عنوان: **سفارشی سازی Binding یک خصوصیت از طریق Attributes**
نویسنده: محمد رعیت پیشه
تاریخ: ۱۳۹۲/۰۸/۲۵ ۲۲:۰
آدرس: www.dotnettips.info
برچسب‌ها: MVC, Enum, custom model binder

اگر با MVC کار کرده باشید حتما با ModelBinding آن آشنا هستید؛ DefaultModelBinder توکار آن که در اکثر مواقع، باری زیادی را از روی دوش برنامه نویسان بر می‌دارد و کار را برای آنان راحتتر می‌کند. اما در بعضی مواقع این مدل بایندر پیش فرض ممکن است پاسخگوی نیاز ما در بایند کردن یک خصوصیت از یک مدل خاص نباشد، برای همین ما نیاز داریم که کمی آن را سفارشی سازی کنیم.

برای این کار ما دو راه داریم:

1) یک مدل بایندر جدید را با پیاده سازی IModelBinder تهیه کنیم. (در این حالت ما مجبوریم که مدل بایندر را از ابتدا جهت بایند کردن کلیه مقادیر شی مدل خود، بازنویسی کنیم و در واقع امکان انتساب آن‌را در سطح فقط یک خصوصیت نداریم.) (نحوه پیاده سازی قبلا در اینجا مطرح شده)

2) ModelBinder پیش فرض را جهت پاسخگویی به نیازمان توسعه دهیم. (که در این مطلب قصد آموزشش را داریم.)

فرض کنید که می‌خواهید بر اساس یک Enum در صفحه، یک DropDownFor معادل را قرار بدید که به طور خودکار رشته انتخاب شده را به یک خصوصیت مدل که از نوع بایت هست بایند بکند.

از طریق کد زیر یک DropDownListFor برای Enum مورد نظر در مدل ایجاد کنیم:

```
@Html.DropDownListFor(model => model.AccountType, new
SelectList(Enum.GetNames(typeof(Enums.AccountType))))
```

و کلاس Enum مورد نظر :

```
public enum AccountType : byte
{
   0 = مدیر,
   1 = کاربر_حقیقی,
   2 = کاربر_حقوقی,
}
```

حالا برای اینکه این مقدار انتخابی به صورت خودکار و از طریق امکان binding توکار خود MVC به خصوصیت AccountType مقدار دهی شود باید یک PropertyBindAttribute سفارشی بنویسیم، برای اینکار یک کلاس جدید با نام CustomBinding می‌سازیم و کدهای زیر را به آن اضافه می‌کنیم :

```
namespace MvcApplication1.Models
{
    [AttributeUsage(AttributeTargets.Property, AllowMultiple = false)]
    public abstract class PropertyBindAttribute : Attribute
    {
        public abstract bool BindProperty(ControllerContext controllerContext,
        ModelBindingContext bindingContext, PropertyDescriptor propertyDescriptor);
    }

    public class ExtendedModelBinder : DefaultModelBinder
    {
        protected override void BindProperty(ControllerContext controllerContext,
            ModelBindingContext bindingContext, PropertyDescriptor propertyDescriptor)
        {
            if (propertyDescriptor.Attributes.OfType<PropertyBindAttribute>().Any())
            {
                var modelBindAttr =
```

```
propertyDescriptor.Attributes.OfType<PropertyBindAttribute>().FirstOrDefault();

                if (modelBindAttr.BindProperty(controllerContext, bindingContext, propertyDescriptor))
                    return;
            }

            base.BindProperty(controllerContext, bindingContext, propertyDescriptor);
        }
    }
}
```

در کد بالا ما تمام کلاس هایی را که از PropertyBindAttribute مشتق شده باشند را به DefaultModelBinder اضافه می‌کنیم.
این کد فقط یک بار نوشته می‌شود و از این به بعد هر بایندر سفارشی که بسازیم به بایندر پیشفرض اضافه خواهد شد.

حالا از طریق کدهای زیر، ما بایندر سفارشیِ خصوصیتِ خودمان را به کلاس اضافه می‌کنیم :

```
public class AccountTypeBindAttribute : PropertyBindAttribute
    {
        public override bool BindProperty(ControllerContext controllerContext,
            ModelBindingContext bindingContext, PropertyDescriptor propertyDescriptor)
        {
            if (propertyDescriptor.PropertyType == typeof(byte))
            {
                HttpRequestBase request = controllerContext.HttpContext.Request;

                byte accountType = (byte)Enum.Parse(typeof(Enums.AccountType),
request.Form["AccountType"]);
                propertyDescriptor.SetValue(bindingContext.Model, accountType);

                return true;
            }
            return false;
        }
    }
```

در کد بالا ما مقدار رشته‌ای را که از DropDownListFor ارسال شده، به مقدار عددی متناظر تعریف شده آن در Enum تبدیل می‌کنیم و آن را به خصوصیت مورد نظر بازگشت می‌دهیم، از این به بعد فقط برای فیلدی که به شکل زیر نشانه گذاری شده باشد، از این کلاس بایندر سفارشی استفاده می‌کنیم و مدل بایندر پیش فرض هم کار خود را خواهد کرد و بقیه مقادیر را بایند خواهد کرد.
اطلاعات بیشتر

```
[AccountTypeBindAttribute]
public byte AccountType { get; set; }
```

حالا باید این کلاس گسترش یافته ModelBinder را به عنوان بایندر پیش فرض MVC قرار بدهیم، برای اینکار کد زیر را به فایل Global.asax.cs اضافه کنید:

```
ModelBinders.Binders.DefaultBinder = new ExtendedModelBinder();
```

کار ما دیگر تمام است و تا اینجای کار همه چیز به‌درستی کار می‌کند ... تا اینکه شما تصمیم میگیرید که از jquery.validate.unobtrusive برای اعتبار سنجی سمت کاربر استفاده کنید و می‌بینید به DropDownListFor شما هم ایراد میگیرید که حتما باید از نوع عددی باشد
نوع کاربر : The field must be a number. برای حل این مشکل هم باید به صورت دستی validation سمت کاربر رو برای این DropDownListFor غیرفعال کرد. برای این منظور باید کدهای DropDownListFor که در صفحه گذاشتید را به شکل زیر تغییر بدید:

```
@Html.DropDownListFor(model => model.AccountType, new
SelectList(Enum.GetNames(typeof(Enums.AccountType))),new Dictionary<string, object>() {{ "data-val",
"false" }})
```

نظرات خوانندگان

نویسنده: مهران بادامی
تاریخ: ۱۳۹۲/۰۸/۲۶ ۲۰:۲۴

سلام
من یهBinding نوشتم برای تاریخ که شمسی از کاربر گرفته به میلادی میدهد برای دخیره تو DB ولی وقتی میخواهم تاریخ که تو دیتابیسم میلادی هست را نشان بدهم همون میلادی نشون میده برای نمایش به شمسی چه کنم؟

نویسنده: محمد رعیت پیشه
تاریخ: ۱۳۹۲/۰۸/۲۸ ۱۰:۲۸

_ یک راه:
ایجاد یک HTML Helper سفارشی.
http://www.dotnettips.info/post/811/asp-net-mvc-8