# 编程语言的分类（编译型-解释型；动态类型-静态类型；强类型-弱类型）

2017年11月18日 19:57:15 [aiming66](https://me.csdn.net/aiming66" \t "https://blog.csdn.net/aiming66/article/details/_blank) 阅读数：623 标签： [编程语言](http://so.csdn.net/so/search/s.do?q=%E7%BC%96%E7%A8%8B%E8%AF%AD%E8%A8%80&t=blog" \t "https://blog.csdn.net/aiming66/article/details/_blank)[动态语言](http://so.csdn.net/so/search/s.do?q=%E5%8A%A8%E6%80%81%E8%AF%AD%E8%A8%80&t=blog" \t "https://blog.csdn.net/aiming66/article/details/_blank)[计算机](http://so.csdn.net/so/search/s.do?q=%E8%AE%A1%E7%AE%97%E6%9C%BA&t=blog" \t "https://blog.csdn.net/aiming66/article/details/_blank)[软考](http://so.csdn.net/so/search/s.do?q=%E8%BD%AF%E8%80%83&t=blog" \t "https://blog.csdn.net/aiming66/article/details/_blank) 更多

个人分类： [python](https://blog.csdn.net/aiming66/article/category/7295893" \t "https://blog.csdn.net/aiming66/article/details/_blank)

所属专栏： [python](https://blog.csdn.net/column/details/21067.html" \t "https://blog.csdn.net/aiming66/article/details/_blank)

版权声明：本文为博主原创文章，未经博主允许不得转载。本人观点或有不当之处，请在评论中及时指正，我会在第一时间内修改。 https://blog.csdn.net/aiming66/article/details/78570784

* + [1高级语言和低级语言](https://blog.csdn.net/aiming66/article/details/78570784" \l "1高级语言和低级语言" \t "https://blog.csdn.net/aiming66/article/details/_self)
  + [2编译型和解释型](https://blog.csdn.net/aiming66/article/details/78570784" \l "2编译型和解释型" \t "https://blog.csdn.net/aiming66/article/details/_self)
  + [3动态语言和静态语言](https://blog.csdn.net/aiming66/article/details/78570784" \l "3动态语言和静态语言" \t "https://blog.csdn.net/aiming66/article/details/_self)
  + [4强类型定义语言和弱类型定义语言](https://blog.csdn.net/aiming66/article/details/78570784" \l "4强类型定义语言和弱类型定义语言" \t "https://blog.csdn.net/aiming66/article/details/_self)

　　在软考期间，学习了程序语言的翻译方式有多种：解释、和编译。那到底什么是解释和编译呢。另外编程语言除了编译型和解释型，还有静态语言、动态语言、强类型语言、弱类型语言。那么每种语言类型是什么样子的呢，下面让我们来看看吧。   
　　但是在此之前，我们需要先了解一下计算机原因的发展史，从低级语言到高级语言的转变。

## **1、高级语言和低级语言：**

　　最初的计算机程序都是用0和1的序列表示的，程序员直接使用的是机器指令，无需翻译，从纸带打孔输入即可执行得到结果。后来为了方便记忆，就将用0、1序列表示的机器指令都用符号助记，这些与机器指令一一对应的助记符就成了汇编指令，从而诞生了汇编语言。无论是机器指令还是汇编指令都是面向机器的，统称为****低级语言****。因为是针对特定机器的机器指令的助记符，所以汇编语言是无法独立于机器(特定的CPU体系结构)的。但汇编语言也是要经过翻译成机器指令才能执行的，所以也有将运行在一种机器上的汇编语言翻译成运行在另一种机器上的机器指令的方法，那就是交叉汇编技术。

　　高级语言是从人类的逻辑思维角度出发的计算机语言，抽象程度大大提高，需要经过编译成特定机器上的目标代码才能执行，一条高级语言的语句往往需要若干条机器指令来完成。高级语言独立于机器的特性是靠编译器为不同机器生成不同的目标代码(或机器指令)来实现的。那具体的说，要将高级语言编译到什么程度呢，这又跟编译的技术有关了，既可以编译成直接可执行的目标代码，也可以编译成一种中间表示，然后拿到不同的机器和系统上去执行，这种情况通常又需要支撑环境，比如解释器或虚拟机的支持，Java程序编译成bytecode，再由不同平台上的虚拟机执行就是很好的例子。所以，说高级语言不依赖于机器，是指在不同的机器或平台上高级语言的程序本身不变，而通过编译器编译得到的目标代码去适应不同的机器。从这个意义上来说，通过交叉汇编，一些汇编程序也可以获得不同机器之间的可移植性，但这种途径获得的移植性远远不如高级语言来的方便和实用性大。

　　大致了解了以上的内容，下面我们正式解释各种语言类型就简单多了。

## **2）编译型和解释型**

　　我们先看看编译型，其实它和汇编语言是一样的：也是有一个负责翻译的程序来对我们的源代码进行转换，生成相对应的可执行代码。这个过程说得专业一点，就称为编译（Compile），而负责编译的程序自然就称为编译器（Compiler）。如果我们写的程序代码都包含在一个源文件中，那么通常编译之后就会直接生成一个可执行文件，我们就可以直接运行了。但对于一个比较复杂的项目，为了方便管理，我们通常把代码分散在各个源文件中，作为不同的模块来组织。这时编译各个文件时就会生成目标文件（Object file）而不是前面说的可执行文件。一般一个源文件的编译都会对应一个目标文件。这些目标文件里的内容基本上已经是可执行代码了，但由于只是整个项目的一部分，所以我们还不能直接运行。待所有的源文件的编译都大功告成，我们就可以最后把这些半成品的目标文件“打包”成一个可执行文件了，这个工作由另一个程序负责完成，由于此过程好像是把包含可执行代码的目标文件连接装配起来，所以又称为链接（Link），而负责链接的程序就叫……就叫链接程序链接程序（Linker）。链接程序除了链接目标文件外，可能还有各种资源，像图标文件啊、声音文件啊什么的，还要负责去除目标文件之间的冗余重复代码，等等，所以……也是挺累的。链接完成之后，一般就可以得到我们想要的可执行文件了。

　　现在再看看解释型。噢，从字面上看，“编译”和“解释”的确都有“翻译”的意思，它们的区别则在于翻译的时机安排不大一样。打个比方：假如你打算阅读一本外文书，而你不知道这门外语，那么你可以找一名翻译，给他足够的时间让他从头到尾把整本书翻译好，然后把书的母语版交给你阅读；或者，你也立刻让这名翻译辅助你阅读，让他一句一句给你翻译，如果你想往回看某个章节，他也得重新给你翻译。

　　两种方式，前者就相当于我们刚才所说的编译型：一次把所有的代码转换成机器语言，然后写成可执行文件；而后者就相当于我们要说的解释型：在程序运行的前一刻，还只有源程序而没有可执行程序；而程序每执行到源程序的某一条指令，则会有一个称之为解释程序的外壳程序将源代码转换成二进制代码以供执行，总言之，就是不断地解释、执行、解释、执行……所以，解释型程序是离不开解释程序的。像早期的BASIC就是一门经典的解释型语言，要执行BASIC程序，就得进入BASIC环境，然后才能加载程序源文件、运行。解释型程序中，由于程序总是以源代码的形式出现，因此只要有相应的解释器，移植几乎不成问题。编译型程序虽然源代码也可以移植，但前提是必须针对不同的系统分别进行编译，对于复杂的工程来说，的确是一件不小的时间消耗，况且很可能一些细节的地方还是要修改源代码。而且，解释型程序省却了编译的步骤，修改调试也非常方便，编辑完毕之后即可立即运行，不必像编译型程序一样每次进行小小改动都要耐心等待漫长的Compiling…Linking…这样的编译链接过程。不过凡事有利有弊，由于解释型程序是将编译的过程放到执行过程中，这就决定了解释型程序注定要比编译型慢上一大截，像几百倍的速度差距也是不足为奇的。

　　编译型与解释型，两者各有利弊。前者由于程序执行速度快，同等条件下对系统要求较低，因此像开发操作系统、大型应用程序、数据库系统等时都采用它，像C/C++、Pascal/Object Pascal（Delphi）、VB等基本都可视为编译语言，而一些网页脚本、服务器脚本及辅助开发接口这样的对速度要求不高、对不同系统平台间的兼容性有一定要求的程序则通常使用解释性语言，如Java、JavaScript、VBScript、Perl、Python等等。

　　但既然编译型与解释型各有优缺点又相互对立，所以一批新兴的语言都有把两者折衷起来的趋势，例如Java语言虽然比较接近解释型语言的特征，但在执行之前已经预先进行一次预编译，生成的代码是介于机器码和Java源代码之间的中介代码，运行的时候则由JVM（Java的虚拟机平台，可视为解释器）解释执行。它既保留了源代码的高抽象、可移植的特点，又已经完成了对源代码的大部分预编译工作，所以执行起来比“纯解释型”程序要快许多。而像VB6（或者以前版本）、C#这样的语言，虽然表面上看生成的是.exe可执行程序文件，但VB6编译之后实际生成的也是一种中介码，只不过编译器在前面安插了一段自动调用某个外部解释器的代码（该解释程序独立于用户编写的程序，存放于系统的某个DLL文件中，所有以VB6编译生成的可执行程序都要用到它），以解释执行实际的程序体。C#（以及其它.net的语言编译器）则是生成.net目标代码，实际执行时则由.net解释系统（就像JVM一样，也是一个虚拟机平台）进行执行。当然.net目标代码已经相当“低级”，比较接近机器语言了，所以仍将其视为编译语言，而且其可移植程度也没有Java号称的这么强大，Java号称是“一次编译，到处执行”，而.net则是“一次编码，到处编译”。呵呵，当然这些都是题外话了。总之，随着设计技术与硬件的不断发展，编译型与解释型两种方式的界限正在不断变得模糊。

## **3）动态语言和静态语言**

　　通常我们所说的动态语言、静态语言是指动态类型语言和静态类型语言。   
1）动态类型语言：   
　　动态类型语言是指在运行期间才去做数据类型检查的语言，也就是说，在用动态类型的语言编程时，永远也不用给任何变量指定数据类型，该语言会在你第一次赋值给变量时，在内部将数据类型记录下来。Python和Ruby就是一种典型的动态类型语言，其他的各种脚本语言如VBScript也多少属于动态类型语言。   
2）静态类型语言：   
　　静态类型语言与动态类型语言刚好相反，它的数据类型是在编译其间检查的，也就是说在写程序时要声明所有变量的数据类型，C/C++是静态类型语言的典型代表，其他的静态类型语言还有C#、JAVA等。

## **4）强类型定义语言和弱类型定义语言**

1）强类型定义语言：   
　　强制数据类型定义的语言。也就是说，一旦一个变量被指定了某个数据类型，如果不经过强制转换，那么它就永远是这个数据类型了。举个例子：如果你定义了一个整型变量a,那么程序根本不可能将a当作字符串类型处理。强类型定义语言是类型安全的语言。   
2）弱类型定义语言：   
　　数据类型可以被忽略的语言。它与强类型定义语言相反, 一个变量可以赋不同数据类型的值。   
　　强类型定义语言在速度上可能略逊色于弱类型定义语言，但是强类型定义语言带来的严谨性能够有效的避免许多错误。另外，“这门语言是不是动态语言”与“这门语言是否类型安全”之间是完全没有联系的！   
　　例如：Python是动态语言，是强类型定义语言（类型安全的语言）; VBScript是动态语言，是弱类型定义语言（类型不安全的语言）; JAVA是静态语言，是强类型定义语言（类型安全的语言）。