

第21期-基于权限策略的授权机制

2020年8月10日 13:01

认证与授权

认证确定用户是谁，授权确定该用户能否访问指定的资源或者操作。

授权机制

ABP 扩展了 ASP.NET Core 的授权机制，将权限自动添加为策略，在应用服务层面使用拦截器进行授权，可自定义策略，在控制器和应用服务中使用以下注解。

兼容支持: [Authorize] 和 [AllowAnonymous]

ASP.NET Core 中基于策略的授权

权限系统

从用户、角色和客户端维度对不同的参与者权限控制。

```
public class MyPermissionDefinitionProvider:PermissionDefinitionProvider
{
    public override void Define(IPermissionDefinitionContext context)
    {
        var myGroup = context.AddGroup("BookStore");

        myGroup.AddPermission("BookStore_Author_Create");
    }
}
```

本地化权限名称

"BookStore": "Book Store",
"Permission:BookStore_Author_Create": "Creating a new author"

```
public class MyPermissionDefinitionProvider : PermissionDefinitionProvider
{
    public override void Define(IPermissionDefinitionContext context)
    {
        var myGroup = context.AddGroup("BookStore",
        LocalizableString.Create<TestResource>("BookStore"));
    }
}
```

```

        myGroup.AddPermission("BookStore_Author_Create",
            LocalizableString.Create<TestResource>("Permission:BookStore_Author_Create"));
    }
}

```

权限系统支持多租户：Host、Tenant 和 Both

禁用权限

```

myGroup.AddPermission("Author_Management", isEnabled: false);

```

权限的层级结构与子权限

```

var authorManagement = myGroup.AddPermission("Author_Management");

authorManagement.AddChild("Author_Management_Create_Books");
authorManagement.AddChild("Author_Management_Edit_Books");
authorManagement.AddChild("Author_Management_Delete_Books");

[Authorize("Author_Management")]
public class AuthorAppService : ApplicationService, IAuthorAppService
{
    public Task<List<AuthorDto>> GetListAsync()
    {
        ...
    }

    public Task<AuthorDto> GetAsync(Guid id)
    {
        ...
    }

    [Authorize("Author_Management_Create_Books")]
    public Task CreateAsync(CreateAuthorDto input)
    {
        ...
    }

    [Authorize("Author_Management_Edit_Books")]
    public Task UpdateAsync(CreateAuthorDto input)
    {
        ...
    }

    [Authorize("Author_Management_Delete_Books")]
    public Task DeleteAsync(CreateAuthorDto input)
    {
        ...
    }
}

```

```
}  
}
```

更改依赖模块的权限定义

```
context.GetPermissionOrNull("<PermissionName>").IsEnabled = false;
```

扩展权限检查提供者

内置：UserPermissionValueProvider、RolePermissionValueProvider 和 ClientPermissionValueProvider 都继承 PermissionValueProvider 抽象类或者实现 IPermissionValueProvider 接口。

```
public class SystemAdminPermissionValueProvider : PermissionValueProvider  
{  
    public SystemAdminPermissionValueProvider(IPermissionStore  
permissionStore) : base(permissionStore)  
    {  
  
        public override string Name => "SystemAdmin";  
  
        public override async Task<PermissionGrantResult>  
CheckAsync(PPermissionValueCheckContext context)  
        {  
            await Task.CompletedTask;  
  
            if (context.Principal?.FindFirst("User_Type")?.Value ==  
"SystemAdmin")  
            {  
                return PermissionGrantResult.Granted;  
            }  
  
            return PermissionGrantResult.Undefined;  
        }  
    }  
}
```

- PermissionGrantResult.Granted 授予用户权限，如果没有其他的授权值提供程序返回 Prohibited,, 那么最后会返回 Granted。
- PermissionGrantResult.Prohibited 禁止授权用户，任何一个授权值提供程序返回了 Prohibited，那么其他的提供程序返回的值都不再重要。
- PermissionGrantResult.Undefined 代表当前无法确定是否授予或禁止权限,返回UnDefined由其他权限值提供程序检查权限。

权限存储机制

IPermissionStore 是唯一需要从持久化源(通常是数据库)中读取权限值的接口，它的实现在权限管理模块。

```
public interface IPermissionStore
{
    Task<bool> IsGrantedAsync(
        [NotNull] string name,
        [CanBeNull] string providerName,
        [CanBeNull] string providerKey
    );
}
```