

《Python与数据挖掘》

第4章 面向对象编程

讲师：武永亮



目录

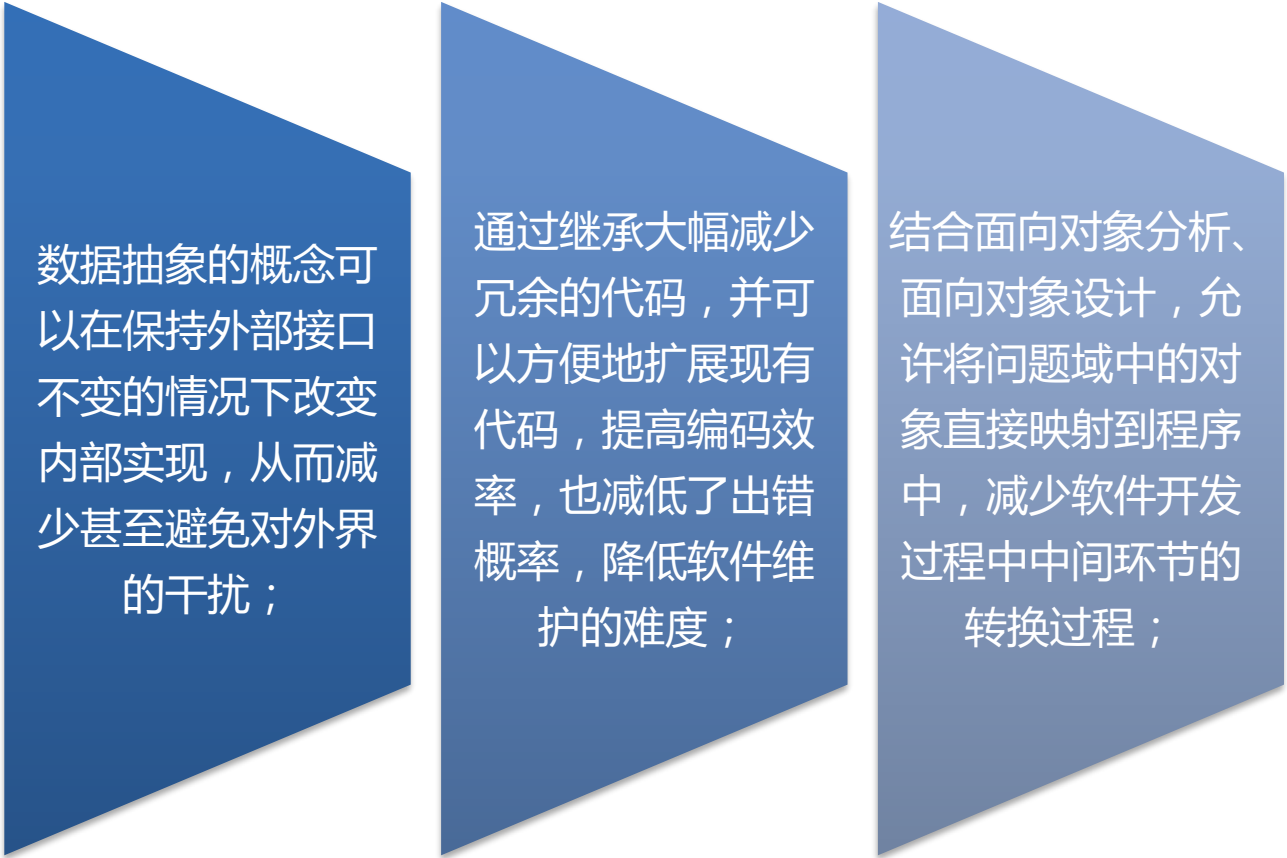
1	简介
2	类与对象
3	_init_方法
4	对象的方法
5	继承

面向对象概述

- 在第二章讲解了Python的主要**内建对象类型**（数字，列表，元组，字典，字符串），而本章我们将介绍我们如何自定义对象。
- Python是一门面向对象编程的语言，因此自定义对象是Python语言的一个核心。本章将先从面向对象的思想开始，然后逐步介绍Python的类和对象。类使得程序设计更加抽象，通过类的继承（inheritance）和组合（composition）使得程序语言接近人类的语言。
- 面向对象是把构成问题事务分解成各个对象，建立对象的目的不是为了完成一个步骤，而是为了描述某个事物在整个解决问题的步骤中的行为。

面向对象优点

- 在面向过程程序设计中，问题被看作一系列需要完成的任务，解决问题的焦点集中于函数。面向对象有如下优点：



数据抽象的概念可以在保持外部接口不变的情况下改变内部实现，从而减少甚至避免对外界的干扰；

通过继承大幅减少冗余的代码，并可以方便地扩展现有代码，提高编码效率，也减低了出错概率，降低软件维护的难度；

结合面向对象分析、面向对象设计，允许将问题域中的对象直接映射到程序中，减少软件开发过程中中间环节的转换过程；

何时使用面向对象编程

- 面向对象的程序是与人类对事物的抽象理解密切相关。
- 举一个例子，虽然我们不知道精灵宝可梦这款游戏（又名口袋妖怪）的具体源码，但我们可以确定的是，它的程序是通过面向对象的思想编写的。我们将游戏中的每种精灵看作一个类，而具体的某只精灵就是其中一个类的一个实例对象。所以每种精灵的程序具有一定的独立性。
- 现在的程序和软件开发都是使用面向对象编程的，最重要的原因还是其良好的抽象性。但对于小型程序和算法来说，面向对象的程序一般会比面向过程的程序慢，所以我们编写程序需要掌握两种思想，发挥出它们的长处。

目录



A vertical table of contents with five items. On the left, a vertical line has five circles numbered 1 to 5. Item 2 is highlighted with a black circle and black text. Each circle is connected to a horizontal bar on the right. Item 2's bar is black with white text, while the others are light gray with black text.

1	简介
2	类与对象
3	_init_方法
4	对象的方法
5	继承

类的定义

- 下面我们正式创建自己的类，这里我们尝试使用Python自定义精灵宝可梦中的小火龙。
- 类的定义就像函数定义，用class语句替代了def语句，同样需要执行了class的整段代码这个类才会生效。进入类定义部分后，会创建出一个新的局部作用域，后面定义的类的数据属性和方法都是属于此作用域的局部变量。上面创建的类很简单，只有一些简单的方法。
- 当捕捉到精灵的时候，我们首先要为其起名字，所以我们先编写函数setName()和getName()。似乎函数中self参数有点奇怪，我们尝试建立具体的对象来探究该参数的作用。

```
>>> pokemon1 = Charmander()
>>> pokemon2 = Charmander()
>>> pokemon1.setName('Bang')
>>> pokemon2.setName('Loop')
>>> print pokemon1.getName()
```

类的定义

Bang

```
>>> print pokemon2.getName()
```

Loop

```
>>> print pokemon1.getInfo()
```

```
<__main__.Charmander instance at 0x02F26B98>
```

```
>>> print pokemon2.getInfo()
```

```
<__main__.Charmander instance at 0x02F26AF8>
```

- 创建对象和调用一个函数很相似，使用类名作为关键字创建一个类的对象。实际上Charmander的括号里是可以有参数的。
- 我们捕捉了两只精灵，一只名字为Bang，另一只为Loop，并且对面它们执行getName()，名字正确返回。观察getInfo()的输出，返回的是包含地址的具体对象的信息，可以看到两个对象的地址是不一样的。

类的属性

- self的作用是与C++的*this指针类似，在调用Charmander的setName和getName函数时，函数都会自动把该对象的地址作为第一个参数传入（该信息包含在参数self中），这就是为什么我们调用函数时不需要写self而在函数的定义都需要把self作为第一个参数。
- 类的每个对象都会有各自的数据属性。Charmander类中有数据属性name,这是通过setName()函数中的语句self.name = name创建的。
- 这个语句中的两个name是不一样的，它们的作用域不一样。第一个name通过self语句声明了作用域是类Charmander()的作用域，将其作为pokemon1的数据属性进行存储，而后面的name的作用域是函数的局部作用域，与参数中的name相同。而后面getName()函数返回的是对象中的name。

目录

1	简介
2	类与对象
3	_init_方法
4	对象的方法
5	继承

__init__方法

- 从深一层的逻辑去说，我们捕捉到精灵的一刻应该已经起好名字，而并非捕捉后再去设置。所以这里我们需要的是一个初始化的手段
- Python中的__init__方法用于初始化类的实例对象。__init__函数的作用一定程度上与C++的构造函数相似，但并不等于。
- C++的构造函数是使用该函数去创建一个类的示例对象，而Python执行__init__方法的时候实例对象已被构造出来。__init__方法会在对象构造出来后自动执行，所以可以用于初始化我们所需要的数据属性。

__init__方法

- 为保持数据类型的一致性，我们使用元组存储，并让小火龙的第二个属性为None。由于小火龙的属性是固定的，所以在__init__的输入参数不需要type。我们创建实例对象测试代码：

```
>>>pokemon1 = Charmander('Bang','male',5)
>>>pokemon2 = Charmander('Loop','female',6)
>>>print pokemon1.getName(),pokemon1.getGender(),pokemon1.getStatus()
Bang male [20, 10, 10, 10, 10, 10]
>>>print pokemon2.getName(),pokemon2.getGender(),pokemon2.getStatus()
Loop female [22, 11, 11, 11, 11, 11]
```
- 这时候创建实例对象就需要参数了，实际上这是__init__函数的参数。
__init__自动将数据属性进行了初始化，然后调用相关函数能够返回我们需要的对象的数据属性。

目录

1	简介
2	类与对象
3	_init_方法
4	对象的方法
5	继承

方法引用

- 本节我们详细探讨对象的方法，类的方法和对象的方法是一样。我们在定义类的方法时程序没有为类的方法分配内存，而创建具体实例对象程序才会为对象的每个数据属性和方法分配内存。
- 我们已经知道定义类的方法是def定义的，具体定义格式与普通函数相似，只不过类的方法的第一个参数需要为self参数。我们可以用普通函数实现对对象函数的引用：

```
>>>pokemon1 = Charmander('Bang','male',5)
>>>getStatus1 = pokemon1.getStatus
>>>print getStatus1()
[20, 10, 10, 10, 10, 10]
```
- 虽然这看上去似乎是调用了一个普通函数，但是getStatus1()这个函数是引用pokmemon1.getStatus()的，意味着程序还是隐性地加入了self参数。

私有化

- 如果要获取对象的数据属性并不需要通过getName(), getType()等方法，直接在程序外部调用数据属性即可：

```
>>> print pokemon1.type , pokemon1.getType()
('fire', None) ('fire', None)
>>> print pokemon1.gender , pokemon1.getGender()
male male
```

- 虽然这似乎很方便，但是这却违反了类的封装的原则。对象的状态对于类外部应该是不可访问的。为什么要这样做？
- 我们查看Python的模块的源码你会发现源码里面定义的很多类，模块中的算法通过使用类实现是很常见的，如果我们使用算法时能够随意访问对象中的数据属性，那么很有可能在不经意中修改了算法中已经设置的参数，这是十分糟糕的。一般封装好的类都会有足够的函数接口给程序员使用，程序员没有必要访问对象的具体数据属性。

私有化

- 为防止程序员无意地修改了对象的状态，我们需要对类的数据属性和方法进行私有化。
- Python不支持直接私有方式，但可以使用一些小技巧达到私有特性的目的。为了让方法数据属性或方法变为私有，只需要在它的名字前面加上双下划线即可。
- 现在在程序外部直接访问私有数据属性是不允许的，我们只能通过设定好的接口函数去调取对象的信息。不过通过双下划线去私有化实际上是“伪私有化”，实际上我们还是可以做到从外部访问私有这些数据属性。

```
>>> print pokemon1._Charmander__type  
('fire', None)
```


私有化

- Python使用的是一种name_mangling技术，将__membername替换成_class__membername，所以在外部使用原来的私有成员时，会提示无法找到，而上面执行pokemon1._Charmander__type是可以访问。
- 可以看到代码中还增加了一个函数level_up()，这个函数用于处理精灵升级时能力的提升。
- 我们不应该在外部修改pokemon的status，所以我们应准备好接口去处理能力发生变化的情景。函数level_up()仅是一个简单的例子，在工业代码中，如此的函数接口是大量的，程序需要对它们进行归类并附上相应的文档进行说明。

迭代器

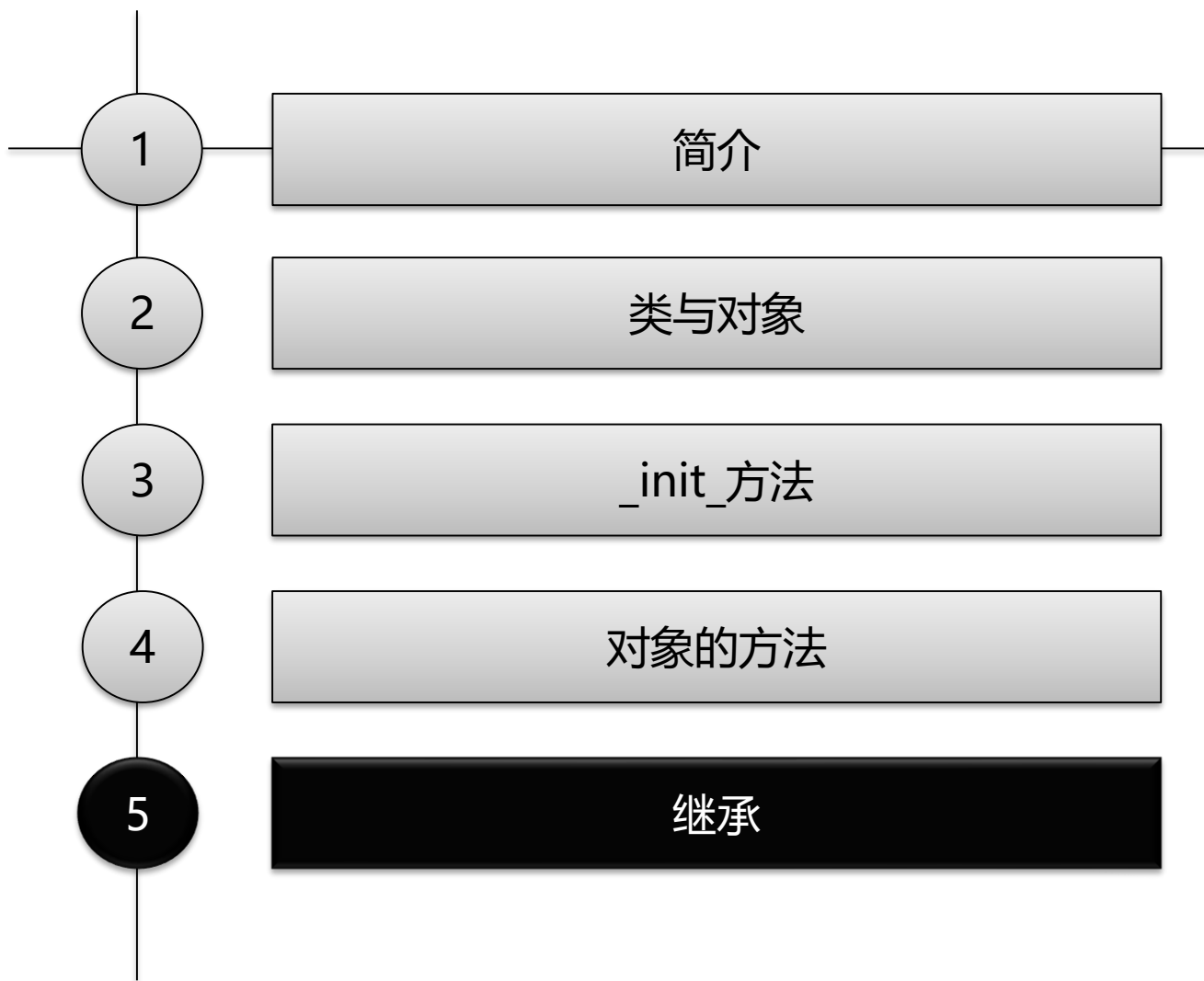
- 我们前面接触到的Python容器对象都可以用for遍历：
- for语句在容器对象上调用了iter(),该函数返回一个定义了next()方法的迭代器对象，它将在容器中逐一访问元素。当容器遍历完毕，next()找不到后续元素时，next()会引发一个StopIteration异常，告知for循环终止。
- 例如：

```
>>> L = [1, 2, 3]
>>> it = iter(L)
>>> it
<listiterator object at 0x0302E050>
>>> it.next()
1
>>> it.next()
2
>>> it.next()
3
```

迭代器

- 当知道迭代器协议背后的机制后，我们便可以把迭代器加入到自己的类中。我们需要定义一个`__iter__()`方法，它返回一个有`next`方法的对象。如果类定义了`next()`，`__iter__()`可以只返回`self`。再次修改类`Charmenda`的代码，通过迭代器能输出对象的全部信息。

目录



继承

- 面向对象的编程带来好处之一是代码的重用，实现这种重用方法之一是通过继承机制。
- 继承是两个类或多个类之间的父子关系，子类继承了基类的所有公有数据属性和方法，并且可以通过编写子类的代码扩充子类的功能。
- 继承实现了数据属性和方法的重用，减少了代码的冗余度。
- 那么我们何时需要使用继承呢？如果我们需要的类中具有公共的成员，且具有一定的递进关系，那么我们的就可以使用继承，且让结构最简单的类作为基类。
- 一般来说，子类是父类的特殊化，如下面的关系：

哺乳类动物 ——> 狗 ——> 特定狗种

继承特点

- 继承语法：class 子类名(基类名1, 基类名2, ...) 基类写在括号里，如果有多个基类，则需要全部都写在括号里，这种情况称为多继承。在Python中继承有下面一些特点：
 1. 在继承中基类初始化方法__init__不会被自动调用。如果希望子类调用基类的__init__方法，需要在子类的__init__方法显示调用它。这与C++和C#区别很大。
 2. 在调用基类的方法时，需要加上基类的类名前缀，且带上self参数变量。注意在类中调用在该类定义的方法是不需要self参数的。
 3. Python总是首先查找对应类的方法，如果在子类没有对应的方法，Python才会在继承链的基类中按顺序查找。
 4. 在Python继承中，子类不能访问基类的私有成员。

Thank You!