目 录

[XXX课程总结 1](#_Toc446076720)

[第1章 一级标题示例 1](#_Toc446076721)

[1.1 二级标题示例 1](#_Toc446076722)

[1.1.1 三级示例 1](#_Toc446076723)

# .pyc是个什么鬼？

## Python是一门解释型语言？

我初学Python时，听到的关于Python的第一句话就是，Python是一门解释性语言，我就这样一直相信下去，直到发现了\*.pyc文件的存在。如果是解释型语言，那么生成的\*.pyc文件是什么呢？c应该是compiled的缩写才对啊！

为了防止其他学习Python的人也被这句话误解，那么我们就在文中来澄清下这个问题，并且把一些基础概念给理清。

## 解释型语言和编译型语言

计算机是不能够识别高级语言的，所以当我们运行一个高级语言程序的时候，就需要一个“翻译机”来从事把高级语言转变成计算机能读懂的机器语言的过程。这个过程分成两类，第一种是编译，第二种是解释。

编译型语言在程序执行之前，先会通过编译器对程序执行一个编译的过程，把程序转变成机器语言。运行时就不需要翻译，而直接执行就可以了。最典型的例子就是C语言。

解释型语言就没有这个编译的过程，而是在程序运行的时候，通过解释器对程序逐行作出解释，然后直接运行，最典型的例子是Ruby。

通过以上的例子，我们可以来总结一下解释型语言和编译型语言的优缺点，因为编译型语言在程序运行之前就已经对程序做出了“翻译”，所以在运行时就少掉了“翻译”的过程，所以效率比较高。但是我们也不能一概而论，一些解释型语言也可以通过解释器的优化来在对程序做出翻译时对整个程序做出优化，从而在效率上超过编译型语言。

此外，随着Java等基于虚拟机的语言的兴起，我们又不能把语言纯粹地分成解释型和编译型这两种。

用Java来举例，Java首先是通过编译器编译成字节码文件，然后在运行时通过解释器给解释成机器文件。所以我们说Java是一种先编译后解释的语言。

## Python到底是什么

其实Python和Java/C#一样，也是一门基于虚拟机的语言，我们先来从表面上简单地了解一下Python程序的运行过程吧。

当我们在命令行中输入python hello.py时，其实是激活了Python的“解释器”，告诉“解释器”：你要开始工作了。可是在“解释”之前，其实执行的第一项工作和Java一样，是编译。

熟悉Java的同学可以想一下我们在命令行中如何执行一个Java的程序：

javac hello.java

java hello

只是我们在用Eclipse之类的IDE时，将这两部给融合成了一部而已。其实Python也一样，当我们执行python hello.py时，他也一样执行了这么一个过程，所以我们应该这样来描述Python，Python是一门先编译后解释的语言。

## 简述Python的运行过程

在说这个问题之前，我们先来说两个概念，PyCodeObject和pyc文件。

我们在硬盘上看到的pyc自然不必多说，而其实PyCodeObject则是Python编译器真正编译成的结果。我们先简单知道就可以了，继续向下看。

当python程序运行时，编译的结果则是保存在位于内存中的PyCodeObject中，当Python程序运行结束时，Python解释器则将PyCodeObject写回到pyc文件中。

当python程序第二次运行时，首先程序会在硬盘中寻找pyc文件，如果找到，则直接载入，否则就重复上面的过程。

所以我们应该这样来定位PyCodeObject和pyc文件，我们说pyc文件其实是PyCodeObject的一种持久化保存方式。