您的潜力，我们的动力

Microsoft
微软（中国）有限公司

《现代软件开发——使用.NET与C#》系列第九讲

基于组件的程序设计

俞晖
MSDN Online Program Manager
dolphin@vip.163.com

msdn

MSDN Webcasts

《现代软件开发——使用.NET与C#》系列

- 回顾上节课的内容 （多层应用程序设计）
- 本节课讲述在C#中的基于组件的开发的问题
- **LEVEL 300**
- 基础:
  - 初级编程
  - 应用程序设计

Date	Topic
2005年8月16日	*ADO.NET*
2005年9月13日	Multi-tier Application Design
2005年9月29日	**Component-based Design and Programming**
.	.

Today's Objectives

“基于组件编程有很多优势：代码重用，团队支持，多语言开发和独立更新。在1990年，COM是微软流行的基于组件开发的模型。.NET替代了COM，提供了一个全新模型：语言独立的，注册自由，版本明确的……”

[Hummel]

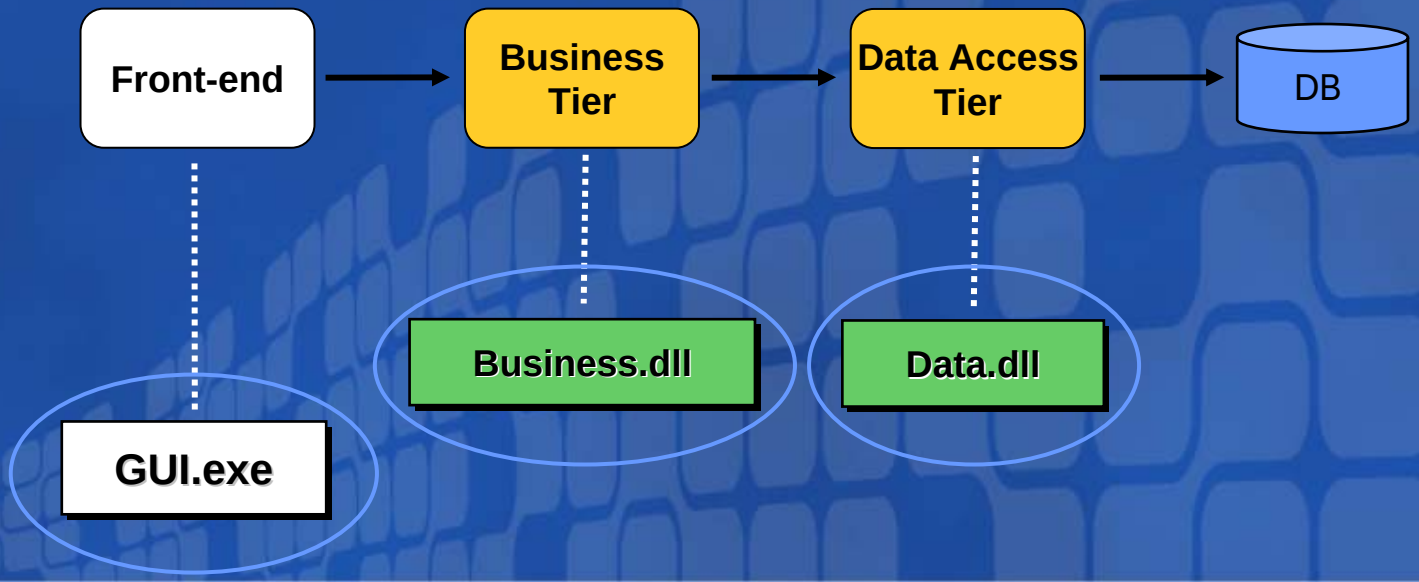
- 议题：
  - 基于组件的开发
  - 组件命名和部署

日程

- *建立组件*
- 部署组件
- 配置文件

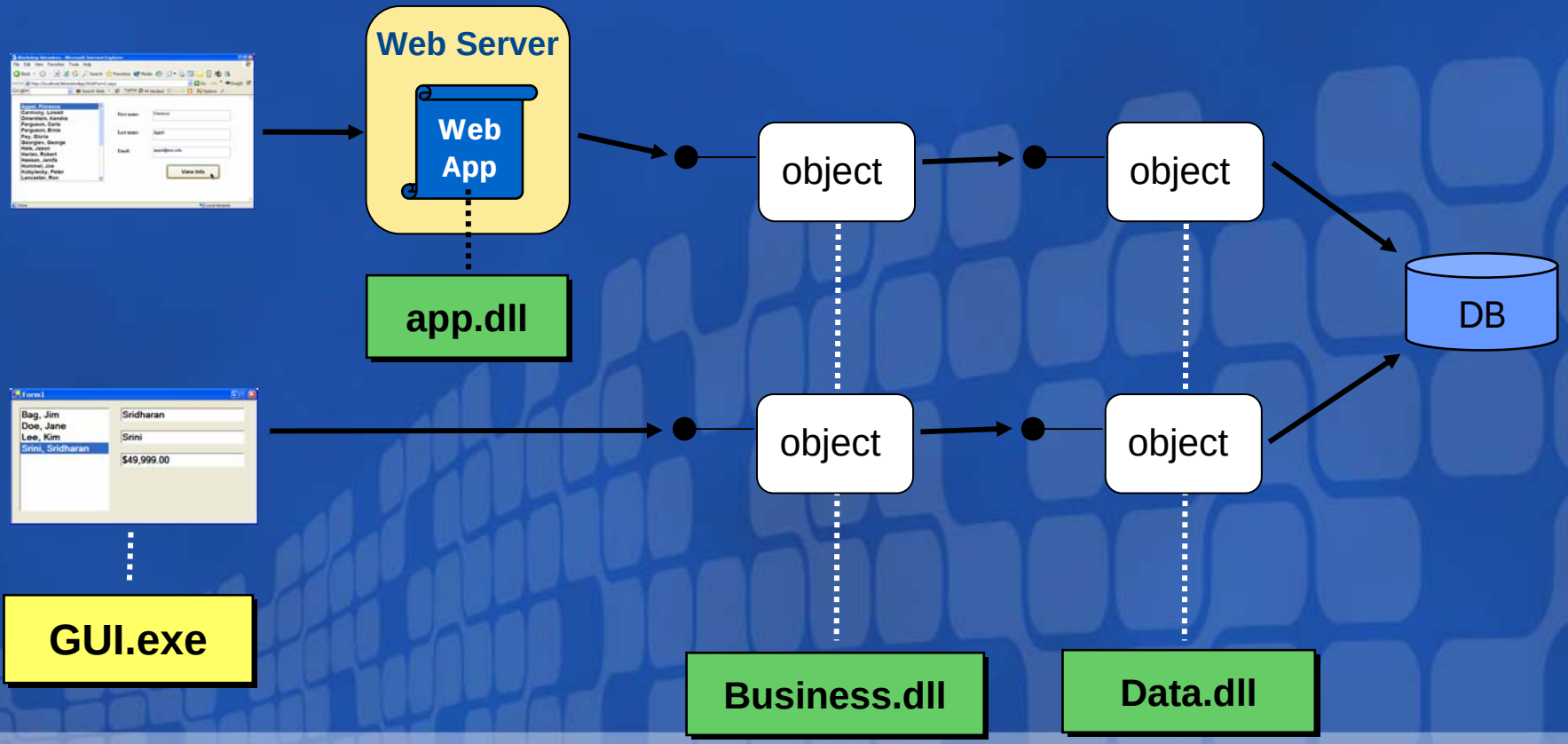
组件

- 什么是组件？
  - 组件 == 编译后实体
  - 在.NET中，组件 == Assembly (exe / .dll)
- 组件与 *物理* 打包相关.....



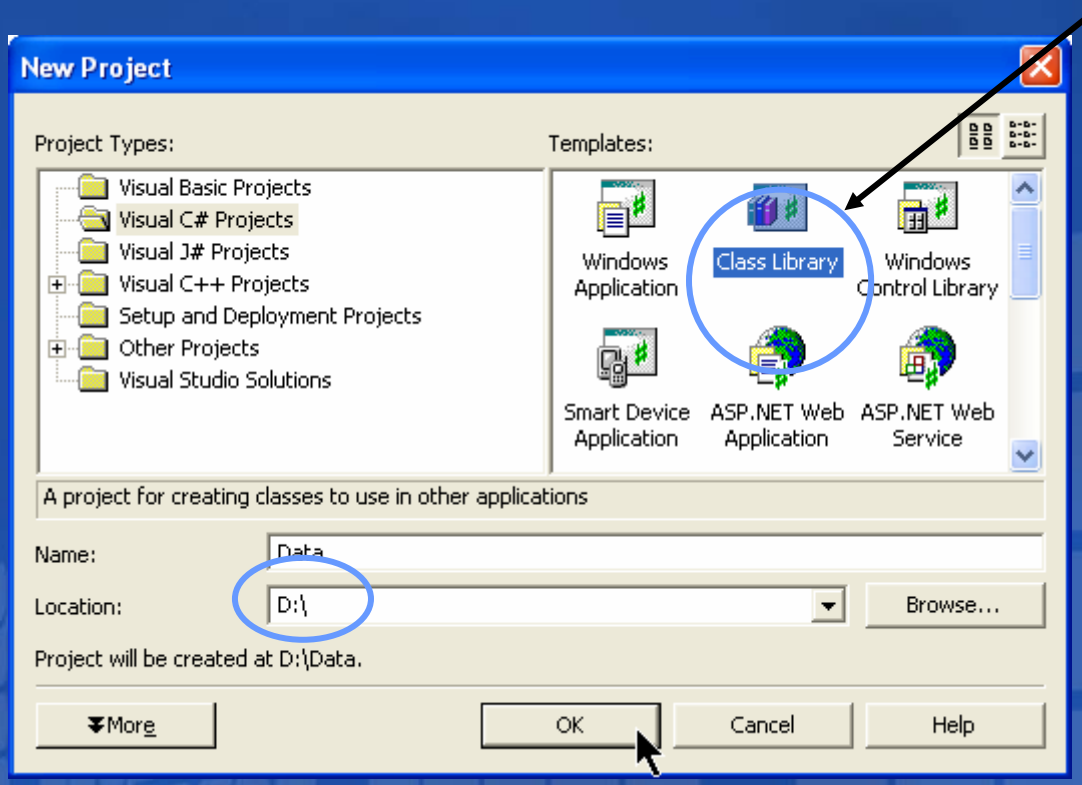
例子?

- 多层应用程序设计是一个很好的表现组件的例子：



创建组件

- 在VS .NET中创建 *Class Library*
  - class libraries 将被编译成DLLs



如何实现？

- 仅仅按照标准的设计和编码过程
  - 使用 *名称空间* 进行逻辑组织
  - 使用类和OO的设计

```
namespace CompanyX.DataAccessTier
{
  public class DataAccess
  {
    .
    .
    .
```

```
namespace CompanyX.DataAccessTier
{
  public class MSAccessDB
  {
    .
    .
    .
```

```
namespace CompanyX.DataAccessTier
{
  public class SqlServerDB
  {
    .
    .
    .
```

Internal 关键字

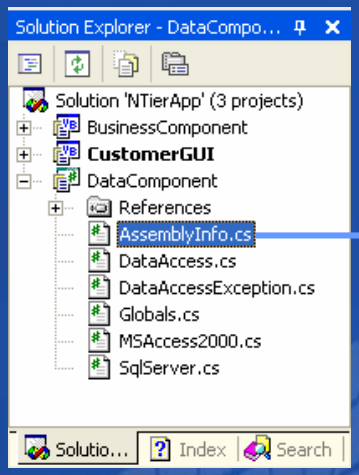
- *使用 internal 关键字来限制访问*
  - 任何被标识为 *internal* 的项只能被组件内部的类访问
  - 在组件外部没有权限访问.....

```
namespace CompanyX.DataAccessTier
{
  public class DataAccess
  {
    .
    .
    .
    internal void LogIt(string msg)
    {
      ...
    }
```

```
namespace CompanyX.DataAccessTier
{
  internal class HelperClass
  {
    .
    .
    .
```

版本

- 版本号：
  - *major.minor.build.revision*
- *AssemblyInfo.cs 中的属性可以控制*
  - 在编译时被设置到Assembly中
  - 默认情况下，版本号在每次build时都会改变




```
using System.Reflection;
using System.Runtime.CompilerServices;

[assembly: AssemblyTitle("")]
[assembly: AssemblyDescription("")]
[assembly: AssemblyCompany("")]
  .
  .
  .

// You can specify values or assign default Revision and Build values
// on each build using '*' as shown below:
[assembly: AssemblyVersion("1.0.*.*")]
```

谁来控制版本？

- 你来!
  - 你来决定何时应该改变版本号（任何时候）
  - 你需要保证在不同版本中的兼容性（如果需要）
- 默认情况下，.NET & CLR 会忽略版本号
  - 如果 .EXE 需要 .DLL，那么任何版本的.DLL都可以
- 如果组件有强命名/强名称(strong name)，那么
  - 如果.EXE 提及需要1.0.3.12 版本的 .DLL，那么它必须查找到 1.0.3.12 版本的 .DLL...

强命名（Strong Name）？

- Assembly 在 .NET 的命名有四部分：
  - friendly, human readable name
  - culture
  - version #
  - public key token

```
DataComponent,
Culture=neutral,
Version=1.0.0.0,
PublicKeyToken=1234567890abcdef
```

- Assembly 如果有公钥token，那么它有*强命名*——这表示它被私钥数字签名过

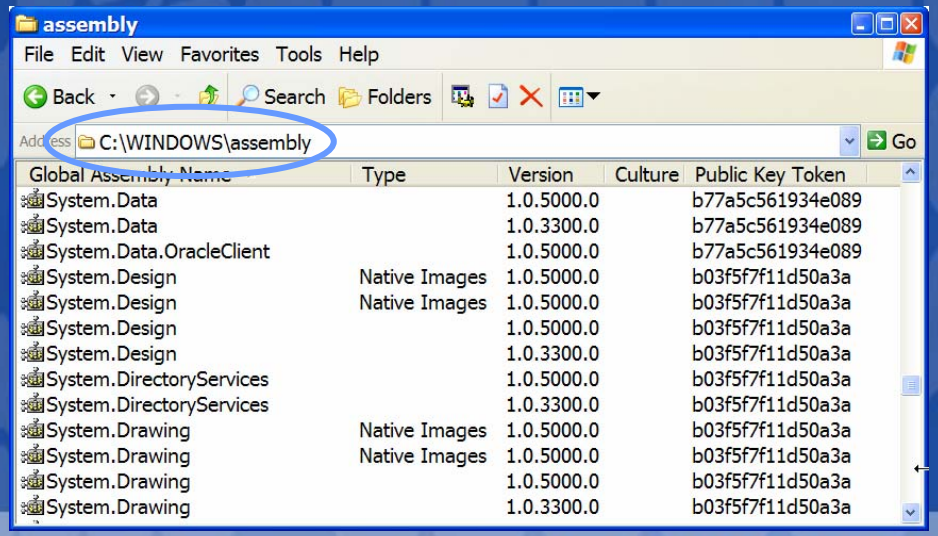
强命名的好处？

- 安全性，组件无法篡改!
- 组件的多个版本可以共存！
  - 应用程序可以使用它建立时的组件版本，而不是第一个找到的DLL。

Demonstration One

demo

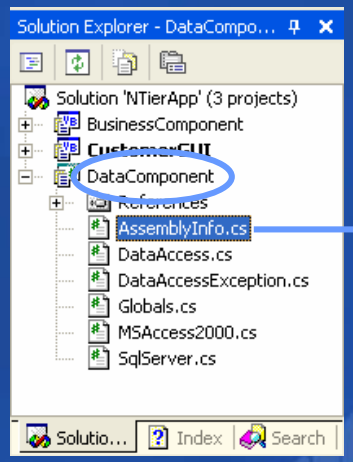
强命名...



命名Assembly

```
DataComponent,
Culture=neutral,
Version=1.0.0.0,
PublicKeyToken=1234567890abcdef
```

- Friendly name == Visual Studio project name
- 通过属性来设置其它的部分：



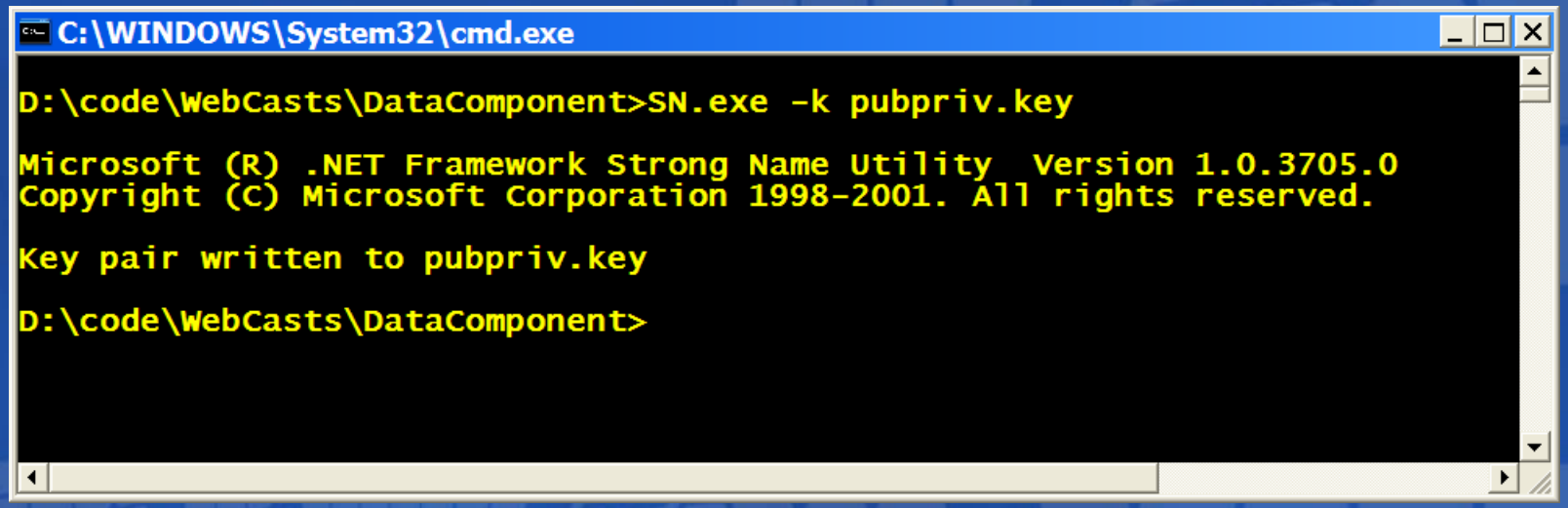

```
using System.Reflection;
using System.Runtime.CompilerServices;

[assembly: AssemblyTitle("")]
  .
  .
  .

[assembly: AssemblyCulture("")]
[assembly: AssemblyVersion("1.0.0.0")]
[assembly: AssemblyKeyFile("..\\..\\pubpriv.key")]
```

4步用来强命名

- 创建强命名组件的步骤：
  1. 生成一组 **public-private key pair**
  2. 将其通过**AssemblyInfo.cs** 的属性应用到组件中。
  3. 重新编译 **assembly**
  4. 重新编译客户端

建立Public/Private Key 文件

- 使用 .NET “SN” 命令行用法
  - -k 选项生成 key pair
  - 把该文件放置在VS solution / project目录下



保护 Key 文件！

- 私钥 key 的安全-recompile
  - 必须保证 key 文件是秘密的
- 每个程序员是否需要一份key文件的拷贝？
  - 可能不会...
  - 因为.NET 支持后签名：

```
.
.
.
[assembly: AssemblyKeyFile("..\\..\\pubpriv.key")]
[assembly: AssemblyDelaySign(true)]
```

    - *在发布之前，部署团队签署 assembly …*

Demonstration Two

demo

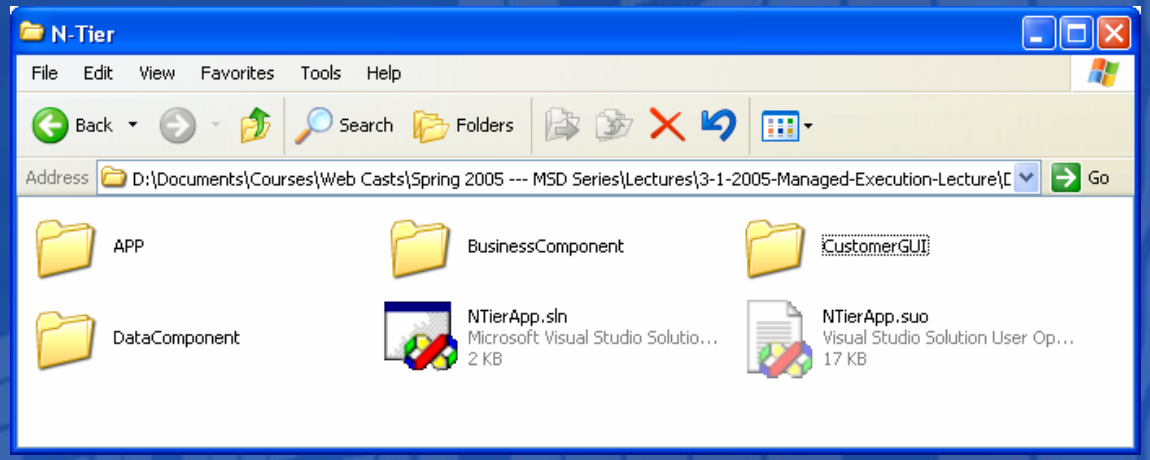
基于组件的应用程序...

VS .NET中基于组件的应用程序

- 两种方法来开发基于组件的应用程序：
  1. *单一的解决方案中包含多个项目*– 这是最简单的方法，如果你有所有的源代码
  2. *多个解决方案，每个都含有一个项目*–典型地在一个大型的项目中或者你没有所有的源代码

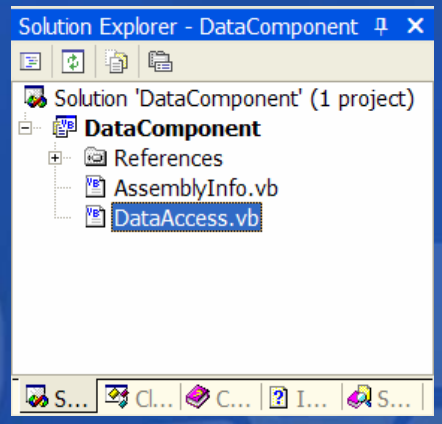
单一解决方案

1. 正常建立组件项目...
2. 创建空解决方案
   - New project, Visual Studio Solutions, Blank Solution
3. 把每个项目添加到该解决方案中
   - File menu, Add Project, Existing Project...



调试单一项目

- 如何自我调试一个项目？
- 配置 **Visual Studio** 来运行**.EXE:**
  - **Project Properties, then**
  - **Configuration Projecties, then**
  - **Debugging:**
    - **Debug Mode: "Program"**
    - **Start Application: "D:\...\GUI.exe"**
  - **Set breakpoints in your code...**
  - **Run!**

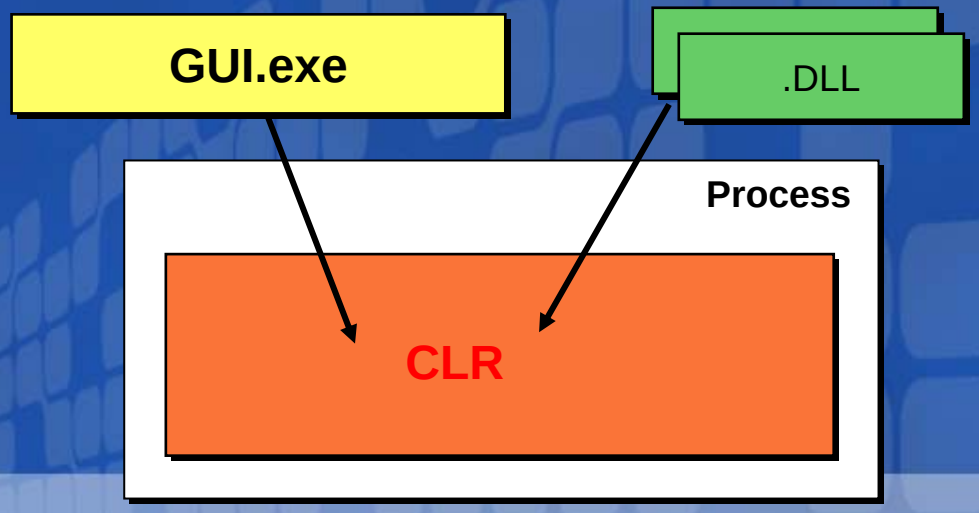


日程

- 建立组件
- *部署组件*
- 配置文件

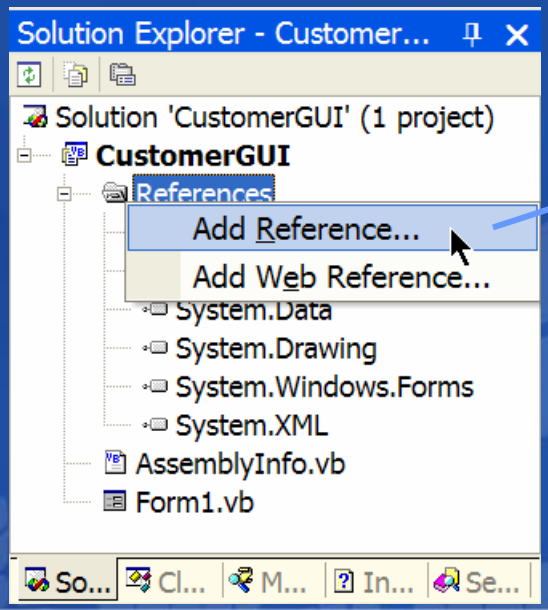
部署

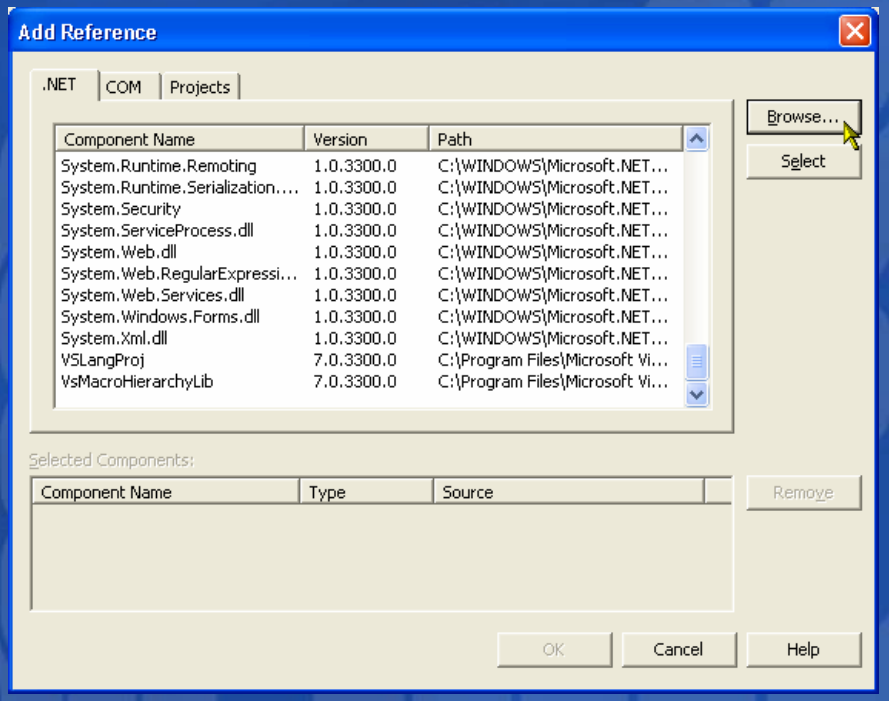
- 前提：
  - .NET 必须安装在目标计算机中
  - 应用程序必须安装在目标计算机中
  - 应用程序必须在运行时态可以定位组件



引用

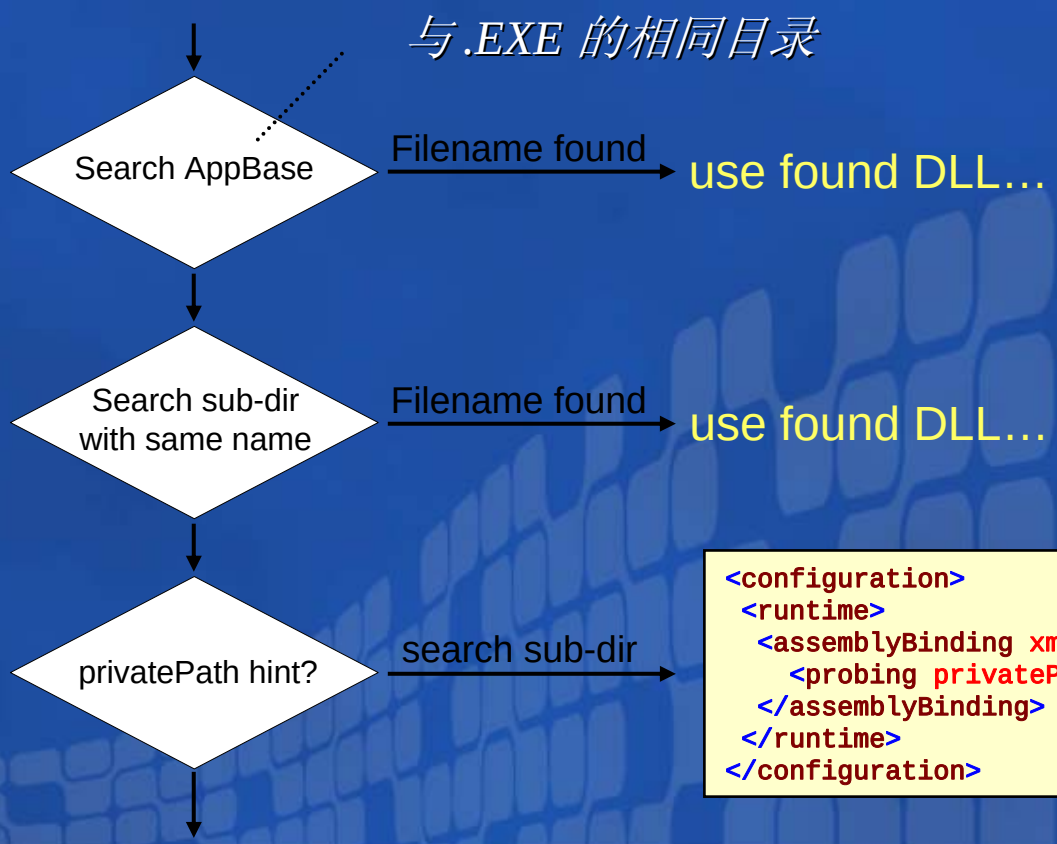
- 引用你需要使用的组件
  - 无论是DLL的文件，还是VS项目中的文件





如果不使用强命名...

- 定位assembly的算法：




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

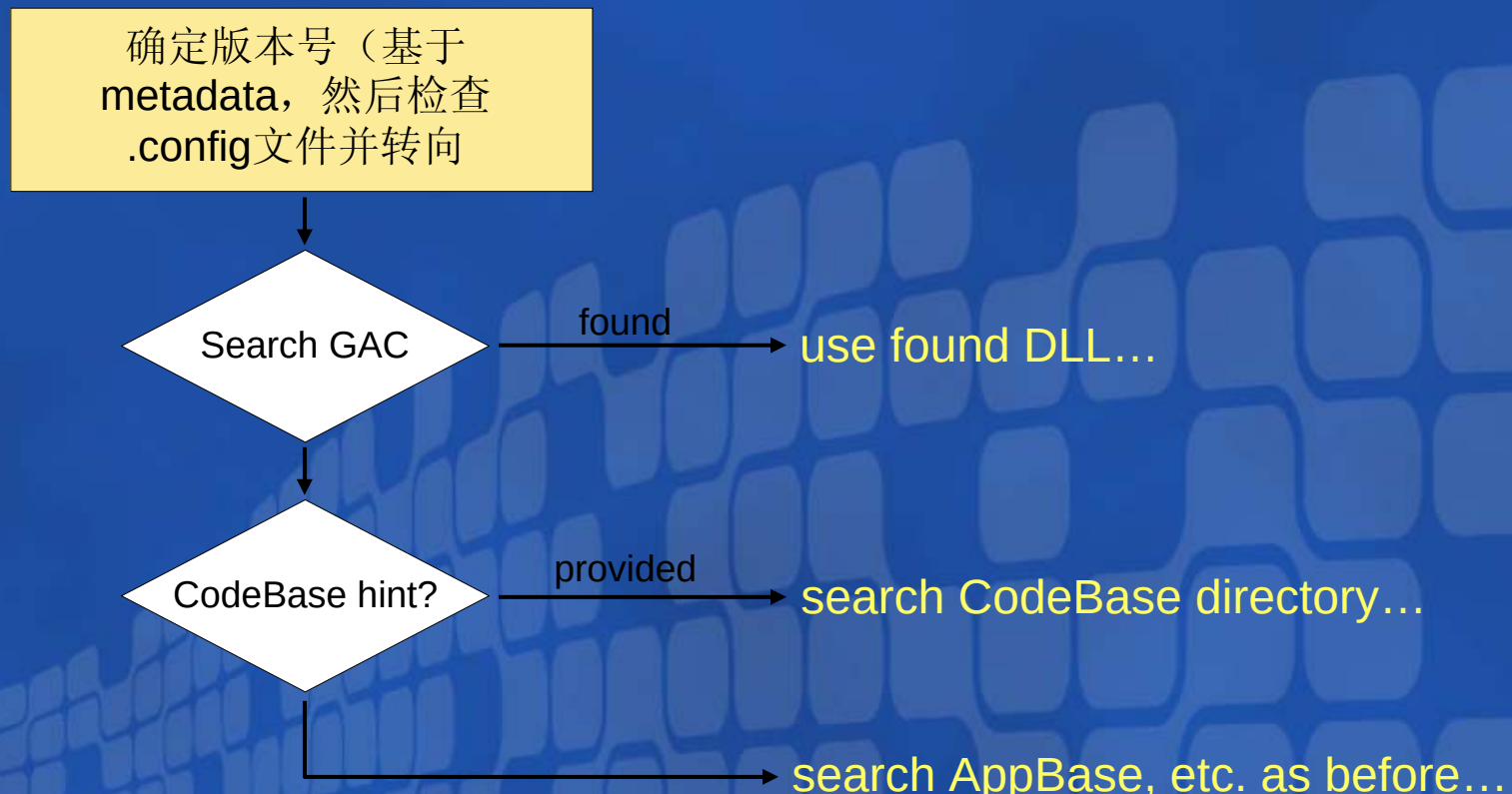
GUI.exe.config

部署选项?

1. “XCopy” 部署:
   - 安装 EXE, DLLs 和支持文件到单一目录
   - .NET 使用AppBase定位所有的组件和文件
2. “Zero-touch” 部署:
   - 为.EXE设置一个URL: http://server/myapp/app.exe
   - 客户端浏览 URL
   - *.NET 在Web服务器远程目录中设置 AppBase!*
   - .NET 从AppBase下载 .EXE 和 .DLLs

如果使用了强命名...

- 定位assembly的算法：



版本转向

- 应用程序的 .config 文件将覆盖 .EXE 的 manifest
- 例:
  - .EXE 引用 1.0.0.0 Business Tier
  - 我们要求.EXE 使用2.0.0.0

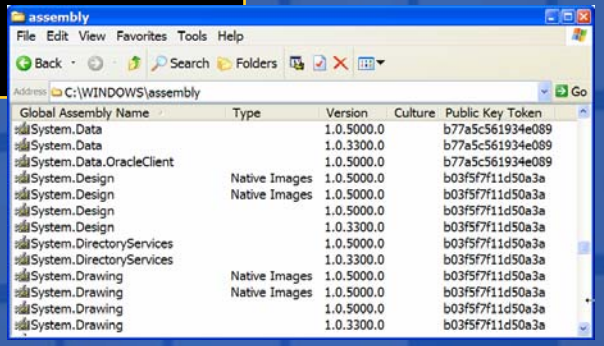
```
<configuration>
 <runtime>
  <assemblyBinding xmlns="urn:schemas-microsoft-com:asm.v1">
   <dependentAssembly>
      <assemblyIdentity name="BusinessComponent"
                        publicKeyToken="1234567890abcdef" />
      <bindingRedirect oldVersion="1.0.0.0“ newVersion="2.0.0.0" />
   </dependentAssembly>
  </assemblyBinding>
 </runtime>
</configuration>
```

GUI.exe.config

安装到GAC

- 如何把组件放置进GAC中？
  - 组件必须是强命名的
  - 使用 **.NET “gacutil”** 命令行工具：

```
> gacutil /i BusinessComponent.dll
```



  - 优势?
    - 共享 **DLLs**
    - 并行安装不同版本
    - 更快的下载时间 （在安装时刻，组件完整的检查）

CodeBase

- 可以转向到不同的目录和机器
  - DLLs 保存用户的 internet 下载缓存
  - 对于每个版本只有一个codebase

```
<configuration>
 <runtime>
  <assemblyBinding xmlns="urn:schemas-microsoft-com:asm.v1">
   <dependentAssembly>
      <assemblyIdentity
        name="BusinessComponent"
        publicKeyToken="1234567890abcdef" />
      <codeBase
        version="2.0.0.0"
        href="http://www.company.com/downloads/BusinessComponent.dll" />
   </dependentAssembly>
  </assemblyBinding>
 </runtime>
</configuration>
```

GUI.exe.config

日程

- 创建组件
- 部署组件
- *配置文件*

Config 文件

- 允许我们覆写一些.NET默认的行为
- 应用程序提供配置信息的机制
  - 例如：一般的练习中通过配置文件指定数据库连接字符串

```
<configuration>

   <appSettings>
      <add key="ConnectionString" value="..." />
   </appSettings>
   .
   .
   .
```

```
using CFG=System.Configuration.ConfigurationSettings;
 .
 .
 .
sConnect = CFG.AppSettings[“ConnectionString”].ToString();
```

但是配置文件有缺点...

潜在的配置冲突：

- machine-wide config file (*machine.config*)
- User's can have a config file
- Applications can have a config file
- Components can have a *policy* file that overrides app

其它参考资源

- **Webcast:**
  - http://www.microsoft.com/china/msdn/webcasts
- **Resources:**
  - MSDN Library
    http://www.microsoft.com/china/MSDN/library/default.mspx
- **MSDN flash:**
  - http://www.microsoft.com/msdn/china/newsletter/
- **Applied MICROSOFT .NET Framework Programming** *Jeffery Richter*

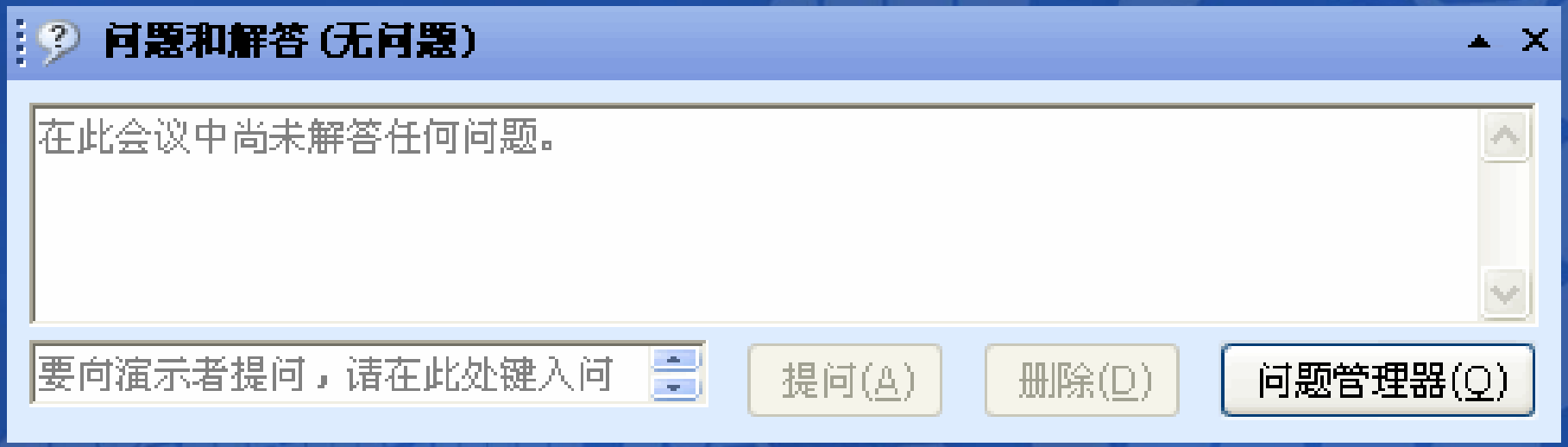
感谢大家的参与!

- 非常感谢大家的参与
- 您的反馈对我们来说是非常重要的

Date	Topic
2005年9月	基于组件的应用程序开发
2005年10月	C#一些技巧的开发

Question & Answer

如需提出问题，请单击“提问”按钮并在随后显示的浮动面板中输入问题内容。一旦完成问题输入后，请单击“提问”按钮。




您的潜力，我们的动力
Microsoft
微软（中国）有限公司
Microsoft
msdn
MSDN Webcasts