

高级编程技术

- 第1章 高级开发概述
- 第2章 组件化开发
- 第3章 XML 技术
- 第4章 文件和数据库访问技术
- 第5章 线程和异步编程
- 第6章 分布式组件技术
- 第7章 XML Web Service
- 第8章 设计模式与构建
- 第9章 面向服务的架构（SOA）
- 第10章 开发智能设备应用程序

第 2 章 组件化开发

- 组件化开发概述
- 创建组件
- 创建简单的控制台客户端程序调用组件
- 创建 Web 客户端应用程序调用组件
- 应用程序部署介绍
- 应用程序部署方案

组件化开发概述

2.1 组件化开发概述

- 组件化技术
- 面向对象技术
- 面向对象开发和组件化开发
- 组件化开发优势
- 组件化开发方法

组件化技术

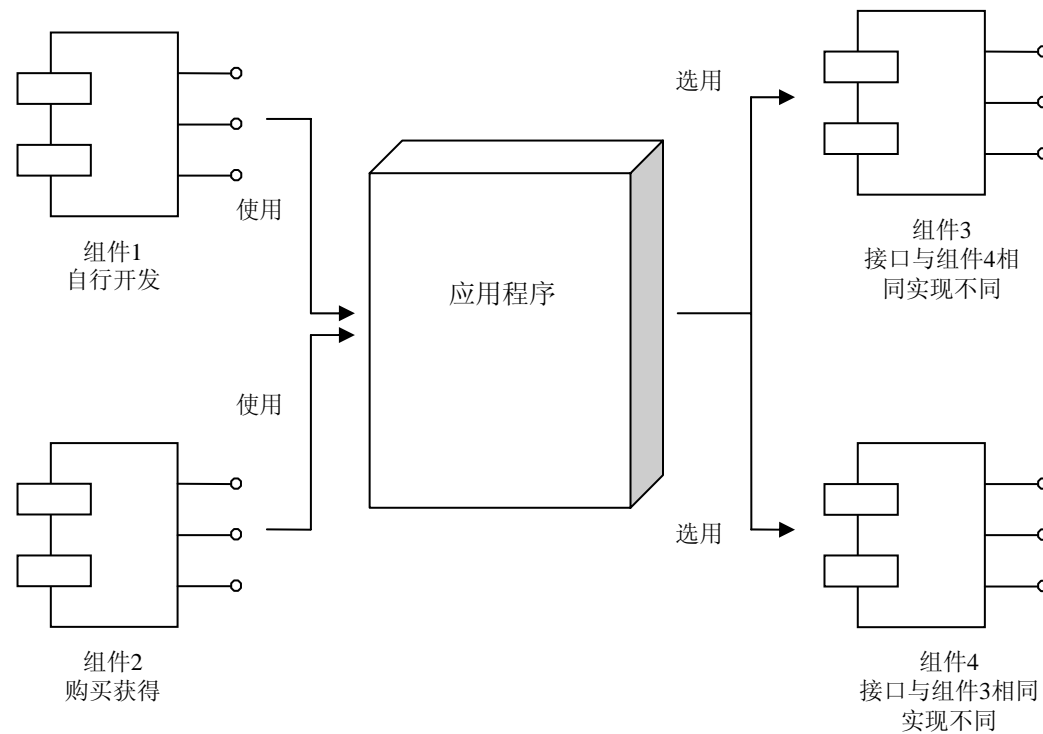
2.1.1 组件化技术

- 标准化与组件技术
 - 组件是指用于重用、发布和部署的二进制代码单元
 - 组件遵循一定的规范
- 接口与实现分离
- 组件技术解决的问题
 - 重用
 - 部署

组件化技术

2.1.1 组件化技术

- 接口与实现分离——组件技术的灵活性



面向对象技术

2.1.2 面向对象技术

- 便于开发复杂应用程序
- 面向对象的基本要点
 - 封装
 - 抽象
 - 继承
 - 多态

面向对象开发和组件开发

2.1.3 面向对象开发和组件开发

- 两者的区别

- 面向对象开发：更侧重于应用程序的微观层次，关心数据如何封装成对象。
- 组件开发：侧重于宏观层次上如何划分模块（即组件），确定模块之间划分得合理性和依赖性。

组件化开发优势

2.1.4 组件化开发优势

- 组件化开发优势：
 - 良好的可重用性
 - 购买第三方组件
 - 具有很好的灵活性

组件化开发方法

2.1.5 组件化开发方法

- 组件化开发方法

- 必须基于某种可靠的组件技术（COM 组件）
- 设计、开发、测试、部署等环节都需要适应组件化开发的特点
- 带来的负面影响

- 常见组件模型

- COM/.NET
- Java Bean
- Corba

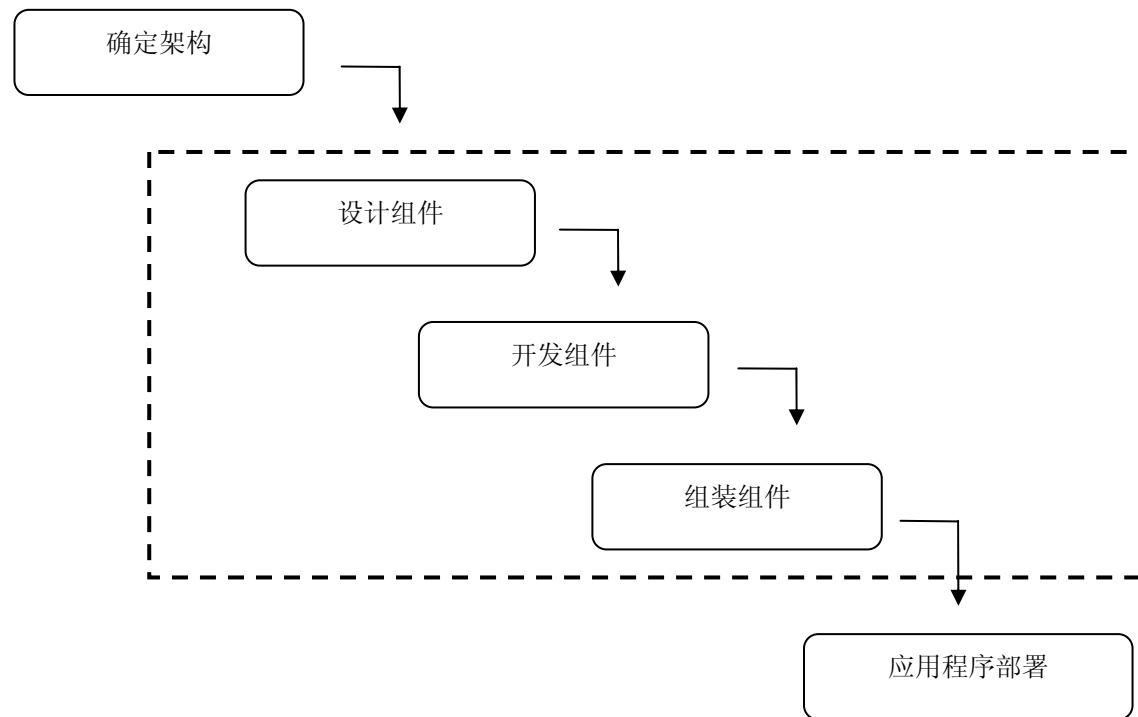
- 组件化开发步骤

- 设计组件
- 开发组件
- 组装组件

组件化技术

2.1.5 组件化开发方法

● 组件化技术的开发流程



第 2 章 组件化开发

- 组件化开发概述
- 创建组件
- 创建简单的控制台客户端程序调用组件
- 创建 Web 客户端应用程序调用组件
- 应用程序部署介绍
- 应用程序部署方案

创建组件

2.2 创建组件

- 使用命名空间和声明类
- 创建类的实现
- 实现结构化异常处理
- 创建属性
- 编译组件

使用命名空间和声明类

2.2.1 使用命名空间和声明类

- 创建命名空间

```
using System;  
namespace CompCS {...}
```

- 定义类

```
public class StringComponent {...}
```

创建类的实现

2.2.2 创建类的实现

- 声明一个私有的字符串数组字段

```
private string[] stringSet;
```

- 创建公有默认构造函数

```
public StringComponent() {...}
```

- 用字符串来初始化字符串数组 stringSet

```
    stringSet = new string[] {  
        "C# String 0",  
        "C# String 1",  
        ...  
    };
```

实现结构化异常处理

2.2.3 实现结构化异常处理

- 实现 GetString 方法

```
public string GetString(int index) {...}
```

- 创建和抛出新类型对象 IndexOutOfRangeException 异常

```
if((index < 0) || (index >= stringSet.Length)) {  
    throw new IndexOutOfRangeException();  
}  
return stringSet[index];
```

- 异常可能会在 try、catch、finally 块的函数调用中被捕获
- 结构化异常处理代理了 COM 中的以 HRESULT 为基础的错误处理机制

实现结构化异常处理（续）

2.2.3 实现结构化异常处理

- 传统上错误处理模型
 - 编程语言上独特的错误处理过程
 - 操作系统提供错误处理机制
- .NET 运行库实现的异常处理具有下列特点
 - 处理异常时不考虑生成异常的语言和处理异常的语言
 - 异常处理时不要求任何特定的语言语法，而是允许每种语言定义自己的语法
 - 允许跨进程甚至跨计算机边界引发异常

实现结构化异常处理（续）

2.2.3 实现结构化异常处理

- 与其他错误通知方法相比，异常具有若干优点
 - 不再有出现错误而不被人注意的情况
 - 无效值不会继续在系统中传播
 - 不必检查返回代码
 - 可以轻松添加异常处理代码，以增加程序的可靠性
 - 运行库的异常处理比基于 Windows 的 C++ 错误处理更快
- 所有异常类都是从基类 **Exception** 类继承的
 - 如 `IndexOutOfRangeException` 类就是 `Exception` 类的子类

创建属性

2.2.4 创建属性

- 创建一个只读属性来取得 stringSet 字符数组中元素数量

```
public int Count {  
    get { return stringSet.Length; }  
}
```

编译组件

2.2.5 编译组件

- 用 `/target: library` 编译器开关创建一个 DLL 组件
 - 否则，就会创建一个以 `.exe` 为扩展名的可执行文件，而不是一个 DLL 的类库

```
csc /out: CompCS.dll /target: library CompCS.cs
```

第 2 章 组件化开发

- 组件化开发概述
- 创建组件
- 创建简单的控制台客户端程序调用组件
- 创建 Web 客户端应用程序调用组件
- 应用程序部署介绍
- 应用程序部署方案

创建简单的控制台客户端程序调用组件

2.3 创建简单的控制台客户端程序调用组件

- 使用类库
- 实例化组件
- 调用组件
- 生成客户端应用程序

使用类库

2.3.1 使用类库

- 引用库，使我们在引用类型的时候不需要整个类型名称

```
using CompA;  
using CompB;
```

- 如果多个命名空间包含相同名称的类型，创建一个命名空间别名来消除歧义

```
using StringCompA = CompA.StringComponent;  
using StringCompB = CompB.StringComponent;
```

实例化组件

2.3.2 实例化组件

- 声明一个 StringComponent 类型的局部变量
- 创建 StringComponent 类的实例

```
//...  
using CSStringComp = CompCS.StringComponent;  
//...  
CSStringComp myCSStringComp = new  
    CSStringComp();
```

调用组件

2.3.3 调用组件

- 遍历 StringComponent 组件的字符串数组，并且把这些字符串输出到控制台

```
for (int index = 0;  
    index < myCSStringComp.Count; index++) {  
    Console.WriteLine  
        (myCSStringComp.GetString(index));  
}
```


生成客户端应用程序

2.3.4 生成客户端应用程序

- 用 /reference 开关来引用包含 StringComponent 类的程序集

```
csc /reference: CompCS.dll, CompVB.dll ↵  
/out: ClientCS.exe ClientCS.cs
```

第 2 章 组件化开发

- 组件化开发概述
- 创建组件
- 创建简单的控制台客户端程序调用组件
- 创建 Web 客户端应用程序调用组件
- 应用程序部署介绍
- 应用程序部署方案

创建 Web 客户端应用程序调用组件

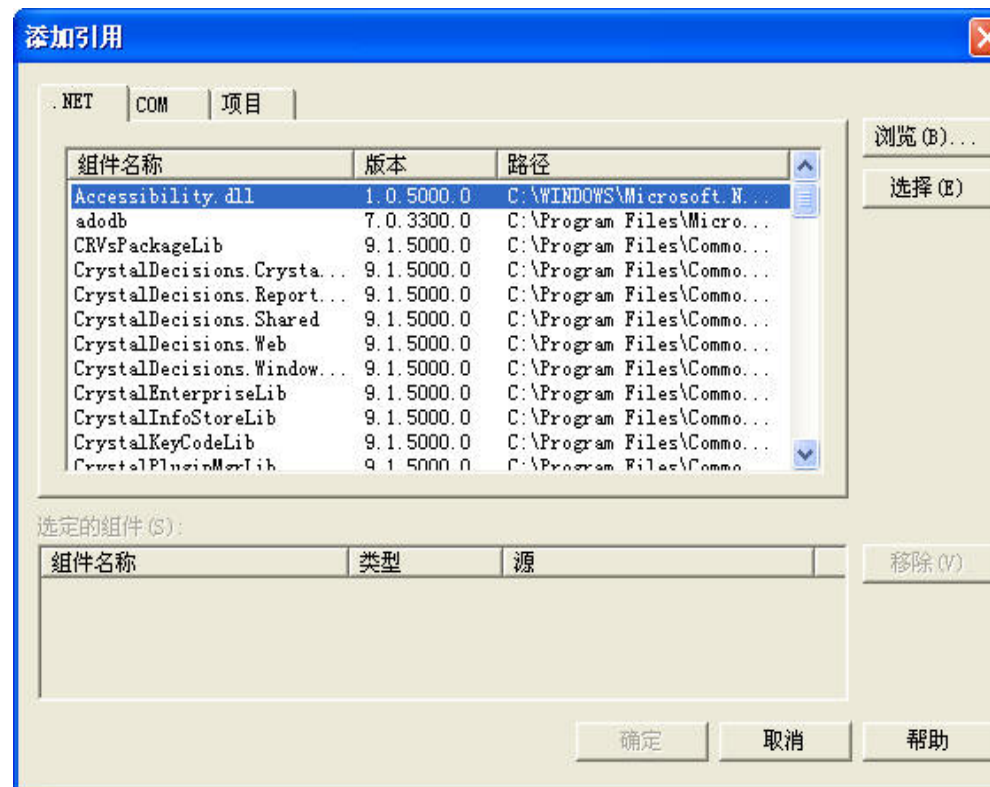
2.4 创建 Web 客户端应用程序调用组件

- 创建 Web 应用程序
- 编写 Page_Load 事件处理程序

在Web应用程序中编写HTML

2.4.1 创建Web应用程序

- 在 VS.NET 中创建 Web 应用程序
- 向 Web 应用程序中添加组件的引用



编写 Page_Load 事件处理程序

2.4.2 编写 Page_Load 事件处理程序

```
void Page_Load(Object sender, EventArgs EvArgs)
{
    StringBuilder Out = new StringBuilder("");
    int Count = 0;
    //遍历并连接组件的所有字符串
    Out.Append("Strings from C# Component \n");
    CompCS.StringComponent myCSStringComp = new
        CompCS.StringComponent();
    for(int index = 0; index < myCSStringComp.Count; index++)
    {
        Out.Append(myCSStringComp.GetString(index));
        Out.Append("\n");
    }
    Out.Append("\n");
    TextBox1.text = Out.ToString();
}
```

编写 Page_Load 事件处理程序（续）

2.4.2 编写 Page_Load 事件处理程序

- ASP.NET 中的事件
 - 在 ASP.NET 页面执行过程中，有一系列的事件发生，其中最重要的是 Init 和 Load 事件
 - Init 事件发生在页面首次被实例化后
 - Load 事件发生在页面中的控件被实例化后，页面内容被输出之前
 - Load 事件很重要，它是放置动态初始化页面控件代码的最好地方

第 2 章 组件化开发

- 组件化开发概述
- 创建组件
- 创建简单的控制台客户端程序调用组件
- 创建 Web 客户端应用程序调用组件
- 应用程序部署介绍
- 应用程序部署方案

应用程序部署介绍

2.5 应用程序部署介绍

- 虚拟执行环境
- 在虚拟执行环境中编译和运行应用程序
- 部署的基本概念
- 简单应用程序
- 组件化应用程序
- 配置和分发

虚拟执行环境

2.5.1 虚拟执行环境

- 虚拟执行环境技术特点

- 虚拟执行环境技术的最大特点在于用软件技术构建了一个硬件和平台的抽象层，虚拟执行环境自身就是一个计算机平台
- 可移植性
- 更多的控制程序运行的手段
- 局限性

- 虚拟执行环境应用

- 常见的虚拟执行环境包括
 - .NET 的托管执行环境：包括 C#、VB.NET 在内的多种编程语言
 - Java 语言所使用的虚拟执行环境 JVM：Java 语言

在虚拟执行环境中编译和运行应用程序

2.5.2 在虚拟执行环境中编译和运行应用程序

- 编译器选项

- Csc.exe

- 元数据

- 元数据是一组数据表，它能够充分描述模块中定义的每个元素
- 元数据的用途
 - 查找和加载类
 - 强制安全性

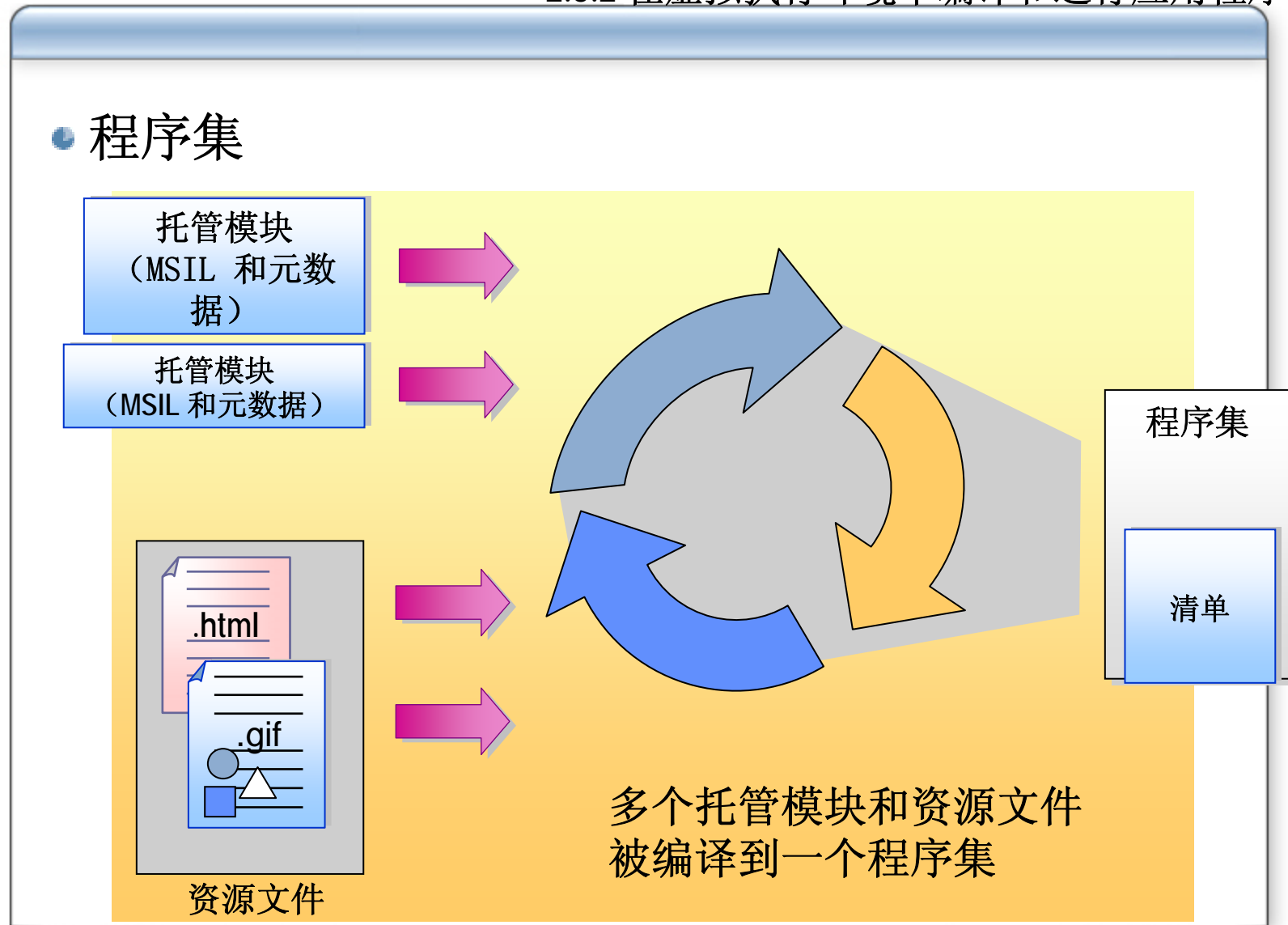
- 中间语言（MSIL）

- 经过编译的 MSIL
- 转换为本机代码

在虚拟执行环境中编译和运行应用程序（续）

2.5.2 在虚拟执行环境中编译和运行应用程序

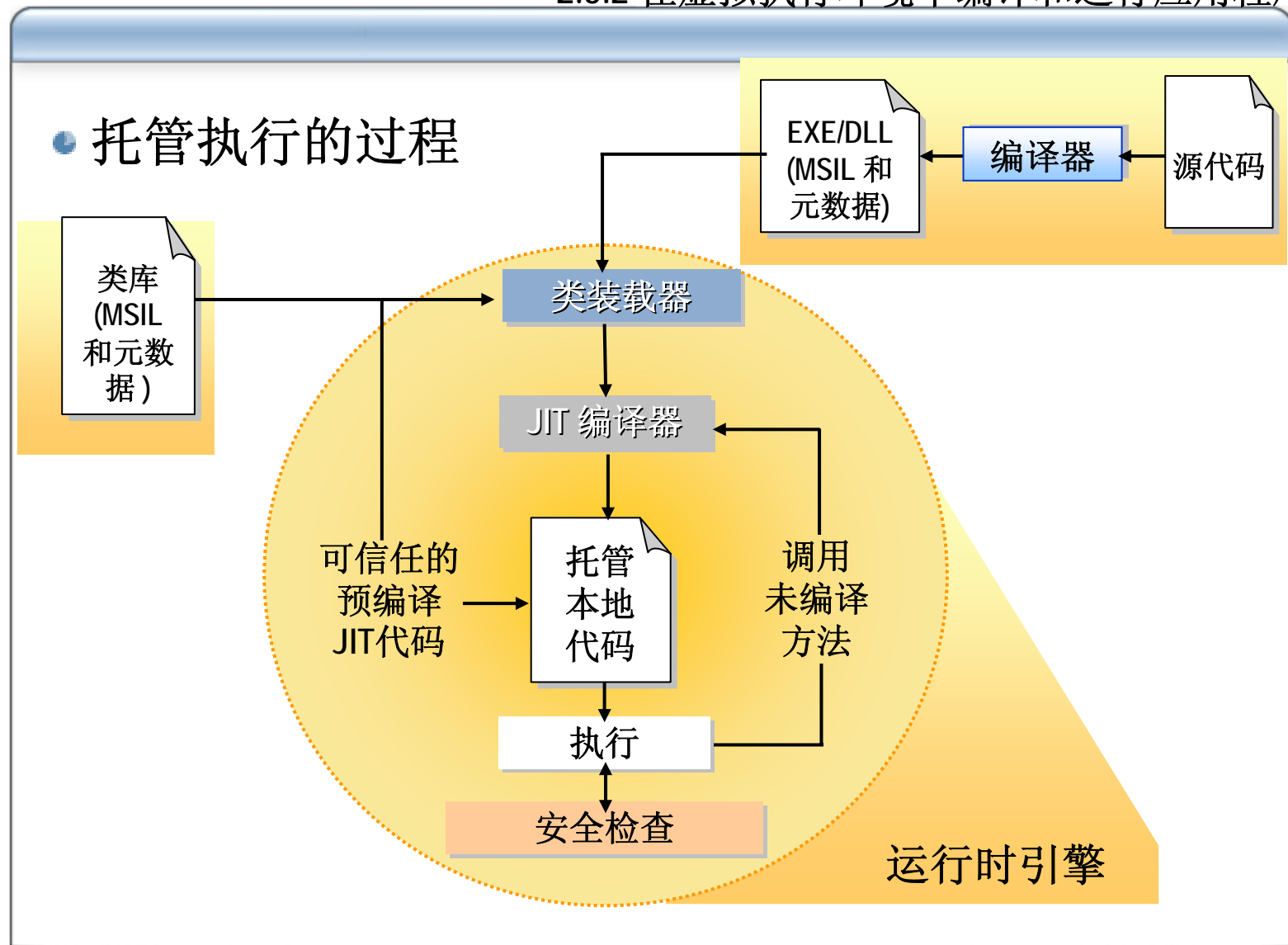
● 程序集



在虚拟执行环境中编译和运行应用程序（续）

2.5.2 在虚拟执行环境中编译和运行应用程序

• 托管执行的过程



在虚拟执行环境中编译和运行应用程序（续）

2.5.2 在虚拟执行环境中编译和运行应用程序

- 公共语言运行库工具
 - MSIL 编译器（ilasm.exe）：从 Microsoft 中间语言（MSIL）生成最终可执行二进制代码
 - MSIL 反编译器（ildasm.exe）：检查元数据和托管二进制代码，利用包含 MSI 代码的 PE 文件，创建适合输入到 MSI 汇编程序（ilasm.exe）的文本文件
- 实时（JIT, Just-In-Time）编译
 - 代码执行过程
- 应用程序域
 - 过去进程边界用来隔离应用程序
 - 现在应用程序域提供应用程序之间的隔离
 - 在一个应用程序中的错误不会影响其他的应用程序
- 垃圾回收
 - 自动回收

部署基本概念

2.5.3 部署的基本概念

- .NET Framework 应用程序中用到的类和类型
 - 被组织在命名空间体系中
 - 保存在 PE 文件中，如 DLL 文件和 EXE 文件
 - 被元数据充分描述
- 程序集
 - 由一个或多个 PE 文件组成
 - 包含标识程序集的清单
 - 详细说明输入输出的类和类型
 - 部署、重用和版本控制的单位

简单应用程序

2.5.4 简单应用程序

- 需要本地计算机上安装 .NET 运行库
- 可以从文件服务器或者本地拷贝直接运行
- 不需要在注册表注册
- 不会影响其他应用程序
 - 防止了 DLL 版本冲突（DLL HELL）
- 通过直接删除卸载应用程序

组件化应用程序

2.5.5 组件化应用程序

- 程序集属应用程序专有
 - 和简单应用程序类似
- 程序集属相关应用程序专有且共享
 - 把程序集部署到一个公共的子目录中
- 程序集被其他不相关应用程序共享
 - 程序集需要一个强名称和版本信息
 - 把程序集部署到全局程序集缓存中

配置和分发

2.5.6 配置和分发

- 配置应用程序
 - 通过以 XML 为格式的文本文件来配置应用程序
 - 不需要开发人员的参与，系统管理员就可以在不同的计算机上定制应用程序
- 部署应用程序
 - 通用分发格式，
 - 例如 .CAB 文件或者 Windows Installer (.MSI) 文件
 - 通用分发机制，如 Windows 2000 IntelliMirror 或微软系统管理服务服务器

第 2 章 组件化开发

- 组件化开发概述
- 创建组件
- 创建简单的控制台客户端程序调用组件
- 创建 Web 客户端应用程序调用组件
- 应用程序部署介绍
- 应用程序部署方案

应用程序部署方案

2.6 应用程序部署方案

- 简单应用程序
- 组件化应用程序
- 指定私有程序集路径
- 两种程序集、两种部署
- 强名称程序集
- 部署共享组件
- 版本化程序集
- 创建强名称程序集的多个版本
- 绑定策略
- 部署多版本的强名称程序集
- 打包和部署工具

简单应用程序

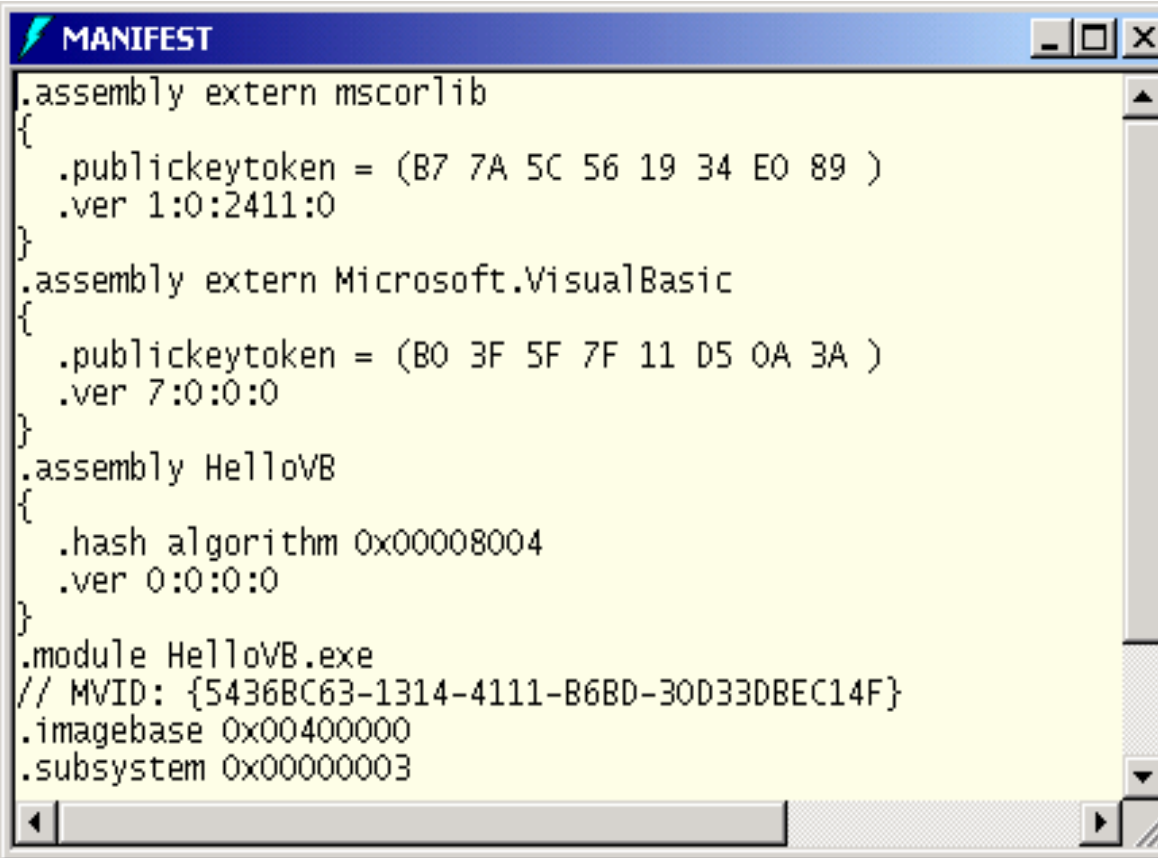
2.6.1 简单应用程序

- 用MSIL反编译器（Ildasm.exe）来查看程序集清单包含的信息
 - 版本信息
 - 输出类型信息
 - 输入类型信息
- 部署应用程序
 - 从文件服务器上直接执行可执行文件，或者通过本地拷贝安装应用程序
 - 通过直接删除文件来卸载应用程序

简单应用程序（续）

2.6.1 简单应用程序

- 程序集清单包含的输入输出类型及版本信息

A screenshot of a text editor window titled "MANIFEST". The window has a blue title bar with standard Windows window controls (minimize, maximize, close). The text inside is a manifest file for an application named "HelloVB.exe". It lists external assemblies: "mscorlib" and "Microsoft.VisualBasic", each with its public key token and version. It also specifies the hash algorithm (0x00008004) and version (0:0:0:0) for the application assembly. Finally, it defines the module "HelloVB.exe" with its MVID, image base (0x00400000), and subsystem (0x00000003).

```
.assembly extern mscorlib
{
    .publickeytoken = (B7 7A 5C 56 19 34 E0 89 )
    .ver 1:0:2411:0
}
.assembly extern Microsoft.VisualBasic
{
    .publickeytoken = (B0 3F 5F 7F 11 D5 0A 3A )
    .ver 7:0:0:0
}
.assembly HelloVB
{
    .hash algorithm 0x00008004
    .ver 0:0:0:0
}
.module HelloVB.exe
// MVID: {5436BC63-1314-4111-B6BD-30D33DBEC14F}
.imagebase 0x00400000
.subsystem 0x00000003
```

指定私有程序集路径

2.6.2 指定私有程序集路径

- 现实中，系统管理员为了管理方便会把程序集组件放到一个单独的子目录中
- 编译时指定程序集的路径
 - 重新编译上一节的源文件，只是编译选项有些不同

```
cd \compapp
csc /target:library /out:MyStringer\Stringer.dll↵
MyStringer\Stringer.cs
csc /reference:MyStringer\Stringer.dll Client.cs
```

指定私有程序集路径（续）

2.6.2 指定私有程序集路径

- 在 **Client.exe.config** 配置文件中用 **privatePath** 标签类指定应用程序加载私有程序集的路径

```
<configuration>
  <runtime>
    <assemblyBinding
      xmlns="urn:schemas-microsoft-com:asm.v1">
      <probing privatePath="MyStringer"/>
    </assemblyBinding>
  </runtime>
</configuration>
```

- 配置文件中 XML 标签是区分大小写的
- 配置文件名是应用程序全名称加“.config”

组件化应用程序

2.6.3 组件化应用程序

- 被应用程序调用的程序集组件
 - 创建程序集Stringer.dll

```
csc /target:library Stringer.cs
```

- 引用程序集来创建客户端应用程序

```
csc /reference:Stringer.dll Client.cs
```

- 通过文件服务器和本地拷贝来部署应用程序
 - 一般情况下可以用拷贝部署应用程序，用直接删除来卸载应用程序

两种程序集、两种部署

2.6.4 两种程序集、两种部署

- .NET Framework 支持两种类型程序集
 - 弱名称程序集
 - 用密钥签名的强名称程序集
- 程序集有两种部署方式
 - 弱名称程序集只能用私有方式部署
 - 强名称程序集既可以用私有方式部署也可以用全局方式部署，即部署到全局程序集缓存中

强名称程序集

2.6.5 强名称程序集

- 全局程序集缓存包含被多个不相关应用程序共享的程序集
- 组件共享中的问题
 - 在 COM 和 COM+ 时代，组件共享严重依赖注册表，如新安装修改了系统注册表就会造成其他应用程序崩溃
 - 在 .NET 平台上，通过并行执行（side-by-side execution）来杜绝了这个缺陷

强名称程序集（续）

2.6.5 强名称程序集

- 强名称

- 强名称包含类型名称、版本号、区域信息、私钥四个要素
- 下面三个字符代表不同的三个组件

```
"MyTypes,Version=1.0.8123.0,Culture=neutral,PublicKeyToken=b77a5c561934e089"
```

```
"MyTypes,Version=1.0.8123.0,Culture="en-US",PublicKeyToken=b77a5c561934e089"
```

```
"MyTypes,Version=2.0.1234.0,Culture=neutral,PublicKeyToken=b77a5c561934e089"
```

强名称程序集（续）

2.6.5 强名称程序集

- 创建强名称组件

- 产生私钥

```
sn -k OrgKey.snk
```

- 向资源文件中指定版本和私钥信息代码

```
#if STRONG  
[assembly:System.Reflection.AssemblyVersion("1.0.0.0")]  
[assembly:System.Reflection.AssemblyKeyFile("orgKey.snk")]  
#endif
```

- 编译组件

```
csc /define:STRONG /target:library /out:AReverser.dll↵  
AReverser.cs
```

部署共享组件

2.6.6 部署共享组件

- 把强名称组件安装到全局程序集缓存中

```
gacutil /i AReverser.dll
```

- 检查全局程序集缓存

```
gacutil /l
```

- 开发者可用 `gacutil /u` 命令自动删除共享组件文件

```
gacutil /u AReverser
```

部署共享组件（续）

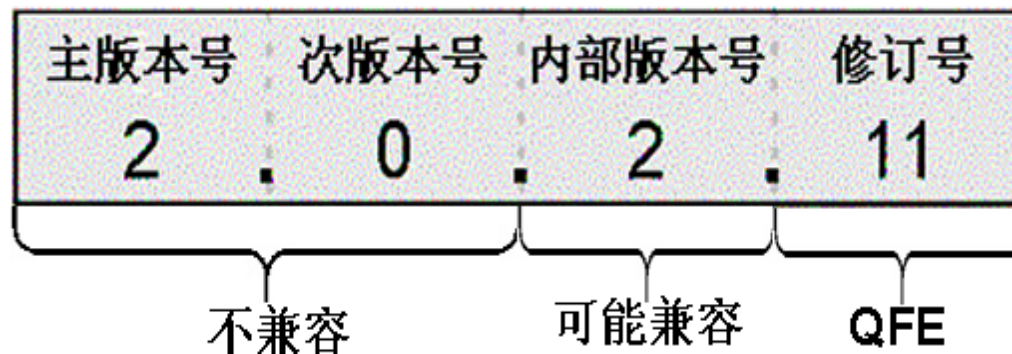
2.6.6 部署共享组件

- 全局程序集缓存安装在 \WindowsDirectory\Assembly 目录中
- 系统管理员可以从 \WindowsDirectory\Assembly 目录中直接删除组件
- 只有具有管理员权限的用户才能把强签名组件安装到全局程序集缓存或删除该组件

版本化程序集

2.6.7 版本化程序集

- 版本号由 4 部分的数字组成，每个部分代表不同的含义，被逻辑上分成三个部分，代表含义如下：
 - 不兼容，产品的主要新版本就属于这种情况
 - 可能兼容，服务包或每日生成的新版本就属于这种情况
 - 快速修复工程（QFE），如紧急安全修补就属于这种情况



版本化程序集（续）

2.6.7 版本化程序集

- 应用程序需要绑定到适当版本的共享程序集
- 每一程序集都有一个特定的兼容性版本号作为其标识的一部分
- 具有不同兼容版本号的共享组件是完全不同的程序集
- 默认情况下应用程序寻找创建时指定版本的程序集
 - 除非有明确的策略指定要加载哪个版本的程序集

版本化程序集（续）

2.6.7 版本化程序集

- 除了版本号，程序集还用区域信息作为程序集的标示，默认是区域信息中性的
 - 不同的区域信息的程序集代表不同的程序集
 - 下表列出了几种区域信息的例子

主标签	次标签	区域信息
de	(none)	德语
de	AT	奥地利德语
de	CH	瑞士德语
en	(none)	英语
en	GB	英国英语
en	US	美国英语

创建强名称程序集的多个版本

2.6.8 创建强名称程序集的多个版本

- 用新的公钥—私钥对创建两个不同版本的 AReverser 组件
 - 一个指定版本 2.0.0.0 ， 另一个指定 2.0.1.0
为 AReverser_v2.0.0.0\AReverser.vb 指定版本

```
#if STRONG  
[assembly:System.Reflection.AssemblyVersion("2.0.0.0")]  
[assembly:System.Reflection.AssemblyKeyFile("orgVerKey.snk")]  
#endif
```

- 编译

```
csc /define:STRONG /target:library /out:AReverser_v2.0.0.0\AReverser.dll  
AReverser_v2.0.0.0\AReverser.cs
```

创建强名称程序集的多个版本（续）

2.6.8 创建强名称程序集的多个版本

- 用 Ildasm.exe 来查看两个版本的不同之处
 - 注意公钥和版本号

```
.assembly AReverser
{
  ...
  .publickey = (00 24 ... 82 B1 F2 A0 )
  .hash algorithm 0x00008004
  .ver 2:0:1:0
}
```

- **Use Ildasm.exe to Verify Strong Names for Both Assemblies**
 - Different .originator property
 - Updated version, 2.0.0.0 or 2.0.1.0

绑定策略

2.6.9 绑定策略

- 允许程序集引用在应用程序部署之后修改
 - 允许程序集引用（在编译时指定）在应用程序部署之后修改，
 - 而不必重新编译涉及的程序集
- 策略解析可能发生在下列阶段
 - 1. 应用程序策略解析
 - 2. 发行者策略解析
 - 3. 管理员策略解析
- 在每个阶段XML配置文件被读取
 - 注意：XML 是大小写敏感的
- 非强名称组件的版本号不检查
- 配置文件中的标签包括
 - privatePath
 - bindingRedirect

绑定策略（续）

2.6.9 绑定策略

- 发行者策略解析允许共享组件供应商在他们的软件不同修订号之间作出兼容性声明
 - 因为发行者策略影响整个系统的版本重定向，所以它受管理员策略的制约
 - 基于上面的原因，程序集应与应用程序分开安装，这一点很重要
- 管理员策略解析是绑定策略解析过程中的最后阶段，也是最具决定性的阶段
 - 管理员策略文件位于
WindowsDirectory\Microsoft.NET\Framework\v1.0.FinalBuildNumber\CONFIG 目录中，叫做 Machine.config

部署多版本的强名称程序集

2.6.10 部署多版本的强名称程序集

- 通过配置文件来修改版本策略
 - 这些配置文件包括应用程序配置文件，发行者配置文件和系统管理员配置文件
 - 以上配置文件指定的版本策略使保存在客户端程序集清单中的版本策略无效
- 编译客户端 VerClient.exe，指定版本 2.0.0.0 的 AReverser 组件

```
csc /reference: MyStringer\Stringer.dll↵  
/reference: AReverser_v2.0.0.0\AReverser.dll VerClient.cs
```

部署多版本的强名称程序集（续）

2.6.10 部署多版本的强名称程序集

- 用版本策略在运行时指定程序集绑定
- 绑定到指定版本的程序集

```
<bindingRedirect  
  oldVersion="2.0.0.0-2.0.0.9" newVersion="2.0.1.0"  
>
```

- 课后习题第二题中将使用这种技术

打包和部署工具

2.6.11 打包和部署工具

- 程序集链接器 (Al.exe)
- 全局程序集缓存工具 (Gacutil.exe)
- MSIL 反编译器 (Ildasm.exe)
- 强名称 (Sn.exe)
- 本地映像文件生成器 (Ngen.exe)
- 程序集绑定日志查看器 (Fuslogvw.exe)
- .NET Framework 部署工具 (Mscorcfg.msc)
- 代码访问安全策略工具 (Caspol.exe)

回顾

学习完本章后，将能够：

- 创建简单的组件
- 实施结构化异常处理
- 创建调用组件的简单控制台应用程序
- 使用 Windows 窗体类库创建 Web 客户端应用程序
- 创建 Web 窗体应用程序并调用组件
- 打包和部署简单应用程序和组件化应用程序
- 创建强名称程序集
- 向全局程序集缓存安装强名称程序集
- 配置应用程序使其绑定特定位置和版本的程序集