

# 深入Python运行原理

## 0. 铺垫

### 1. 翻译过程

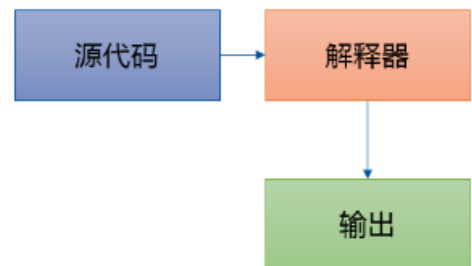
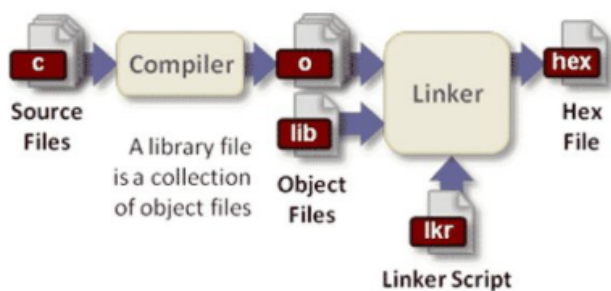
计算机要识别高级语言必须经历翻译这个过程, 该过程分为两类: 编译和解释。

### 2. 编译

在程序运行之前, 通过编译器编译成机器文件, 运行时直接运行即可, 运行效率高无法跨平台, C语言。

### 3. 解释

在程序运行时, 通过解释器将程序逐行做出解释, 运行效率低可以跨平台, Ruby语言。



## 编译器：先整体编译再执行

## 解释器：边解释边执行

## 1. Python运行

### 1. 纠正

准确来讲Python不是一门解释性语言, 是基于虚拟机的语言。

### 2. 过程

源代码---编译器---虚拟机(解释器)---cpu  
先编译后解释。

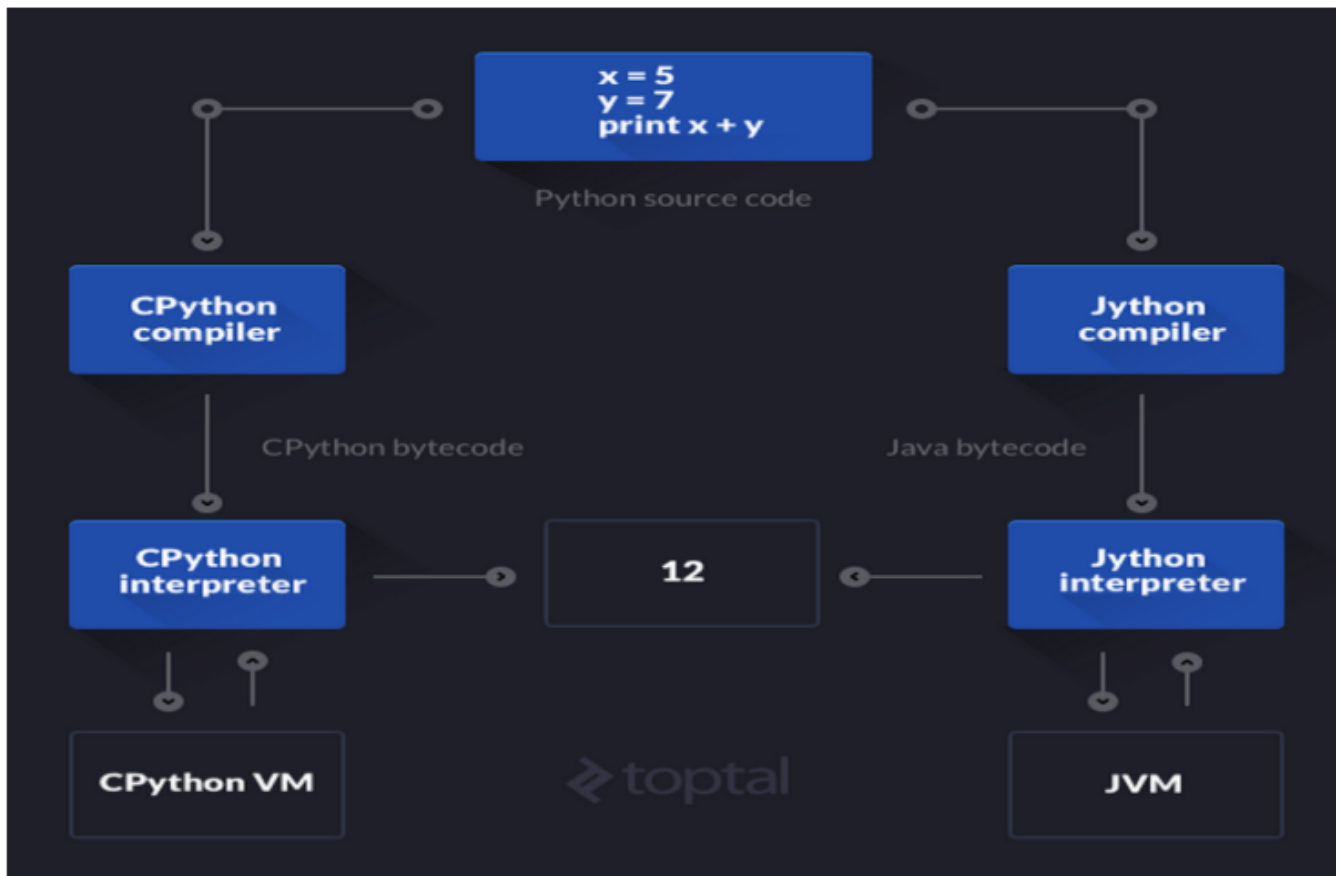
### 3. 剖析

先编译: 编译器真正译成的结果是PyCodeObject, 存在于内存。

后解释: PVM生成可供cpu执行的二进制字节码文件。

结束: 当Python程序运行结束时, Python解释器则将PyCodeObject写回到pyc文件中。

ps: 类似JAVA运行, 但是Python的pyc并不等同于JAVA的字节码!



## 2.pyc文件

### 1. 定位

pyc文件其实是PyCodeObject的一种持久化保存方式。

### 2. 位置

pyc文件存在于\_\_pycache\_\_文件夹中。

### 3. 意义

它先检查本地是否有上述字节码文件(.pyc)和比较时间戳，是就直接运行pyc文件，否则重新编译在解释。是一种程序运行加速机制。

### 4. 查看

```
python -m py_compile hello.py
```

## 3.pyc思考

### 1. pyc目的

要把py文件编译成pyc文件，最大的优点在于我们在运行程序时，不需要重新对该模块进行再次解释。需要编译成pyc文件的应该是那些可以重用的模块，这于我们在设计类时是一样的目的。所以Python的解释器认为：只有import进来的模块，才是需要被重用的模块。

### 2. 生成pyc的标准

对于当前调用的主程序不会生成pyc文件。

以import xxx或from xxx import xxx等方式导入主程序的模块才会生成pyc文件。

如果没有修改，直接使用pyc文件代替模块。

main不会生成pyc文件!!!

**module\_a:**


```
"""  
被导入的模块  
"""  
pass
```

导入

**module\_main:**

```
import module_a  
print("module_a将生成pyc文件!")
```

运行  
结果

名称	修改日期	类型	大小
 module_a.cpython-36.pyc	2017/5/30 9:22	Compiled Pytho...	1 KB

### 3. 是否写入pyc文件

测试:不会生成pyc文件

web文件:不会生成pyc文件,启动服务器,一直监视端口运行即可,把PyCodeObject一直放在内存中就足够了,完全没有必要持久化到硬盘上。

脚本:负责的应该是Model之间的调度,不存在逻辑结构只需要传递修改参数即可,持久化pyc画蛇添足。

\*\*\*Python解释器的意图,Python解释器只是把我们可能重用到的模块持久化成pyc文件。

### 4. pyc文件的过期时间

Import模块的源码,它在写入pyc文件的时候,写了一个整型变量,变量的内容则是文件的最近修改日期,在pyc文件中,每次在载入之前都会检查一下py文件和pyc文件保存的最后修改日期,如果不一致则重新生成新的pyc文件。

## 5.根据pyc启发

从Python解释器的做法上学到一些处理问题的方式和方法:

1. 在Python中判断是否生成pyc文件和我们在设计缓存系统时是一样的,我们可以仔细想想,到底什么是值得扔在缓存里面的,什么是不值得的。
2. 在运行一个耗时的Python脚本时,我们如何能够做到稍微压榨一些程序的运行时间呢?就是将模块从主模块分开,虽然往往这都不是瓶颈。
3. 在设计一个软件系统时,重用和非重用的东西是不是也可以分开来对待,这是软件设计原则的重要部分。
4. 在设计缓存系统(或者其他系统)时,我们如何来避免程序的过期,其实Python解释器为我们提供了一个特别常见而且有效的解决方案。