

第55期-使用规则注解特性验证数据

2019年9月12日 09:45

从业务层面来讲，针对 EF Core 的所有操作都不会对模型进行验证，从数据库层面来讲，无论是否验证数据，关系数据库都会保证数据的完全性，根据字段约束限制非法输入，基于字段的约束，只有当 SQL 命令发送到远程数据库时才执行约束验证，验证失败时抛出异常，在业务中提前验证模型可避免不必要的数据库性能开销。

捕获数据库异常

当您调用 SaveChanges 或 SaveChangesAsync 时，可能会发生一系列异常，通过这些异常可以发现数据库执行错误，如：DbUpdateConcurrencyException（保存到数据时遇到并发冲突时引发的异常），DbUpdateException（保存到数据库时遇到错误时引发的异常）。

```
try
{
    _context.SaveChanges();
}
catch (DbUpdateException e)
{
    var sqlException = e?.InnerException as SqlException;
    Console.WriteLine(sqlException.Number);
}
```

SQL Server 数据库引擎错误

基于数据注解的验证

使用 System.ComponentModel.DataAnnotations 程序集中的注解特性可进行模型验证，微软提供很多的默认规则，例如 Required、MinLength 和 MaxLength 等等，除此之外，我们还可以扩展自己的验证注解特性。

System.ComponentModel.DataAnnotations Namespace

```
public class Blog
{
    [Range(1, int.MaxValue)]
    public int BlogId { get; set; }

    [Required, MinLength(3)]
    public string Name { get; set; }

    [Url]
    public string Url { get; set; }
}
```

基于注解特性的验证是一种惯用，许多 .NET 客户端应用程序原生就识别这些批注，例如 ASP.NET Core MVC 中，可通过这些批注为客户端和服务端提供验证规则，详见零度 ASP.NET Core 视频教程。

做一个简单的验证示例

```
public class Book
{
    [Range(1, 100)]
    public int BookId { get; set; }

    [MinLength(2), MaxLength(10)]
    public string BookName { get; set; }
}

static void Main(string[] args)
{
    Book book = new Book { BookId = -2, BookName = "C++" };

    var validationContext = new ValidationContext(book);
    var validationResults = new List<ValidationResult>();

    if (!Validator.TryValidateObject(book, validationContext, validationResults, true))
    {
        validationResults.ForEach(vr => Console.WriteLine(vr.ErrorMessage));
    }
}
```

实现通用验证扩展方法

```
public override int SaveChanges()
{
    var entities = ChangeTracker.Entries()
        .Where(e => e.State == EntityState.Added || e.State == EntityState.Modified)
        .Select(e => e.Entity);

    foreach (var entity in entities)
    {
        var validationContext = new ValidationContext(entity);
        Validator.ValidateObject(entity, validationContext, true);
    }

    return base.SaveChanges();
}

try
{
    _context.SaveChanges();
}
catch (ValidationException e)
{
    Console.WriteLine(e.ValidationResult.ErrorMessage);
}
```

编写一个属于自己的 Attribute 注解特性

```

public class StringContainsAttribute : ValidationAttribute
{
    public string ContainsSubString { get; }

    public StringContainsAttribute(string containsSubString)
    {
        ContainsSubString = containsSubString;
    }

    public override bool IsValid(object value)
    {
        string valueString = value.ToString();

        if (string.IsNullOrEmpty(ContainsSubString))
        {
            throw new InvalidOperationException("Substring cannot be null or empty.");
        }

        if (ContainsSubString.Length > valueString.Length)
        {
            throw new InvalidOperationException("Invalid length of the substring.");
        }

        return valueString.Contains(ContainsSubString);
    }

    public override string FormatErrorMessage(string name)
    {
        return $"{name} does not contain {ContainsSubString}";
    }
}

public class Blog
{
    public int BlogId { get; set; }

    [StringContains("NB")]
    public string Name { get; set; }
}

```

支持实现 IValidatableObject 接口验证模型

```

public class Blog : IValidatableObject
{
    public int BlogId { get; set; }

    [Required, MinLength(3)]
    public string Name { get; set; }

    [Url]
    public string Url { get; set; }

    public IEnumerable<ValidationResult> Validate(ValidationContext validationContext)
    {
        if (!Name.Contains("NB"))
        {

```

```
yield return new ValidationResult("Blog name does not contain NB .", new[]  
{ nameof(Name) });  
}  
}
```