

第28期-系统设置配置

2020年9月7日 9:22

配置系统是在启动时配置应用程序很好的方式，除了配置之外，ABP提供了另外一种设置和获取应用程序设置的方式。

设置模块

Install-Package Volo.Abp.Settings

```
[DependsOn(typeof(AbpSettingsModule))]
```

定义设置

```
public class EmailSettingProvider : SettingDefinitionProvider
{
    public override void Define(ISettingDefinitionContext context)
    {
        context.Add(
            new SettingDefinition("Smtp.Host", "127.0.0.1"),
            new SettingDefinition("Smtp.Port", "25"),
            new SettingDefinition("Smtp.UserName"),
            new SettingDefinition("Smtp.Password", isEncrypted: true),
            new SettingDefinition("Smtp.EnableSsl", "false")
        );
    }
}
```

使用常量是一种比较好的习惯。 `SettingDefinition` 类具有以下属性：

Name：应用程序中设置的唯一名称。是具有约束的唯一属性，在应用程序获取/设置此设置的值(设置名称定义为常量而不是magic字符串是个好主意)。

DefaultValue：设置的默认值。

DisplayName：本地化的字符串，用于在UI上显示名称。

Description：本地化的字符串，用于在UI上显示描述。

IsVisibleToClients：布尔值，表示此设置是否在客户端可用。默认为false，避免意外暴露内部关键设置。

IsInherited：布尔值，此设置值是否从其他提供程序继承。如果没有为请求的提供程序设置设定值，那么默认值是true并回退到下一个提供程序(参阅设置值提供程序部分了解更多)。

IsEncrypted：布尔值，表示是否在保存值是加密，读取时解密。在数据库中存储加密的值。

Providers：限制可用于特定的设置值提供程序(参阅设置值提供程序部分了解更多)。

Properties：设置此值的自定义属性 名称/值 集合，可以在之后的应用程序代码中使用。

在模块中更改设置定义

下面的示例中获取了由 Volo.Abp.Emailing 包定义的设置并将其更改：

```
public class MySettingDefinitionProvider : SettingDefinitionProvider
{
    public override void Define(ISettingDefinitionContext context)
    {
        var smtpHost = context.GetOrNull("Abp.Mailing.Smtp.Host");
        if (smtpHost != null)
        {
            smtpHost.DefaultValue = "mail.mydomain.com";
            smtpHost.DisplayName =
                new LocalizableString(
                    typeof(MyLocalizationResource),
                    "SmtpServer_DisplayName"
                );
        }
    }
}
```

获取设置值

ISettingProvider 是非常常用的服务，一些基类中(如IApplicationService)已经将其属性注入，这种情况下可以直接使用 SettingProvider 即可。

```
public class MyService
{
    private readonly ISettingProvider _settingProvider;

    //Inject ISettingProvider in the constructor
    public MyService(ISettingProvider settingProvider)
    {
        _settingProvider = settingProvider;
    }

    public async Task FooAsync()
    {
        //Get a value as string.
        string userName = await
            _settingProvider.GetOrNullAsync("Smtp.UserName");

        //Get a bool value and fallback to the default value (false) if
        not set.
        bool enableSsl = await _settingProvider.GetAsync<bool>
            ("Smtp.EnableSsl");

        //Get a bool value and fallback to the provided default value
        (true) if not set.
    }
}
```

```

        bool enableSsl = await _settingProvider.GetAsync<bool>(
            "Smtp.EnableSsl", defaultValue: true);

        //Get a bool value with the IsTrueAsync shortcut extension method
        bool enableSsl = await
        _settingProvider.IsTrueAsync("Smtp.EnableSsl");

        //Get an int value or the default value (0) if not set
        int port = (await _settingProvider.GetAsync<int>("Smtp.Port"));

        //Get an int value or null if not provided
        int? port = (await
        _settingProvider.GetOrNullAsync("Smtp.Port"))?.To<int>();
    }
}

```

在客户端读取设置值

如果**允许在客户端显示**某个设置，可以使用 JavaScript 代码读取设置值，示例：

```

//Gets a value as string.
var language = abp.setting.get('Abp.Localization.DefaultLanguage');

//Gets an integer value.
var requiredLength = abp.setting.getInt('Abp.Identity.Password.RequiredLength');

//Gets a boolean value.
var requireDigit = abp.setting.getBoolean('Abp.Identity.Password.RequireDigit');

```

设置值提供程序

设置系统是可扩展的，你可以定义设置值提供程序扩展它，根据任何条件从任何来源获取设置值，`ISettingProvider` 使用设置值提供程序来获取设置值，如果值提供程序无法获取设置值，则会回退到下一个值提供程序。

有五个预构建设置值提供程序按以下顺序注册：

- `DefaultValueSettingValueProvider`：从设置定义获取值(参见上面的`SettingDefinition`部分)。
- `ConfigurationSettingValueProvider`：从`IConfiguration`服务中获取值。
- `GlobalSettingValueProvider`：获取设置的全局(系统范围)值。
- `TenantSettingValueProvider`：获取当前租户的设置值(参阅 多租户文档)。
- `UserSettingValueProvider`：获取当前用户的设置值(参阅 当前用户 文档)。

全局、租户和用户设置值提供程序使用 `ISettingStore` 从数据源读取值。

在应用程序配置中设置值

```
{
    "Settings": {
        "Abp.Mailing.DefaultFromAddress": "noreply@mydomain.com",
        "Abp.Mailing.DefaultFromDisplayName": "My Application",
        "Abp.Mailing.Smtp.Host": "mail.mydomain.com",
        "Abp.Mailing.Smtp.Port": "547",
        "Abp.Mailing.Smtp.UserName": "myusername",
        "Abp.Mailing.Smtp.Password": "mySecretPassW00rd",
        "Abp.Mailing.Smtp.EnableSsl": "True"
    }
}
```

自定义设置值提供程序

实现 `ISettingValueProvider` 接口或者继承 `SettingValueProvider` 类，实现 `ISettingValueProvider` 接口，这时需要记得将其注册到依赖注入。

```
public class CustomSettingValueProvider : SettingValueProvider
{
    public override string Name => "Custom";

    public CustomSettingValueProvider(ISettingStore settingStore) :
base(settingStore)
    {
    }

    public override Task<string> GetOrNullAsync(SettingDefinition setting)
    {
        /* Return the setting value or null Use the SettingStore or another
data source */
    }
}

Configure<AbpSettingOptions>(options =>
{
    options.ValueProviders.Add<CustomSettingValueProvider>();
});
```

实现 `ISettingStore` 接口

尽管设置值提供程序可以自由使用任何来源来获取设置值，但 `ISettingStore` 服务是设置值的默认来源，全局、租户和用户设置值提供者都使用它。

加密解密配置服务

ISettingEncryptionService 用于在设置定义的 isencryption 属性设置为 true 时加密解密设置值。

你可以在依赖项入系统中替换此服务,自定义实现加密解密过程。

默认实现 StringEncryptionService 使用AES算法(参见字符串加密文档学习更多)

设置模块实现原理

ISettingDefinitionManager->ISettingDefinitionProvider->*SettingDefinition

ISettingProvider->

ISettingDefinitionManager+ISettingEncryptionService+ISettingValueProviderManager

ISettingValueProviderManager->*ISettingValueProvider->ISettingStore

使用设置管理存储模块

Install-package Volo.Abp.SettingManagement.EntityFrameworkCore

Install-package Volo.Abp.SettingManagement.Web

```
[DependsOn(typeof(AbpSettingManagementWebModule), typeof(AbpSettingManagementEntityFrameworkCoreModule))]
```

```
public DbSet<Setting> Settings { get; set; }
```

```
modelBuilder.ConfigureSettingManagement();
```

```
Add-Migration AddSettingTable
```

```
Update-Database
```

默认的 ISettingStore 实现也是 NullSettingStore 实现, 它为所有设置值返回 null 值。

ISettingStore->ISettingManagementStore->

IDistributedCache+ISettingRepository+ISettingDefinitionManager

ISettingManager->

ISettingDefinitionManager+ISettingEncryptionService+*ISettingManagementProvider

ISettingManagementProvider->ISettingManagementStore->

SettingRepository+ISettingDefinitionManager+IDistributedCache

设置管理页面贡献者

```
Configure<SettingManagementPageOptions>(options =>
{
    options.Contributors.Add(new MySettingPageContributor());
});
```