### 语言类型

**编译型语言**

使用专门的编译器，针对特定的平台，将高级语言源代码一次性的编译成可被该平台硬件执行的机器码，并包装成该平台所能识别的可执行性程序的格式。

特点

在编译型语言写的程序执行之前，需要一个专门的编译过程，把源代码编译成机器语言的文件，如exe格式的文件，以后要再运行时，直接使用编译结果即可，如直接运行exe文件。因为只需编译一次，以后运行时不需要编译，所以编译型语言执行效率高。

总结

一次性的编译成平台相关的机器语言文件，运行时脱离开发环境，运行效率高；

与特定平台相关，一般无法移植到其他平台；(可移植性差，只能在兼容的操作系统上运行)

现有的C、C++、Objective等都属于编译型语言。



解释型语言

使用专门的解释器对源程序逐行解释成特定平台的机器码并立即执行。

特点

解释型语言不需要事先编译，其直接将源代码解释成机器码并立即执行，所以只要某一平台提供了相应的解释器即可运行该程序。

总结

解释型语言每次运行都需要将源代码解释称机器码并执行，效率较低；

只要平台提供相应的解释器，就可以运行源代码，所以可以方便源程序移植；

Python等属于解释型语言。



对比:

编译型语言和解释型语言的区别在于，编译是对高级语言程序进行一次性翻译，这样的好处是，一旦源程序被彻底翻译，它就可以重复运行，且今后都不再需要编译器和源代码；

而解释型语言，则高级语言程序每次运行，都需要借助源程序和解释器，其最大的好处就是，程序有很好的可移植性。(非独立)

编译型程序比解释型程序消耗的内存更少,执行速度更快。

为了运行解释型语言编写的程序，相关的解释器必须首先运行。解释器是复杂的，智能的，大量消耗资源的程序并且它们会占用很多CPU周期和内存。

解释型应用的decode-fetch-execute（解码-抓取-执行）的周期，它们比编译型程序慢很多。

解释器也会做很多代码优化，运行时安全性检查；这些额外的步骤占用了更多的资源并进一步降低了应用的运行速度。

编译型语言：程序在执行之前需要一个专门的编译过程，把程序编译成为机器语言的文件（即exe文件），运行时不需要重新编译，直接用编译后的文件（exe文件）就行了。

优点：执行效率高

缺点：跨平台性差

解释型语言：程序不需要编译，程序在运行的过程中才用解释器编译成机器语言，边编译边执行（没有exe文件）。

优点：跨平台性好

缺点：执行效率低

安全上的对比。

编译型程序不支持代码中实现安全性——例如，一个编译型的程序可以访问内存的任何区域，并且可以对你的PC做它想做的任何事情（大部分病毒是使用编译型语言编写的）,由于松散的安全性和平台依赖性，编译型语言不太适合开发因特网或者基于Web的应用。而解释型语言可以保证高度的安全性,比如解释器在执行过程中会做很多安全性的检查；