CrystalWall 权限系统用户指南 Version1.0.1

概述

本指南是如何使用 Cyrstalwall 的用户指南，主要针对使用 crystalwall 进行权限控制的用户。建议阅读本指南之前，首先阅读《Crystalwall 权限系统开发者指南--基本概念》

身份提供者 PrincipalProvider

使用 crystalwall 之前，首先需要配置身份提供者实现。Crystalwall v1.0 版本中已经实现了一个默认的 DBPrincipalProvider，这个身份提供者可以在 App.config 配置文件中按照如下方式配置：

```
<?xml version="1.0"?>
<configuration>
  <configSections>
   <!--在这里添加处理身份提供者的处理器-->
    <sectionGroup name="principal-providers">
       <section name="provider" type="CrystalWall.Config.PrincipalProviderSectionHandler,
CrystalWall, Version=1.0.0.0, Culture=neutral, PublicKeyToken=null" allowLocation="true"
allowDefinition="Everywhere"/>
    </sectionGroup>
    <!-- Other <section> and <sectionGroup> elements.  -->
  </configSections>
  <!--在这里配置身份提供者配置节-->
  <principal-providers>
    <provider class="CrystalWall.Auths.DBPrincipalProvider">
      <connection>Data Source=**;Initial Catalog=***;User
ID=sa;Password=***;</connection>
      <!--<conn-provider>数据库提供者名称</conn-provider>（可不配置默认为sql server）-->
      <principal-table>principal</principal-table><!--这里配置存储用户身份的表-->
      <!--<user-indentity>pname</user-indentity>（可不配置，默认为pname）?-->
      <permission-table>permission</permission-table><!--这里配置权限表-->
```

```
        <!--以下配置权限和用户身份表的中间表，但此配置是可选的，默认中间表为身份表名_权限表名
        <foreign-table name="principal_permission">
          <foreign-user>principal_id</foreign-user><!--这里配置指向身份表的外键字段名称-->
          <foreign-permission>permission_id</foreign-permission><!--这里配置指向权限表的外
键字段名称-->
        </foreign-table>
        -->
      </provider>
      <!--以下配置其他身份提供者，在crystalwall后续版本中将陆续增加-->
      <!--other provider section
      <provider class="LDAPPrincipalProvider"/>
      <provider class="XmlPricipalProvider">
        <file>~/web/principal/principal.xml</file>
      </provider>
      -->
      </principal-providers>
</configuration>
```

DB 身份提供者实现，必须按照配置在 connection 指定的数据库中提供三张表：身份表、权限表、和身份_权限中间表。这些表必须具有 crystalwall 主张的一些惯例，他们是：

- 所有的主键必须为 id 字段，且为唯一的 32~36 位字符串（sql server 中为 uniqueidentifier 类型）
- user-indentity 配置指定用户身份表的唯一标识名称，可以为 id 字段也可以为其他，一般标识用户名
- permission 权限表中必须至少具有三个字段分别存储 name、action、和 class 用于存储权限信息的名称、action、和类型

【当然，如果你不希望遵守 crystalwall 的惯例，你需要重写 DBPrincipalProvider 类，并进行相应的配置】

注意：如果你不使用微软的ConfigurationSection元注释类配置提供者，且不使

用默认的配置文件，请不要用微软的ConfigurationManager.GetSection或者Configuration.GetSection方法获得配置节，而应该使用crystalwall封装的ConfigurationFile类获得提供程序处理对象：

```
string path = Assembly.GetAssembly(typeof(IPrincipalProvider)).Location;
ConfigurationFile configuration = new ConfigurationFile(path);
IList<IPrincipalProvider> providers = configuration.GetSection("principal-providers") as
IList<IPrincipalProvider>;
```

如果你使用ConfigurationSection元注释类配置提供者，请不要使用ConfigurationFile类的GetSection方法，此时你应该使用ConfigurationFile内部包装的Configuration类的GetSection，然后直接使用微软的Configuration.GetSection方法获取，例如：

```
ConfigurationFile configurationFile = new ConfigurationFile(path);
//使用微软的配置获取元继承ConfigurationElement的配置节：
ConfigurationSection o = configurationFile.Configuration.GetSection("");
```

当前用户存储器 IPrincipalTokenStorage 和身份持有者 PrincipalTokenHolder

前者是一个接口，专门用于存储当前使用系统的用户身份，而后者是一个静态类，主要提供所有的身份，并可以设置和获得当前的身份。

对于web系统，crystalwall提供了一个默认实现-WebPrincipalTokenStorage。此默认实现将当前使用系统的token身份令牌存入到当前的Session会话中。

以下是使用 PrincipalTokenHolder 存储、获取身份令牌的代码：

```
PrincipalTokenHolder.Storage = new WebPrincipalTokenStorage();
IPrincipalProvider provider = ....//获取或new DBPrincipalProvider
PrincipalTokenHolder.PrincipalProviders.Add(provider );

//web页面中登录，获得用户名称
String currentUser = “” ;
PrincipalTokenHolder.CurrentPrincipal = PrincipalTokenHolder.GetPrincipal(currentUser );
```

```
//logout注销登录时调用：
PrincipalTokenHolder.ClearCurrentToken();
```

注意：实际上，你完全无需进行如上的设置存储器、身份提供者列表的代码编写，在应用程序启动时，PrincipalTokenHolder会自动从配置文件中读取配置的存储器和身份提供者实现。（目前暂时不支持注入容器，因此PrincipalTokenHolder为静态类）。而你只需要在应用程序启动的入口，例如Global.asax.cs文件的Application_Start方法中进入如下调用即可：

PrincipalTokenHolder.InitWeb();

如果是普通客户端应用，则您需要调用PrincipalTokenHolder.Init();

注意，PrincipalTokenHolder是整个应用程序的全局对象，在应用程序启动之后Storage 、PrincipalProviders就不应该将其改变，PrincipalTokenHolder虽然支持身份提供列表的设置方法，只是为了容器注入预留接口而已。

编程方式的权限控制

Crystalwall控制权限最核心的地方就是处理PermissionInfo。实际上，这也是crystalwall最灵活的地方。使用PermissionInfo，你可以在您代码的任何位置进行权限控制。因此，crystalwall提供了PermissionInfo对象上的"++"操作符，在您需要进行权限控制的方法中，您只需要：

```
  public void AccessCheckTest()
{
  PrincipalTokenHolder.CurrentPrincipal = PrincipalTokenHolder.GetPrincipal("admin");
  PermissionInfo p = new TestPermissionInfo("test", "test");
  p++;
}
```

当你执行p++时，crystalwall会自动判断当前用户是否具有p权限，如果不具有，则代码将抛出AccessException异常，此时，你可以根据自己应用的实际情况进行相应的处理

通过++操作符，crystalwall使得您将权限控制的焦点转移到PermissionInfo对象的编写上，对于你想要控制的每个资源，你只需要使用PermissionInfo进行封装，而无需关心权限是如何控制的细节。（注：PermissionInfo的基本概念请参考《Crystalwall开发者指南--基本概念》）

使用权限点元注释的权限控制

Crystalwall真正进行权限控制的接口是IAccessDecider。通过这个接口，crystalwall判断并解析需要权限控制对象中的“权限点"。此接口只有一个方法：

```
void Decide(IPrincipalToken principal, object context);
```

第一个参数为要判断的令牌，第二个参数为要检测的上下文对象。这个上下文对象很重要，他提供了需要进行权限控制的上下文信息。

Crystalwall建议不要直接实现此接口，调用者可以继承AbstractDecider抽象类。你可以为这个抽象类配置ConfuseElect冲突解决器，以及实现GetPointResolves获取权限点解析器的方法。

AbstractDecider抽象类还提供了AccessDenyed事件，此事件将在权限检查不通过时执行（执行事件，但异常继续抛出）

权限点解析器是IPointResolveStrategy接口，此接口负责从context对象中解析出权限点。AbstractDecider抽象类将遍历解析策略列表解析指定上下文对象为权限点列表。另外，如果context本身就是实现IPermissionPointProvider接口的对象，则AbstractDecider会直接使用此接口提供的权限点判断所需的权限。Crystalwall建议进行权限判断时，尽量传递实现IPermissionPointProvider接口的上下文对象。

开发者可以使用元特性PermissionPointAttribute在方法上定义某个方法需要某种权限，如下所示：

```
[PermissionPoint(Name = "test3", Action = "test3", Type =
"Crystalwall.Test.Auths.TestPermissionInfoPoint, Crystalwall.Test,
Version=1.0.0.0, Culture=neutral, PublicKeyToken=null")]
public virtual string YourMethod()
{
    //your code
}
```

在 PermissionPoint 属性定义中，你只需要定义权限名称、action 列表以及此 PermissionPoint 的具体类型全名。（实际上，权限点中的 type 字符属性也可以直接指定 PermissionInfo 具体子类的类型，crystalwall 框架会自动判断你定义的类型是权限点类型还是直接的 PermissionInfo ）

PermissionPoint是一个抽象类，他有一个抽象方法：

public abstract PermissionInfo NewPermission();

你可以根据需要自己定义权限点是如何生成权限对象的

通过上面的方式，你可以定义任意虚拟方法是否需要权限判断。但是，光有权限定义还不够，如何才能让crystalwall知道某个方法上需要权限判断呢？

Crystalwall提供了一个叫做CrystalwallSite对象，这个对象将对由context属性定义的上下文对象加入权限代码。请不要自己构造此对象，要得到这个对象，请使用静态的Find方法：

CrystalwallSite site = CrystalwallSite.Find(object context);

site.InitSite();

site.Decider.decider(principal, context);

另外，要使用Site对象，你需要在你的配置文件中加入如下配置：

```
<sites>
    <site context="Crystalwall.AOP.MethodInvocation, CrystalWall , Version=1.0.0.0,
Culture=neutral, PublicKeyToken=null" class="SiteType类型（可选）">
    <decider class="可选">
        <resolves>
          <resolve class="权限解析器类型1">
          <resolve class="权限解析器类型2">
        </resolves>
    </decider>
    </site>
</sites>
```

Decider配置都是可选的，因此你可以简化为：

```
<sites>
    <site context="Castle.DynamicProxy.IInvocation, Castle.Core , Version=2.5.1.0"/>
</sites>
```

详细的配置，可以参考源码core目录下的App.config配置文件。

可以看到，crystalwall默认支持Castle的动态代理的方法调用对象。正式因为这个原因，如果您需要对一个对象的使用PermissionPoint元特性定义的方法进行权限判断，你必须使用Castle动态代理技术获取代理后的对象，不过Crystalwall提供了一个简单的工具类，开发者可以使用此类获得代理对象，从而自动的进行权限控制。在以下的例子中，你希望对ProxyTestClass对象上的两个方法判断权限：

```
public class ProxyTestClass
    {
        /// <summary>
        /// 执行此方法，会首先判断当前用户是否具有test权限
        /// </summary>
        [PermissionPoint(Name = "test", Action = "test", Type =
"Crystalwall.Test.Auths.TestPermissionInfoPoint, Crystalwall.Test, Version=1.0.0.0,
Culture=neutral, PublicKeyToken=null")]
        public virtual string MyTestHasPermissionMethod()
        {
            return   "test";//你的方法主体          }
    }
```

使用CrystalwallDynamicProxyor类创建此类的实例：

```
var actual = CrystalwallDynamicProxyor.ProxyClass<ProxyTestClass>();
```

这样，当你执行MyTestHasPermissionMethod方法时：

```
actual.MyTestHasPermissionMethod();
```

Crystalwall框架即会自动判断当前用户（PrincipalTokenHolder.CurrentPrincipal）是否具有方法上定义的PermissionPoint指定的test权限。如果当前用户具有test权限，则方法会继续执行，否则将抛出AccessException异常。

总结

使用Crystalwall框架进行权限控制，总体来说可以按照如下步骤进行：

1、定义自己的PermissionInfo权限对象，并定义相应的PermissionPoint

2、将需要进行权限控制的方法上使用PermissionPoint元特性

3、使用身份提供者PrincipalProvider建立身份对应的Permission关系

4、使用CrystalwallDynamicProxyor生成实例，而不使用new创建实例

5、如果需要，可以在配置文件例如App.config中重新定义site、provider等定义（此步骤可选，crystalwall提供了很多默认选项，除非你的应用想要改变默认的设定）

通过以上的方式，crystalwall使得开发人员从”如何判断权限“、”何时判断权限“等细节中解脱出来，开发人员只要专心定义”身份需要什么权限“就可以了（即编写PermissionInfo的子类）