： 90
```

【例 3.12】 设计一个学生类。这个学生类中包含学生的学号、姓名和成绩，并能根据学生人数计算平均成绩。

程序代码如下：

```
class Student:

    def __init__(self):
        self.s = 0
        self.count = 0

    def cot(self):
        return self.s/self.count
```

在构造方法中初始化变量

计算平均成绩

```
    def prnid(self, data):
        for i in data:
          self.sid = i['sid']
          self.name = i['name']
          self.scro = i['scro']
          self.s = self.s + self.scro
          print('学号:',self.sid, '姓名:',self.name, '成绩:',self.scro)
          self.count += 1

data = [{'sid':'a1001','name':'张大山','scro':92},
       {'sid':'a1002','name':'李晓丽','scro':82},
       {'sid':'a1003','name':'赵志勇','scro':97}]
stu = Student()
stu.prnid(data)
s = stu.cot()
print('平均成绩：',int(s))
```

获取列表中的字典元素

程序运行结果如下：

```
学号: a1001 姓名: 张大山 成绩: 92
学号: a1002 姓名: 李晓丽 成绩: 82
学号: a1003 姓名: 赵志勇 成绩: 97
平均成绩： 90
```

习　题　3

1．编写一个具有加、减、乘、除功能的模块，然后通过导入另一个程序中调用。

2．设计一个商品类，该类有商品编号、品名、价格、数量。应用该类，统计 3 种商品的总金额。