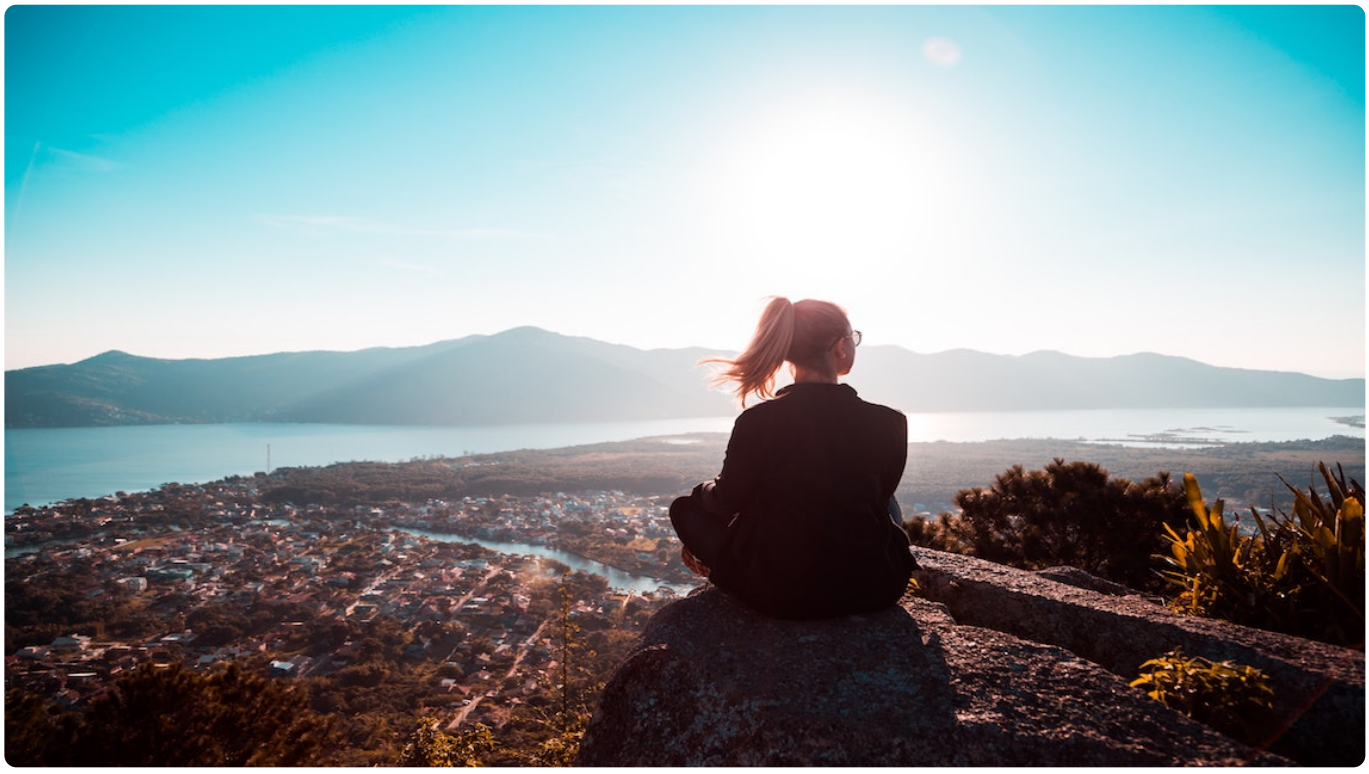
11 | 面向对象（上）：从生活中的类比说起

2019-06-03 景霄

Python核心技术与实战 进入课程 >



讲述：冯永吉
时长 15:37 大小 14.31M

你好，我是景霄。

很多朋友最开始学编程的时候，是从 C++ 或者 JAVA 语言入手的。他们好不容易磕磕绊绊地搞懂了最基本的数据类型、赋值判断和循环，却又迎面撞上了 OOP (object oriented programming) 的大墙，一头扎进公有私有保护、多重继承、多态派生、纯函数、抽象类、友元函数等一堆专有名词的汪洋大海中找不到彼岸，于是就放弃了进阶之路。

相比之下，Python 是一门相对友好的语言，它在创立之初就鼓励命令交互式的轻量级编程。理论上，Python 的命令式语言是图灵完备的，也就是说命令式语言，理论上可以做到其他任何语言能够做到的所有的事情，甚至进一步，仅仅依靠汇编语言的 MOV 指令，就能实现图灵完备编程。

那么为什么不这样做呢？其实，“上古时代”的程序员就是这么做的，可是随着程序功能复杂性的逐步提升，以及需求的不断迭代，很多老旧的代码修改起来麻烦无比，牵一发而动全身，根本无法迭代和维护，甚至只能推倒重来，这也是很多古老的代码被称为“屎山”的原因。

传统的命令式语言有无数重复性代码，虽然函数的诞生减缓了许多重复性，但随着计算机的发展，只有函数依然不够，需要把更加抽象的概念引入计算机才能缓解（而不是解决）这个问题，于是 OOP 应运而生。

Python 在 1989 年被一位程序员打发时间创立之后，一步步攻城掠地飞速发展，从最基础的脚本程序，到后来可以编写系统程序、大型工程、数据科学运算、人工智能，早已脱离了当初的设计，因此一些其他语言的优秀设计之处依然需要引入。我们必须花费一定的代价掌握面向对象编程，才能跨越学习道路中的瓶颈期，走向下一步。

接下来，我将用两节课来讲解面向对象编程，从基础到实战。第一讲，我将带你快速但清晰地疏通最基础的知识，确保你能够迅速领略面向对象的基本思想；第二讲，我们从零开始写一个搜索引擎，将前面所学知识融会贯通。

这些内容可能和你以往看到的所有教程都不太一样，我会尽可能从一个初学者的角度来审视这些难点。同时我们面向实战、面向工程，不求大而全，但是对最核心的思想会有足够的勾勒。我可以保证内容清晰易懂，但想要真正掌握，仍要求你能用心去阅读和思考。真正的提高，永远要靠自己才能做到。

对象，你找到了吗？

我们先来学习，面向对象编程中最基本的概念。

为了方便你理解其中的抽象概念，我先打个比方带你感受一下。生物课上，我们学过“界门纲目科属种”的概念，核心思想是科学家们根据各种动植物、微生物的相似之处，将其分化为不同的类型方便研究。生活中我们也是如此，习惯对身边的事物进行分类：

猫和狗都是动物；

直线和圆都是平面几何的图形；

《哈利波特》和《冰与火之歌》（即《权力的游戏》）都是小说。

自然，同一类事物便会有着相似的特性：

动物会动；

平面图形有面积和周长；

小说也都有相应的作者和大致情节等各种元素。

那回到我们的 Python 上，又对应哪些内容呢？这里，我们先来看一段最基本的 Python 面向对象的应用代码，不要被它的长度吓到，你无需立刻看懂所有代码，跟着节奏来，我会一点点为你剖析。

复制代码

```
class Document():
    def __init__(self, title, author, context):
        print('init function called')
        self.title = title
        self.author = author
        self.__context = context # __ 开头的属性是私有属性

    def get_context_length(self):
        return len(self.__context)

    def intercept_context(self, length):
        self.__context = self.__context[:length]

harry_potter_book = Document('Harry Potter', 'J. K. Rowling', '... Forever Do not belie

print(harry_potter_book.title)
print(harry_potter_book.author)
print(harry_potter_book.get_context_length())

harry_potter_book.intercept_context(10)

print(harry_potter_book.get_context_length())

print(harry_potter_book.__context)

########## 输出 ##########

init function called
Harry Potter
J. K. Rowling
77
10

------------------------------------------------------------------------
```

```
AttributeError                            Traceback (most recent call last)
<ipython-input-5-b4d048d75003> in <module>()
     22 print(harry_potter_book.get_context_length())
     23
---> 24 print(harry_potter_book.__context)

AttributeError: 'Document' object has no attribute '__context'
```

参照着这段代码，我先来简单解释几个概念。

类：一群有着相似性的事物的集合，这里对应 Python 的 class。

对象：集合中的一个事物，这里对应由 class 生成的某一个 object，比如代码中的 harry_potter_book。

属性：对象的某个静态特征，比如上述代码中的 title、author 和 __context。

函数：对象的某个动态能力，比如上述代码中的 intercept_context () 函数。

当然，这样的说法既不严谨，也不充分，但如果你对面向对象编程完全不了解，它们可以让你迅速有一个直观的了解。

这里我想多说两句。回想起当年参加数学竞赛时，我曾和一个大佬交流数学的学习，我清楚记得我们对数学有着相似的观点：很多数学概念非常抽象，如果纯粹从数理逻辑而不是更高的角度去解题，很容易陷入僵局；而具体、直观的想象和类比，才是迅速打开数学大门的钥匙。虽然这些想象和类比不严谨也不充分，很多时候甚至是错误或者异想天开的，但它们确实能帮我们快速找到正确的大门。

就像很多人都有过的一个疑惑，“学霸是怎样想到这个答案的？”。德国数学家克莱因曾说过，“推进数学的，主要是那些有卓越直觉的人，而不是以严格的证明方法见长的人。”编程世界同样如此，如果你不满足于只做一个 CRUD“码农”，而是想成为一个优秀的工程师，那就一定要积极锻炼直觉思考和快速类比的能力，尤其是在找不到 bug 的时候。这才是编程学习中能给人最快进步的方法和路径。

言归正传，继续回到我们的主题，还是通过刚刚那段代码，我想再给类下一个更为严谨的定义。

类，一群有着相同属性和函数的对象的集合。

虽然有循环论证之嫌（lol），但是反复强调，还是希望你能对面向对象的最基础的思想，有更真实的了解。清楚记住这一点后，接下来，我们来具体解读刚刚这段代码。为了方便你的阅读学习，我把它重新放在了这段文字下方。

复制代码

```
class Document():
    def __init__(self, title, author, context):
        print('init function called')
        self.title = title
        self.author = author
        self.__context = context # __ 开头的属性是私有属性

    def get_context_length(self):
        return len(self.__context)

    def intercept_context(self, length):
        self.__context = self.__context[:length]

harry_potter_book = Document('Harry Potter', 'J. K. Rowling', '... Forever Do not belie

print(harry_potter_book.title)
print(harry_potter_book.author)
print(harry_potter_book.get_context_length())

harry_potter_book.intercept_context(10)

print(harry_potter_book.get_context_length())

print(harry_potter_book.__context)

########## 输出 ##########

init function called
Harry Potter
J. K. Rowling
77
10

---------------------------------------------------------------------------
AttributeError                            Traceback (most recent call last)
<ipython-input-5-b4d048d75003> in <module>()
     22 print(harry_potter_book.get_context_length())
     23
---> 24 print(harry_potter_book.__context)

AttributeError: 'Document' object has no attribute '__context'
```

可以看到，class Document 定义了 Document 类，再往下能看到它有三个函数，这三个函数即为 Document 类的三个函数。

其中，**init** 表示构造函数，意即一个对象生成时会被自动调用的函数。我们能看到，`harry_potter_book = Document(...)`这一行代码被执行的时候，`'init function called'`字符串会被打印出来。而 get_context_length() 和 intercept_context() 则为类的普通函数，我们调用它们来对对象的属性做一些事情。

class Document 还有三个属性，title、author 和 __context 分别表示标题、作者和内容，通过构造函数传入。这里代码很直观，我们可以看到， intercept_context 能修改对象 harry_potter_book 的 __context 属性。

这里唯一需要强调的一点是，如果一个属性以 __ （注意，此处有两个 _） 开头，我们就默认这个属性是私有属性。私有属性，是指不希望在类的函数之外的地方被访问和修改的属性。所以，你可以看到，title 和 author 能够很自由地被打印出来，但是`print(harry_potter_book.__context)`就会报错。

老师，能不能再给力点?

掌握了最基础的概念，其实我们已经能做很多很多的事情了。不过，在工程实践中，随着复杂度继续提升，你可能会想到一些问题：

> 如何在一个类中定义一些常量，每个对象都可以方便访问这些常量而不用重新构造?
>
> 如果一个函数不涉及到访问修改这个类的属性，而放到类外面有点不恰当，怎么做才能更优雅呢?
>
> 既然类是一群相似的对象的集合，那么可不可以是一群相似的类的集合呢?

前两个问题很好解决，不过，它们涉及到一些常用的代码规范，这里我放了一段代码示例。同样的，你无需一口气读完这段代码，跟着我的节奏慢慢学习即可。

复制代码

```
class Document():

    WELCOME_STR = 'Welcome! The context for this book is {}.'

    def __init__(self, title, author, context):
        print('init function called')
```

```
        self.title = title
        self.author = author
        self.__context = context

    # 类函数
    @classmethod
    def create_empty_book(cls, title, author):
        return cls(title=title, author=author, context='nothing')

    # 成员函数
    def get_context_length(self):
        return len(self.__context)

    # 静态函数
    @staticmethod
    def get_welcome(context):
        return Document.WELCOME_STR.format(context)


empty_book = Document.create_empty_book('What Every Man Thinks About Apart from Sex', 'I


print(empty_book.get_context_length())
print(empty_book.get_welcome('indeed nothing'))

########## 输出 ##########

init function called
7
Welcome! The context for this book is indeed nothing.
```

第一个问题，在 Python 的类里，你只需要和函数并列地声明并赋值，就可以实现这一点，例如这段代码中的 WELCOME_STR。一种很常规的做法，是用全大写来表示常量，因此我们可以在类中使用 self.WELCOME_STR ，或者在类外使用 Entity.WELCOME_STR ，来表达这个字符串。

而针对第二个问题，我们提出了类函数、成员函数和静态函数三个概念。它们其实很好理解，前两者产生的影响是动态的，能够访问或者修改对象的属性；而静态函数则与类没有什么关联，最明显的特征便是，静态函数的第一个参数没有任何特殊性。

具体来看这几种函数。一般而言，静态函数可以用来做一些简单独立的任务，既方便测试，也能优化代码结构。静态函数还可以通过在函数前一行加上 @staticmethod 来表示，代码中也有相应的示例。这其实使用了装饰器的概念，我们会在后面的章节中详细讲解。

而类函数的第一个参数一般为 cls，表示必须传一个类进来。类函数最常用的功能是实现不同的 **init** 构造函数，比如上文代码中，我们使用 create_empty_book 类函数，来创造新的书籍对象，其 context 一定为 `'nothing'`。这样的代码，就比你直接构造要清晰一些。类似的，类函数需要装饰器 @classmethod 来声明。

成员函数则是我们最正常的类的函数，它不需要任何装饰器声明，第一个参数 self 代表当前对象的引用，可以通过此函数，来实现想要的查询 / 修改类的属性等功能。

继承，是每个富二代的梦想

接下来，我们来看第三个问题，既然类是一群相似的对象的集合，那么可不可以是一群相似的类的集合呢?

答案是，当然可以。只要抽象得好，类可以描述成任何事物的集合。当然你要小心、严谨地去定义它，不然一不小心就会引起第三次数学危机 XD。

类的继承，顾名思义，指的是一个类既拥有另一个类的特征，也拥有不同于另一个类的独特特征。在这里的第一个类叫做子类，另一个叫做父类，特征其实就是类的属性和函数。

复制代码

```
class Entity():
    def __init__(self, object_type):
        print('parent class init called')
        self.object_type = object_type

    def get_context_length(self):
        raise Exception('get_context_length not implemented')

    def print_title(self):
        print(self.title)

class Document(Entity):
    def __init__(self, title, author, context):
        print('Document class init called')
        Entity.__init__(self, 'document')
        self.title = title
        self.author = author
        self.__context = context

    def get_context_length(self):
        return len(self.__context)

class Video(Entity):
```

```
    def __init__(self, title, author, video_length):
        print('Video class init called')
        Entity.__init__(self, 'video')
        self.title = title
        self.author = author
        self.__video_length = video_length

    def get_context_length(self):
        return self.__video_length

harry_potter_book = Document('Harry Potter(Book)', 'J. K. Rowling', '... Forever Do not
harry_potter_movie = Video('Harry Potter(Movie)', 'J. K. Rowling', 120)

print(harry_potter_book.object_type)
print(harry_potter_movie.object_type)

harry_potter_book.print_title()
harry_potter_movie.print_title()

print(harry_potter_book.get_context_length())
print(harry_potter_movie.get_context_length())

########## 输出 ##########

Document class init called
parent class init called
Video class init called
parent class init called
document
video
Harry Potter(Book)
Harry Potter(Movie)
77
120
```

我们同样结合代码来学习这些概念。在这段代码中，Document 和 Video 它们有相似的地方，都有相应的标题、作者和内容等属性。我们可以从中抽象出一个叫做 Entity 的类，来作为它俩的父类。

首先需要注意的是构造函数。每个类都有构造函数，继承类在生成对象的时候，是不会自动调用父类的构造函数的，因此你必须在 **init**() 函数中显式调用父类的构造函数。它们的执行顺序是 子类的构造函数 -> 父类的构造函数。

其次需要注意父类 get_context_length() 函数。如果使用 Entity 直接生成对象，调用 get_context_length() 函数，就会 raise error 中断程序的执行。这其实是一种很好的写法，叫做函数重写，可以使子类必须重新写一遍 get_context_length() 函数，来覆盖掉原有函数。

最后需要注意到 print_title() 函数，这个函数定义在父类中，但是子类的对象可以毫无阻力地使用它来打印 title，这也就体现了继承的优势：减少重复的代码，降低系统的熵值（即复杂度）。

到这里，你对继承就有了比较详细的了解了，面向对象编程也可以说已经入门了。当然，如果你想达到更高的层次，大量练习编程，学习更多的细节知识，都是必不可少的。

最后，我想再为你扩展一下抽象函数和抽象类，我同样会用一段代码来辅助讲解。

复制代码

```
from abc import ABCMeta, abstractmethod

class Entity(metaclass=ABCMeta):
    @abstractmethod
    def get_title(self):
        pass

    @abstractmethod
    def set_title(self, title):
        pass

class Document(Entity):
    def get_title(self):
        return self.title

    def set_title(self, title):
        self.title = title

document = Document()
document.set_title('Harry Potter')
print(document.get_title())

entity = Entity()

########## 输出 ##########

Harry Potter

---------------------------------------------------------------------------
TypeError                                 Traceback (most recent call last)
```

```
<ipython-input-7-266b2aa47bad> in <module>()
     21 print(document.get_title())
     22
---> 23 entity = Entity()
     24 entity.set_title('Test')

TypeError: Can't instantiate abstract class Entity with abstract methods get_title, set_
```

你应该发现了，Entity 本身是没有什么用的，只需拿来定义 Document 和 Video 的一些基本元素就够了。不过，万一你不小心生成 Entity 的对象该怎么办呢？为了防止这样的手误，必须要介绍一下抽象类。

抽象类是一种特殊的类，它生下来就是作为父类存在的，一旦对象化就会报错。同样，抽象函数定义在抽象类之中，子类必须重写该函数才能使用。相应的抽象函数，则是使用装饰器 @abstractmethod 来表示。

我们可以看到，代码中`entity = Entity()`直接报错，只有通过 Document 继承 Entity 才能正常使用。

这其实正是软件工程中一个很重要的概念，定义接口。大型工程往往需要很多人合作开发，比如在 Facebook 中，在 idea 提出之后，开发组和产品组首先会召开产品设计会，PM（Product Manager，产品经理） 写出产品需求文档，然后迭代；TL（Team Leader，项目经理）编写开发文档，开发文档中会定义不同模块的大致功能和接口、每个模块之间如何协作、单元测试和集成测试、线上灰度测试、监测和日志等等一系列开发流程。

抽象类就是这么一种存在，它是一种自上而下的设计风范，你只需要用少量的代码描述清楚要做的事情，定义好接口，然后就可以交给不同开发人员去开发和对接。

总结

到目前为止，我们一直在强调一件事情：面向对象编程是软件工程中重要的思想。正如动态规划是算法中的重要思想一样，它不是某一种非常具体的技术，而是一种综合能力的体现，是将大型工程解耦化、模块化的重要方法。在实践中要多想，尤其是抽象地想，才能更快掌握这个技巧。

回顾一下今天的内容，我希望你能自己回答下面两个问题，作为今天内容的总结，写在留言区里。

第一个问题，面向对象编程四要素是什么？它们的关系又是什么？

第二个问题，讲了这么久的继承，继承究竟是什么呢？你能用三个字表达出来吗？

> 这里不开玩笑，Facebook 很多 Launch Doc （上线文档）中要求用五个单词总结你的文档，因为你的文档不仅仅是你的团队要看，往上走甚至会到 VP 或者 CTO 那里，你需要言简意赅，让他们快速理解你想要表达的意思。

思考题

最后，再给你留一道思考题。既然你能通过继承一个类，来获得父类的函数和属性，那么你能继承两个吗？答案自是能的，这就叫做多重继承。那么问题来了。

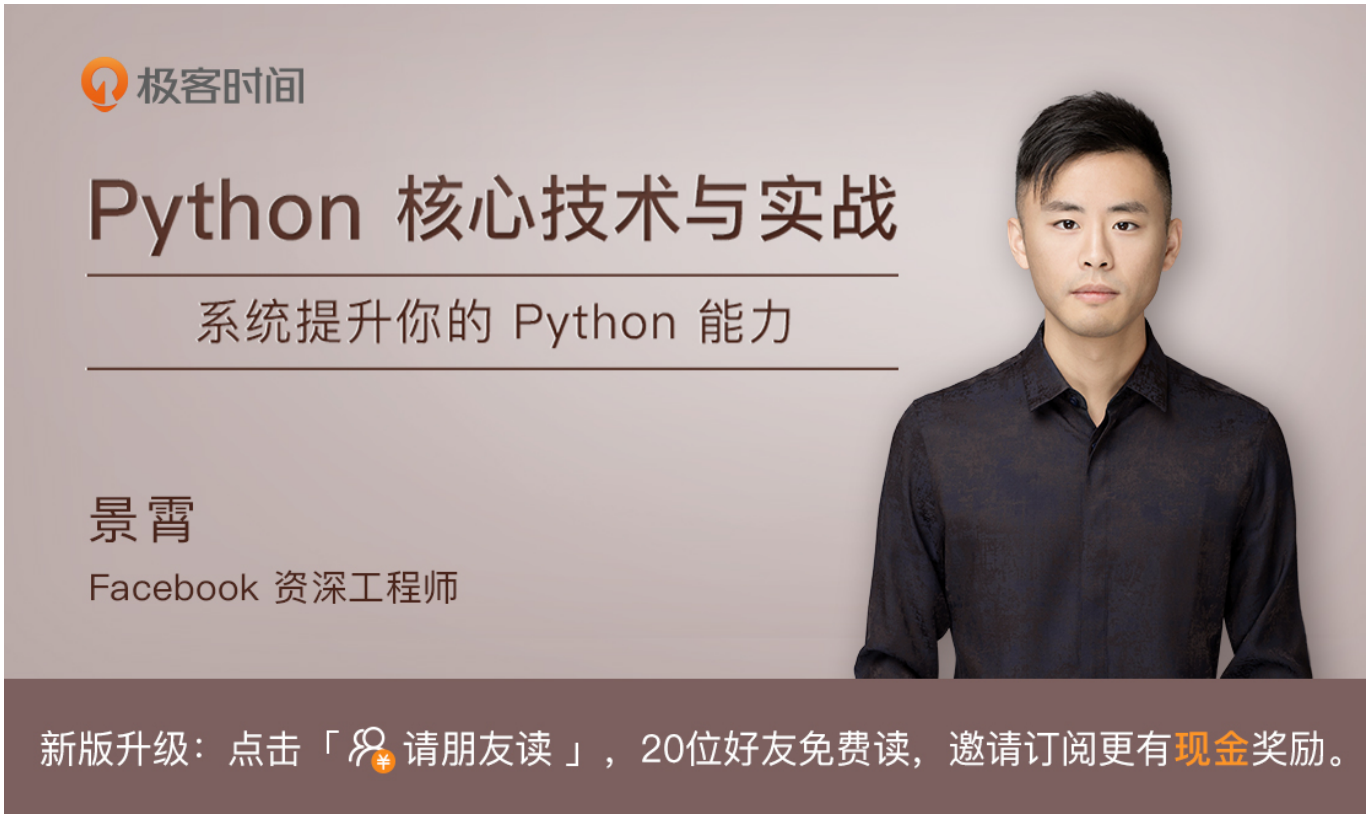
我们使用单一继承的时候，构造函数的执行顺序很好确定，即子类 -> 父类 -> 爷类 ->... 的链式关系。不过，多重继承的时候呢？比如下面这个例子。

复制代码

```
  --->B---
A-       -->D
  --->C---
```

这种继承方式，叫做菱形继承，BC 继承了 A，然后 D 继承了 BC，创造一个 D 的对象。那么，构造函数调用顺序又是怎样的呢？

欢迎在留言区写下你的答案想法，与我讨论，也欢迎你把这篇文章分享给你的同事、朋友。



上一篇　10 | 简约不简单的匿名函数

下一篇　12 | 面向对象（下）：如何实现一个搜索引擎?

精选留言 (50)

写留言

Jingxiao 置顶
2019-06-04

7

思考题答案：庄小P 同学的写法很好，非常明确的表明了菱形继承潜在的问题：一个基类的初始化函数可能被调用两次。在一般的工程中，这显然不是我们所希望的。正确的做法应该是使用 super 来召唤父类的构造函数，而且 python 使用一种叫做方法解析顺序的算法（具体实现算法叫做 C3），来保证一个类只会被初始化一次。

...

展开

庄小P
2019-06-03

13

class A():

```
    def __init__(self):
        print('A class called')

class B(A):...
```

展开

---

hlz-123
2019-06-03

11

第一个问题，面向对象编程四要素是什么？它们的关系又是什么？
答：面向对象编程四要素是类，属性，函数，对象，
　　它们关系可以总结为：类是一群具有相同属性和函数的对象的集合。
第二个问题，讲了这么久的继承，继承究竟是什么呢？你能用三个字表达出来吗？
三个字：父与子。儿子可以使用自己的东西，没有的可以使用父亲的东西。

展开

不瘦到140...
2019-06-03

6

思考题：多重继承，是基于mro进行查找，使用的是一种C3的算法。总结一下规律就是：
B F

C G

...

展开

Geek_59f23...
2019-06-04

4

哥，能不能教教怎么搭梯子？给的链接都是国外的，进不了咋整。。

展开

LiANGZE
2019-06-03

4

只知道多重继承时会通过mro算法生成一个顺序，可以通过 xxx.__mro__ 查看继承的顺序，但其中原理确实没深入研究过 🤔

DX3906
2019-06-03

3

init双下方法是初始化方法，构造方法是双下new

展开

HelloWorld
2019-06-04

2

面向对象编程的四要素： 类、属性、函数（方法）、对象（实例）

下面来展开总结：

类： 一群有着相同属性和函数(方法)的对象(实例)的集合，也可以具象化的理解为是一群...

展开

奔跑的蜗牛
2019-06-03

2

经典类：深度优先，F->D->B->A->E->C->H
新式类：广度优先，F->D->B->E->C->H->A

```
class A:
    # def test(self):...
```

展开

作者回复: 👍

程序员人生
2019-06-03

2

类函数有什么特别之处?

展开

Geek_59f23...
2019-06-04

1

1. 类 对象 属性 函数
2. DRY (don't repeat yourself)
3. D B A C (C3算法)

展开

Fergus
2019-06-03

1

对于思考题，感谢提到c3算法的小伙伴，给补充了知识。但可能过度解读老师的意图了吧

Fergus
2019-06-03

1

先写答案再看答案：
Q1.面向对象编程四要素是什么？它们的关系又是什么？

A. 类，实例（对象），属性，方法（函数）；
...
展开

kevinsu
2019-06-03

1

第一题
由类，属性，函数，对象联想到了哺乳动物这块儿例如猴子和人，人的特征属性和行为，猴子的特征属性和行为，以及根据物种起源学说我们可能有共同的老祖先且继承了老祖先的一些属性和函数。这样来理解这几个概念非常深刻。
第二题...
展开

SCAR
2019-06-03

1

总结的两问题：
1.oop的四要素：类，对象(实例)，属性，方法(函数)，其中的关系是类是产生对象的工厂，从类产生的对象都继承该类的属性和方法。
2.继承究竟是什么？用三个字表达就是“逆搜索”，意思是实例对象需要的属性或者方法是对象逆着从父类爬取出来的。...
展开

开心
2019-06-03

1

python也有抽像类

展开

Eline

2019-06-06

python3用的是新式类，C3算法计算MRO，
class D(object):
    def __init__(self):
        print('Base D')
...

展开

Geek_00bd9...

2019-06-06

作者您好，引用您的例子，我想问个问题，执行到enter B的时候为什么没有输出enter A，B类不是继承了A类了吗？ 反而输出enter C，然后才输出enter A。这个地方比较疑惑?
class A():
    def __init__(self):...

展开

Fergus

2019-06-05

感谢老师对答案的补充，回头复习看到答案又增加了新知。

展开

羽翼1982

2019-06-05

有一个问题，Java中没有classmethod和staticmethod的区分，感觉python中classmethod能做更多的事情，包括用来实现工厂类，设置和访问类变量等等，为什么python要有staticmethod呢？ 有哪些场景是一定要用staticmethod的呢?