# Python面向对象

> 面向对象非常的重要。这个文档必不可能涵盖所有的内容。希望大家可以通过csdn等方式在课后继续学习面向对象。

> 面向对象的三大特性：封装、继承、多态

## 1 面向过程与面向对象

- 面向过程

我们之前编写程序就是面向过程编程。是为了实现一个目的而编写的代码。所有的代码仅仅为了完成当前这个目标。这是种自上而下的编程方式。首先通过目标来确定代码核心框架，然后再向框架中增添详细功能实现代码。这种编程方式对于较大体量的程序来说，既不方便编写，也不方便维护。为了定位实现某一步功能的代码，可能需要将代码完整的浏览一遍。

- 面向对象

我们将有着相似特性和行为的一类代码进行抽象封装，让它形成一个具有一定属性和功能的类来方便我们调用。通过构建一个个类并调用类来构成我们的程序。这是种自下而上的编程方式。首先要构建相关类，通过一个个组件逐步构建程序。

## 2 类与对象

### 2.1 什么是类

在现实生活中，我们会给东西分类。比如鼠标，尽管有不同的品牌，不同的形状等等，它们都因为有左右按键以及一个滚轮并可以操作系统中的光标而被称为鼠标。我们通过总结事物所具有的属性和功能，将相似的归为一类。

对于代码，也是这样。我们可以将具有相似属性和功能的代码块进行抽象总结，构建一个类。

比如登陆这个非常常见的操作。它出现在非常多的网站上，有着不同的登陆界面。我们可以对其进行总结抽象。

首先，有三个最基础的功能：输入账号、输入密码以及提交。有两个属性：账号与密码。

通过上面的抽象总结，我们可以把登陆这个模块设计成一个类。

像这样把相似属性和功能代码放置在一个类中的操作称为封装。

```python
class Login():
    # 两个属性
    username = None
    password = None

    # 三个方法
    def getUsername(self):
        pass

    def getPassword(self):
        pass

    def submit(self):
        pass
```

## 2.2 什么是对象

> 学好了这里七夕的时候大家就可以有好多对象陪着了ヾ(≧▽≦)ゝ

当将一个类被实例化的时候，就可以获得一个对象。

好比我们讲鼠标，这只是一个抽象的概念，它指那一类物品，并不具体到哪一个。

当你买的罗技G502被生产出来的时候，就是鼠标这个类被实例化了。你的G502就是一个对象，它是鼠标类生成的一个对象。

我们使用前面的Login类来进行讲解

```python
# 实例化Login类
a = Login() # 这时候a就是一个对象，是一个Login对象
```

类是无法被直接调用的，我们可以通过实例化类获得这个类的对象来调用其中的方法。

## 2.3 类的基本结构

### 2.3.1 属性与方法

类内部由属性和方法两部分组成。

- 属性：对象的数据描述，为类的成员变量。
- 方法：对象的方法

使用class来声明一个类。

```
```

```python
class Login():

    # 可以直接在这里声明（创建）属性，在没有初始值的情况下推荐优先初始化为None。
    # 在后面学了构造方法之后更推荐在构造方法中初始化相关属性。
    username = None
    password = None

    # 在类中声明的函数即为类的方法
    def  getUsername(self):
        pass

    def getPassword(self):
        pass

    def submit(self):
        pass
```

> self的事情等等再讲。

### 2.3.2 self变量

self变量在类中处处存在，也是类内数据交换的桥梁。

self指向这个方法的调用对象。谁调用了这个方法，在这个方法内self就指向谁。 self由调用者自动传入，不需要我们传入。

```python
class Login():

    def getUsername(self):
        print(self)

a = Login()
b = Login()
a.getUsername() # 不需要传入self，self由调用者a传入
print(a)
b.getUsername()
print(b)
```

运行结果

```
<__main__.Login object at 0x000001FD1F059FD0>
<__main__.Login object at 0x000001FD1F059FD0>
<__main__.Login object at 0x000001FD1F059FA0>
<__main__.Login object at 0x000001FD1F059FA0>
```

可以看到，a指向的对象和a调用getUsername方法后getUsername中self指向的对象是一致的。

有了self我们就可以在类内部调用对象的属性以及方法了。

```
class Login():
    username = None
    password = None

    def getUsername(self):
        print(self.username)
        print("get username ...")
        self.getUsername()

a = Login()
print(a.username)
a.getUsername()
```

### 2.3.3 构造方法

通过实例化类可以获得类的对象。而这个实例化过程由类的构造方法提供。

```
class Login():

    # 名为__init__的函数即为Python中类构造方法。注意：这里的__是两个下划线_
    def __init__(self):
        pass
```

但是在之前的例子中我们并没有编写过任何构造方法，类是如何实例化的?

如果一个类没有提供任何构造方法，那么在这个类被加载的时候，Python解释器会为这个类自动添加一个空参构造器（即没有任何参数的构造器）（就是上面示例中写的那个样子）。

有了构造方法就可以在实例化类的时候为对象传入一定参数了。

比如我们想让前面那个Login类在实例化的时候就获得一个默认用户名，就可以使用带参构造器来实现。

```
class Login():

    def __init__(self, username):
        self.username = username
        self.password = None


a = Login('Python')
print(a.username)
```

### 2.3.4 访问权限

> Python极大弱化了访问权限这个属性，推荐大家在课后去了解Java中的访问权限修饰符。

你手里有两本书，如果别人想看，你不是很介意。你手里有2000块钱，别人想用，你恨不得直接把他按在地上。

你是这么想的，你的对象也是这么想的。那该如何实现访问控制呢。

Python并没有像Java或C++那样提供访问控制符。我们只需要在在变量名/方法名前加上__就无法从外部访问了，这个变量/方法将会转为私有变量/私有方法。

```
class Login() :

    def __init__(self):
        self.__username = "Python"
        self.__password = None

    def getUsername(self) :
        print(self.__username)

a = Login()
print(a.__username)
```

这时候解释器会直接抛出AttributeError，告诉你对象中这个变量不存在（实际上是存在的，只是不允许你在这个对象内部以外的地方调用）

```
b = Login()
b.getUsername()
print(b.__username)
```

通过运行结果我们可以看到，__username这个私有变量只能在对象内部调用，无法从外部访问。

通过创建私有变量/私有方法的方式可以有效防止数据泄露。

虽然我们无法直接访问私有变量，但是我们可以提供公共方法来访问私有变量。比如经常使用的一种逻辑：

```
class User():

    def __init__(self, username, password):
        self.__username = username
        self.__password = password

    def getUsername(self):
        return self.__username

    def setUsername(self, username):
        self.__username = username

```

通过get...和set...来完成私有变量的访问。

> 但是！但是！！但是！！！
>
> 实质上Python是没有私有变量的，Python中所谓的私有变量，只是将其重命名为："_类名__变量名"
>
> 比如前文中的__username，无法直接通过实例化对象访问__username，但是可以访问_User__username，这二者在内存数据指向上是等价的。

# 3 继承与多态

> 继承与多态大大扩展了类的能力，方便了代码复用以及扩展。可以说是类的精髓所在。

> 继承与多态比较难，希望大家能多尝试，多问，一定要学会。

## 3.1 继承

顾名思义，从父辈那里获得他们已有的东西。

在类这个层面上，我们可以为类指定父亲，让他去继承指定类所拥有的属性和方法。

比如这里有个类

```
class Person():

    def __init__(self, name, gender, age):
        self.__name = name
        self.__gender = gender
        self.__age = age

    def getName(self):
        return self.__name

    def getGender(self):
        return self.__gender

    def getAge(self):
        return self.__age
```

众所周知，用户一般是人，所以我们可以通过扩展Person这个类来简化User类的实现，即使用User类继承Person`类

```
class User(Person):

    def __init__(self, id, name, gender, age):
        super(User, self).__init__(name, gender, age)
        self.__id = id

    def getId(self):
        return self.__id

# 上面的代码等价于下面这种写法，哪种更好一目了然
class User(Person):

    def __init__(self, id, name, gender, age):
        self.__name = name
        self.__gender = gender
        self.__age = age
        self.__id = id
```

```
18
19      def getName(self):
20          return self.__name
21
22      def getGender(self):
23          return self.__gender
24
25      def getAge(self):
26          return self.__age
27
28      def getId(self):
29          return self.__id
```

> super()类，用于调用父类内部的方法。
>
> 比如super(User, self)，可以指向User类的父类。在上面继承过程中，使用了super(User, self).__init__(name, gender, age)，即调用父类的构造方法。
>
> 之所以这么做，是因为在继承过程中，子类被实例化的时候不会主动调用父类中的有参构造方法。
>
> 当然，直接使用Person.__init__(name, gender, age)来调用父类构造器也是可以的。

Python支持多继承，即同时继承多个类。

```
1  class sample(clazz1, clazz2):
2      pass
```

## 3.2 多态

多态有两种出现形式：重载与重写

### 3.2.1 重载

有的时候，同一个功能方法我们可能根据需求不同传入不同的参数，其内部处理逻辑有一定的不同。如果为每一个都创建一个独立的方法名，可能调用会非常的不方便。为了方便调用，出现了重载这个概念。

重载，指在同一个类中，创建数个相同的方法名，但是参数列表不完全相同方法。

解释器会根据你传入的参数列表长度来自动确定你所调用的方法。

```
1  class UserFactory() :
2
```

```
    def __init__(self) :
        pass

    def getUser(self):
        id = None
        name = None
        gender = None
        age = None
        return User(id, name, gender, age)

    # 重载
    @overload
    def getUser(self, id):
        id = id
        name = None
        gender = None
        age = None
        return User(id, name, gender, age)
```

> Python作为弱类型语言简化了重载这个概念，不然重载能多写一页纸。

### 3.2.2 重写

有的时候我们并不满意父类所提供的某个方法，可以通过重写（overwrite）来重新编写这个方法。

```
class User(Pserson):

    def __init__(self, id, name, gender, age):
        super(User, self).__init__(name, gender, age)
        self.__id = id

    @overwrite
    def getName(self):
        print("try to get name ....")
        print("success")
        return self.__name

    def getId(self):
        return self.__id
```

这时候我们再使用User对象调用getName方法，就不止会返回用户姓名了。

> 虽然重写了但是仍然可以使用 super() 来调用父类中的方法。

## 4 填坑

在前面个基础知识的时候我挖了不少坑。我来兑现承诺填坑了。