عنوان: **قابلیت Templated Razor Delegate**
نویسنده: سیروان عفیفی
تاریخ: ۱۳۹۳/۰۹/۲۳ ۱۰:۵
آدرس: www.dotnettips.info
گروه‌ها: MVC, Razor

Razor دارای قابلیتی با نام Templated Razor Delegates است. همانطور که از نام آن مشخص است، یعنی Razor Template هایی که Delegate هستند. در ادامه این قابلیت را با ذکر چند مثال توضیح خواهیم داد. **مثال اول:**
می‌خواهیم تعدادی تگ li را در خروجی رندر کنیم، این کار را می‌توانیم با استفاده از Razor helpers نیز به این صورت انجام دهیم:

```
@helper ListItem(string content) {
 <li>@content</li>
}
<ul>
 @foreach(var item in Model) {
 @ListItem(item)
 }
</ul>
```

همین کار را می‌توانیم توسط Templated Razor Delegate به صورت زیر نیز انجام دهیم:

```
@{
 Func<dynamic, HelperResult> ListItem = @<li>@item</li>;
}
<ul>
 @foreach(var item in Model) {
 @ListItem(item)
 }
</ul>
```

برای اینکار از نوع Func استفاده خواهیم کرد. این Delegate یک پارامتر را می‌پذیرد. این پارامتر می‌تواند از هر نوعی باشد. در اینجا از نوع dynamic استفاده کرده‌ایم. خروجی این Delegate نیز یک HelperResult است. همانطور که مشاهده می‌کنید آن را برابر با الگویی که قرار است رندر شود تعیین کرده‌ایم. در اینجا از یک پارامتر ویژه با نام item استفاده شده است. نوع این پارامتر dynamic است؛ یعنی همان مقداری که برای پارامتر ورودی Func انتخاب کردیم. در نتیجه پارامتر ورودی یعنی رشته item جایگزین @item درون Delegate خواهد شد. در واقع دو روش فوق خروجی یکسانی را تولید میکنند. برای حالت‌هایی مانند کار با آرایه‌ها و یا Enumerations بهتر است از روش دوم استفاده کنید؛ از این جهت که نیاز به کد کمتری دارد و نگهداری آن خیلی از روش اول ساده‌تر است.

مثال دوم:

اجازه دهید یک مثال دیگر را بررسی کنیم. به طور مثال معمولاً در یک فایل Layout برای بررسی کردن وجود یک section از کدهای زیر استفاده می‌کنیم:

```
<header>
    @if (IsSectionDefined("Header"))
    {
        @RenderSection("Header")
    }
    else
    {
        <div>Default Content for Header Section</div>
    }
</header>
```

روش فوق به درستی کار خواهد کرد اما می‌توان آن را با یک خط کد، درون ویو نیز نوشت. در واقع می‌توانیم با استفاده از Templated Razor Delegate یک متد الحاقی برای کلاس ViewPage بنویسیم؛ به طوریکه یک محتوای پیش‌فرض را برای حالتی که

section خاصی وجود ندارد، نمایش دهد:

```
public static HelperResult RenderSection(this WebViewPage page, string name,
    Func<dynamic, HelperResult> defaultContent)
{
    if (page.IsSectionDefined(name))
    {
        return page.RenderSection(name);
    }
    return defaultContent(null);
}
```

بنابراین درون ویو می‌توانیم از متد الحاقی فوق به این صورت استفاده کرد:

```
<header>
   @this.RenderSection("Header", @<div>Default Content for Header Section</div>)
</header>
```

نکته: جهت بوجود نیامدن تداخل با نمونه اصلی RenderSection درون ویو، از کلمه this استفاده کرده‌ایم.

مثال سوم: شبیه‌سازی کنترل Repeater:
یکی از ویژگی‌های جذاب WebForm کنترل Repeater است. توسط این کنترل به سادگی می‌توانستیم یکسری داده را نمایش دهیم؛ این کنترل در واقع یک کنترل DataBound و همچنین یک Templated Control است. یعنی در نهایت کنترل کاملی بر روی Markup آن خواهید داشت. برای نمایش هر آیتم خاص داخل لیست می‌توانستید از ItemTemplate استفاده کنید. همچنین می‌توانستید از AlternatingItemtemplate استفاده کنید. یا اگر می‌خواستید هر آیتم را با چیزی از یکدیگر جدا کنید، می‌توانستید از SeparatorTemplate استفاده کنید. در این مثال می‌خواهیم همین کنترل را در MVC شبیه‌سازی کنیم.
به طور مثال ویوی Index ما یک مدل از نوع IEnumerable<string> را دارد:

```
@model IEnumerable<string>
@{
    ViewBag.Title = "Test";
}
```

و اکشن متد ما نیز به این صورت اطلاعات را به ویوی فوق پاس میدهد:

```
public ActionResult Index()
{
    var names = new string[]
    {
        "Vahid Nasiri",
        "Masoud Pakdel",
        ...
     };

    return View(names);
}
```

اکنون در ویوی Index می‌خواهیم هر کدام از اسامی فوق را نمایش دهیم. اینکار را می‌توانیم درون ویو با یک حلقه‌ی foreach و بررسی زوج با فرد بودن ردیف‌ها انجام دهیم اما کد زیادی را باید درون ویو بنویسیم. اینکار را می‌توانیم درون یک متد الحاقی نیز انجام دهیم. بنابراین یک متد الحاقی برای HtmlHelper به صورت زیر خواهیم نوشت:

```
public static HelperResult Repeater<T>(this HtmlHelper html,
    IEnumerable<T> items,
    Func<T, HelperResult> itemTemplate,
    Func<T, HelperResult> alternatingitemTemplate = null,
    Func<T, HelperResult> seperatorTemplate = null)
{
    return new HelperResult(writer =>
    {
        if (!items.Any())
        {
```

```
            return;
        }
        if (alternatingitemTemplate == null)
        {
            alternatingitemTemplate = itemTemplate;
        }
        var lastItem = items.Last();
        int ii = 0;
        foreach (var item in items)
        {
           var func = ii % 2 == 0 ? itemTemplate : alternatingitemTemplate;
           func(item).WriteTo(writer);
           if (seperatorTemplate != null && !item.Equals(lastItem))
           {
               seperatorTemplate(item).WriteTo(writer);
           }
           ii++;
        }
    });
}
```

توضیح کدهای فوق: خوب، همانطور که ملاحظه می‌کنید متد را به صورت Generic تعریف کرده‌ایم، تا بتواند با انواع نوع‌ها به خوبی کار کند. زیرا ممکن است لیستی از اعداد را داشته باشیم. از آنجائیکه این متد را برای کلاس HtmlHelper می‌نویسیم، پارامتر اول آن را از این نوع می‌گیریم. پارامتر دوم آن، آیتم‌هایی است که می‌خواهیم نمایش دهیم. پارامترهای بعدی نیز به ترتیب برای ItemTemplate، AlternatingItemtemplate و SeperatorItemTemplate تعریف شده‌اند و از نوع Delegate با پارامتر ورودی T و خروجی HelperResult هستند. در داخل متدمان یک HelperResult را برمیگردانیم. این کلاس یک Action را از نوع TextWriter از ورودی می‌پذیرد. اینکار را با ارائه یک Lambda Expression با نام writer انجام می‌دهیم. در داخل این Delegate به تمام منطقی که برای نمایش یک آیتم نیاز هست دسترسی داریم.

ابتدا بررسی کرده‌ایم که آیا آیتم برای نمایش وجود دارد یا خیر. سپس اگر AlternatingItemtemplate برابر با null بود همان ItemTemplate را در خروجی نمایش خواهیم داد. مورد بعدی دسترسی به آخرین آیتم در Collection است. زیرا بعد از هر آیتم باید یک SeperatorItemTemplate را در خروجی نمایش دهیم. سپس توسط یک حلقه درون آیتم‌ها پیمایش میکنیم و ItemTemplate و AlternatingItemtemplate را توسط متغیر func از یکدیگر تشخیص می‌دهیم و در نهایت درون ویو به این صورت از متد الحاقی فوق استفاده می‌کنیم:

```
@Html.Repeater(Model, @<div>@item</div>, @<p>@item</p>, @<hr/>)
```

متد الحاقی فوق قابلیت کار با انواع ورودی‌ها را دارد به طور مثال مدل زیر را در نظر بگیرید:

```
public class Product
{
        public int Id { set; get; }
        public string Name { set; get; }
}
```

می‌خواهیم اطلاعات مدل فوق را در ویوی مربوط درون یک جدول نمایش دهیم، می‌توانیم به این صورت توسط متد الحاقی تعریف شده اینکار را به این صورت انجام دهیم:

```
<table>
    <tr>
        <td>Id</td>
        <td>Name</td>
    </tr>
    @Html.Repeater(Model, @<tr><td>@item.Id</td><td>@item.Name</td></tr>)
</table>
```