您的潜力，我们的动力

Microsoft
微软（中国）有限公司

设计安全的桌面智能客户端应用程序

付仲恺
微软特邀开发专家

msdn

MSDN Webcasts

议题

- 安全性概览
- 验证和授权
- 自定义验证
- 私密信息和完整性
- Microsoft Security Application Blocks

课程准备

- 您需要具备：
  - .NET开发人员
  - 熟悉桌面应用程序的开发

Level 300

议题

- *安全性概览*
- 验证和授权
- 自定义验证
- 私密信息和完整性
- Microsoft Security Application Blocks

概览

- 关于安全的五条原则：
  - 验证
  - 授权
  - 私密信息保护
  - 数据完整性
  - 漏洞

概览

1. *验证：* 用户是谁？
2. *授权：* 他们都有哪些权限？
3. *私密信息：* 是否是私有数据？
4. *完整性：* 数据是否易于被篡改？
5. *漏洞：* 能否避免已知漏洞？

如何处理这些原则

1. *验证？* 登陆 / 密码
2. *授权？* 检查用户的权限
3. *私密信息？* 加密
4. *完整性？* 数字签名
5. *漏洞？* 在设计和编码时尽量避免

议题

- 安全性概要
- *验证和授权*
- 自定义验证
- 私密信息和完整性
- Microsoft Security Application Blocks

方法

- 在.NET 1.1/2.0中的验证/授权：
  1. *基于角色*
     - 基于用户身份标示符的安全性
     - 用户通过密码（或者指纹等其它方式）来提供他们的标示符
     - 用户基于角色授权进行操作
       - » 类似于“组”的概念
     - 例如：Microsoft® Windows®
  2. *基于代码*
     - 安全性基于代码自身，而不是运行它的用户
     - 代码基于证据来标示
       （例如：程序的下载位置）
     - 代码的授权基于：
       - » 从调用代码继承 / 授予权限
       - » 从Enterprise，Machine，User，Application获得策略
     - 例如：Code Access Security (CAS)

考虑下列情况...

- 是否正在构建整个应用程序？
  - 基于角色的安全...
- 是否要调用不完全信任的组件？
- 构建出来的组件是否用于外部环境？
  - 基于代码的安全性...

.NET中基于角色的安全性

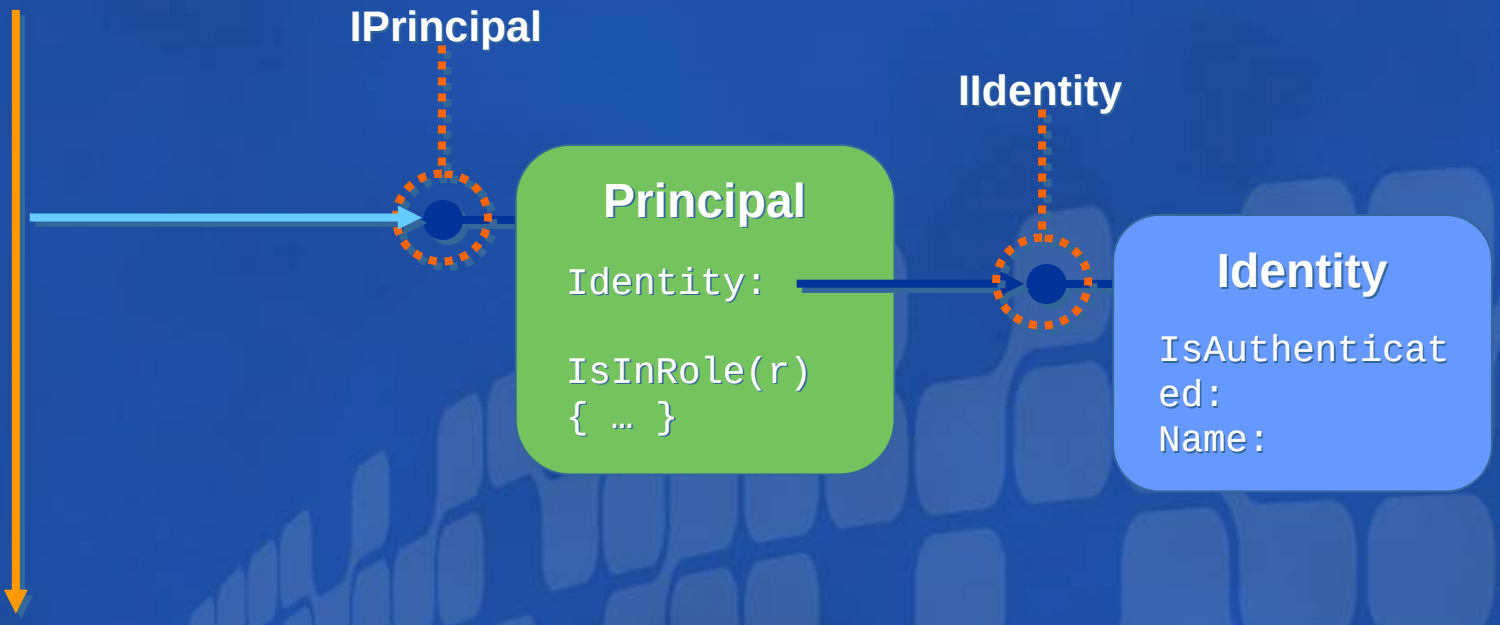
- 基于两个概念：
  - 用户的标识符（*Identity*）
  - *Principal* 会记录下用户的标示符和所扮演的角色
- 方法：
  - 在程序启动的时候验证用户的标识符
  - 根据用户角色，授予用户的行为权利
- 例如：
  - “*fzk*” 通过密码获得验证
  - “*fzk*” 是“*Managers*”组的一员，因此可以浏览敏感信息

demo

Principal, Identity 和 Roles…

演示回顾（1）

当前线程

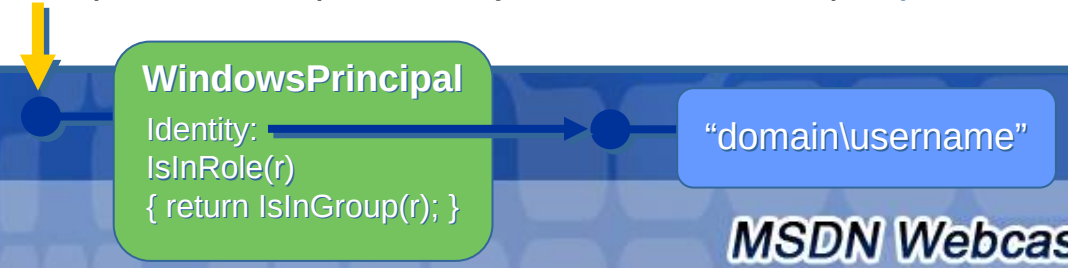



```
// C#:
System.Security.Principal.IPrincipal  principal;
System.Security.Principal.IIdentity   user;

principal = System.Threading.Thread.CurrentPrincipal;
user = principal.Identity;
MessageBox.Show(user.Name + ": " + principal.IsInRole("Managers"));
```

演示回顾（2）

- 基于接口的可替换模型：
  - 可以使用 *Windows authentication* 来创建对象
  - 你能够创建你自己的验证子系统
- 使用Windows验证：
  - 标示符 = Windows用户名，角色 = Windows组

```
// C#:
static void Main()
{
   // 对当前用户使用Windows验证方式;
   System.AppDomain.CurrentDomain.SetPrincipalPolicy(
      System.Security.Principal.PrincipalPolicy.WindowsPrincipal);
```



演示回顾（3）

- .NET 提供 *命令式* 和 *声明式* 方式
- 命令式： 通过代码实现，细粒度

```
// C#:
private void ViewSensitiveInfoMenuItem_Click(...)
{
    if (!System.Threading.Thread.CurrentPrincipal.IsInRole("Managers"))
        throw new System.Security.SecurityException("Only Managers may...");
```

- 声明式风格：通过声明实现， .NET自动完成
  - *较粗粒度（例如：通过这种方式来保证整个类的安全性）*

```
// C#:
[System.Security.Permissions.PrincipalPermission(
        System.Security.Permissions.SecurityAction.Demand,Role="Managers")]
private void ViewSensitiveInfoMenuItem_Click(...)
{
    // 如果用户属于Manager组，则下面的代码允许执行…
    ...
```

演示总结（4）

- 基于角色的安全性 != Windows 安全
- 例如：

```
//
// Whose “identity” is used to check if we have permission to open this file?
//
System.IO.FileStream  file;
file = new System.IO.FileStream(“filename”, FileMode.Open, FileAccess.Read);
```

  - 线程的 *CurrentPrincipal.Identity*，或者进程的标识符？
  - *暗示？基于角色的安全性是内建于.NET的，其不会影响通过操作系统来实现的安全性*

议题

- 安全性概览
- 验证和授权
- *自定义验证*
- 私密信息和完整性
- Microsoft Security Application Blocks

创建自定义验证...

- 是否能够确信？
  - 用户是否需要其他类型的验证？
  - 是否愿意由开发人员自己来处理密码？
    - » *添加salt，绝对不要明文保存密码，或者明文传输密码...*

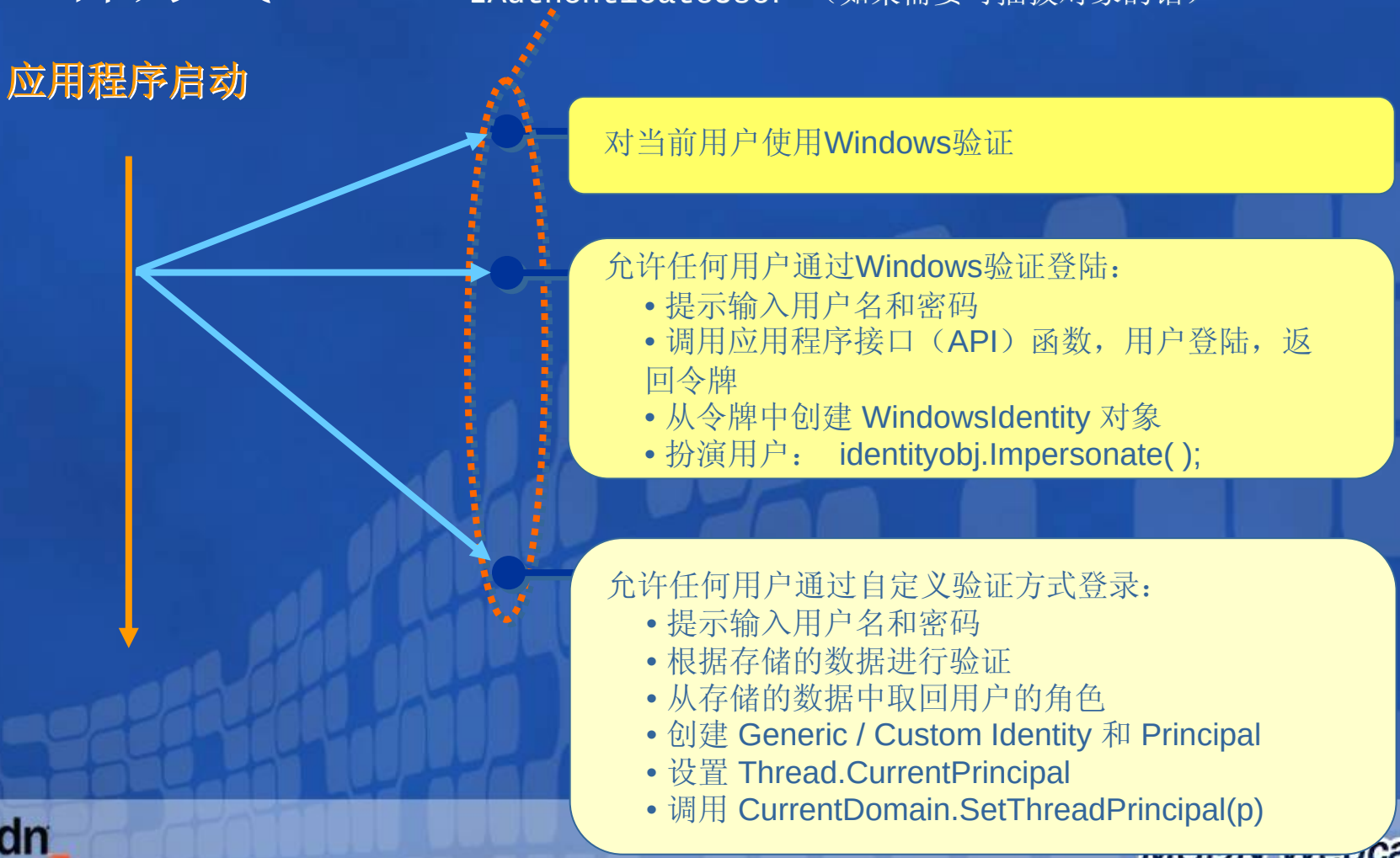
```
using SSP = System.Security.Principal;
static void Main()
{
   // 我们将实现自己的安全子系统
   System.AppDomain.CurrentDomain.SetPrincipalPolicy(
      System.Security.Principal.PrincipalPolicy.UnauthenticatedPrincipal);

   string username = <<authenticate user>>;
   string[] roles  = <<retrieve user's roles>>
   SSP.GenericIdentity i  = new SSP.GenericIdentity(username);
   SSP.GenericPrincipal p = new SSP.GenericPrincipal(i, roles);

   System.Threading.Thread.CurrentPrincipal = p;  // 连接到当前线程
   System.AppDomain.CurrentDomain.SetThreadPrincipal(p);  // 以后的线程也采用同样的设置
```

权衡利弊

- 优势：
  - 不依赖于Windows帐户，组，活动目录等等。
- 劣势：
  - 需要实现验证子系统
  - 需要认真考虑子系统的安全性
- 考虑：*创建一个智能客户端应用程序？*
  - 需要运行在 Microsoft Windows 和 .NET 之上…
  - … 如果允许的话，最好使用Windows验证

验证体系结构

- 3 种方式：


IAuthenticateUser （如果需要可插拔对象的话）
应用程序启动
对当前用户使用Windows验证
允许任何用户通过Windows验证登陆：
• 提示输入用户名和密码
• 调用应用程序接口（API）函数，用户登陆，返回令牌
• 从令牌中创建 WindowsIdentity 对象
• 扮演用户： identityobj.Impersonate( );
允许任何用户通过自定义验证方式登录：
• 提示输入用户名和密码
• 根据存储的数据进行验证
• 从存储的数据中取回用户的角色
• 创建 Generic / Custom Identity 和 Principal
• 设置 Thread.CurrentPrincipal
• 调用 CurrentDomain.SetThreadPrincipal(p)

安全框架

- IssueVision 应用程序（Microsoft® Visual Basic® 和 Microsoft® Visual C#®）：
  - 展示了如何设计数据存储以保存加密密码
  - http://www.windowsforms.net/Applications/application.aspx?PageID=40&tabindex=8
- CSLA:
  - Rocky Lhotka 的基于组件的，可扩展的，逻辑体系结构
  - 展示了如何设计自定义 Identity 和 Principal 对象
  - 书籍： “*VB .NET Business Objects*” 和 “*Expert C# Business Objects*”
- ASP.NET 2.0 自定义验证管理：
  - ASP.NET 2.0 为自定义，非Windows验证提供了扩展支持
  - “*Security: Manage Custom Credentials in Windows Forms*”, MSDN Magazine, Apr 2005

提示输入用户名 / 密码

- 能够很容易地创建自己的登陆页面
  - 需要例子？详见 *IssueVision* 的源代码
- 调用应用程序编程接口（API）中的 *CredUIPromptForCredentials*函数
  - 在 Windows XP 和 2003 上可用
  - K. Brown, "*.NET Developer's Guide to Windows Security*"
    - Item 71
    - http://www.pluralsight.com/wiki/default.aspx/Keith.GuideBook.HomePage

用户名 / 密码 数据存储

- 两张表：用户表和角色表
  - 用户表：UserID, Username, PwdHash, Salt
  - 角色表： UserID, Role
- 避免通过明文的方式保存密码
  - 加密密码，例如：通过单向Hash运算
  - “salted”用于防范字典攻击

```
// C#和.NET 2.0对用户的密码使用基于工业标准的单向散列算法:
System.Security.Cryptography.Rfc2898DeriveBytes db = new
       System.Security.Cryptography.Rfc2898DeriveBytes(pwd, 32, 1000);

byte[]  salt, pwdhash;
salt    = db.Salt;
pwdhash = db.GetBytes(32);
// 如果需要添加用户，保存salt和散列后的密码到数据存储设备
// 如果需要验证用户，从数据存储设备中读出slat，使用salt来计算出密码的散列值，并且与
保存在数据存储设备中的密文密码进行比较
```

demo

创建自定义验证系统...

议题

- 安全性概览
- 验证和授权
- 自定义验证
- *私密信息和完整性*
- Microsoft Security Application Blocks

五项原则

1. *验证：* 用户是谁？
2. *授权：* 他们都有哪些权限？
3. *私密信息：* 是否是私密数据？
4. *完整性：* 数据是否易于被篡改？
5. *漏洞：* 能否避免已知漏洞？

私密信息

- 加密是保护私密数据的基本方法
  - encrypted-data = *encrypt*(data, key)
  - data = *decrypt*(encrypted-data, key)
- 问题：
  - 选择合适的算法
  - 保证密钥的安全性
- 解决方案？
  - *Data Protection API* (DPAPI)
  - 强加密算法
  - 密钥基于用户的信任，由操作系统保护

数据保护 API

- 在 .NET 2.0中, DPAPI 已经被托管代码包装起来：

```
using SSC = System.Security.Cryptography;          // 需要添加引用 System.Security assembly

byte[]  data, salt, encrypted;
data = ... ;   // = System.Text.Encoding.ASCII.GetBytes("testing 123");
salt = ... ;   // = System.Windows.Forms.Application.ProductName; -一些不会随着时间变化的数据
encrypted = SSC.ProtectedData.Protect(data, salt, SSC.DataProtectionScope.CurrentUser);
 .
 .  // 写入文件，保存到数据库，等等…
 .
data = SSC.ProtectedData.Unprotect(encrypted, salt, SSC.DataProtectionScope.CurrentUser);
```

  - 在 .NET 1.1中，必须由开发人员自己去实现DPAPI调用的封装
  - 详见 *IssueVision* 的源代码： DataProtection.cs / .vb

完整性

- 可靠的校验和通常用于确保完整性：
  - 数据没有被他人编辑过
  - 数据没有被他人替换过
  - 等等
- 好消息？
  - *Data Protection API* 也支持完整性验证！

数据保护 API （2）

- DPAPI 整合了消息验证代码（MAC）的概念
  - MAC = 通过密钥保护消息的校验和
  - *ProtectData.Protect* 在结果中自动包含MAC
  - *ProtectData.Unprotect* 自动检查MAC，如果发现非法则抛出异常

与前面相同的代码！

```
byte[]  data, salt, encrypted;
data = ... ;
salt = ... ;
encrypted = SSC.ProtectedData.Protect(data, salt, SSC.DataProtectionScope.CurrentUser);
 .
 .
 .
data = SSC.ProtectedData.Unprotect(encrypted, salt, SSC.DataProtectionScope.CurrentUser);
```

您的潜力，我们的动力

Microsoft
微软（中国）有限公司

demo

保护配置文件设置...

msdn

MSDN Webcasts

演示回顾

- 步骤：
  - 使用无保护的配置文件来构建应用程序
  - 当我们准备去发布时：
    - » 分别加密私有设置，并且粘贴到.config文件中
    - » 对于应用程序设置： *SSC.DataProtectionScope.LocalMachine*
    - » 对于用户设置：*SSC.DataProtectionScope.CurrentUser*
  - 修改 *Settings* 类，在Get / Set操作的时候进行解密 / 加密
- 资源：
  - http://www.developer.com/net/vb/article.php/3500906
  - K. Brown, "*.NET Developer's Guide to Windows Security*"
    - Item 70
    - http://www.pluralsight.com/wiki/default.aspx/Keith.GuideBook.HomePage

代码回顾

- 加密 / 解密 字符串的封装：

```
using SSC = System.Security.Cryptography;          // 需要添加引用System.Security assembly

public class MyDPAPI
{
   public static string Encrypt(string data, string salt, SSC.DataProtectionScope scope)
   {
     byte[] encrypted;
     encrypted = SSC.ProtectedData.Protect(System.Text.Encoding.UTF8.GetBytes(data),
                   System.Text.Encoding.UTF8.GetBytes(salt), scope);
     return System.Convert.ToBase64String(encrypted);
   }

   public static string Decrypt(string edata, string salt, SSC.DataProtectionScope scope)
   {
     byte[] data;
     data = SSC.ProtectedData.Unprotect(System.Convert.FromBase64String(edata),
              System.Text.Encoding.UTF8.GetBytes(salt), scope);
     return System.Text.Encoding.UTF8.GetString(data);
   }
}
```

DPAPI的限制

- 必须在同一台机器上进行加密/解密
- *DataProtectionScope.CurrentUser*:
  - 只有该用户可以解密数据
- *DataProtectionScope.LocalMachine*:
  - 在这台机器上面的任何用户都可以解密数据

- *暗示？*
  - 实现了本地数据保密和完整性的策略

需要注意的问题

- 意识到当安全出现漏洞的情况…
- 例如：
  - 使用DPAPI，用户所运行的任何应用程序都能够读取用户的设置（user settings）
  - 使用DPAPI，在同一台机器上的任何应用程序都能够读取应用程序的设置（application settings）
  - Salt只有获得充分保护，才能够阻止非法解密…
- 解决方案？
  - 不要在配置文件中保存任何保密信息！
  - 使用不同的加密API — 但是，密钥保存在哪里？如何传递密钥？如何保护它们？

威胁模型

- 什么是你的威胁模型？
  - 防止非法/临时用户查看数据？
  - 是否需要防范密集，集中，长时间的攻击？
- 这将会帮助你决定如何去正确地工作
  - 安全设计要根据你的威胁模型而定

议题

- 安全性概览
- 验证和授权
- 自定义验证
- 私密信息和完整性
- ***Microsoft Security Application Blocks***

与安全相关的应用程序块

- Microsoft提供了三个与安全相关的应用程序块：
  - *Authorization and Profile* application block (单独发布， .NET 1.1): http://msdn.microsoft.com/library/en-us/dnpag2/html/security1.asp (see link in downloads section)
  - *Security* application block (企业库的一部分， .NET 1.1): http://msdn.microsoft.com/library/en-us/dnpag2/html/security1.asp
  - *Cryptography* application block (企业库的一部分， .NET 1.1): http://msdn.microsoft.com/library/en-us/dnpag2/html/crypto1.asp
  - Crypto应用程序块对配置文件的设置提供非常好的支持
  - Security应用程序块扩展了基于角色的模型，支持自定义验证管理（类似于ASP.NET 2.0）

总结

- 安全性概览
- 验证和授权
- 自定义验证
- 私密信息和完整性
- Microsoft Security Application Blocks

您的潜力，我们的动力

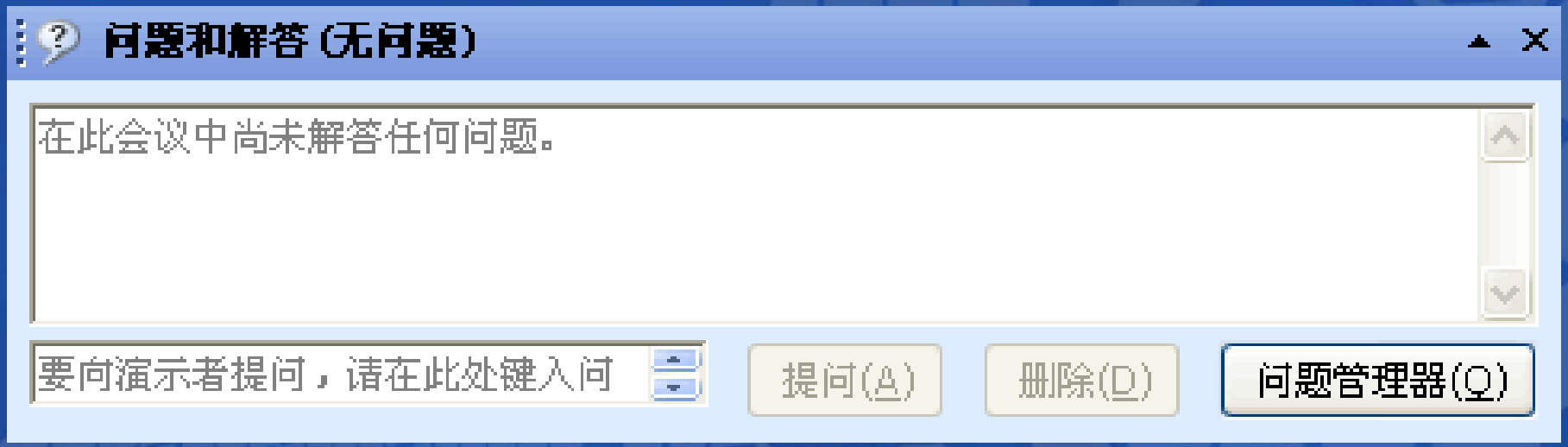
Microsoft
微软（中国）有限公司

获取更多MSDN资源

- **MSDN中文网站**
  http://www.microsoft.com/china/msdn
- **MSDN中文网络广播**
  http://www.msdnwebcast.com.cn
- **MSDN Flash**
  http://www.microsoft.com/china/newsletter/case/msdn.aspx
- **MSDN开发中心**
  http://www.microsoft.com/china/msdn/DeveloperCenter/default.mspx

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