.NET 框架中基于角色的安全性研究

田 原
(荆门职业技术学院计算机系 荆门 448000)

摘 要 商务应用程序经常根据用户提供的凭据提供对数据或资源的访问。通常,这种应用程序会检查用户的角色,并根据该角色提供对资源的访问。在.NET Framework 中,本文研究了根据 Windows 帐户或自定义标识提供基于角色安全性的授权支持技术。

关键词 角色,安全性,.NET 框架

Research of Role-Based Security in .NET Framework

TIAN Yuan
(Jingmen Vocational Technical College, Jingmen 448000)

Abstract The commercial application often provides services for accessing data or resource according to the voucher provided by the user. The application often checks the role of user, and provides the access for resource. In .NET framework, this paper studies the technology about supporting authorization based on role security according to Windows identity or self-definition identity.

Keywords Role, Security, .NET framework

商务应用程序经常根据用户提供的凭据提供对数据或资源的访问。通常情况下,这种应用程序会检查用户的角色,并根据该角色提供对资源的访问。.NET Framework 根据 Windows 帐户或自定义标识提供基于角色的授权支持。

1 基于角色的安全性介绍

在商务应用程序中经常使用角色强制策略。例如,应用程序可能根据提出请求的用户是不是指定角色的成员,对要处理的事务大小加以限制。职员有权处理的事务可能小于指定的阈值,主管拥有的权限可能比职员的高,而副总裁的权限可能还更高(或根本不受限制)。当应用程序需要多个批准完成某项操作时,也可以使用基于角色的安全性。例如一个采购系统,在该系统中,任何雇员均可生成采购请求,但只有采购代理人可以将此请求转换成可发送给供应商的采购订单。

.NET Framework 基于角色的安全性通过生成可供当前线程使用的主体信息来支持授权,而主体是用关联的标识构造的。标识(及其帮助定义的主体)可以基于 Windows 帐户,也可以是同 Windows 帐户无关的自定义标识。.NET Framework 应用程序可以根据主体的标识或角色成员条件(或者这两者)做出授权决定。角色是指在安全性方面具有相同特权的一组命名主体(如出纳或经理)。一个主体可以是一个或多个角色的成员。因此,应用程序可以使用角色成员条件来确定主体是否有权执行某项请求的操作。

.NET Framework 提供了灵活且可扩展的基于角色的安全性支持,足以满足广泛的应用程序的需要。可选择同现有的身份验证结构(如 COM+ 1.0 服务)相互操作,或创建自定义身份验证系统。.NET Framework 基于角色的安全性既可用于客户端,也可用于服务器,尤其适用于主要在服务器处理的 ASP.NET Web 应用程序。

田 原 副教授,研究方向为软件工程、信息系统。

2 主体和标识对象

托管代码可通过 Principal 对象发现主体的标识或角色,该对象包含对 Identity 对象的引用。标识和主体对象类似于用户与组帐户这样的常见概念。在多数网络环境中,用户帐户表示人员或程序,而组帐户表示特定类别的用户及其拥有的权限。同样,.NET Framework 中的标识对象表示用户,而角色表示成员条件与安全性上下文。在.NET Framework 中,主体对象同时封装标识对象和角色。.NET Framework 应用程序根据主体的标识或角色成员条件(后者更常见)来向主体授予权限。

2.1 基于角色的安全性结构

基于角色的安全性结构如图 1 所示。具体说明见 2.2 节和 2.3 节。

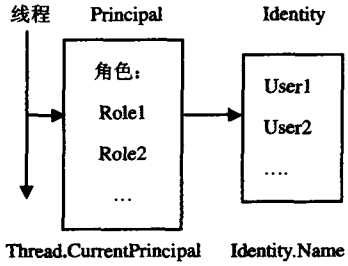

图 1 基于角色的安全性结构

2.2 标识对象

标识对象封装有关正在验证的用户或实体的信息。在最基本的级别上,标识对象包含名称和身份验证类型。名称可以是用户名或 Windows 帐户名,而身份验证类型可以是所支持的登录协议(如 Kerberos V5)或自定义值。.NET Frame-

work 定义了 GenericIdentity 和 WindowsIdentity 两个对象，前者用于自定义登录方案，后者用于依赖于 Windows 身份验证的情况中。此外，还可以定义自己的标识类来封装自定义用户信息。

Identity 接口定义用于访问名称和身份验证类型（如 Kerberos V5 或 NTLM）的属性。所有 Identity 类均实现 Identity 接口。Identity 对象同执行当前线程所用的 Windows NT 进程标记之间不需要有什么关系。但是，如果 Identity 对象是 WindowsIdentity 对象，则假定标识表示 Windows NT 安全标记。

2.3 主体对象

主体对象表示代码运行时所在的安全上下文。实现基于角色的安全性的应用程序将基于与主体对象关联的角色来授予权限。同标识对象类似，.NET Framework 提供 GenericPrincipal 对象和 WindowsPrincipal 对象。还可以定义自己的自定义主体类。

IPrincipal 定义一个属性和一种方法，前者用于访问关联的 Identity 对象，而后者用于确定 Principal 对象所标识的用户是否为给定角色的成员。所有 Principal 类都实现 IPrincipal 接口以及任何必需的附加属性和方法。例如，公共语言运行库提供了 WindowsPrincipal 类，该类实现将 Windows NT 或 Windows 2000 组成员条件映射到角色的附加功能。

Principal 对象在应用程序域（AppDomain）中绑定到调用上下文（CallContext）对象。默认的调用上下文总是用每个新的 AppDomain 创建的，因此总是存在可用于接受 Principal 对象的调用上下文。创建新线程的同时也为该线程创建 CallContext 对象。Principal 对象引用从创建线程自动复制到新线程的 CallContext 中。如果运行库无法确定哪个 Principal 对象属于线程的创建者，它将遵循 Principal 和 Identity 对象创建的默认策略。

可配置的应用程序域特定策略定义了一些规则，用以决定同新的应用程序域关联的 Principal 对象类型。在安全策略的允许范围内，运行库可创建 Principal 和 Identity 对象来反射同当前执行线程关联的操作系统标记。默认情况下，运行库使用 Principal 和 Identity 对象表示未经身份验证的用户。运行库不创建这些默认的 Principal 和 Identity 对象，除非代码试图访问它们。

创建应用程序域的受信任代码可设置应用程序域策略，以控制默认 Principal 和 Identity 对象的构造。此应用程序域特定的策略适用于该应用程序域中的所有执行线程。非托管、受信任的宿主本身就具有设置此策略的能力，但托管代码必须具有控制域策略的 System.Security.Permissions.SecurityPermission 才能设置此策略。

在不同的应用程序域之间、但在同一进程内（因此在同一台计算机上）传输 Principal 对象时，远程基础结构将同调用方上下文相关联的、对 Principal 对象的引用复制到被调用方的上下文中。

2.4 基于角色的有效性检查

由于负责引用 WindowsIdentity 的 WindowsPrincipal 直接与 Windows 用户相关，而且由独立的资源授权此用户，所以这种身份类型非常强壮。为了能够执行基于角色的有效性检查，必须创建一个 WindowsPrincipal 对象。有两种方式可以实现它，分别依赖于是仅做一次用户和角色的有效性检查还是要重复做多次。一次有效性解决方案如下：

```
'初始化 WindowsIdentity 对象的一个实例
Dim WinIdent as WindowsIdentity = WindowsIdentity.GetCurrent()
'创建一个 WindowsPrincipal 对象实例，把它和 WindowsIdentity 绑定在一起
Dim WinPrinc as New WindowsPrincipal(WindIdent)
'访问 WindowsIdentity 对象和 WindowsPrincipal 的属性
Dim PrincName As String=WinPrinc.Identity.Name
Dim IdentName As String=WinIdent.Name
Dim IdentType As String = WinIdent.AuthenticationType
```

如果要重复执行基于角色有效性的操作，绑定 WindowsPrincipal 到线程上。如果创建了一个新的线程，也要将负责人拷贝给它。具体实现如下：

1）创建 1 个基于 WindowsPrincipal 的主体对象策略，并绑定它到当前的线程，这将初始化 WindowsIdentity 对象的 1 个实例，创建 WindowsPrincipal 对象的 1 个实例，绑定 WindowsIdentity 到它上面，并且绑定 WindowsPrincipal 到当前线程上。所有的这些可以用下面的一个简单语句实现：

```
AppDomain.CurrentDomain.SetPrincipalPolicy(PrincipalPolicy.Windowsprincipal)
```

2）得到绑定到线程的 WindowsPrincipal 对象的一份拷贝：

```
Dim WinPrinc As WindowsPrincipal = Ctype(Thread.CurrentPrincipal,Windowsprincipal)
```

在第一个创建的方法里将 WindowsPrincipal 绑定到线程上去是可行的。然而为了这样做，必须给代码 SecurityPermission 权限。使用如下代码绑定主体对象到线程上：

```
Thread.CurrentPrincipal=WinPrinc
```

不想信赖 Windows 授权时，可以使用 GenericPrincipal。假设应用程序要求从用户那里得到用户名和密码，靠应用程序自己的授权数据库检验用户名和密码，并创建一个用户的身份。为此，必须创建一个 GenericPrincipal，让它在应用程序里执行基于角色的验证：

1）为刚刚授权的用户创建一个 GenericIdentity 对象：

```
Dim GenIdent As New GenericIdentity("User1")
```

2）创建 GenericPrincipal 对象，绑定 GenericIdentity 对象到它上面，并向 GenericPrincipal 中添加角色：

```
Dim UserRoles as String( ) = {"Role1","Role2","Role3"}
Dim GenPrinc As New Genricprincipal(GenIdent,UserRoles)
```

3）绑定 GenericPrincipal 到线程，同样需要 SecurityPermission：

```
Thread.CurrentPrincipal=GenPrinc
```

3 基于角色的安全检查

定义了标识和主体对象后，可采用下列方法之一对其进行安全检查：使用命令式安全检查；使用声明式安全检查；直接访问 Principal 对象。

托管代码可使用命令式或声明式安全检查来确定以下内容：特定主体对象是否是已知角色的成员，是否具有已知的身份，或者是否表示一种角色中的一个已知身份。若要通过命令式或声明式安全性进行安全检查，必须对适当构造的 PrincipalPermission 对象生成一个安全请求。安全检查期间，公共语言运行库检查调用方的主体对象，确定其身份和角色是否与所请求的 PrincipalPermission 表示的身份和角色相匹配。如果主体对象不匹配，则引发 SecurityException。此外，

可以直接访问主体对象的值，并在不使用 PrincipalPermission 对象的情况下执行检查。在这种情况下，只需读取当前线程主体的值或使用 IsInRole 方法执行身份验证。

利用 PrincipalPermission 可以检查活动的主体对象是 WindowsPrincipal 还是 GenericPrincipal。活动的主体对象可能是为执行一次性检查而创建的，也可能是绑定到线程的那一个。PrincipalPermission 既可以用声明式方式又可以用命令式方式使用。

以命令式方式实现 PrincipalPermission 类，创建该类的一个新实例，并用希望用户在访问代码时具有的名称和角色来初始化该实例。例如，以下代码以 Tom 身份和 Teller 角色对此对象的一个新实例进行初始化。

```
Dim id As String="Tom"
Dim role As String="Teller"
Dim principalPerm As New PrincipalPermission(id,role)
```

为了用声明式方式使用 PrincipalPermission，需要如下使用 Principal Permission Attribute 对象：

```
Public Shared Function
〈PrincipalPermissiobAttribute (SecurityAction. Demand,_
Name:="User1",Role:="Role1")〉Act2()
As Integer
    'body of the function
End Function
〈assembly: PrincipalPermissionAttribute (SecurityAction. Demand,_
            Role:='Administrator')〉
```

为了使用命令式方式，可以像上面所示的那样做 PrincipalPermission 检查：

```
Dim PrincPerm As New PrincipalPermission("User1","Role1")
PrincPerm. Demand()
```

以如下两种方式使用强制设置 PrincipalPermission 对象也是可行的：

```
Dim PrincState As PermissionState=Unrestricted
Dim PrincPerm As New PrincipalPermission(PrincState)
```

权限状态(PrincState)可以是 None，也可以是 Unrestricted，None 意味着主体对象没有授权。因此用户是 Nothing，角色是 Nothing，Authenticated 为假。Unrestricted 匹配其他的主体对象。

```
Dim PrincAuthenticated As Boolean=True
Dim PrincPerm As New PrincipalPermission("User1","Role1",PrincAuthenticated)
```

IsAutonticated 域(PrincAuthenticated)可以是真或是假。

在 PrincipalPemrission. Demand 容许不止一个用户或角色联合时，可以合并两个 PrincipalPermission 对象。当且仅当这些对象是相同类型的才可行。如果一个 PrincipalPermission 对象已设立了用户/角色，而其他对象要使用 PermissionState 时，CLR 将会抛出一个异常。合并操作如下所示：

```
Dim PrincPerm1 As New PrincipalPermission("User1","Role1")
Dim PrincPerm2 As New PrincipalPermission("User2","Role2")
PrincPerm1. Union(PrincPerm2). Demand()
```

只要主体对象在角色 Role1 中有用户 User1 或者角色 Role2 中有 User2，Demand 都会成功。任何其他的合并都将会失败。

直接访问主体对象的身份对象，即不使用 PrincipalPermission 执行安全检查。除能检查更多的信息外，还可以阻止处理因使用 PrincipalPermission 产生的异常。为此，可以像 PrincipalPermission 做这件事的方法一样来查询 Windows 主体对象：

• 靠检查 WindowsPrincipal. Identity. Name 的值来得到用户名

```
If(WinPrinc. Identity. Name="User1")or _
    WinPrinc. Identity. Name. Equals("DOMAIN1\User11")Then
End If
```

• 靠调用 IslnRole 方法来得到一个角色

```
If(WinPrinc. IsInRole("Role1"))Then
End If
```

• 靠检查 WindowsPrincipal. Identity. lsAuthenticated 的值来决定是否给主体对象授权

```
If(Winprinc. Identity. IsAuthenticated)Then
End If
```

另外，对 PrincipalPermission 可以检查下面的 Windows Identity 属性：

• AuthenticationType 决定使用的授权类型，绝大部分公共值为 NTLM 和 Kerberosos。

• IsAnonymous 决定一个用户是否等同于匿名账号。

• sGuest 决定一个用户是否等同于宾客账户。

• IsSystem 决定一个用户是否等同于系统账户。

• Token 返回用户的 Windows 账户标记。

结束语 .NET 安全性有两个重要的分支，基于角色的安全性和代码访问安全性。.NET Framework 提供了灵活且可扩展的基于角色的安全性支持，足以满足广泛的应用程序的需要，尤其适用于主要在服务器处理的 ASP. NET Web 应用程序。

参 考 文 献

1 许春根，贡生. 一种新型的基于角色访问控制的角色管理模型. 计算机工程，2003，29(8)：26～28

2 毛碧波，孙玉芳. 角色访问控制. 计算机科学，2003，30(1)：121～123

3 陈军，杨善林. 一种基于中间件的角色访问控制的实现. 计算机应用研究，2004，(10)：84～86

4 孙国刚，范学峰. 基于 WS-Security 和角色访问控制的 Web 服务安全性研究. 微计算机应用，2005，(1)：37～39

5 李晓林，张彦铎. 信息系统中基于角色的访问控制. 微机发展，2004，14(9)：112～115

6 何斌，顾健. 基于角色访问控制的权限管理系统. 计算机工程，2004，30(B12)：326～328

7 Adam Sills，等著，战晓苏，等译. XML、NET 编程指南. 电子工业出版社，2003，1：111～156