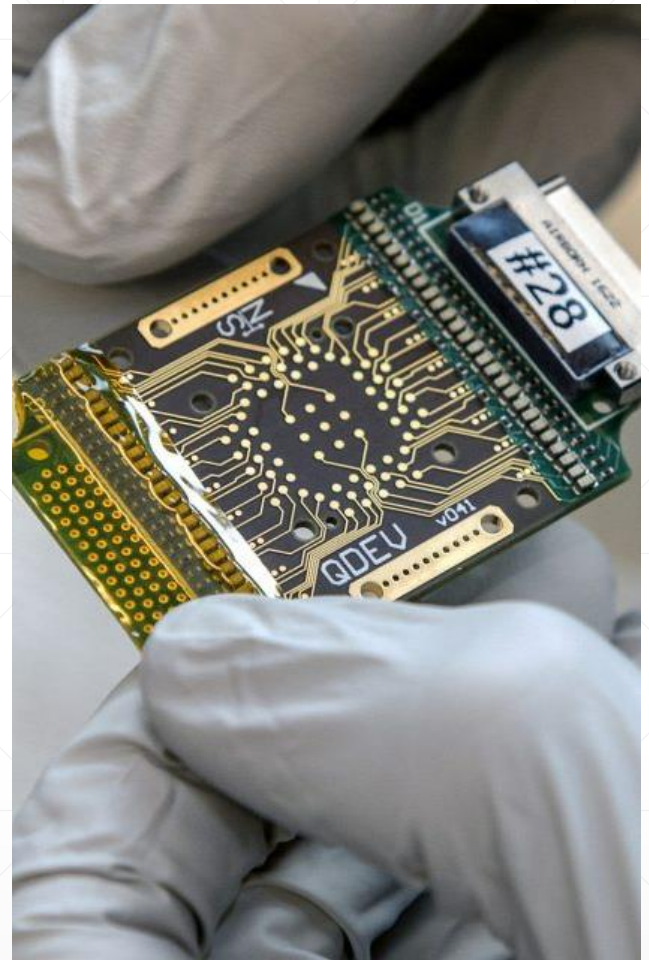


# .NET虚拟机工作原理

---

北京理工大学计算机学院  
金旭亮

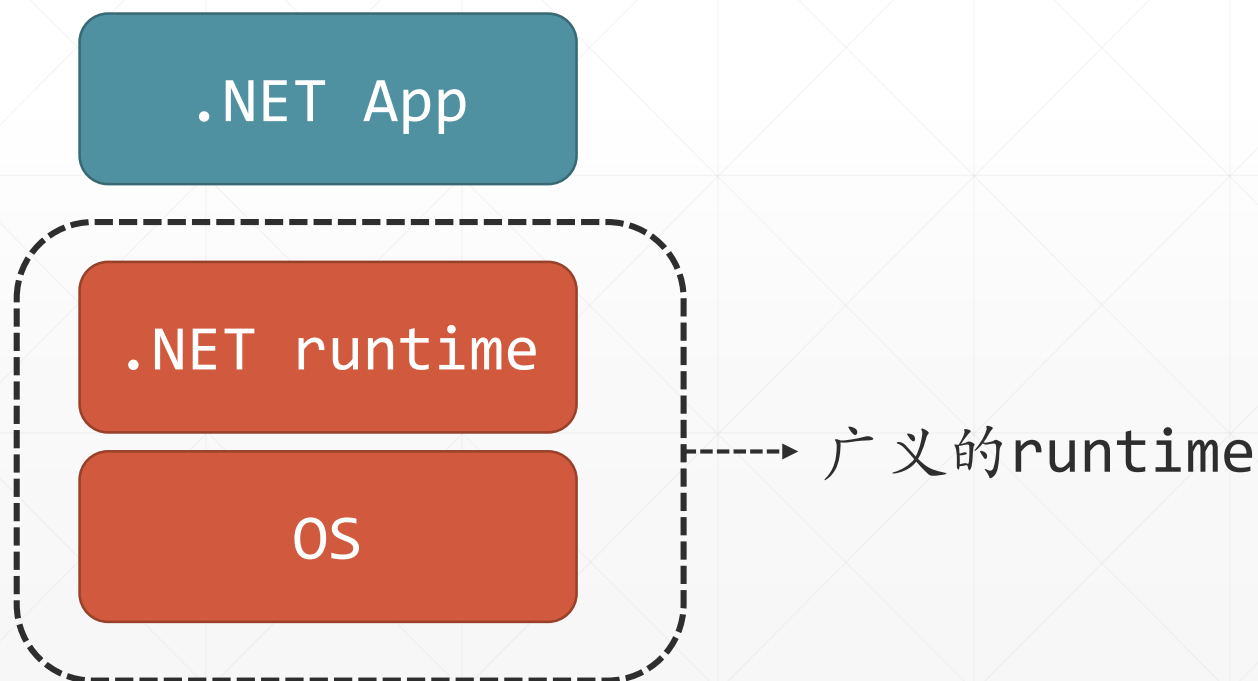


# 理解“运行时 (runtime)”的概念

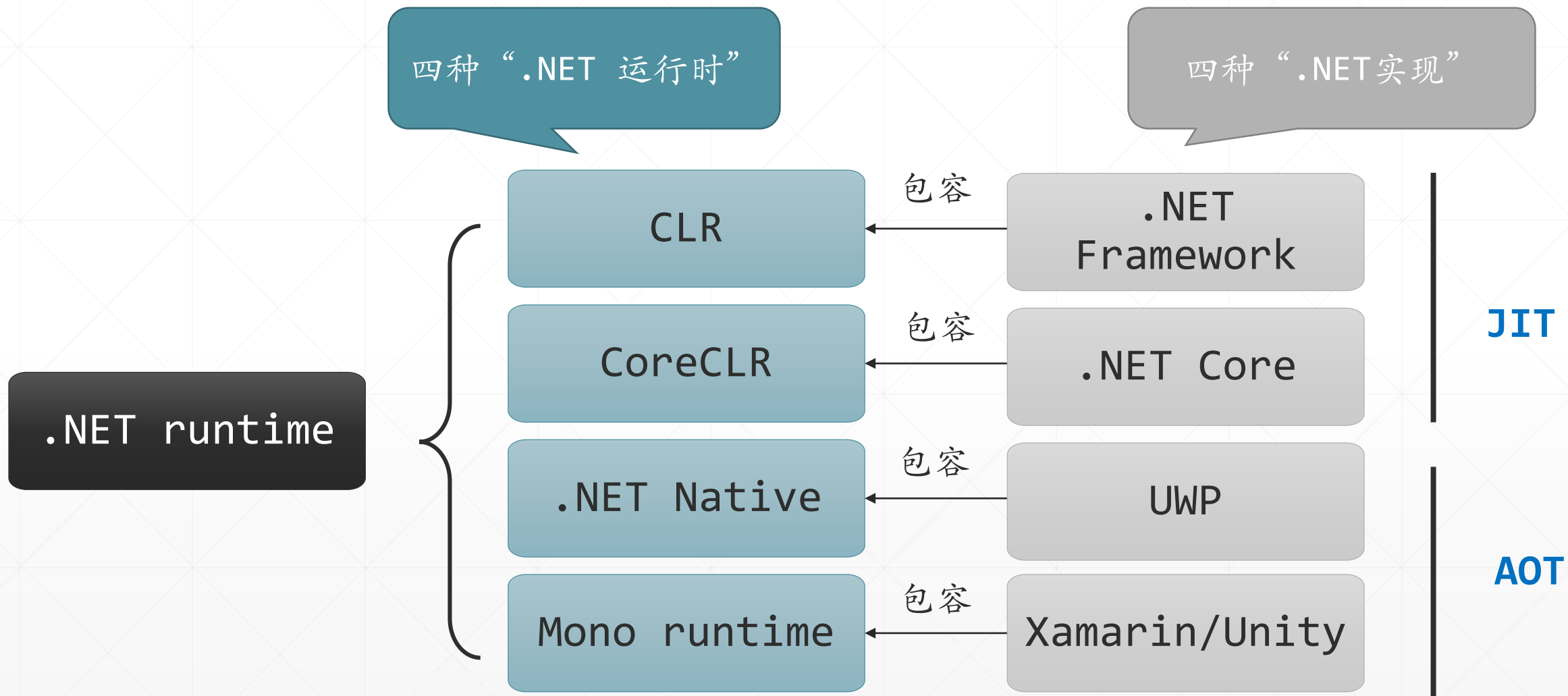


A runtime is the execution environment for a managed program.

.NET Runtime  $\Leftrightarrow$  .NET 虚拟机



# “.NET运行时”与“.NET实现”之间的关系



# .NET运行时（JIT模式）的主要职责

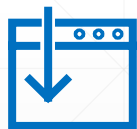
## 主要职责



内存动态分配与智能回收



混合语言开发

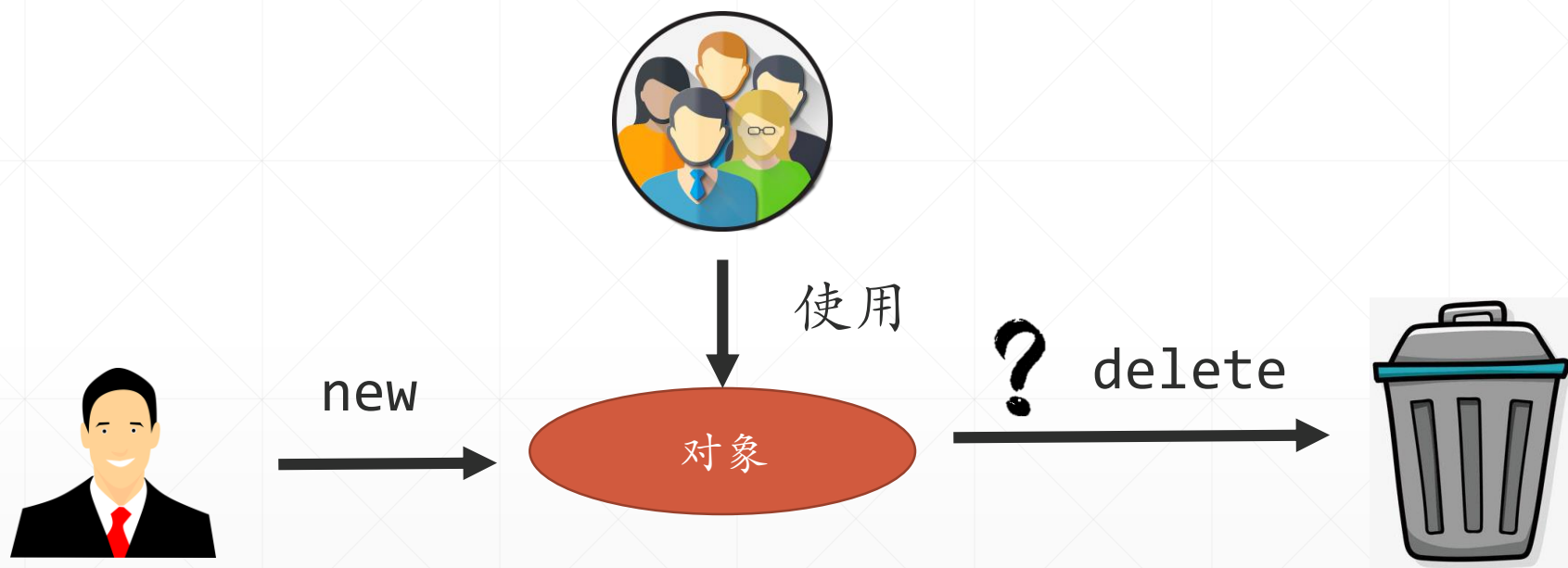


平台调用

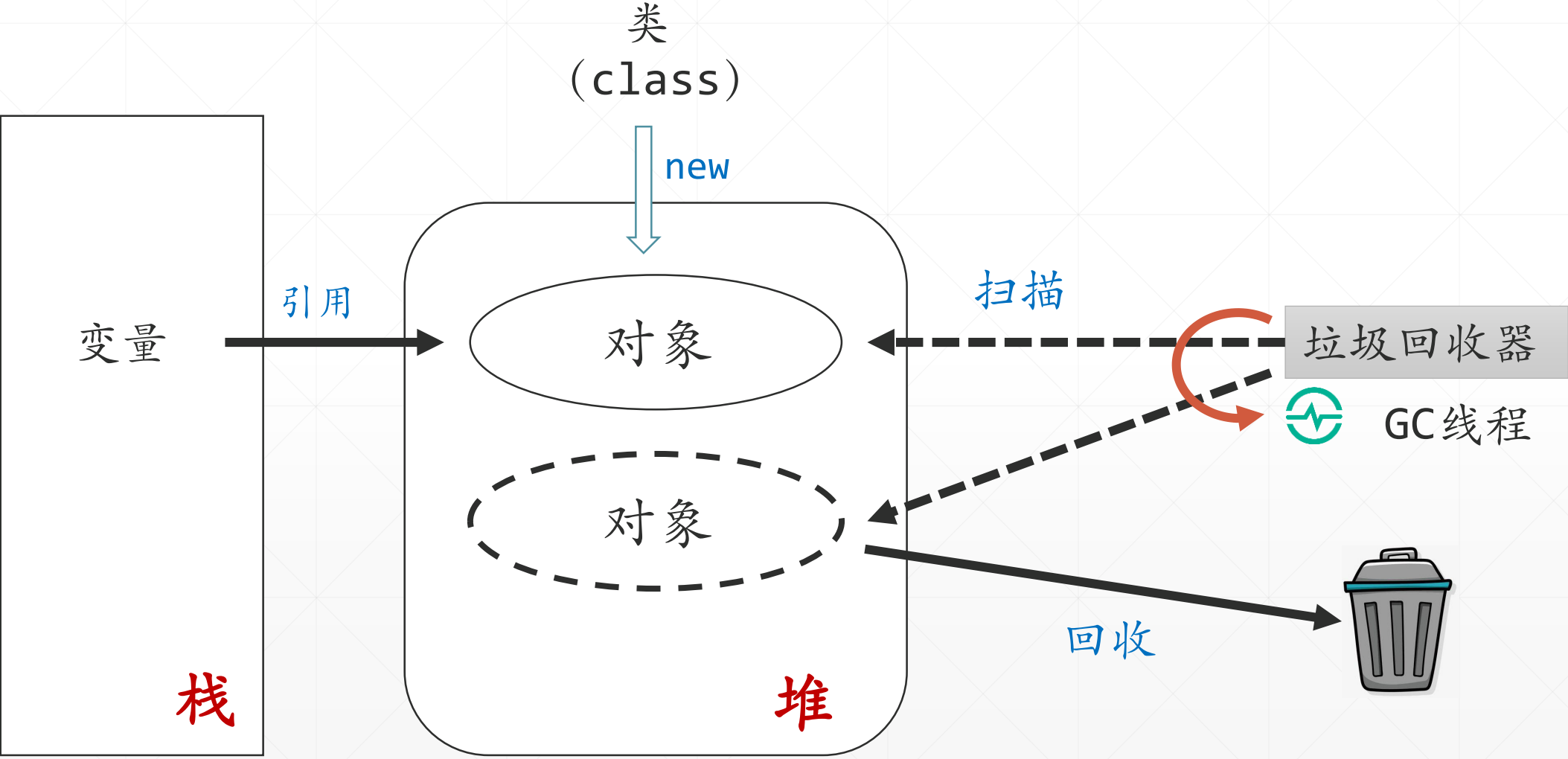


使用JIT编译器将程序集中的IL代码转换为机器码，并执行之

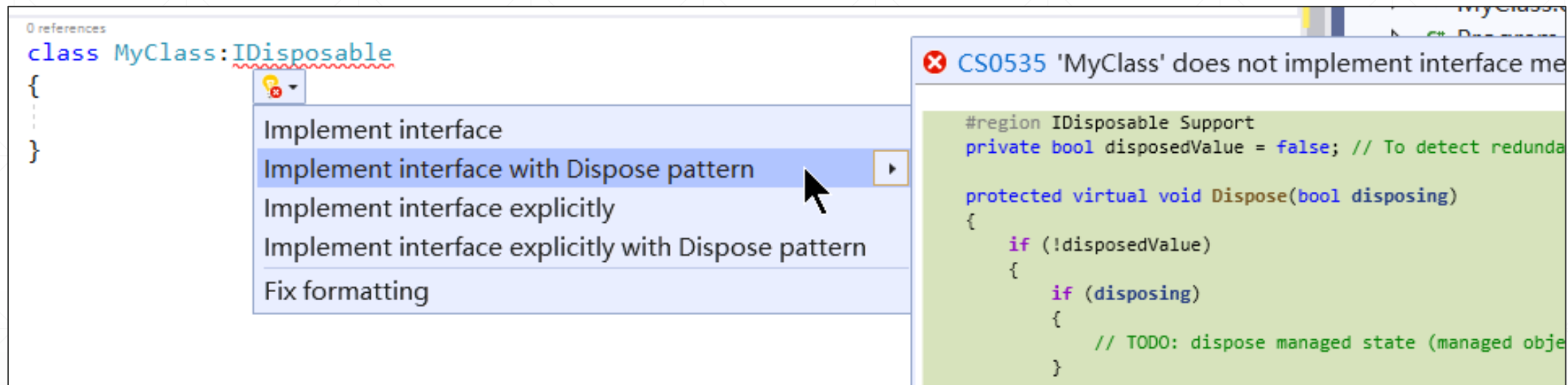
# 谁负责“倒”垃圾？



# 虚拟机职责一：内存的自动分配与智能回收



# C#的Dispose编程模式



The image shows a Visual Studio editor window with a C# class `MyClass` that implements the `IDisposable` interface. A context menu is open over the `IDisposable` interface name, offering several options. The option "Implement interface with Dispose pattern" is selected, indicated by a mouse cursor. To the right, a code snippet is displayed, showing a `#region IDisposable Support` block. This block includes a `private bool disposedValue = false;` to detect redundant calls, and a `protected virtual void Dispose(bool disposing)` method. Inside this method, there is an `if (!disposedValue)` check, followed by an `if (disposing)` block where the user is prompted to dispose managed state (commented as `// TODO: dispose managed state (managed objects)`).

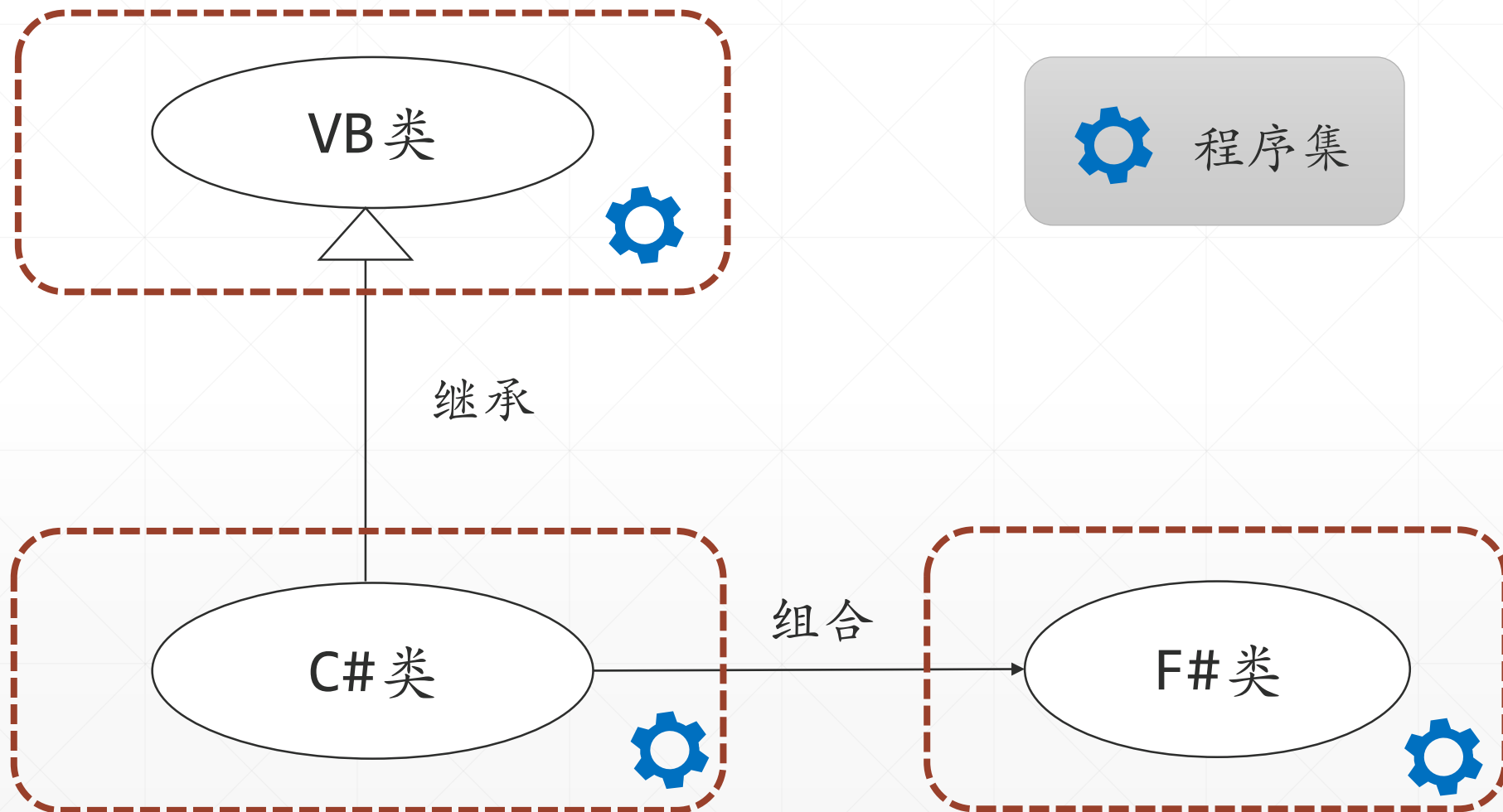
```
0 references
class MyClass:IDisposable
{
    ...
}

Implement interface
Implement interface with Dispose pattern
Implement interface explicitly
Implement interface explicitly with Dispose pattern
Fix formatting
```

```
CS0535 'MyClass' does not implement interface member
#region IDisposable Support
private bool disposedValue = false; // To detect redundant calls

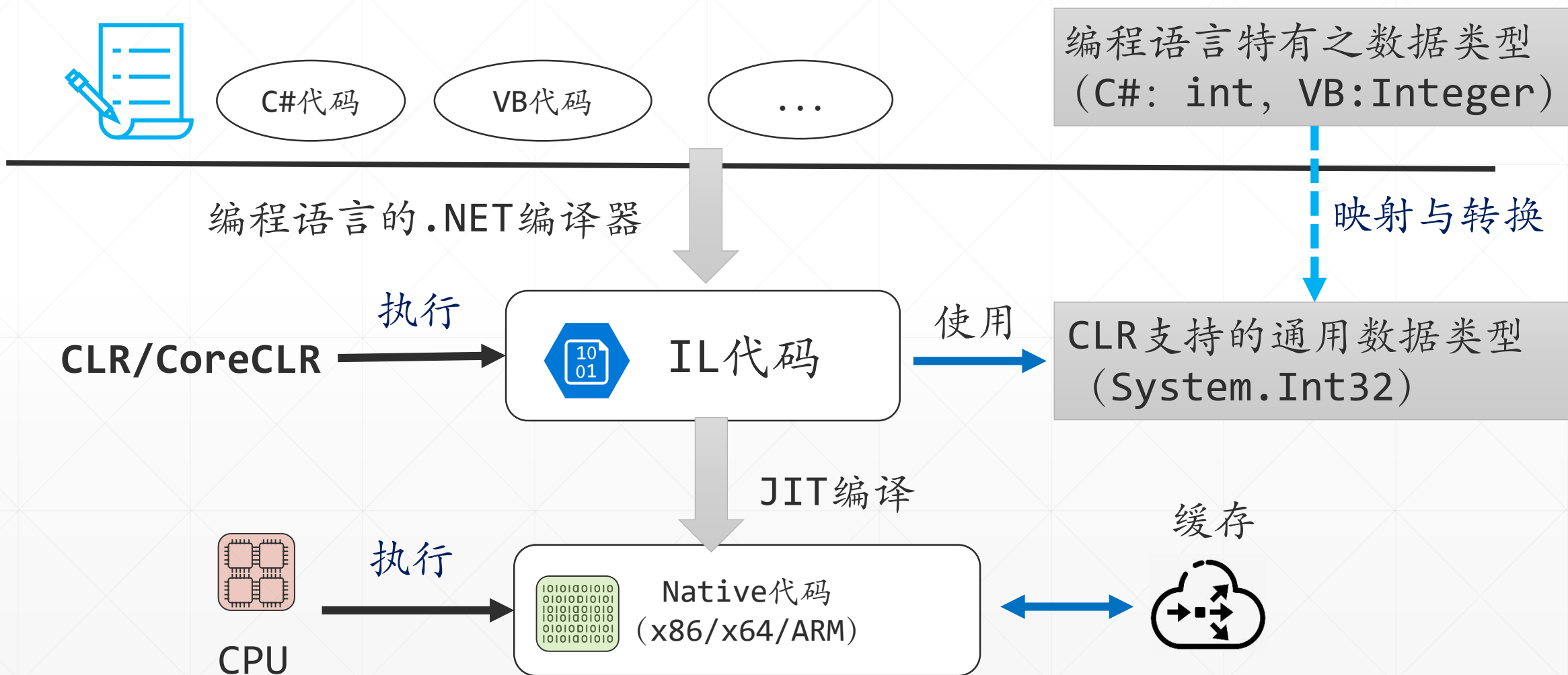
protected virtual void Dispose(bool disposing)
{
    if (!disposedValue)
    {
        if (disposing)
        {
            // TODO: dispose managed state (managed objects)
        }
    }
}
```

## 虚拟机职责二：不同语言代码之间可以相互集成

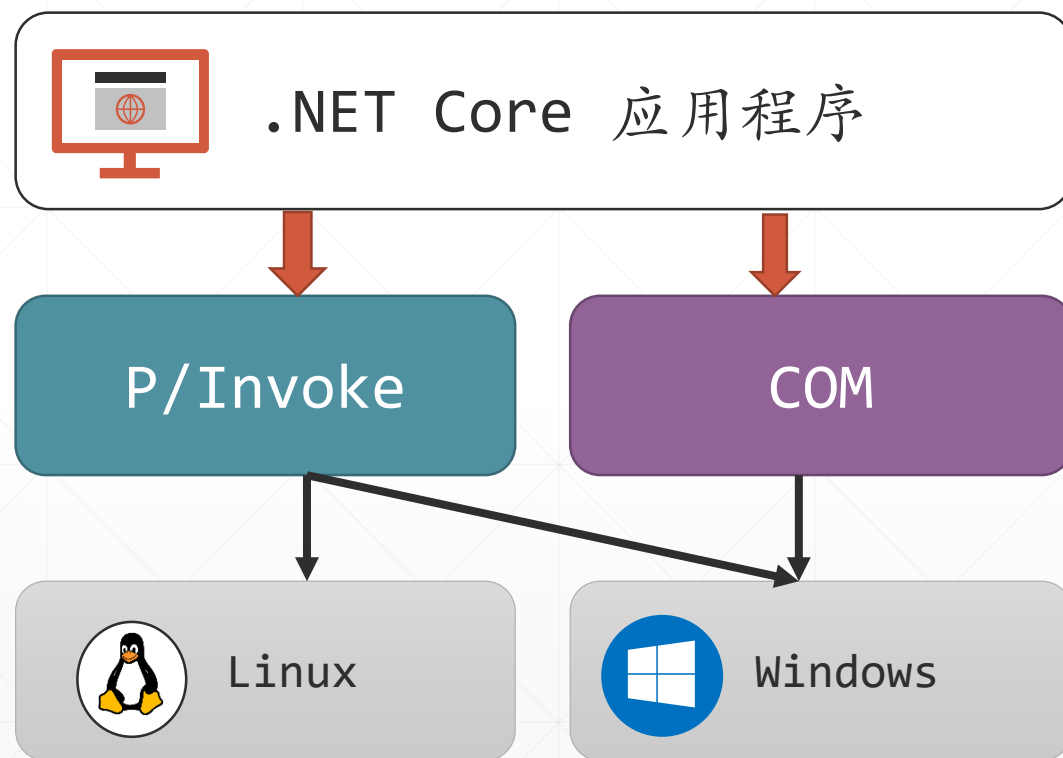




# .NET跨语言编程特性之实现原理



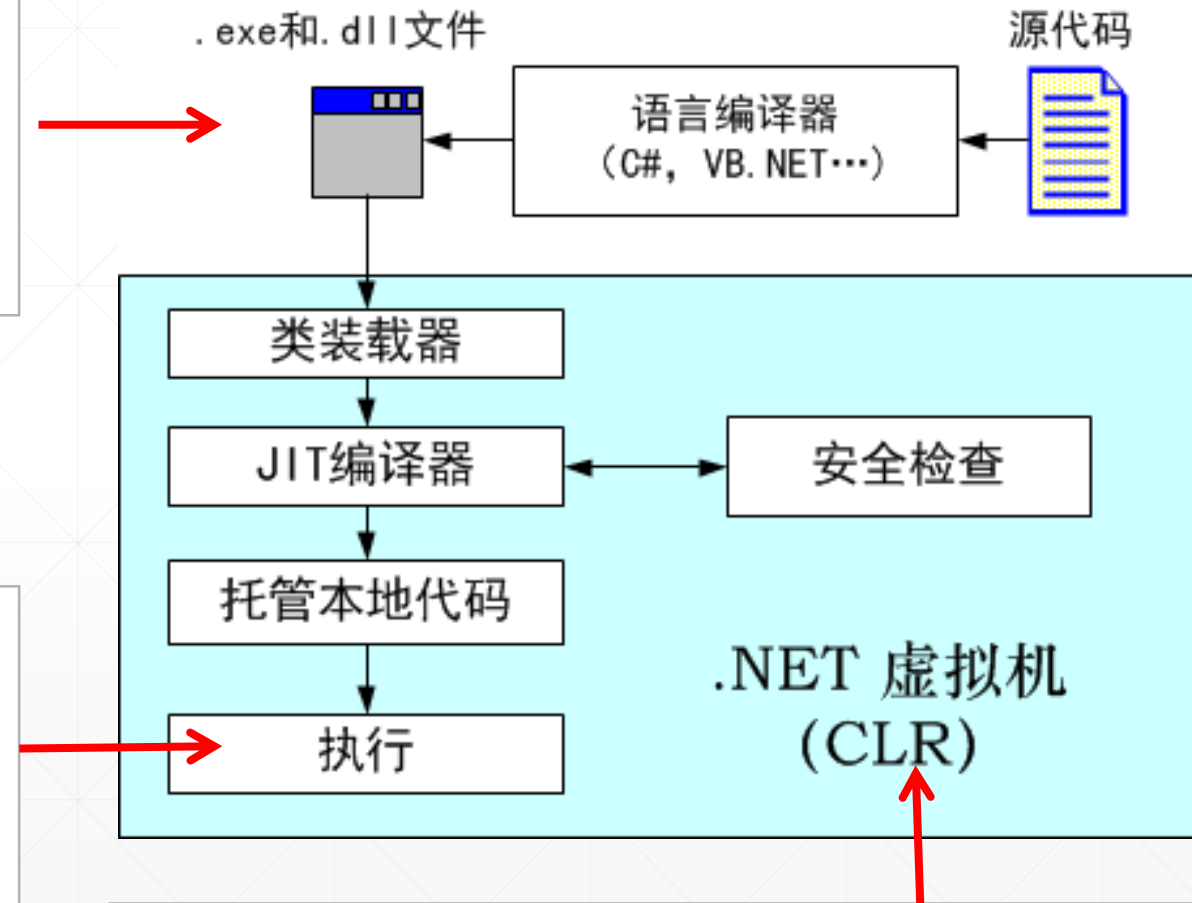
## 虚拟机职责三：平台调用



# CLR的工作原理 (以.NET Framework为例)

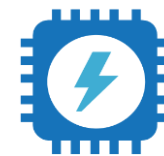
.exe和.dll在.NET中统称为“**程序集 (Assembly)**”，其中保存的是“**IL (中间语言)**”指令。

真正被CPU执行的还是机器指令，它是由JIT编译器在程序运行时动态地由IL指令翻译而成，称为“**本地代码**”。



每个.NET应用程序运行时，操作系统会创建一个CLR实例，最终由CLR负责装入.NET应用程序并执行之。

# 两种.NET程序的编译方式



## JIT (Just In Time Compilation)

- 在程序运行时动态地把IL转换为本地代码
- 具体实现：CLR和CoreCLR

## AOT (Ahead Of Time Compilation)

- 将程序直接编译为本机代码
- 具体实现：Mono、.NET Native和CoreRT

# 一个正在开发中的新运行时—CoreRT



CoreRT - A .NET Core Runtime for AOT

CoreRT = CoreCLR - JIT功能 + AOT功能

CoreRT官网:

<https://github.com/dotnet/corert>

想了解其实现原理, 可以参考以下文章:

<https://mattwarren.org/2018/06/07/CoreRT-.NET-Runtime-for-AOT/>

# 小结:

