托管的执行过程包括以下步骤，在本主题后面将对此进行详细讨论：

1. 选择编译器。

若要获取公共语言运行时提供的好处，必须使用一个或多个面向运行时的语言编译器。

2. 将代码编译为 MSIL。

编译将你的源代码转换为 Microsoft 中间语言 (MSIL) 并生成必需的元数据。

3. 将 MSIL 编译为本机代码。

在执行时，实时 (JIT) 编译器将 MSIL 转换为本机代码。 在此编译期间，代码必须通过检查 MSIL 和元数据的验证过程以查明是否可以将代码确定为类型安全。

4. 运行代码。

公共语言运行时提供启用要发生的执行的基础结构以及执行期间可使用的服务。

选择编译器

若要获取公共语言运行时 (CLR) 提供的好处，必须使用一个或多个面向运行时的语言编译器，如 Visual Basic、C#、Visual C++、F# 或众多第三方编译器之一，如 Eiffel、Perl 或 COBOL 编译器。 **因为它是一个多语言的执行环境，运行时支持各种各样的数据类型和语言功能。 你使用的语言编译器确定哪些运行时功能是可用的，并且可以使用这些功能设计你的代码。 编译器（而非运行时）设定代码必须使用的语法（我所理解的编译器其实就是所选择的语言，也就决定了所能使用的语法。不同的编译器，要使用不同语法来实现对应的功能）。** 如果组件必须完全可供以其他语言编写的组件使用，则组件的导出类型必须仅公开 Language Independence and Language-Independent Components (CLS) 中包含的语言功能。 可以使用 CLSCompliantAttribute 属性来确保你的代码符合 CLS。 有关详细信息，请参阅 “语言独立性和与语言无关的组件”。

返回页首

编译为 MSIL

编译为托管代码时，编译器将源代码转换为 Microsoft 中间语言 (MSIL)，这是一组独立于 CPU 且可以有效地转换为本机代码的说明。 MSIL 包括有关加载、存储、初始化和调用对象方法的说明，以及有关算术和逻辑运算、控制流、直接内存访问、异常处理和其他操作的说明。 代码可以运行之前，必须将 MSIL 转换为特定于 CPU 的代码，通常通过 实时 (JIT) 编译器实现。 由于公共语言运行时为其支持的每个计算机基础结构提供一个或多个 JIT 编译器，同一组的 MSIL 可以在任何受支持的基础结构上进行 JIT 编译和运行。

当编译器生成 MSIL 时，它还生成元数据。 元数据描述**代码中的类型**，包括每种**类型的定义**、**每种类型的成员的签名**、**代码引用的成员**以及**运行时在执行时间使用的其他数据。** MSIL 和元数据包含在一个可移植的可执行 (PE) 文件中，该文件基于且扩展已发布的 Microsoft PE 和历来用于可执行内容的通用对象文件格式 (COFF)。 容纳 MSIL 或本机代码以及元数据的这种文件格式使操作系统能够识别公共语言运行时映像。 **文件中元数据的存在以及 MSIL 使代码能够描述自身，这意味着将不需要类型库或接口定义语言 (IDL)。 运行时在执行期间会根据需要从文件中查找并提取元数据。**

返回页首

将 MSIL 编译为本机代码

运行 Microsoft 中间语言 (MSIL) 前，必须根据公共语言运行时将其编译为目标计算机基础结构的本机代码。 .NET 提供两种方法来执行此转换：

- .NET 实时 (JIT) 编译器。
- Ngen.exe（本机映像生成器）。

由 JIT 编译器编译

在加载和执行程序集的内容时，JIT 编译在应用程序运行时按需将 MSIL 转换为本机代码。 由于**公共语言运行时为每个受支持的 CPU 基础结构提供 JIT 编译器**，开发人员可以构建一组 MSIL 程序集，这些程序集可以进行 JIT 编译并可在具有不同计算机基础结构的不同计算机上运行。 但是，如果**你的托管代码调用特定于平台的本机 API 或特定于平台的类库，它将仅在该操作系统上运行。**

JIT 编译将执行期间可能永远不会调用的某些代码的可能性考虑在内。 它根据需要在执行期间转换 MSIL，而不是使用时间和内存来将 PE 文件中所有 MSIL 转换为本机代码，并在内存中存储生成的本机代码，以便该进程上下文中的后续调用可以对其进行访问。 加载类型并将其初始化时，加载程序创建并将存根附加到类型中的每个方法。 第一次调用某个方法时，存根将控件传递给 JIT 编译器，后者将该方法的 MSIL 转换为本机代码，并将存根修改为直接指向生成的本机代码。 因此，对 JIT 编译的方法的后续调用会直接转到本机代码。

使用 NGen.exe 的安装时代码生成

由于在调用该程序集中定义的各个方法时，JIT 编译器将程序集的 MSIL 转换为本机代码，因此它在运行时中对性能产生负面影响。 在大多数情况下，这种性能降低的程度是可以接受的。 更为重要的是，**由 JIT 编译器生成的代码会绑定到触发编译的进程上。** 它无法在多个进程之间进行共享。 若要允许生成的代码跨应用程序的多个调用或跨共享一组程序集的多个进程进行共享，则公共语言运行时支持预编译模式。 这种预编译模式使用 Ngen.exe（本机映像生成器）将 MSIL 程序集转换为本机代码，非常类似 JIT 编译器执行的操作。 但是，Ngen.exe 的操作在三个方面不同于 JIT 编译器的操作：

- 它在运行应用程序之前而非运行该应用程序时，将 MSIL 转换为本机代码。
- 它一次编译整个程序集，而不是一次编译一种方法。
- 它将本机映像缓存中生成的代码作为磁盘上的文件保存。

代码验证

作为其编译为本机代码的一部分，MSIL 代码必须通过验证过程，除非管理员已经设定允许代码忽略验证的安全策略（可以通过代码控制来让编译器跳过代码检查）。 验证过程检查 MSIL 和元数据，以找出**代码是否为类型安全**，这意味着**它仅访问其有权访问的内存位置。 类型安全有助于将对象相互隔离，并帮助保护它们免受无意或恶意损坏。** 它还保障**可以可靠地对代码强制执行安全限制。**

运行时基于以下语句对于可验证类型安全代码为 true 这一事实：

- **对类型的引用严格符合所引用的类型。**
- **在对象上只调用正确定义的操作。**
- **标识与声称的要求一致。**

在验证过程中，检查 MSIL 代码以试图确认代码可以访问内存位置，并仅通过正确定义的类型调用方法。 例如，代码不允许以允许内存位置溢出的方式访问对象的字段。 此外，验证检查代码以确定是否已正确生成 MSIL，因为错误的 MSIL 可能会导致违反类型安全规则。 验证过程传递一组定义完善的

类型安全代码，并且仅传递类型安全的代码。 但是，由于验证过程的一些限制，并且按照设计，某些语言不生成可验证的类型安全代码，某些类型安全代码可能无法通过验证。 如果安全策略要求提供类型安全代码，而该代码不能通过验证，则在运行该代码时将引发异常。

运行代码

公共语言运行时提供启用要发生的托管执行的基础结构以及执行期间可使用的服务。 方法可以运行之前，必须编译为特定于处理器的代码。 **当第一次调用，然后运行时，为其生成 MSIL 的每种方法都是 JIT 编译的。** 下次运行该方法时，将运行现有的 JIT 编译的本机代码。 重复 JIT 编译，然后运行代码的过程，直到执行完毕。

在执行期间，托管代码接收服务，如垃圾收集、安全性、与非托管代码的互操作性、跨语言调试支持以及增强的部署和版本控制支持。

在 Microsoft Windows Vista 中，操作系统加载程序通过检查 COFF 标头中的一个位检查托管模块。 所设置的位表示托管模块。 如果加载程序检测到托管模块，它将加载 mscoree.dll， `_CorValidateImage` 并且 `_CorImageUnloading` 在加载和卸载托管模块映像时通知加载程序。 `_CorValidateImage` 执行以下操作：

1. 确保代码是有效的托管代码。
2. 将映像中的入口点更改为运行时中的入口点。

在 64 位 Windows 上， `_CorValidateImage` 通过将其从 PE32 转换为 PE32+ 格式修改内存中的映像。