**基本授权**

[Authorize]

public class AuthorAppService : ApplicationService, IAuthorAppService

{

    public Task<List<AuthorDto>> GetListAsync()

    {

        ...

    }

    [AllowAnonymous]

    public Task<AuthorDto> GetAsync(Guid id)

    {

        ...

    }

    [Authorize("BookStore\_Author\_Create")]

    public Task CreateAsync(CreateAuthorDto input)

    {

        ...

    }

}

* Authorize：表示AuthorAppService服务需要登录才能访问
* AllowAnonymous：表示AuthorAppService的GetAsync方法不需要登录也能访问
* Authorize("BookStore\_Author\_Create")：用户需要具有BookStore\_Author\_Create权限才能访问

**定义权限**

我们只需要继承PermissionDefinitionProvider便可在里面定义权限了

public class BookStorePermissionDefinitionProvider

    : PermissionDefinitionProvider

{

    public override void Define(IPermissionDefinitionContext context)

    {

        var myGroup = context.AddGroup("BookStore");

        myGroup.AddPermission("BookStore\_Author\_Create");

    }

}

**适用于多租户的权限**

如下我们可以针对多租户配置权限

myGroup.AddPermission(

    "BookStore\_Author\_Create",

    LocalizableString.Create<BookStoreResource>("Permission:BookStore\_Author\_Create"),

    multiTenancySide: MultiTenancySides.Tenant //set multi-tenancy side!

);

* Host: 权限仅适用于宿主.
* Tenant: 权限仅适用于租户.
* Both (默认): 权限适用与宿主和租户.

**启用/禁用权限**

权限默认为启用，它也可以被禁用，禁用权限所有的用户将无法使用它. 你仍然可以检查这个权限，它总是会返回被禁止

myGroup.AddPermission("Author\_Management", isEnabled: false);

**子权限**

如下我们演示子权限的用法，看完你应该知道如何使用子权限了

var authorManagement = myGroup.AddPermission("Author\_Management");

authorManagement.AddChild("Author\_Management\_Create\_Books");

authorManagement.AddChild("Author\_Management\_Edit\_Books");

authorManagement.AddChild("Author\_Management\_Delete\_Books");

使用权限

[Authorize("Author\_Management")]

public class AuthorAppService : ApplicationService, IAuthorAppService

{

    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)

    {

        ...

    }

}

**自定义策略覆盖已有权限**

如果你想为某个权限自定义验证规则，那么你可以新建一个与权限名相同的策略，验证时会使用该策略进行验证，详情请参考

<https://docs.microsoft.com/zh-cn/aspnet/core/security/authorization/policies>

**IAuthorizationService**

在业务代码中，我们可以使用IAuthorizationService进行手动权限检查

public async Task CreateAsync(CreateAuthorDto input)

{

    var result = await AuthorizationService

        .AuthorizeAsync("Author\_Management\_Create\_Books");

    if (result.Succeeded == false)

    {

        //throw exception

        throw new AbpAuthorizationException("...");

    }

    //continue to the normal flow...

}

**权限管理**

常规系统都具有权限管理功能，Abp提供了 IPermissionManager 供我们进行权限管理

public class MyService : ITransientDependency

{

    private readonly IPermissionManager \_permissionManager;

    public MyService(IPermissionManager permissionManager)

    {

        \_permissionManager = permissionManager;

    }

    public async Task GrantPermissionForUserAsync(Guid userId, string permissionName)

    {

        await \_permissionManager.SetForUserAsync(userId, permissionName, true);

    }

    public async Task ProhibitPermissionForUserAsync(Guid userId, string permissionName)

    {

        await \_permissionManager.SetForUserAsync(userId, permissionName, false);

    }

}

* SetForUserAsync 方法用于设置用户的权限 (true/false). 类似的还有 SetForRoleAsync 和 SetForClientAsync 扩展方法.

详细功能请参考

<https://docs.abp.io/zh-Hans/abp/latest/Modules/Permission-Management>

**AlwaysAllowAuthorizationService**

AlwaysAllowAuthorizationService 类可以绕过授权服务. 通常用于在集成测试中.

使用如下方法禁用权限检查

public override void ConfigureServices(

ServiceConfigurationContext context)

{

    context.Services.AddAlwaysAllowAuthorization();

}