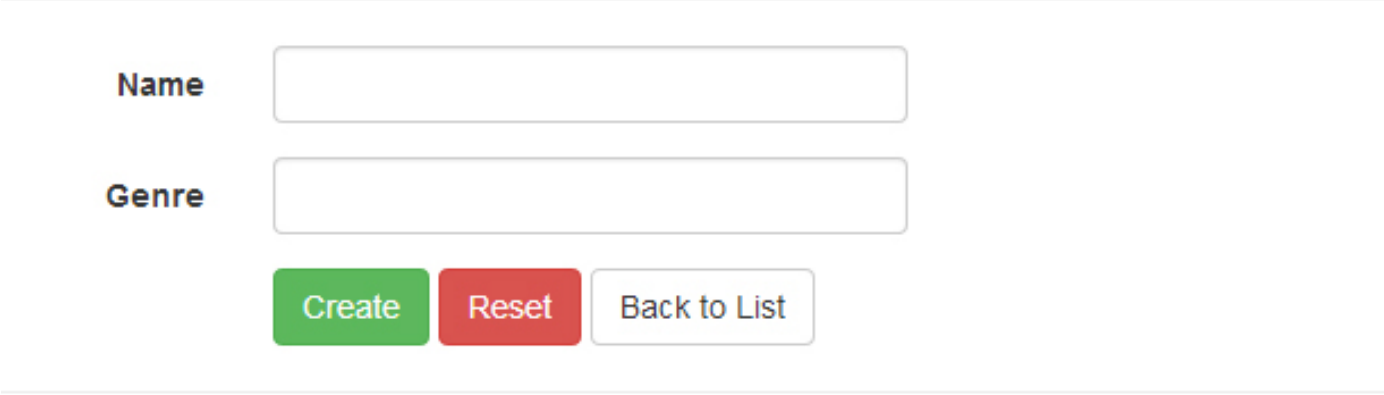
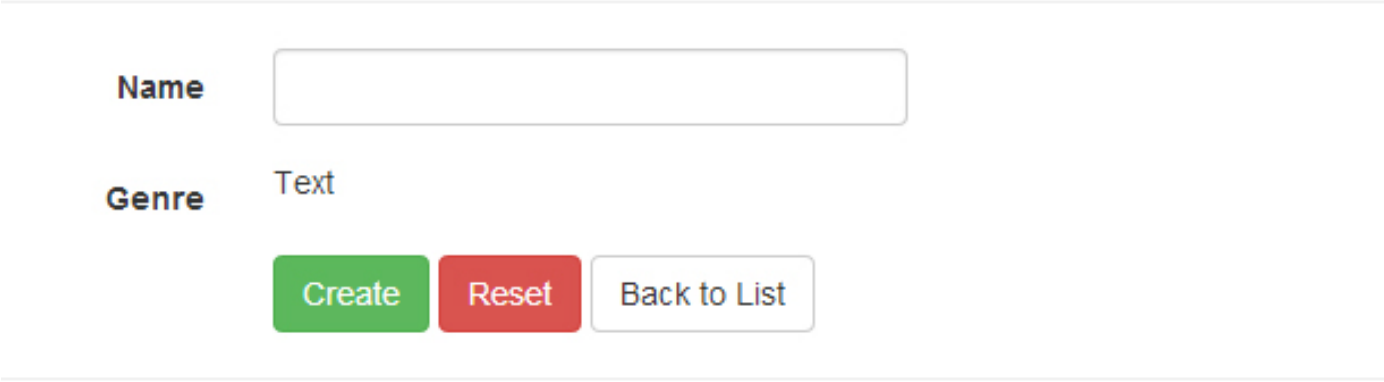
عنوان: **قالب‌های سفارشی برای HtmlHelperها**
نویسنده: علی اکبر کورش فر
تاریخ: ۱۳۹۳/۰۵/۱۶ ۱۳:۳۵
آدرس: www.dotnettips.info
گروه‌ها: MVC, HTML helpers

در ابتدای بحث، برای آشنایی بیشتر با HTML Helperها به مطالعه این مقاله بپردازین.
در این مقاله قرار است برای یک HTML Helper خاص، قالب نمایشی اختصاصی خودمان را طراحی کنیم و به نحوی HTML Helper موجود را سفارشی سازی کنیم. به عنوان مثال می‌خواهیم خروجی یک EditorFor() برای یک نوع خاص، به حالت دلخواهی باشد که ما خودمان آن را تولیدش کردیم؛ یا اصلا نه. حتی می‌شود برای خروجی یک EditorFor() که خصوصیتی از جنس string را می‌خواهیم به آن انتساب دهیم، به جای تولید input، یک مقدار متنی را برگردانیم. به این حالت:

```
<div>
            @Html.LabelFor(model => model.Name, htmlAttributes: new { @class = "control-label col-md-2"
})
            <div>
                @Html.EditorFor(model => model.Name, new { htmlAttributes = new { @class = "form-
control" } })
                @Html.ValidationMessageFor(model => model.Name, "", new { @class = "text-danger" })
            </div>
        </div>

        <div>
            @Html.LabelFor(model => model.Genre, htmlAttributes: new { @class = "control-label col-md-
2" })
            <div>
                @Html.EditorFor(model => model.Genre, new { htmlAttributes = new { @class = "form-
control" } })
                @Html.ValidationMessageFor(model => model.Genre, "", new { @class = "text-danger" })
            </div>
        </div>
```





در ادامه یک پروژه‌ی عملی را شروع کرده و در آن کاری را که می‌خواهیم، انجام می‌دهیم. پروژه‌ی ما به این شکل می‌باشد که قرار است در آن به ثبت کتاب بپردازیم و برای هر کتاب هم یک سبک داریم و قسمت سبک کتاب‌های ما یک Enum است که از قبل می‌خواهیم مقدارهایش را تعریف کنیم.

مدل برنامه

```
public class Books
    {
        public int Id { get; set; }
        [Required]
        [StringLength(255)]
        public string Name { get; set; }
        public Genre Genre { get; set; }
    }
```

```
public enum Genre
    {
        [Display(Name = "Non Fiction")]
        NonFiction,
        Romance,
        Action,
        [Display(Name = "Science Fiction")]
        ScienceFiction
    }
```

در داخل کلاس Books یک خصوصیت از جنس Genre برای سبک کتاب‌ها داریم و در داخل نوع شمارشی Genre، سبک‌های ما تعریف شده‌اند. همچنین هر کدام از سبک‌ها هم به ویژگی Display مزین شده‌اند تا بتونیم بعدا از مقدار آنها استفاده کنیم.

کنترلر برنامه

```
public class BookController : Controller
    {
        // GET: Book
        public ActionResult Index()
        {
            return View(DataAccess.DataContext.Book.ToList());
        }

        public ActionResult Create()
        {
            return View();
        }

        [HttpPost]
        [ValidateAntiForgeryToken]
        public ActionResult Create(Books model)
        {
            if (!ModelState.IsValid)
                return View(model);

            try
            {
                DataAccess.DataContext.Book.Add(model);
                DataAccess.DataContext.SaveChanges();
                return RedirectToAction("Index");
            }
            catch (Exception ex)
            {
                ModelState.AddModelError("", ex.Message);
                return View(model);
            }
        }

        public ActionResult Edit(int id)
        {
            try
            {
                var book = DataAccess.DataContext.Book.Find(id);
                return View(book);
```

```
            }
            catch (Exception ex)
            {
                return View("Error");
            }
        }

        [HttpPost]
        [ValidateAntiForgeryToken]
        public ActionResult Edit(Books model)
        {
            if (!ModelState.IsValid)
                return View(model);

            try
            {
                DataAccess.DataContext.Book.AddOrUpdate(model);
                DataAccess.DataContext.SaveChanges();
                return RedirectToAction("Index");
            }
            catch (Exception ex)
            {
                ModelState.AddModelError("", ex.Message);
                return View(model);
            }
        }

        public ActionResult Details(int id)
        {
            try
            {
                var book = DataAccess.DataContext.Book.Find(id);
                return View(book);
            }
            catch (Exception ex)
            {
                return View("Error");
            }
        }
    }
```

در قسمت کنترلر هم کار خاصی جز عملیات اصلی نوشته نشده‌است. لیست کتاب‌ها را از پایگاه داده بیرون آوردیم و از طریق اکشن Index به نمایش گذاشتیم. با اکشن‌های Edit، Create و Details هم کارهای روتین مربوط به خودشان را انجام دادیم. نکته‌ی قابل تذکر، DataAccess می‌باشد که کلاسی است که با آن ارتباط برقرار شده با EF و سپس اطلاعات واکشی و تزریق می‌شوند.

View مربوط به اکشن Create برنامه

```
@using Book.Entities
@model Book.Entities.Books

@{
    ViewBag.Title = "Create";
}
<h2>New Book</h2>
@using (Html.BeginForm())
{
    @Html.AntiForgeryToken()

    <div>
        <h4>Books</h4>
        <hr />
        @Html.ValidationSummary(true, "", new { @class = "text-danger" })
        <div>
            @Html.LabelFor(model => model.Name, htmlAttributes: new { @class = "control-label col-md-2"
})
            <div>
                @Html.EditorFor(model => model.Name, new { htmlAttributes = new { @class = "form-
control" } })
                @Html.ValidationMessageFor(model => model.Name, "", new { @class = "text-danger" })
            </div>
        </div>

        <div>
```

```
            @Html.LabelFor(model => model.Genre, htmlAttributes: new { @class = "control-label col-md-
2" })
            <div>
                @Html.EditorFor(model => model.Genre, new { htmlAttributes = new { @class = "form-
control" } })
                @Html.ValidationMessageFor(model => model.Genre, "", new { @class = "text-danger" })
            </div>
        </div>

        <div>
            <div>
                <input type="submit" value="Create" />
                <input type="reset" value="Reset" />
                @Html.ActionLink("Back to List", "Index", null, new {@class="btn btn-default"})
            </div>
        </div>
    </div>
}

@section Scripts {
    @Scripts.Render("~/bundles/jqueryval")
}
```

View برنامه هم همان ویویی است که خود Visual Studio برای ما ساخته‌است. به جز یک سری دست‌کاری‌هایی داخل سی‌اس‌اس، هدف از گذاشتن View مربوط به Create این بود که قرار است بر روی این قسمت کار کنیم. اگر پروژه رو اجرا کنید و به قسمت Create بروید، مشاهده خواهید کرد که برای Genre یک input ساخته شده‌است که کاربر باید در آن مقدار وارد کند. ولی اگر یادتان باشد، ما سبک‌های نگارشی خودمان را در نوع شمارشی Genre ایجاد کرده بودیم. پس عملا باید یک لیست به کاربر نشان داده شود که تا از آن لیست، نوع را انتخاب کند. می‌توانیم بیایم همینجا در داخل View مربوطه، به‌جای استفاده از HTML Helper پیش‌فرض، از DropDownList یا EnumFor استفاده کنیم و به طریقی این لیست را ایجاد کنیم. ولی چون قرار است در این مثال به شرح موضوع مقاله خودمان بپردازیم، این کار را انجام نمی‌دهیم.

در حقیقت می‌خوایم متد EditorFor را طوری سفارشی سازی کنیم که برای نوع شمارشی Genre، به صورت خودکار یک لیست ایجاد کرده و برگرداند. از نسخه‌ی سوم ASP.NET MVC به بعد این امکان برای توسعه دهنده‌ها فراهم شده‌است. شما می‌توانید در پوشه‌ی Shared داخل پوشه Views برنامه، پوشه‌ای را به اسم EditorTemplates ایجاد کنید؛ همینطور DisplayTemplates و برای نوع خاصی که می‌خواهید سفارشی‌سازی را برای آن انجام دهید، یک PartialView بسازید.

```
Views/Shared/DisplayTemplates/<type>.cshtml
```

یک PartialView در داخل پوشه EditorTemplates به نام Genre.cshtml ایجاد کنید. برای اینکه مشاهده کنید چطور کار می‌کند، کافی‌است یک فایل متنی اینجا تهیه کرده و بعد پروژه را اجرا کرده و به قسمت Create روید تا تغییرات را مشاهده کنید. بله! به‌جای inputی که از قبل وجود داشت، فقط متن شما آنجا نوشته شده‌است. (به عکسی که در بالا قرار دارد هم می‌تونید نگاه کنید)

کاری که الان میخواهیم انجام دهیم این است که یک SelectListItem ایجاد کرده تا مقدارهای نوع Genreمان داخلش باشد و بتوانیم به راحتی برای ساختن DropDownList از آن استفاده کنیم. برای این کار Helper مخصوص خودمان را ایجاد می‌کنیم. پوشه‌ای به اسم Helpers در کنار پوشه‌های Controllers، Models ایجاد می‌کنیم و در داخل آن کلاسی به اسم EnumHelpers می‌سازیم.

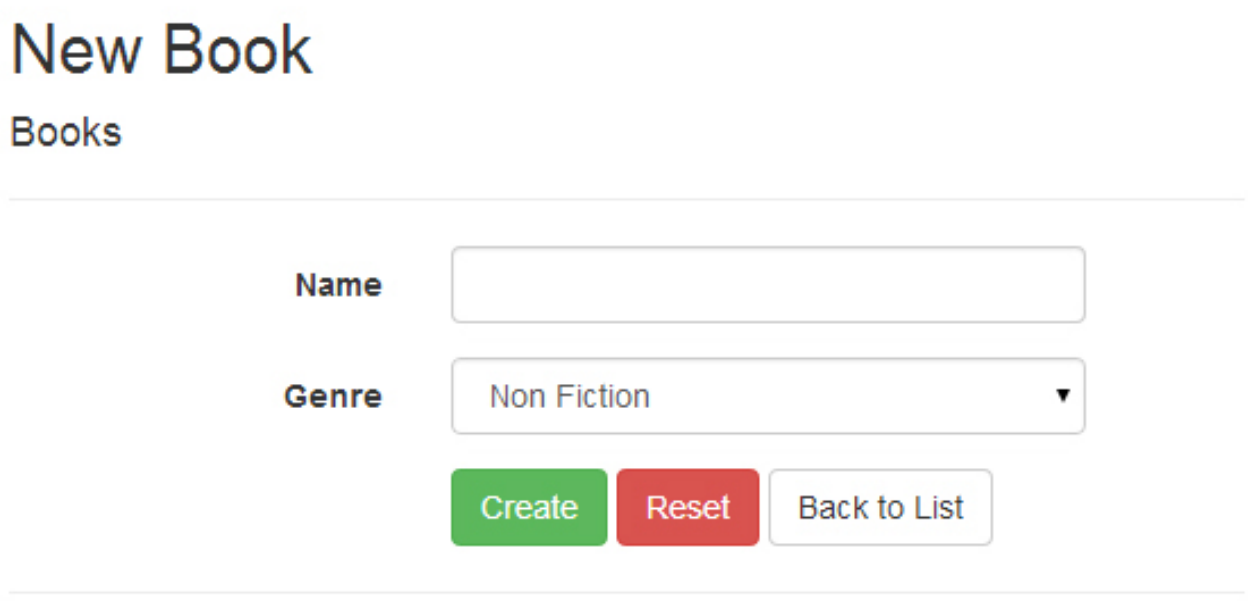
```
public static class EnumHelpers
    {
        public static IEnumerable<SelectListItem> GetItems(
            this Type enumType, int? selectedValue)
        {
            if (!typeof(Enum).IsAssignableFrom(enumType))
            {
                throw new ArgumentException("Type must be an enum");
            }

            var names = Enum.GetNames(enumType);
            var values = Enum.GetValues(enumType).Cast<int>();

            var items = names.Zip(values, (name, value) =>
```

```
                    new SelectListItem
                    {
                        Text = GetName(enumType, name),
                        Value = value.ToString(),
                        Selected = value == selectedValue
                    }
                );
            return items;
        }

        static string GetName(Type enumType, string name)
        {
            var result = name;

            var attribute = enumType
                .GetField(name)
                .GetCustomAttributes(inherit: false)
                .OfType<DisplayAttribute>()
                .FirstOrDefault();

            if (attribute != null)
            {
                result = attribute.GetName();
            }

            return result;
        }
    }
```

در توضیح کد بالا عنوان کرد که متدها به‌صورت متدهای الحاقی به نوع Type نوشته شدند. کار خاصی در بدنه‌ی متدها انجام نشده‌است. در بدنه‌ی متد اول لیست آیتم‌ها را تولید کردیم. در هنگام ساخت SelectListItem برای گرفتن Text، متد GetName را صدا زدیم. برای اینکه بتوانیم مقدار ویژگی Display که در هنگام تعریف نوع شمارشی استفاده کردیم را بدست بیاریم، باید چک کنیم ببینیم که آیا این آیتم به این ویژگی مزین شده‌است یا نه. اگر شده بود مقدار را می‌گیریم و به خصوصیت Text متد اول انتساب می‌دهیم.

```
@using Book.Entities
@using Book.Web.Helpers
@{
    var items = typeof(Genre).GetItems((int?)Model);
}

@Html.DropDownList("", items, new {@class="form-control"})
```

کدهایی که در بالا مشاهده می‌کنید کدهایی می‌باشند که قرار است داخل PartialViewی Genre قرار دهیم که در پوشه‌ی EditorTemplates ساختیم. ابتدا آمدیم آیتم‌ها را گرفتیم و بعد به DropDownList دادیم تا لیست نوع را برای ما بسازد. حالا اگه برنامه را اجرا کنید می‌بینید که EditorFor برای شما یه لیست از نوع شمارشی ساخته و حالا قابل استفاده هست.



کدهای کامل این مثال را از اینجا می‌توانید دریافت کنید

HtmlHelpersEditor.rar