应用程序域是.NET引入的一个新概念，它比进程所占用的资源要少，可以被看作是一个轻量级的进程。  
在一个进程中可以包含多个应用程序域，一个应用程序域可以装载一个可执行程序（\*.exe）或者多个程序集（\*.dll）

应用程序域之间深度隔离，即使进程中的某个应用程序域出现错误，也不会影响其他应用程序域的正常运作

当一个程序集同时被多个应用程序域调用时，会出现两种情况：  
第一种情况：CLR分别为不同的应用程序域加载此程序集。  
第二种情况：CLR把此程序集加载到所有的应用程序域之外，并实现程序集共享

**AppDomain的属性与方法**

System命名空间存在AppDomain类，用于管理应用程序域。下面是AppDomain类的常用属性：

|  |  |
| --- | --- |
| 属性 | 说明 |
| ActivationContext | 获取当前应用程序域的激活上下文。 |
| ApplicationIdentity | 获得应用程序域中的应用程序标识。 |
| BaseDirectory | 获取基目录。 |
| CurrentDomain | 获取当前 Thread 的当前应用程序域。 |
| Id | 获得一个整数，该整数唯一标识进程中的应用程序域。 |
| RelativeSearchPath | 获取相对于基目录的路径，在此程序集冲突解决程序应探测专用程序集。 |
| SetupInformation | 获取此实例的应用程序域配置信息。 |

表2.0

AppDomain类中有多个方法，可以用于创建一个新的应用程序域，或者执行应用程序域中的应用程序。

|  |  |
| --- | --- |
| 方法 | 说明 |
| CreateDomain | 创建新的应用程序域。 |
| CreateInstance | 创建在指定程序集中定义的指定类型的新实例。 |
| DoCallBack | 在指定的应用程序域中执行本程序域的委托 |
| ExecuteAssembly | 执行指定文件中包含的程序集。 |
| ExecuteAssemblyByName | 执行程序集。 |
| GetAssemblies | 获取已加载到此应用程序域的执行上下文中的程序集。 |
| GetCurrentThreadId | 获取当前线程标识符。 |
| GetData | 根据指定名称，获取存储在程序域中的值 |
| IsDefaultAppDomain | 返回一个值，指示应用程序域是否是进程的默认应用程序域。 |
| SetData | 将数据保存到程序域中 |
| Load | 将 Assembly 加载到此应用程序域中。 |
| Unload | 卸载指定的应用程序域。 |

表2.1

AppDomain类中有多个事件，用于管理应用程序域生命周期中的不同部分。

|  |  |
| --- | --- |
| 事件 | 说明 |
| AssemblyLoad | 在加载程序集时发生。 |
| AssemblyResolve | 在对程序集的解析失败时发生。 |
| DomainUnload | 在即将卸载 AppDomain 时发生。 |
| ProcessExit | 当默认应用程序域的父进程存在时发生。 |
| ReflectionOnlyAssemblyResolve | 当程序集的解析在只反射上下文中失败时发生。 |
| ResourceResolve | 当资源解析因资源不是程序集中的有效链接资源或嵌入资源而失败时发生。 |
| TypeResolve | 在对类型的解析失败时发生。 |
| UnhandledException | 当某个异常未被捕获时出现。 |

表2.2

下面将举例详细介绍一下AppDomain的使用方式

**在AppDomain中加载程序集**

通过CreateDomain方法可以建立一个新的应用程序域。  
下面的例子将使用CreateDomain建立一个应用程序域，并使用Load方法加载程序集Model.dll。最后使用GetAssemblies方法，列举此应用程序域中的所有程序集。

static void Main(string[] args)

{

    // 创建程序域 NewAppDomain

    var appDomain = AppDomain.CreateDomain("NewAppDomain");

    // 程序域 NewAppDomain 加载 Model.dll

    appDomain.Load("Model");

    // 获取程序域以载入的程序集

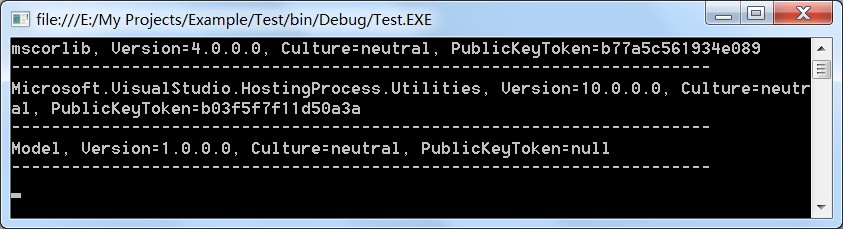
    foreach (var assembly in appDomain.GetAssemblies())

        Console.WriteLine(string.Format("{0}\n----------------------------", assembly.FullName));

    Console.ReadKey();

}

运行结果



注意：当加载程序集后，就无法把它从AppDomain中卸载，只能把整个AppDomain卸载。

当需要在AppDomain加载可执行程序时，可以使用ExecuteAssembly方法。

// 当加载的程序集是 dll 时，dll 的 ModuleStart 方法会被运行

// 同理，加载的是 exe 时，其 main 方法会被运行

appDomain.ExecuteAssembly("Example.exe");

**卸载AppDomain**

通过Unload可以卸载AppDomain，卸载时将会触发DomainUnload事件。

static void Main(string[] args)

{

    // 新建名为 NewAppDomain 的应用程序域

    AppDomain newAppDomain = AppDomain.CreateDomain("NewAppDomain");

    // 设置其 AssemblyLoad 事件处理方法

    newAppDomain.AssemblyLoad +=

        (obj, e) =>

        {

            Console.WriteLine(string.Format("{0} is loading!", e.LoadedAssembly.GetName()));

        };

    // 设置其 DomainUnload 事件处理方法

    newAppDomain.DomainUnload +=

        (obj, e) =>

        {

            Console.WriteLine("NewAppDomain Unload!");

        };

    // 加载程序集

newAppDomain.Load("Model");

    // 模拟操作

    for (int n = 0; n < 5; n++)

        Console.WriteLine("  Do Work.......!");

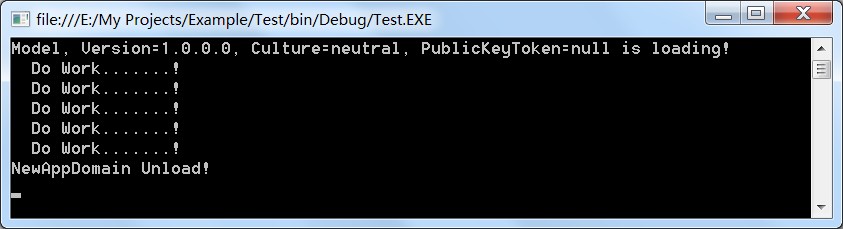
    // 卸载AppDomain

    AppDomain.Unload(newAppDomain);

    Console.ReadKey();

}

运行结果



**在AppDomain中实例指定类的对象**

使用CreateInstance方法，能实例指定类的对像。 但使用此方法将返回一个ObjectHandle对象，若要将此值转化为原类型，可调用Unwrap方法。  
下面例子会实例Model.dll程序集中的Model.Person对象。

namespace Test

{

     public class Program

    {

        static void Main(string[] args)

        {

            // 实例对象 Model.Person

            var person=(Person)AppDomain.CurrentDomain

                            .CreateInstance("Model","Model.Person").Unwrap();

            person.ID = 1;

            person.Name = "Leslie";

            person.Age = 29;

            Console.WriteLine(string.Format("{0}'s age is {1}!",person.Name,person.Age));

            Console.ReadKey();

        }

    }

}

namespace Model {

    public class Person {

        public int ID { get; set; }

        public string Name { get; set; }

        public int Age { get; set; }

    }

}