.NET Framework的部署目标

Windows多年来一直因为不稳定和过于复杂而口碑不佳，要归咎于几方面的原因：

①所有应用程序都使用来自Microsoft或者其他厂商动态链接库(Dynamic-Link Library,DLL)--DLL hell。

②安装的复杂性。大多数应用程序在安装时会影响到系统的全部组件。

③涉及安全性。应用程序安装时会带来各种文件，其中许多是由不同的公司开发的。此外，Web应用程序经常会悄悄下载一些代码(比如ActiveX控件)，用户根本注意不到自己的机器上安装了这些代码。并且这些代码能执行类似删除文件或发送电子邮件的操作。

2、.NET Framework正在尝试彻底解决DLL hell的问题，还在很大程度上解决了应用程序状态在用户硬盘中四处分散的问题。如：类型不再需要注册表中的设置。

将类型生成到模块中：CSC.exe

响应文件：是一个文本文件(扩展名rsp)，包含一组编译器命令行开关（包含引用的库）。执行CSC.exe时可以直接读取并应用这些开关。

什么是元数据？元数据是描述数据的数据。元数据是一个二进制数据块，由几个表组成。这些表分为三个类别：定义表、引用表和清单表。

① 常用的定义表：ModuleDef（包含对模块进行标识的记录项。包含模块的文件名、扩展名、版本ID）、TypeDef（包含每个类的记录项。包含类名、基类、类的标志(即作用域如public，private)和一些指向其他表的索引）、MethodDef（包含每个方法的记录项。包含方法名，方法的标志(如virtual、public等)、方法的签名(由方法的名称和形参列表组成)以及方法的IL代码在模块中的偏移量）、FieldDef（包含每个字段的记录项。包含标志、类型、名称）、ParamDef（包含每个参数的记录项。包含标志(如in、out、retval等)、类型、名称）、PropertyDef（包含每个属性的记录项。包含标志、类型、名称）、EventDef（包含每个事件的记录项。包含标志和名称）。

② 常用的引用表：AssemblyRef（包含模块引用的程序集的记录项。包含程序集名称、版本号、语言文化、公钥令牌，每个项目还包含一些标志(flag)和一个哈希值）、ModuleRef（包含该模块引用的类型的每个托管模块的记录项。包含了每个托管模块的文件名和扩展名。作用是建立同实现该模块中类型的类型的绑定关系）、TypeRef（包含模块引用的每一个类型的记录项。包含类的名称和指向类型的引用）、MemberRef（包含模块引用的每个成员的记录项。每个记录项都包含成员的名称和签名，并且指向对成员进行定义的那个类型的TypeRef记录项）。

③ 清单表：AssemblyDef（如果模块标识的是程序集，那么就包含程序集的名称、语言文化、一些标志、哈希算法以及发布者公钥）、FileDef（包含程序集每个PE文件、资源文件的记录项（除了清单本身）。包含文件名和扩展名、哈希值和一些标志（如果程序集只包含“主模块”，则该表为空））、ManifestResourceDef（包含每个资源的记录项。包含资源名称、一些可见性的标志、FileDef的索引。（如果资源不是独立文件如gif等，那么资源就是包含在PE文件中的流，对这些“嵌入文件”记录项包含一个偏移量指出资源流在PE文件中的起始位置））、ExportedTypesDef（包含程序集中从PE模块导出每个public类的记录项。包含类名、FileDef表的一个索引和TypeDef表的一个索引）。

程序集所有文件中，有一个文件容纳了清单。清单是一个元数据表的集合，表中主要包含作为程序集组成部分的那些文件的名称。

清单使得程序集具有自描述性（因为它包含了程序集的重要信息，如程序集的名称、版本、文化、公钥标记和依赖关系等。这些信息使得CLR能够在运行时确定程序集的身份和依赖关系，并且能够正确地加载和使用程序集）。包含清单文件不会指出自己是程序集中的哪一部分。

补充第一章笔记中对于程序集的组成，程序集可以由多个文件组成，一些是含有清单的PE文件(托管模块)，另一部分是资源文件如gif、jpg。

程序集的特点：1.程序集定义了可重用的类型。2.程序集用一个版本号标记。3.程序集可以关联安全信息。

AL和CSC程序 生成一个PE文件程序集时，还会在PE文件中嵌入一个标准的Win32版本资源。

版本号格式：major(主版本号)+minor(次版本号)+build(内部版本号)+revision(修订号)。

部署到和应用程序相同目录中的程序集称为 私有部署的程序集。

阅读十八章 定制特性时产生的困惑：

1.在编译刚开始之前就生成元数据的定义表和引用表吗？

在编译刚开始之前并不会生成元数据的定义表和引用表，而是在编译过程中逐步生成。在编译过程中，编译器会扫描各项名称并将它们加入元数据定义表。元数据定义表包含了程序集中所有类型和成员的定义，以及对其他程序集中类型和成员的引用。这些信息在编译过程中被收集并存储在程序集的元数据部分，以便在运行时被公共语言运行时（CLR）使用。

2. 那如果一个类使用了还没有加入元数据定义表的部分，此时编译器会做什么？

由上一个问题的解答可以知道，如果一个类使用了还没有加入元数据定义表的部分，编译器会在编译过程中解析和处理所有的类型和依赖关系，以确保所有的类型都被正确编译。编译器会扫描各项名称并将它们加入元数据定义表，引用的其他程序集的类型会被加入引用表，然后定义表引用引用表中的对应项。

3. 清单不会指出自己是程序集的哪一部分，那么加载程序集时如何找到清单文件呢？

当CLR加载程序集时，它会根据程序集的名称和位置来查找清单文件。程序集的开发人员可以在创建程序集时指定清单文件的名称和位置，以便在运行时被CLR使用。

4. 如果开发人员没有特意指定清单文件的位置，CLR如何获取清单文件呢？

如果开发人员没有特意指定清单文件的位置，CLR会使用默认的规则来查找清单文件。对于托管程序集，CLR会在程序集所在的目录中查找具有相同名称的文件，该文件的扩展名为.dll或.exe。