语言类型

一、编译型与解释型

1、编译型语言

- 定义：

> 在程序运行之前，**通过编译器将源程序编译成机器码(可运行的二进制代码)**，以后执行这个程序时，就不用再进行编译了。

- 优点：

> 编译器一般会有**预编译的过程**对代码进行优化。**因为编译只做一次，运行时不需要编译，所以编译型语言的程序执行效率高可以脱离语言环境独立运行**

- 缺点：

> **编译之后如果需要修改就需要整个模块重新编译。**编译的时候根据对应的运行环境生成机器码，**不同的操作系统之间移植就会有问题**，需要根据运行的操作系统环境编译不同的可执行文件。

- 总结：

> **执行速度快、效率高；依靠编译器、跨平台性差些。**

- 代表语言：

> C、C++、Pascal、Object-C以及Swift。

2、解释型语言

- 定义：

> **解释型语言的源代码不是直接翻译成机器码，而是先翻译成中间代码，再由解释器对中间代码进行解释运行。在运行的时候才将源程序翻译成机器码，翻译一句，然后执行一句，直至结束。**

- 优点：

> **有良好的平台兼容性，在任何环境中都可以运行**，前提是安装了解释器（虚拟机）。灵活，修改代码的时候直接修改就可以，**可以快速部署，不用停机维护。**

- 缺点：

> **每次运行的时候都要解释一遍，性能上不如编译型语言。**

- 总结：

> **执行速度慢、效率低；依靠解释器、跨平台性好。**

- 代表语言：

> JavaScript、Python、Erlang、PHP、Perl、Ruby。

3、混合型语言

- 定义：

> 既然编译型和解释型各有缺点就会有人想到把两种类型整合起来，取其精华去其糟粕，就出现了半编译，半解释型语言。

- 概述：

> 比如C#，C#在编译的时候不是直接编译成机器码而是中间码，.NET平台提供了中间语言运行库运行中间码，中间语言运行库类似于Java虚拟机。.NET在编译成IL代码后，保存在dll中，首次运行时由JIT在编译成机器码缓存在内存中，下次直接执行。严格来说混合型语言属于解释型语言，C#更接近编译型语言。

- **Java即是编译型的，也是解释型语言，总的来说Java更接近解释型语言。**
  - 可以说它是`编译型`的。因为所有的Java代码都是要编译的，.java不经过编译就什么用都没有。同时围绕JVM的效率问题，会涉及一些如JIT、AOT等优化技术，例如**JIT技术，会将热点代码编译成机器码。而AOT技术，是在运行前，通过工具直接将字节码转换为机器码。**
  - 可以说它是`解释型`的。因为Java代码编译后不能直接运行，它是解释运行在JVM上的，所以它是解释运行的。

二、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`：**在运行期间才去做数据类型检查的语言**，说的是数据类型。动态类型语言的数据类型不是在编译阶段决定的，而是把类型绑定延后到了运行阶段。
>
> `代表语言`：Python、Ruby、Erlang、JavaScript、Swift、PHP、Perl。

2、静态类型语言

> `静态类型语言`：**数据类型是在编译期间（或运行之前）确定的，编写代码的时候要明确确定变量的数据类型。**
>
> `代表语言`：C、C++、C#、Java、Object-C。

三、动态语言和静态语言

1、动态语言

动态类型语言和动态语言是完全不同的两个概念

> `动态语言`：**说的是运行时改变结构，说的是代码结构。**在运行时可以改变其结构的语言：例如新的函数、对象、甚至代码可以被引进，已有的函数可以被删除或是其他结构上的变化。通俗点说就是在**运行时代码可以根据某些条件改变自身结构。**
>
> `代表语言`：Object-C、C#、JavaScript、PHP、Python、Erlang。

2、静态语言

> 静态语言：与动态语言相对应的，**运行时结构不可变的语言就是静态语言。**
>
> 代表语言： Java、C、C++。