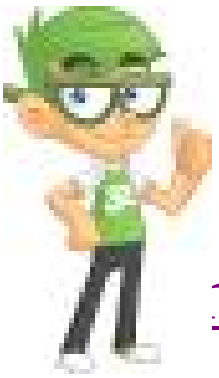
پایگاه تخصصی آموزشی کد سیتی

دانلود فیلم های آموزشی به زبان فارسی

دانلود نرم افزار - کتاب الکترونیکی – مقالات آموزشی – پروژه های دانشجویی

اخبار کنکور – دانشگاه – موقعیت های شغلی – تحصیل در خارج

همه و همه در وب سایت کد سیتی



http://www.codecity.ir/

دریافت جدیدترین مطالب آموزشی در ایمیل شما

دریافت جدیدترین فیلم های آموزشی فارسی و زبان اصلی

دریافت جدیدترین کتابهای آموزشی

دریافت جدیدترین مقالات آموزشی

دریافت جدیدترین پروژه های دانشجویی

و ....

جهت دریافت جدیدترین مطالب سایت در گروه کد سیتی عضو شوید

جهت عضویت در گروه اینجا کلیک کنید

عنوان: **ASP.NET MVC #9**
نویسنده: وحید نصیری
تاریخ: ۱۳۹۱/۰۱/۱۵ ۱۳:۳۶:۰۰
آدرس: www.dotnettips.info
برچسب‌ها: MVC

مروری بر HTML Helpers استاندارد مهیا در ASP.NET MVC

یکی از اهداف وجودی Server controls در ASP.NET Web forms، رندر خودکار HTML است. برای مثال Menu control، TreeView control، GridView و امثال آن کار تولید تگ‌های table، tr و بسیاری موارد دیگر را در پشت صحنه برای ما انجام می‌دهند. اما در ASP.NET MVC، هدف رسیدن به یک markup ساده و تمیز است که 100 درصد بر روی اجزای آن کنترل داشته باشیم و این مورد به صورت ضمنی به این معنا است که در اینجا تمام این HTMLها را باید خودمان تولید کنیم. البته در عمل خیر. یک نمونه از آن‌را در قسمت قبل مشاهده کردیم که چطور می‌توان منطق تولید تگ‌های HTML را کپسوله سازی کرد و بارها مورد استفاده قرار داد. به علاوه فریم ورک ASP.NET MVC نیز به همراه تعدادی HTML helper توکار ارائه شده است مانند CheckBox، ActionLink، RenderPartial و غیره که کار تولید تگ‌های HTML ضروری و پایه را برای ما ساده می‌کنند.
یک مثال:

```
@Html.ActionLink("About us", "Index", "About")
```

در اینجا از متدی به نام ActionLink استفاده شده است. شیء Html هم وهله‌ای از کلاس HtmlHelper است که در تمام Viewها قابل دسترسی می‌باشد.
در این متد، اولین پارامتر، متن نمایش داده شده به کاربر را مشخص می‌کند، پارامتر سوم، نام کنترلری است که مورد استفاده قرار می‌گیرد و پارامتر دوم، نام متد یا اکشنی در آن است که فراخوانی خواهد شد (البته هر کدام از این HtmlHelperها به همراه تعداد قابل توجهی overload هم هستند).
زمانیکه این صفحه را رندر کنیم، به خروجی زیر خواهیم رسید:

```
<a href="/About">About us</a>
```

در این لینک نهایی خبری از متد Index ایی که معرفی کردیم، نیست. چرا؟
متد ActionLink بر اساس تعاریف پیش فرض مسیریابی برنامه، سعی می‌کند بهترین خروجی را ارائه دهد. مطابق تعاریف پیش فرض برنامه، متد Index، اکشن پیش فرض کنترلرهای برنامه است. بنابراین ضرورتی به ذکر آن ندیده است.

مثالی دیگر:
همان کلاس‌های Product و Products قسمت هفتم را در نظر بگیرید (قسمت بررسی «ساختار پروژه مثال جاری» در آن مثال).
همچنین به اطلاعات «نوشتن HTML Helpers ویژه، به کمک امکانات Razor» قسمت هشتم هم نیاز داریم.
اینبار می‌خواهیم بجای نمایش لیست ساده‌ای از محصولات، ابتدا نام آن‌ها را به صورت لینک‌هایی در صفحه نمایش دهیم. در ادامه پس از کلیک کاربر روی یک نام، توضیحات بیشتری از محصول انتخابی را در صفحه‌ای دیگر ارائه نمائیم. کدهای View ما اینبار به شکل زیر تغییر می‌کنند:

```
@using MvcApplication5.Models
@model MvcApplication5.Models.Products
@{
    ViewBag.Title = "Index";
}
@helper GetProductsList(List<Product> products)
    {
```

```
    <ul>
        @foreach (var item in products)
        {
            <li>@Html.ActionLink(item.Name, "Details", new { id = item.ProductNumber })</li>
        }
    </ul>
}
<h2>Index</h2>

@GetProductsList(@Model)
```

توضیحات:

ابتدا یک helper method را تعریف کرده‌ایم و به کمک Html.ActionLink، از نام و شماره محصول، جهت تولید لینک‌های نمایش جزئیات هر یک از محصولات کمک گرفته‌ایم. بنابراین در کنترلر خود نیاز به متد جدیدی به نام Details خواهیم داشت که پارامتری از نوع ProductNumber را دریافت می‌کند. سپس جزئیات این محصول را یافته و در View متناظر با خودش ارائه خواهد داد. پارامتر سومی که در متد ActionLink بکارگرفته شده در اینجا مشاهده می‌کنید، یک anonymously typed object است و توسط آن خواصی را تعریف خواهیم کرد که توسط تعاریف مسیریابی تعریف شده در فایل Global.asax.cs، قابل تفسیر و تبدیل به لینک‌های مرتبط و صحیحی باشد.

اکنون اگر این مثال را اجرا کنیم، اولین لینک تولیدی آن به این شکل خواهد بود:

```
http://localhost/Home/Details/D123
```

در اینجا به یک نکته مهم هم باید دقت داشت؛ نام کنترلر به صورت خودکار به این لینک اضافه شده است. بنابراین بهتر است از ایجاد دستی این نوع لینک‌ها خودداری کرده و کار را به متدهای استاندارد فریم ورک واگذار نمود تا بهترین خروجی را دریافت کنیم.

البته اگر الان بر روی این لینک کلیک نمائیم، با پیغام 404 مواجه خواهیم شد. برای تکمیل این مثال، متد Details را به کنترلر تعریف شده اضافه خواهیم کرد:

```
using System.Linq;
using System.Web.Mvc;
using MvcApplication5.Models;

namespace MvcApplication5.Controllers
{
    public class HomeController : Controller
    {
        public ActionResult Index()
        {
            var products = new Products();
            return View(products);
        }

        public ActionResult Details(string id)
        {
            var product = new Products().FirstOrDefault(x => x.ProductNumber == id);
            if (product == null)
                return View("Error");
            return View(product);
        }
    }
}
```

در متد Details، ابتدا ProductNumber دریافت شده و سپس شیء محصول متناظر با آن، به View این متد، بازگشت داده می‌شود. اگر بر اساس ورودی دریافتی، محصولی یافت نشد، کاربر را به View ایی به نام Error که در پوشه Views/Shared قرار گرفته است، هدایت می‌کنیم.

برای اضافه کردن این View هم بر روی متد کلیک راست کرده و گزینه Add view را انتخاب کنید. چون یک شیء strongly typed از نوع Product را قرار است به View ارسال کنیم (مانند مثال قسمت پنجم)، می‌توان در صفحه باز شده تیک Create a strongly typed view را گذاشت و سپس Model class را از نوع Product انتخاب کرد و در قسمت Scaffold template هم Details را انتخاب نمود. به این ترتیب Code generator توکار VS.NET قسمتی از کار تولید View را برای ما انجام داده و بدیهی است اکنون سفارشی سازی این View تولیدی که قسمت عمده‌ای از آن تولید شده است، کار ساده‌ای می‌باشد:

```
@model MvcApplication5.Models.Product

@{
    ViewBag.Title = "Details";
}

<h2>Details</h2>

<fieldset>
    <legend>Product</legend>

    <div class="display-label">ProductNumber</div>
    <div class="display-field">@Model.ProductNumber</div>

    <div class="display-label">Name</div>
    <div class="display-field">@Model.Name</div>

    <div class="display-label">Price</div>
    <div class="display-field">@String.Format("{0:F}", Model.Price)</div>
</fieldset>
<p>
    @Html.ActionLink("Edit", "Edit", new { /* id=Model.PrimaryKey */ }) |
    @Html.ActionLink("Back to List", "Index")
</p>
```

در اینجا کدهای مرتبط با View نمایش جزئیات محصول را مشاهده می‌کنید که توسط VS.NET به صورت خودکار از روی مدل انتخابی تولید شده است.
اکنون یکبار دیگر برنامه را اجرا کرده و بر روی لینک نمایش جزئیات محصولات کلیک نمائید تا بتوان این اطلاعات را در صفحه‌ی بعدی مشاهده نمود.

یک نکته:
اگر سعی کنیم متد helper GetProductsList@ فوق را در پوشه App_Code، همانند قسمت قبل قرار دهیم، به متد Html.ActionLink دسترسی نخواهیم داشت. چرا؟
پیغام خطایی که ارائه می‌شود این است:

```
'System.Web.WebPages.Html.HtmlHelper' does not contain a definition for 'ActionLink'
```

به این معنا که در وهله‌ای از شیء System.Web.WebPages.Html.HtmlHelper، به دنبال متد ActionLink می‌گردد. در حالیکه ActionLink مورد نظر به کلاس System.Web.Mvc.HtmlHelper مرتبط می‌شود.
یک راه حل آن به صورت زیر است. به هر متد helper یک آرگومان WebViewPage page را اضافه می‌کنیم (به همراه دو فضای نامی که به ابتدای فایل اضافه می‌شوند)

```
@using System.Web.Mvc
@using System.Web.Mvc.Html

@using MvcApplication5.Models

@helper GetProductsList(WebViewPage page, List<Product> products)
    {
    <ul>
```

```
        @foreach (var item in products)
        {
            <li> @page.Html.ActionLink(item.Name, "Details", new { id = item.ProductNumber })</li>
        }
    </ul>
}
```

سپس برای استفاده از آن در یک View خواهیم داشت:

```
@MyHelpers.GetProductsList(this, @Model)
```

متد ActionLink و عبارات فارسی

متد ActionLink آدرس‌های وبی را که تولید می‌کند، URL encoded هستند. برای نمونه اگر رشته‌ای که قرار است به عنوان پارامتر به اکشن متد ما ارسال شود، مساوی Hello World است، آن‌را به صورت Hello%20World در صفحه درج می‌کند. البته این مورد مشکلی را در سمت متدهای کنترلرها ایجاد نمی‌کند، چون کار URL decoding خودکار است. اما ... اگر مقداری که قرار است ارسال شود مثلا «مقدار یک» باشد، آدرس تولیدی این شکل را خواهد داشت:

```
http://localhost/Home/Details/%D9%85%D9%82%D8%AF%D8%A7%D8%B1%20%D9%8A%D9%83
```

و اگر این URL encoding انجام نشود، فقط اولین قسمت قبل از فاصله به متد ارسال می‌گردد.
مرورگرهایی مثل فایرفاکس و کروم، مشکلی با نمایش این لینک به شکل اصلی فارسی آن ندارند (حین نمایش، URL decoding را اعمال می‌کنند). اما اگر مرورگر مثلا IE8 باشد، کاربر دقیقا به همین شکل آدرس‌ها را در نوار آدرس مرورگر خود مشاهده خواهد کرد که آنچنان زیبا نیستند.
حل این مشکل، یک نکته کوچک را به همراه دارد. اگر href تولیدی به شکل زیر باشد:

```
<li><a href="/Home/Details/مقدار یک">Super Fast Bike</a></li>
```

IE حین نمایش نهایی آن، آن‌را فارسی نشان خواهد داد. حتی زمانیکه کاربر بر روی آن کلیک کند، به صورت خودکار کاراکترهایی را که لازم است encode نماید، به نحو صحیحی در URL نهایی قابل مشاهده در نوار آدرس‌ها ظاهر خواهد کرد. برای مثال 20% را به صورت خودکار اضافه می‌کند و نگرانی از این لحاظ وجود نخواهد داشت که الان بین دو کلمه فاصله‌ای وجود دارد یا خیر (مرورگرهای دیگر هم دقیقا همین رفتار را در مورد لینک‌های داخل صفحه دارند).
خلاصه این توضیحات متد کمکی زیر است:

```
@helper EmitCleanUnicodeUrl(MvcHtmlString data)
{
    @Html.Raw(HttpUtility.UrlDecode(data.ToString()))
}
```

و برای نمونه نحوه استفاده از آن به شکل زیر خواهد بود:

```
@helper GetProductsList(List<Product> products)
    {
    <ul>
        @foreach (var item in products)
        {
            <li>@EmitCleanUnicodeUrl(@Html.ActionLink(item.Name, "Details", new { id =
item.ProductNumber }))</li>
        }
    </ul>
}
```

ضمن اینکه باید درنظر داشت کلا این نوع طراحی مشکل دارد! برای مثال فرض کنید که در این مثال، جزئیات، نمایش دهنده مطلب ارسالی در یک بلاگ است. یعنی یک سری عنوان و جزئیات متناظر با آن‌ها در دیتابیس وجود دارند. اگر آدرس مطالب به این شکل باشد http://site/blog/details/text، به این معنا است که این text مساوی است با primary key جدول بانک اطلاعاتی. یعنی وبلاگ نویس سایت شما فقط یکبار در طول عمر این برنامه می‌تواند بگوید «سال نو مبارک!». دفعه‌ی بعد به علت تکراری بودن، مجاز به ارسال پیام تبریک دیگری نخواهد بود! به همین جهت بهتر است طراحی را به این شکل تغییر دهید http://site/blog/details/id/text. در اینجا id همان primary key خواهد بود. Text هم عنوان مطلب. Id به جهت خوشایند بانک اطلاعاتی و Text هم برای خوشایند موتورهای جستجو در این URL قرار دارند. مطابق تعاریف مسیریابی برنامه، Text فقط حالت تزئینی داشته و پردازش نخواهد شد.

از این نوع ترفندها زیاد به کار برده می‌شوند. برای نمونه به URL مطالب انجمن‌های معروف اینترنتی دقت کنید. عموما یک عدد را به همراه text مشاهده می‌کنید. عدد در برنامه پردازش می‌شود، متن هم برای موتورهای جستجو درنظر گرفته شده است.