CLR(Common Language RunTime)——公共语言进行时，是一个可由多种编程语言使用的运行时。一个类似于JVM的虚拟机。

CLR的核心功能：内存管理，程序集加载，安全性，异常处理，线程同步。面向CLR的所有语言都能使用这些功能。并且“运行时”允许创建线程，因此面向它的所有语言都能创建线程。

CLR面向许多语言，正因如此可以在开发时选择最适合的语言进行开发，会节省许多开发时间。

CLR的执行模型：

1.将源代码编译成托管模块

2.将托管模块合并成程序集

3.加载公共语言运行时

4.执行程序集代码

1.将源代码编译成托管模块（源码编译过程）

什么是托管模块？托管模块是由PE(Portable Executable,可以指执行体)头，CLR头，元数据，IL代码组成。

文本

中度可信度描述已自动生成

面向CLR的编译器做的工作是在每个托管模块中生成完整的元数据，把代码编译为IL代码。

什么是IL代码？IL代码也称托管代码，是一种类似汇编语言的“中间语言”，可以将其理解为伪汇编语言。使用CLR支持的语言编写的程序被编译为IL中间语言，通过中间语言能够实现跨平台移植提高程序的灵活性。IL是与CPU无关的机器语言。

元数据总是和IL代码的文件关联。由于编译器同时生成元数据和代码，他们绑定在一起嵌入最终生成的托管模块，所以元数据和IL代码永远不会失去同步。

图示

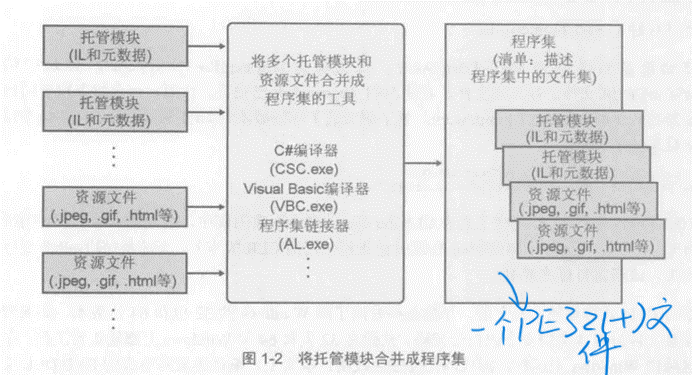
描述已自动生成

关于PE头[(74条消息) PE文件结构详解 --（完整版）\_「已注销」的博客-CSDN博客](https://blog.csdn.net/lyshark_lyshark/article/details/125847139?utm_medium=distribute.pc_relevant.none-task-blog-2~default~baidujs_baidulandingword~default-0-125847139-blog-11903197.pc_relevant_3mothn_strategy_and_data_recovery&spm=1001.2101.3001.4242.1&utm_relevant_index=3)

2.将托管模块合成程序集

什么是程序集？程序集是一个或多个(托管)模块/资源文件的逻辑性分组，CLR实际不直接和托管模块工作，实际是和程序集工作，程序集是重用，安全性以及版本控制的最小单元，在CLR中，程序集就相当于“组件”。利用Assembly，一组文件可以作为一个实体看待。

程序集和命名空间的区别？命名空间用于对类型进行逻辑分组。程序集则是程序的物理分组，对应于一个dll或exe文件。



编译器默认将生成的托管模块转换为程序集，那么其实编译器生成的是一个含有清单的托管模块。那么对于一个无资源文件的仅有一个托管模块的文件项目，那么程序集就是托管模块，生成程序集的过程也不需要执行任何其他的步骤。

3.加载公共语言运行时

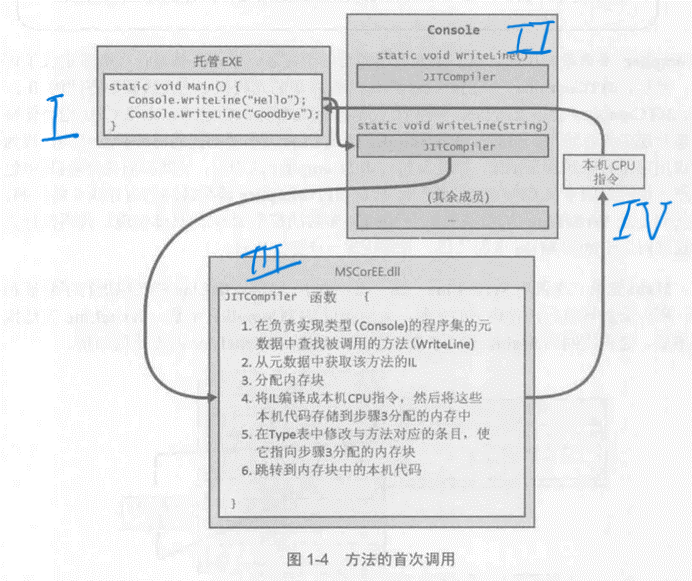
图示

描述已自动生成

用户计算机执行exe文件时，检查exe文件头，决定创建32位进程还是64位进程，随后在进程地址空间加载MSCorEE.dll的x86，x64或者ARM版本，然后进程的主线程调用MSCorEE.dll中定义的一个方法，这个方法初始化CLR，加载exe程序集，再调用程序集中的入口方法（main），由此exe启动完成并开始运行。

4.执行程序集代码（执行一条代码的经历的过程）

为了执行程序集的代码，首先需要把方法的IL代码转换为本机CPU指令，这是CLR的JIT编译器的职责，



图示

描述已自动生成

执行步骤：

1.在Main方法执行之前，CLR会检测出Main的代码引用的所有类型（引用类型和值类型）。使CLR分配一个内部数据结构（栈，IL代码基于栈）来管理对引用类型的访问（II）。例如在这个数据结构中，Console类定义的每个方法都有一条对应的记录项（entry，即函数入口），每条记录项都含有一个地址，根据此地址即可找到方法的实现。对这个结构初始化时，CLR将每个记录项都设置成（指向）包含在CLR内部的一个未编档函数，我们称之为JITCompiler。

2.Main方法首次调用WriteLine时，JITCompiler函数被调用，执行III中的操作。

3.代码执行完毕后，回到Main中，继续执行下一条语句。

由于步骤2中已经对WriteLine的代码进行了验证和编译，所以会直接执行内存块中的代码，完全跳过JITCompiler函数。

由以上步骤可知：

1.方法仅在第一次被调用时才有一些性能损失。以后的调用都以本机代码的形式全速运行。

2.JIT编译器将得到的将本机CPU指令保存在动态内存中。也就是说，一旦应用程序关闭，编译好的代码也会丢失。也就是说，再次运行应用程序时JIT编译器需要把IL代码再次编译为本机CPU指令。

托管代码和非托管代码的区别？

1.什么是非托管代码？非托管代码，直接编译成目标机器码。不在CLR上运行（也就是没有编译成IL代码），在公共语言运行库环境的外部，由操作系统直接执行的代码，代码必须自己提供垃圾回收，类型检查，安全支持等服务（如需要内存管理等服务，必须显示调用操作系统的接口，通常调用Windows SDK所提供的API来实现内存管理）。非托管代码的运行速度比托管代码运行速度快。

2.两者区别：1、托管代码是一种中间语言，运行在CLR上；

非托管代码被编译为机器码，运行在机器上。2、托管代码独立于平台和语言，能更好的实现不同语言平台之间的兼容；非托管代码依赖于平台和语言。3、托管代码可享受CLR提供的服务（如安全检测、垃圾回收等），不需要自己完成这些操作；非托管代码需要自己提供安全检测、垃圾回收等操作。

IL和验证：IL基于栈并且是无类型的。将IL编译成本机CPU指令时，CLR会执行一个名为“验证”的过程。这个过程会检查IL高级IL代码，确定代码所做的操作都是安全的。（当然也可以编写不安全代码，直接操作内存地址和地址存储的字节）

使用托管代码的优点：

1，托管代码会将所有Windows进程放入独立的地址空间，将获得健壮性和稳定性，一个进程干扰不到另一个进程；

2，但如果进程数量太多，会损害性能并制约可用的资源，而托管代码可以用一个进程运行多个应用程序，以此减少进程数，从而增强性能，减少所需的资源

公共语言规范(CLS)

CLS定义了所有语言都必须支持的最小功能集。

CLR/CTS、各种语言和CLS的关系图

图示

描述已自动生成