

本指南將協助您快速開始使用 FormValidators。

安裝

透過 NuGet 安裝

使用 NuGet Package Manager 安裝 FormValidators：

```
Install-Package CloudyWing.FormValidators
```

或使用 .NET CLI：

```
dotnet add package CloudyWing.FormValidators
```

支援的框架

- .NET 10
- .NET Standard 2.0
- .NET Framework 4.5

第一個驗證範例

以下是一個簡單的範例，展示如何使用 FormValidators 進行基本驗證：

```
using CloudyWing.FormValidators;

// 建立批量驗證容器
BulkValidator validators = new();

// 新增必填驗證
validators.Add(new RequiredValidator("使用者名稱", userName));

// 新增整數驗證
validators.Add(new IntegerValidator("年齡", age));

// 執行驗證
if (!validators.Validate()) {
    // 取得錯誤訊息
    string errorMessage = validators.ErrorMessage;
    Console.WriteLine(errorMessage);
}
```

使用 ValidationProvider 簡化程式碼

FormValidators 提供了更簡潔的語法來建立驗證：

```
BulkValidator validators = new(cfg => {
    cfg.Add("使用者名稱", userName,
        opt => opt.Required()
    );
    cfg.Add("年齡", age,
        opt => opt.Required(),
        opt => opt.Integer()
    );
    cfg.Add("Email", email,
        opt => opt.Email()
    );
});

if (!validators.Validate()) {
    Console.WriteLine(validators.ErrorMessage);
}
```

下一步

- [驗證器介紹](#) - 了解所有可用的驗證器
- [使用範例](#) - 查看更多實際應用範例
- [客製化驗證](#) - 學習如何建立自訂驗證器
- [API 參考](#) - 查看完整的 API 文檔

FormValidators 提供多種預設驗證器，涵蓋常見的資料驗證需求。

基本驗證

RequiredValidator

驗證 `Value` 是否為必填。如果值為 `null`、空字串或空格，視為沒有值。

相關 API: [RequiredValidator](#)

ValueLengthValidator

驗證 `Value` 的長度是否符合指定範圍。

相關 API: [ValueLengthValidator](#)

BulkValidator

批量驗證容器，可包含多個驗證器。參數 `isStoppedIfFail` 可控制驗證不通過時，是否繼續後續驗證。

相關 API: [BulkValidator](#)

型別驗證

提供驗證值是否可轉型成特定型別，及最大值和最小值驗證。

IntegerValidator

整數和極限值驗證，資料型別為 `long`。

參數:

- `allowedThousands`: 是否允許千分位符號

相關 API: [IntegerValidator](#)

NumberValidator

數值和極限值驗證，資料型別為 `decimal`。

參數:

- `allowedThousands`: 是否允許千分位符號

相關 API: [NumberValidator](#)

DateTimeValidator

日期和極限值驗證，資料型別為 `DateTime`。

相關 API: [DateTimeValidator](#)

比較驗證

CompareValidator

比較兩欄位的值是否相等。

相關 API: [CompareValidator](#)

IntegerLessThanValidator

整數欄位比較, `Value` 必須小於比較的值。

參數:

- `allowedEqual`: 設為 `true` 時, 可以等於比較值

相關 API: [IntegerLessThanValidator](#)

NumberLessThanValidator

數值欄位比較, `Value` 必須小於比較的值。

參數:

- `allowedEqual`: 設為 `true` 時, 可以等於比較值

相關 API: [NumberLessThanValidator](#)

DateTimeLessThanValidator

日期欄位比較, `Value` 必須小於比較的值。

參數:

- `allowedEqual`: 設為 `true` 時, 可以等於比較值

相關 API: [DateTimeLessThanValidator](#)

格式驗證

RegexValidator

使用正規表示式驗證 `Value`。

相關 API: [RegexValidator](#)

EmailValidator

Email 格式驗證。

相關 API: [EmailValidator](#)

MobilePhoneValidator

手機格式驗證。

參數:

- `MobilePhoneFormats`: 可指定允許的手機號碼格式

相關 API: [MobilePhoneValidator](#)

IdCardValidator

身分證驗證。

參數:

- `IdCardTypes`: 設定驗證允許格式，包括：
 - 國民身分證號
 - 臺灣地區居留證統一證號
 - 外僑居留證統一證號
 - 遊民證號
 - 新式外來人口統一證號

預設允許全部格式。

相關 API: [IdCardValidator](#)

CreditCardValidator

信用卡號碼驗證。

相關 API: [CreditCardValidator](#)

UrlValidator

URL 格式驗證。

相關 API: [UrlValidator](#)

IPAddressValidator

IP 位址驗證。

參數:

- `IIPAddressTypes`: 可指定允許的 IP 類型 (IPv4、IPv6 或兩者)

相關 API: [IIPAddressValidator](#)

Boolean 驗證

TrueAssertValidator

當條件為 `true` 時，驗證通過。

特色：參數可使用 `Func<bool>` 將條件式延後到 `Validate()` 時執行。

相關 API: [TrueAssertValidator](#)

FalseAssertValidator

當條件為 `false` 時，驗證通過。

特色：參數可使用 `Func<bool>` 將條件式延後到 `Validate()` 時執行。

相關 API: [FalseAssertValidator](#)

重要注意事項

(X) IMPORTANT

除了 `RequiredValidator`、`BulkValidator` 和 `Boolean` 驗證外，當 `Value` 為 `null`、空字串或空格時，皆視為驗證通過。

這意味著如果您需要驗證必填欄位，必須同時使用 `RequiredValidator` 和其他型別驗證器。

範例

查看 [使用範例](#) 了解如何在實際應用中使用這些驗證器。

本文提供 FormValidators 的各種使用情境範例。

範例 1：驗證全部項目，並取得全部驗證的錯誤訊息

情境：您需要驗證表單中的所有欄位，並在驗證失敗時顯示所有錯誤訊息。

```
// 建立容器，並將 Validator 加入 Bulk 裡
BulkValidator validators = new();

// 驗證是否有值
validators.Add(new RequiredValidator("欄位一", "值一"));

// 驗證基本原則，當值為 Null、空格和空字串由 RequiredValidator 驗證
// 其他驗證遇到上述情況都回傳 true
validators.Add(new IntegerValidator("欄位一", "值一"));

// 一些特別情況可以使用 TrueAssertValidator 判斷第一個參數為 true 來表示驗證通過
// 也可用 FalseAssertValidator 判斷 false 為通過
validators.Add(new TrueAssertValidator(condition, "錯誤訊息"));

// 進行資料驗證
if (!validators.Validate()) {
    // 驗證失敗使用 validators.ErrorMessage 取得錯誤訊息
    // 多個錯誤訊息使用 <br /> 隔開
    // ErrorMessageWithBR 等同於 ErrorMessage

    // 您也可以使用其他格式：
    // ErrorMessageWithLF：使用 \n 隔開
    // ErrorMessageWithNewLine：使用 Environment.NewLine 隔開

    string errorMessage = validators.ErrorMessage;
}
```

輸出範例：

欄位一為必填。
欄位一必須為整數。
錯誤訊息

範例 2：驗證失敗後，停止驗證後續項目，只取得第一個錯誤訊息

情境：您希望在遇到第一個錯誤時立即停止驗證，提升效能並簡化錯誤訊息。

```

// 建立容器，並將 Validator 加入批量裡
// 第一個參數設為 true，表示遇到錯誤即停止
BulkValidator validators = new(true);

// TrueAssert 遇到 true，驗證成功
validators.Add(new TrueAssertValidator(true, "錯誤訊息一"));

// TrueAssert 遇到 false，驗證失敗（會在這裡停止）
validators.Add(new TrueAssertValidator(false, "錯誤訊息二"));

// 這個驗證不會執行
validators.Add(new TrueAssertValidator(false, "錯誤訊息三"));

if (!validators.Validate()) {
    // 驗證失敗使用 validators.ErrorMessage 取得錯誤訊息
    // 錯誤訊息為「錯誤訊息二」
    string errorMessage = validators.ErrorMessage;
}

```

輸出範例：

錯誤訊息二

範例 3：混合使用批量驗證

情境：您需要對某些欄位群組進行「遇錯即停」驗證，但整體表單仍要收集所有錯誤。

```

BulkValidator validators = new() {
    // 這個內層 Bulk 設定為 true (遇錯即停)
    new BulkValidator(true) {
        new TrueAssertValidator(true, "錯誤訊息一"),
        new TrueAssertValidator(false, "錯誤訊息二"), // 在這裡停止
        new TrueAssertValidator(false, "錯誤訊息三") // 不會執行
    },
    // 外層繼續執行其他驗證
    new TrueAssertValidator(true, "錯誤訊息四"),
    new TrueAssertValidator(false, "錯誤訊息五"),
    new TrueAssertValidator(false, "錯誤訊息六")
};

if (!validators.Validate()) {
    // validators.ErrorMessage 為 "錯誤訊息二<br />錯誤訊息五<br />錯誤訊息六"
}

```

```
        string errorMessage = validators.ErrorMessage;
    }
```

輸出範例：

```
錯誤訊息二<br />錯誤訊息五<br />錯誤訊息六
```

範例 4：使用 ValidationProvider 簡化建立程式碼

情境：使用 Fluent API 語法，讓程式碼更簡潔易讀。

```
BulkValidator validators = new(cfg => {
    // 增加一個驗證
    cfg.Add("欄位一", "值一",
        opt => opt.Required()
    );

    // 增加多個驗證
    cfg.Add("欄位二", "值二",
        opt => opt.Required(),
        opt => opt.DateTime()
    );

    // 條件式驗證：只有當 condition 為 true 時，才會驗證
    cfg.AddIf(
        condition, "欄位三", "值三",
        opt => opt.Required(),
        opt => opt.DateTime()
    );

    // 直接加入斷言驗證
    cfg.AddTrueAssert(true, "錯誤訊息");

    // 巢狀 Bulk 驗證
    cfg.AddBulk(_cfg => {
        _cfg.Add("欄位四", "值四",
            opt => opt.Required()
        );
    });
});

if (!validators.Validate()) {
    Console.WriteLine(validators.ErrorMessage);
}
```

範例 5：實際表單驗證範例

情境：驗證使用者註冊表單。

```
public class RegisterForm {
    public string UserName { get; set; }
    public string Email { get; set; }
    public string Password { get; set; }
    public string ConfirmPassword { get; set; }
    public string Age { get; set; }
    public string PhoneNumber { get; set; }

    public bool Validate(out string errorMessage) {
        BulkValidator validators = new(cfg => {
            // 使用者名稱：必填，長度 3-20
            cfg.Add("使用者名稱", UserName,
                opt => opt.Required(),
                opt => opt.ValueLength(3, 20)
            );

            // Email：必填，格式驗證
            cfg.Add("Email", Email,
                opt => opt.Required(),
                opt => opt.Email()
            );

            // 密碼：必填，長度至少 8
            cfg.Add("密碼", Password,
                opt => opt.Required(),
                opt => opt.ValueLength(8)
            );

            // 確認密碼：必填，與密碼相同
            cfg.Add("確認密碼", ConfirmPassword,
                opt => opt.Required(),
                opt => opt.Compare(Password, "密碼")
            );

            // 年齡：必填，整數，範圍 18-100
            cfg.Add("年齡", Age,
                opt => opt.Required(),
                opt => opt.Integer(18, 100)
            );

            // 手機號碼：非必填，但有值時需符合格式
            cfg.Add("手機號碼", PhoneNumber,

```

```

        opt => opt.MobilePhone()
    );
});

bool isValid = validators.Validate();
errorMessage = validators.ErrorMessage;
return isValid;
}
}

```

使用方式：

```

RegisterForm form = new() {
    UserName = "john_doe",
    Email = "john@example.com",
    Password = "password123",
    ConfirmPassword = "password123",
    Age = "25",
    PhoneNumber = "0912345678"
};

if (!form.Validate(out string errorMessage)) {
    Console.WriteLine("驗證失敗：");
    Console.WriteLine(errorMessage);
} else {
    Console.WriteLine("驗證通過！");
    // 繼續處理註冊邏輯...
}

```

進階技巧

動態條件驗證

使用 `AddIf` 根據條件決定是否進行驗證：

```

bool isCorporateAccount = true;

BulkValidator validators = new(cfg => {
    cfg.Add("公司名稱", companyName,
        opt => opt.Required()
    );

    // 只有企業帳號需要驗證統一編號
    cfg.AddIf(

```

```
isCorporateAccount, "統一編號", taxId,  
opt => opt.Required(),  
opt => opt.ValueLength(8, 8)  
);  
});
```

延遲執行驗證

使用 `Func<bool>` 延遲條件判斷：

```
cfg.AddTrueAssert(() => {  
    // 這個條件會在 Validate() 時才執行  
    return DateTime.Now.Hour >= 9 && DateTime.Now.Hour < 17;  
}, "服務時間為 09:00 - 17:00");
```

相關資源

- [驗證器介紹](#) - 了解所有可用的驗證器
- [客製化驗證](#) - 建立自訂驗證器
- [錯誤訊息設定](#) - 自訂錯誤訊息
- [API 參考](#) - 完整 API 文檔

當內建的驗證器無法滿足您的需求時，您可以建立自訂的驗證器。

方法 1：撰寫新的 FormValidator

實作 [IFormValidator](#) 介面來建立自訂驗證器。

實作介面

```
using CloudyWing.FormValidators;

public class CustomFormValidator : IFormValidator {
    public CustomFormValidator(string column, string value) {
        Column = column;
        Value = value;
    }

    public string Column { get; }

    public string Value { get; }

    public string ErrorMessage { get; private set; }

    public bool IsValid { get; private set; }

    public bool Validate() {
        // 實作您的驗證邏輯
        if (string.IsNullOrWhiteSpace(Value)) {
            IsValid = true; // 空值視為通過（遵循一般驗證器慣例）
            return true;
        }

        // 您的驗證邏輯
        bool isValid = YourValidationLogic(Value);

        if (!isValid) {
            ErrorMessage = $"{Column} 驗證失敗。";
        }
    }

    IsValid = isValid;
    return isValid;
}

private bool YourValidationLogic(string value) {
    // 在此撰寫您的驗證邏輯
    return true;
}
```

```
    }
}
```

使用自訂驗證器

```
BulkValidator validators = new();
validators.Add(new CustomFormValidator("欄位名稱", "欄位值"));

if (!validators.Validate()) {
    Console.WriteLine(validators.ErrorMessage);
}
```

方法 2：撰寫 ValidationProvider 的擴充方法

為了讓自訂驗證器也能使用 Fluent API 語法，您可以建立擴充方法。

建立擴充方法

```
using CloudyWing.FormValidators.Core;

public static class ValidationProviderExtensions {
    public static Func<string, string, CustomFormValidator> Custom(this
ValidationProvider provider)
        => (column, value) => new CustomFormValidator(column, value);
}
```

使用擴充方法

```
BulkValidator validators = new(cfg => {
    cfg.Add("欄位名稱", "欄位值",
        opt => opt.Required(),
        opt => opt.Custom()
    ); // 使用自訂驗證器
});

if (!validators.Validate()) {
    Console.WriteLine(validators.ErrorMessage);
}
```

實際範例：URL Slug 驗證器

以下是一個實際的範例，驗證字串是否為有效的 URL slug（如 `my-blog-post`）。

驗證器實作

```
using System.Text.RegularExpressions;
using CloudyWing.FormValidators;

public class UrlSlugValidator : IFormValidator {
    private static readonly Regex SlugRegex = new(
        @"^([a-z0-9]+(-[a-z0-9]+)*$",
        RegexOptions.Compiled
    );

    public UrlSlugValidator(string column, string value) {
        Column = column;
        Value = value;
    }

    public string Column { get; }

    public string Value { get; }

    public string ErrorMessage { get; private set; }

    public bool IsValid { get; private set; }

    public bool Validate() {
        if (string.IsNullOrWhiteSpace(Value)) {
            IsValid = true;
            return true;
        }

        bool isValid = SlugRegex.IsMatch(Value);

        if (!isValid) {
            ErrorMessage = $"{Column} 必須為小寫字母、數字和連字號組成，且不能以連字號開頭或
結尾。";
        }
    }

    IsValid = isValid;
    return isValid;
}
}
```

擴充方法

```
using CloudyWing.FormValidators.Core;

public static class CustomValidationProviderExtensions {
    public static Func<string, string, UrlSlugValidator> UrlSlug(this
ValidationProvider provider)
        => (column, value) => new UrlSlugValidator(column, value);
}
```

使用範例

```
BulkValidator validators = new(cfg => {
    cfg.Add("文章網址", articleSlug,
        opt => opt.Required(),
        opt => opt.UrlSlug()
    );
});

if (!validators.Validate()) {
    // 輸出：「文章網址 必須為小寫字母、數字和連字號組成，且不能以連字號開頭或結尾。」
    Console.WriteLine(validators.ErrorMessage);
}
```

最佳實踐

1. 遵循空值慣例

大多數內建驗證器在遇到空值時會回傳 `true` (除了 `RequiredValidator`)。建議您的自訂驗證器也遵循這個慣例：

```
public bool Validate() {
    if (string.IsNullOrWhiteSpace(Value)) {
        IsValid = true;
        return true; // 空值視為通過
    }

    // 您的驗證邏輯...
}
```

2. 提供有意義的錯誤訊息

錯誤訊息應明確告知使用者問題所在：

```
// X 不良範例  
ErrorMessage = $"{Column} 無效。";  
  
// ✓ 良好範例  
ErrorMessage = $"{Column} 必須為 8-20 位英數字元，且至少包含一個大寫字母。";
```

3. 考慮效能

如果驗證邏輯複雜，考慮使用快取或預編譯：

```
// 使用 static readonly 預編譯正規表示式  
private static readonly Regex Pattern = new(  
    @"^pattern$",  
    RegexOptions.Compiled  
);
```

4. 支援參數化

讓驗證器更靈活：

```
public class RangeValidator : IFormValidator {  
    private readonly int min;  
    private readonly int max;  
  
    public RangeValidator(string column, string value, int min, int max) {  
        Column = column;  
        Value = value;  
        this.min = min;  
        this.max = max;  
    }  
  
    // ... 實作驗證邏輯  
}
```

相關資源

- [驗證器介紹](#) - 了解內建驗證器的設計
- [使用範例](#) - 查看驗證器的實際應用
- [錯誤訊息設定](#) - 自訂錯誤訊息
- [IFormValidator](#) - 介面文檔
- [ValidationProvider](#) - ValidationProvider API

FormValidators 允許您自訂每個驗證器的預設錯誤訊息，讓錯誤提示更符合您的應用需求。

概述

使用 [ErrorMessageProvider](#) 類別中的各個 `SetAccessor` 方法來設定驗證器的預設錯誤訊息。

基本用法

```
using CloudyWing.FormValidators.Core;

// 設定必填欄位的預設錯誤訊息
ErrorMessageProvider.SetValueIsRequiredAccessor((column, value) => $"{column} is required.");

// 設定整數驗證的預設錯誤訊息
ErrorMessageProvider.SetValueIsIntegerAccessor((column, value) => $"{column} must be a valid integer.");
```

TIP

建議在應用程式啟動時（如 `Program.cs` 或 `Startup.cs`）統一設定錯誤訊息，確保全域一致性。

Validator 與 Accessor 對應關係

RequiredValidator

- **SetValueIsRequiredAccessor:** 欄位為必填時的錯誤訊息

```
ErrorMessageProvider.SetValueIsRequiredAccessor((column, value) =>
    $"{column}為必填。");
```

IntegerValidator

- **SetValueIsIntegerAccessor:** 值不是有效整數
- **SetValueGreaterOrEqualAccessor:** 值小於最小值
- **SetValueLessOrEqualAccessor:** 值大於最大值
- **SetValueInRangeAccessor:** 值不在指定範圍內

```
ErrorMessageProvider.SetValueIsIntegerAccessor((column, value)
    => $"{column}必須為整數。");
```

```
ErrorMessageProvider.SetValueGreaterOrEqualAccessor((column, value, min)
```

```
=> "${column}不得小於 ${min}。");  
  
ErrorMessageProvider.SetValueLessOrEqualAccessor((column, value, max)  
=> "${column}不得大於 ${max}。");  
  
ErrorMessageProvider.SetValueInRangeAccessor((column, value, min, max)  
=> "${column}必須介於 ${min} 與 ${max} 之間。");
```

NumberValidator

- **SetValueIsNumberAccessor**: 值不是有效數值
- **SetValueGreaterOrEqualAccessor**: 值小於最小值
- **SetValueLessOrEqualAccessor**: 值大於最大值
- **SetValueInRangeAccessor**: 值不在指定範圍內

```
ErrorMessageProvider.SetValueIsNumberAccessor((column, value) =>  
"${column}必須為數值。");
```

DateTimeValidator

- **SetValueIsDateTimeAccessor**: 值不是有效日期
- **SetValueGreaterOrEqualAccessor**: 日期早於最小日期
- **SetValueLessOrEqualAccessor**: 日期晚於最大日期
- **SetValueInRangeAccessor**: 日期不在指定範圍內

```
ErrorMessageProvider.SetValueIsDateTimeAccessor((column, value) =>  
"${column}必須為有效日期。");
```

ValueLengthValidator

- **SetValueLengthGreaterOrEqualAccessor**: 長度小於最小長度
- **SetValueLengthLessOrEqualAccessor**: 長度大於最大長度
- **SetValueLengthInRangeAccessor**: 長度不在指定範圍內

```
ErrorMessageProvider.SetValueLengthGreaterOrEqualAccessor((column, value, minLength)  
=> "${column}長度不得少於 ${minLength} 個字元。");
```

```
ErrorMessageProvider.SetValueLengthLessOrEqualAccessor((column, value, maxLength)  
=> "${column}長度不得超過 ${maxLength} 個字元。");
```

```
ErrorMessageProvider.SetValueLengthInRangeAccessor((column, value, minLength, maxLength)  
=> "${column}長度必須介於 ${minLength} 與 ${maxLength} 個字元之間。");
```

RegexValidator

- **SetValueMatchRegexAccessor:** 值不符合正規表示式

```
ErrorMessageProvider.SetValueMatchRegexAccessor((column, value, pattern)
=> $"{column}格式不正確。");
```

EmailValidator

- **SetValueIsEmailAccessor:** 值不是有效的 Email 格式

```
ErrorMessageProvider.SetValueIsEmailAccessor((column, value)
=> $"{column}必須為有效的 Email 地址。");
```

MobilePhoneValidator

- **SetValueIsMobilePhoneAccessor:** 值不是有效的手機號碼

```
ErrorMessageProvider.SetValueIsMobilePhoneAccessor((column, value)
=> $"{column}必須為有效的手機號碼。");
```

IdCardValidator

- **SetValueIsIdCardAccessor:** 值不是有效的身分證字號

```
ErrorMessageProvider.SetValueIsIdCardAccessor((column, value)
=> $"{column}必須為有效的身分證字號。");
```

CompareValidator

- **SetValueCompareAnotherColumnNameValueAccessor:** 兩個欄位的值不相等

```
ErrorMessageProvider.SetValueCompareAnotherColumnNameValueAccessor(
(column, value, compareColumn, compareValue)
=> $"{column}必須與{compareColumn}相同。");
```

IntegerLessThanValidator

- **SetValueLessThanAnotherColumnNameValueAccessor:** 整數值不小於比較值

```
ErrorMessageProvider.SetValueLessThanAnotherColumnNameValueAccessor(
(column, value, compareColumn, compareValue, allowedEqual)
```

```
=> allowedEqual
? $"{column}必須小於或等於{compareColumn}。"
: $"{column}必須小於{compareColumn}。");
```

NumberLessThanValidator

- **SetValueLessThanAnotherColumnValueAccessor**: 數值不小於比較值

i **NOTE**

與 `IntegerLessThanValidator` 共用相同的 Accessor。

DateTimeLessThanValidator

- **SetValueLessThanAnotherColumnValueAccessor**: 日期不早於比較日期

i **NOTE**

與 `IntegerLessThanValidator` 共用相同的 Accessor。

完整範例

以下範例展示如何在應用程式啟動時設定所有錯誤訊息：

```
using CloudyWing.FormValidators.Core;

public class Program {
    public static void Main(string[] args) {
        // 設定錯誤訊息
        ConfigureErrorMessages();

        // 您的應用程式邏輯...
    }

    private static void ConfigureErrorMessages() {
        // 基本驗證
        ErrorMessageProvider.SetValueIsRequiredAccessor((column, value)
            => $"{column}為必填欄位。");

        // 型別驗證
        ErrorMessageProvider.SetValueIsIntegerAccessor((column, value)
            => $"{column}必須為整數。");
    }
}
```

```

ErrorMessageProvider.SetValueIsNumberAccessor((column, value)
=> $"{column}必須為數值。");

ErrorMessageProvider.SetValueIsDateTimeAccessor((column, value)
=> $"{column}必須為有效日期。");

// 範圍驗證
ErrorMessageProvider.SetValueInRangeAccessor((column, value, min, max)
=> $"{column}必須介於 {min} 與 {max} 之間。");

// 長度驗證
ErrorMessageProvider.SetValueLengthInRangeAccessor((column, value,
minLength, maxLength)
=> $"{column}長度必須介於 {minLength} 與 {maxLength} 個字元之間。");

// 格式驗證
ErrorMessageProvider.SetValueIsEmailAccessor((column, value)
=> $"請輸入有效的 Email 地址。");

ErrorMessageProvider.SetValueIsMobilePhoneAccessor((column, value)
=> $"請輸入有效的手機號碼。");

// 比較驗證
ErrorMessageProvider.SetValueCompareAnotherColumnValueAccessor(
(column, value, compareColumn, compareValue)
=> $"{column}必須與{compareColumn}相同。");
}

}

```

多語言支援

您可以根據使用者的語言偏好動態設定錯誤訊息：

```

public static void ConfigureErrorMessages(string language) {
    switch (language) {
        case "en-US":
            ErrorMessageProvider.SetValueIsRequiredAccessor((column, value)
=> $"{column} is required.");
            ErrorMessageProvider.SetValueIsEmailAccessor((column, value)
=> $"{column} must be a valid email address.");
            break;

        case "zh-TW":
            ErrorMessageProvider.SetValueIsRequiredAccessor((column, value)

```

```

        => $"{column}為必填欄位。");
    ErrorMessageProvider.SetValueIsEmailAccessor((column, value)
        => $"{column}必須為有效的 Email 地址。");
    break;
}
}

```

最佳實踐

1. 統一管理

將所有錯誤訊息設定集中在一個方法中，便於維護：

```

public static class ErrorMessageConfig {
    public static void Configure() {
        // 所有錯誤訊息設定...
    }
}

```

2. 提供詳細資訊

錯誤訊息應包含足夠的資訊，讓使用者了解如何修正：

```

// X 不夠明確
ErrorMessageProvider.SetValueLengthInRangeAccessor((column, value, min, max)
    => $"{column}長度錯誤。");

// ✓ 明確且有幫助
ErrorMessageProvider.SetValueLengthInRangeAccessor((column, value, min, max)
    => $"{column}長度必須介於 {min} 與 {max} 個字元之間 (目前長度: {value?.Length ?? 0})。");

```

3. 考慮使用者體驗

使用友善的語氣，避免過於技術性的用語：

```

// X 過於技術性
ErrorMessageProvider.SetValueIsIntegerAccessor((column, value)
    => $"{column}: Invalid integer format. Expected: [-]?[0-9]+");

// ✓ 使用者友善
ErrorMessageProvider.SetValueIsIntegerAccessor((column, value)
    => $"{column}必須為整數 (例如: 123、-456)。");

```

相關資源

- [驗證器介紹](#) - 了解所有驗證器
- [使用範例](#) - 查看驗證器的實際應用
- [ErrorMessageProvider](#) - ErrorMessageProvider API