عنوان: **FluentValidation #1**
نویسنده: محمّد زارع
تاریخ: ۱۳۹۱/۰۸/۲۰ ۱۰:۵
آدرس: www.dotnettips.info
برچسب‌ها: Validation, FluentValidation

FluentValidation یک پروژه سورس باز برای اعتبارسنجی Business Objectها با استفاده از Fluent Interface و Lambda Expressions می‌باشد.

جهت نصب این کتابخانه دستور زیر را در Package Manager Console وارد نمایید:

```
PM> Install-Package FluentValidation
```

ایجاد یک Validator

برای تعریف مجموعه قوانین اعتبارسنجی برای یک موجودیت ابتدا بایستی یک کلاس ایجاد کرد که از AbstractValidator<T> مشتق می‌شود که T در اینجا برابر موجودیتی است که می‌خواهیم اعتبارسنجی کنیم. به عنوان مثال کلاس مشتری به صورت زیر را در نظر بگیرید:

```
public class Customer
{
      public int Id { get; set; }
      public string Surname { get; set; }
      public string Forename { get; set; }
      public decimal Discount { get; set; }
      public string Address { get; set; }
}
```

مجموعه قوانین اعتبارسنجی با استفاده از متد RuleFor و داخل متد سازنده کلاس Validator تعریف می‌شوند. به عنوان مثال برای اطمینان از اینکه مقدار خاصیت Surname برابر Null نباشد باید به صورت زیر عمل کرد:

```
using FluentValidation;

public class CustomerValidator : AbstractValidator<Customer>
{
      public CustomerValidator
      {
           RuleFor(customer => customer.Surname).NotNull();
      }
}
```

اعتبارسنجی زنجیره ای برای یک خاصیت

برای اعتبارسنجی یک خاصیت، می‌توان از چندین Validator باهم نیز استفاده کرد:

```
RuleFor(customer => customer.Surname).NotNull().NotEqual("foo");
```

در اینجا خاصیت Surname نباید Null باشد و همچنین مقدار آن نباید برابر "Foo" باشد.
برای اجراکردن اعتبارسنجی، ابتدا یک نمونه از کلاس Validator مان را ساخته و شیء ای را که می‌خواهیم اعتبارسنجی کنیم به متد Validate آن میفرستیم:

```
Customer customer = new Customer();
CustomerValidator validator = new CustomerValidator();
ValidationResult results = validator.Validate(customer);
```

خروجی متد Validate، یک ValidationResult است که شامل دو خاصیت زیر می‌باشد:

IsValid: از نوع bool برای تعیین اینکه اعتبارسنجی موفقیت آمیز بوده یا خیر.
Errors: یک مجموعه از ValidationFailure که جزئیات تمام اعتبارسنجی‌های ناموفق را شامل می‌شود.

به عنوان مثال قطعه کد زیر، جزئیات اعتبارسنجی‌های ناموفق را نمایش می‌دهد:

```
Customer customer = new Customer();
CustomerValidator validator = new CustomerValidator();

ValidationResult results = validator.Validate(customer);

if(! results.IsValid)
{
     foreach(var failure in results.Errors)
     {
          Console.WriteLine("Property " + failure.PropertyName + " failed validation. Error was: " +
failure.ErrorMessage);
     }
}
```

پرتاب استثناها (Throwing Exceptions)

به جای برگرداندن ValidationResult شما میتوانید با کمک متد ValidateAndThrow به FluentValidation بگویید که هنگام اعتبارسنجی ناموفق یک استثنا پرتاب کند:

```
Customer customer = new Customer();
CustomerValidator validator = new CustomerValidator();
validator.ValidateAndThrow(customer);
```

در این صورت Validator یک ValidationException را پرتاب خواهد کرد که دربردارنده‌ی پیام‌های خطا در خاصیت Errors خود می‌باشد.

استفاده از Validatorها برای Complex Properties

جهت درک این ویژگی تصور کنید که کلاس‌های مشتری و آدرس و همچنین کلاس‌های مربوط به اعتبارسنجی آن‌ها را به صورت زیر داریم:

```
public class Customer
{
     public string Name { get; set; }
     public Address Address { get; set; }
}

public class Address
{
     public string Line1 { get; set; }
     public string Line2 { get; set; }
     public string Town { get; set; }
     public string County { get; set; }
     public string Postcode { get; set; }
}

public class AddressValidator : AbstractValidator<Address>
{
     public AddressValidator()
     {
          RuleFor(address => address.Postcode).NotNull();
          //etc
     }
}

public class CustomerValidator : AbstractValidator<Customer>
{
```

```
     public CustomerValidator()
     {
          RuleFor(customer => customer.Name).NotNull();
          RuleFor(customer => customer.Address).SetValidator(new AddressValidator())
      }
}
```

در این صورت وقتی متد Validate کلاس CustomerValidator را فراخوانی نمایید AddressValidator نیز فراخوانی خواهد شد و نتیجه این اعتبارسنجی به صورت یکجا در یک ValidationResult برگشت داده خواهد شد.

استفاده از Validatorها برای مجموعه‌ها (Collections)

Validatorها همچنین می‌توانند بر روی خاصیت هایی که شامل مجموعه ای از یک شیء دیگر هستند نیز استفاده شوند. به عنوان مثال یک مشتری که دارای لیستی از سفارشات است را در نظر بگیرید:

```
public class Customer
{
     public IList<Order> Orders { get; set; }
}

public class Order
{
     public string ProductName { get; set; }
     public decimal? Cost { get; set; }
}

var customer = new Customer();
customer.Orders = new List<Order>
{
     new Order { ProductName = "Foo" },
     new Order { Cost = 5 }
};
```

کلاس OrderValidator نیز به صورت زیر خواهد بود:

```
public class OrderValidator : AbstractValidator<Order>
{
     public OrderValidator()
     {
          RuleFor(x => x.ProductName).NotNull();
          RuleFor(x => x.Cost).GreaterThan(0);
     }
}
```

این Validator می‌تواند داخل CustomerValidator مورد استفاده قرار بگیرد (با استفاده از متد SetCollectionValidator):

```
public class CustomerValidator : AbstractValidator<Customer>
{
     public CustomerValidator()
     {
          RuleFor(x => x.Orders).SetCollectionValidator(new OrderValidator());
     }
}
```

می توان با استفاده از متد Where یا Unless روی اعتبارسنجی شرط گذاشت:

```
RuleFor(x => x.Orders).SetCollectionValidator(new OrderValidator()).Where(x => x.Cost != null);
```

گروه بندی قوانین اعتبارسنجی

RuleSetها به شما این امکان را می‌دهند تا بعضی از قوانین اعتبارسنجی را داخل یک گروه قرار دهید تا با یکدیگر اجرا شوند. در حالی که دیگر قوانین نادیده گرفته می‌شوند.
برای مثال تصور کنید شما سه خاصیت در کلاس Person دارید که شامل (Id, Surname, Forename) می‌باشند و همچنین یک قانون برای هرکدام از آن ها. میتوان قوانین مربوط به Surname و Forename را در یک RuleSet مجزا به نام Names قرار داد:

```
public class PersonValidator : AbstractValidator<Person>
{
     public PersonValidator()
     {
          RuleSet("Names", () =>
          {
               RuleFor(x => x.Surname).NotNull();
               RuleFor(x => x.Forename).NotNull();
          });

          RuleFor(x => x.Id).NotEqual(0);
      }
}
```

در اینجا دو خاصیت Surname و Forename با یکدیگر داخل یک RuleSet به نام Names گروه شده اند. برای اعتبارسنجی جداگانه این گروه نیز به صورت زیر می‌توان عمل کرد:

```
var validator = new PersonValidator();
var person = new Person();
var result = validator.Validate(person, ruleSet: "Names");
```

این ویژگی به شما این امکان را می‌دهد تا یک Validator پیچیده را به چندین قسمت کوچکتر تقسیم کرده و توانایی اعتبارسنجی این قسمت‌ها را به صورت جداگانه داشته باشید.