عنوان: **Globalization در ASP.NET MVC - قسمت سوم**
نویسنده: یوسف نژاد
تاریخ: ۱۳۹۲/۰۲/۱۸ ۰:۵۰
آدرس: www.dotnettips.info
گروه‌ها: Globalization, Resource, .resx, Global Resource, Local Resource

قبل از ادامه، بهتر است یک مقدمه کوتاه درباره انواع منابع موجود در ASP.NET ارائه شود تا درک مطالب بعدی آسانتر شود.

نکات اولیه

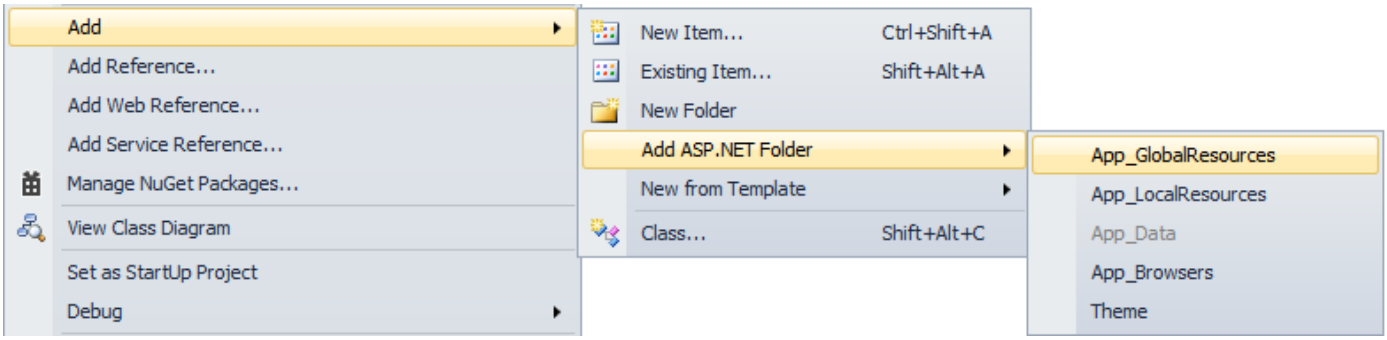
- یک فایل Resource درواقع یک فایل XML شامل رشته هایی برای ذخیره سازی مقادیر (منابع) موردنیاز است. مثلا رشته هایی برای ترجمه به زبانهای دیگر، یا مسیرهایی برای یافتن تصاویر یا فایلها و ... . پسوند این فایلها resx. است (مثل MyResource.resx).
- این فایلها برای ذخیره منابع از جفت داده‌های کلید-مقدار (key-value pair) استفاده می‌کنند. هر کلید معرف یک ورودی مجزاست. نام این کلیدها حساس به حروف بزرگ و کوچک **نیست (Not Case-Sensitive)** .
- برای هر زبان (مثل fa برای فارسی) یا کالچر موردنظر (مثل fa-IR برای فارسی ایرانی) می‌توان یک فایل Resource جداگانه تولید کرد. عناون زبان یا کالچر باید جزئی از نام فایل Resource مربوطه باشد (مثل MyResource.fa.resx یا MyResource.fa-IR.resx). هر منبع باید دارای یک فایل اصلی (پیش‌فرض) Resource باشد. این فایل، فایلی است که برای حالت پیش‌فرض برنامه (بدون کالچر) تهیه شده است و در عنوان آن از نام زبان یا کالچری استفاده نشده است (مثل MyResource.resx). برای اطلاعات بیشتر به قسمت اول این سری مراجعه کنید.
- تمامی فایل‌های Resource باید دارای کلیدهای یکسان با فایل اصلی Resource باشند. البته لزومی ندارد که این فایل‌ها حاوی تمامی کلیدهای منبع پیش‌فرض باشند. درصورت عدم وجود کلیدی در یک فایل Resource عملیات پیش فرض موجود در دات نت با استفاده از فرایند مشهور به fallback مقدار کلید موردنظر را از نزدیکترین و مناسبترین فایل موجود انتخاب می‌کند (درباره این رفتار در قسمت اول توضیحاتی ارائه شده است).
- در زمان اجرا موتور پیش فرض مدیریت منابع دات نت با توجه به کالچر UI در ثرد جاری اقدام به انتخاب مقدار مناسب برای کلیدهای درخواستی (به همراه فرایند fallback) می‌کند. فرایند نسبتا پیچیده fallback در اینجا شرح داده شده است.

منابع Global و Local

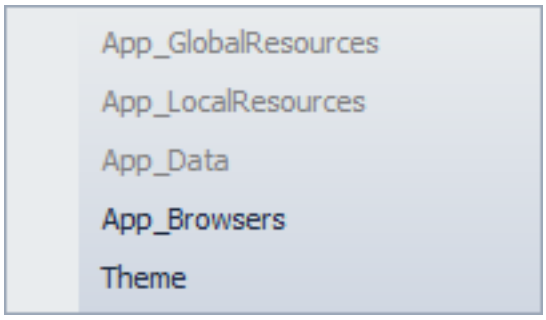
در ASP.NET دو نوع کلی Resource وجود دارد که هر کدام برای موقعیت‌های خاصی مورد استفاده قرار می‌گیرند:

- Resourceهای **Global** : منابعی کلی هستند که در تمام برنامه در دسترسند. این فایل‌ها در مسیر رزرو شده **APP_GlobalResources** در ریشه سایت قرار می‌گیرند. محتوای هر فایل resx. موجود در این فولدر دارای دسترسی کلی خواهد بود.

- Resourceهای **Local** : این منابع همان‌طور که از نامشان پیداست محلی! هستند و درواقع مخصوص همان مسیری هستند که در آن تعبیه شده اند! در استفاده از منابع محلی به ازای هر صفحه وب (aspx. یا master.) یا هر یوزرکنترل (ascx.) یک فایل resx. تولید می‌شود که تنها در همان صفحه یا یوزرکنترل در دسترس است. این فایل‌ها درون فولدر رزرو شده **APP_LocalResources** در مسیرهای موردنظر قرار می‌گیرند. درواقع در هر مسیری که نیاز به این نوع از منابع باشد، باید فولدری با عنوان App_LocalResources ایجاد شود و فایلهای resx. مرتبط با صفحه‌ها یا یوزرکنترل‌های آن مسیر در این فولدر مخصوص قرار گیرد.
در تصویر زیر چگونگی افزودن این فولدرهای مخصوص به پروژه وب اپلیکیشن نشان داده شده است:



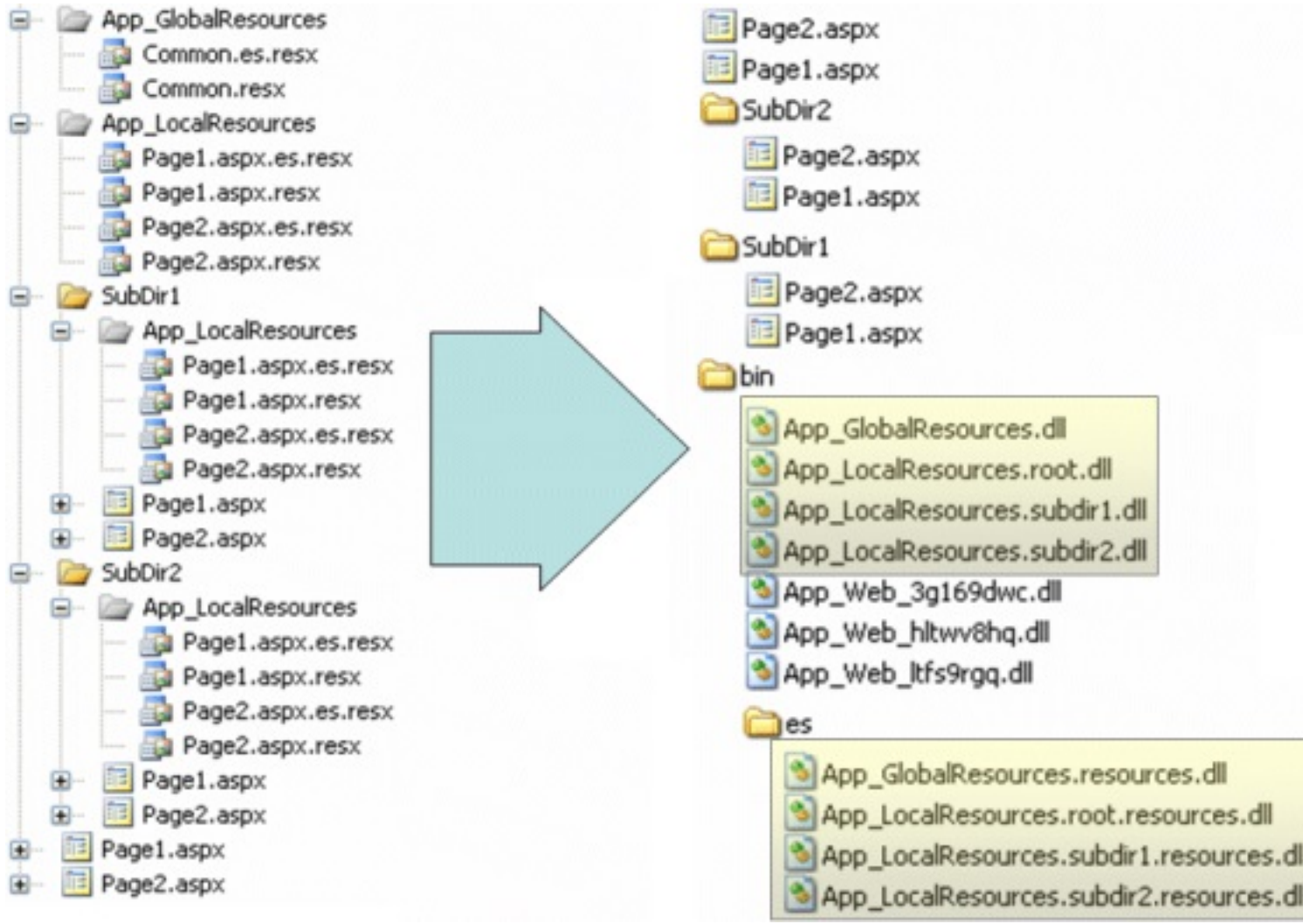
نکته: دقت کنید که تنها یک فولدر App_GlobalResources به هر پروژه می‌توان افزود. همچنین در ریشه هر مسیر موجود در پروژه تنها می‌توان یک فولدر Appp_LocalResources داشت. پس از افزودن هر یک از این فولدرهای مخصوص، منوی فوق به صورت زیر در خواهد آمد:



نکته: البته با تغییر نام یک فولدر معمولی به این نام‌های رزرو شده نتیجه یکسانی بدست خواهد آمد.

نکته: در زمان اجرا، عملیات استخراج داده‌های موجود در این نوع منابع، به صورت **خودکار** توسط ASP.NET انجام می‌شود. این داده‌ها پس از استخراج در حافظه سرور کش خواهند شد.

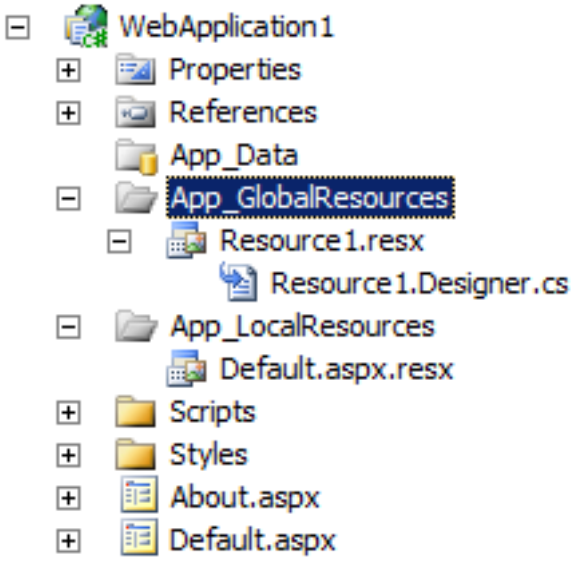
برای روشن‌تر شدن مطالب اشاره شده در بالا به تصویر فرضی! زیر توجه کنید (اسمبلی‌های تولید شده برای منابع کلی و محلی فرضی است):



در تصویر بالا محل قرارگیری انواع مختلف فایلهای Resource و نیز محل نهایی فرضی اسمبلی‌های ستلایت تولید شده، برای حداقل یک زبان غیر از زبان پیش فرض برنامه، نشان داده شده است.

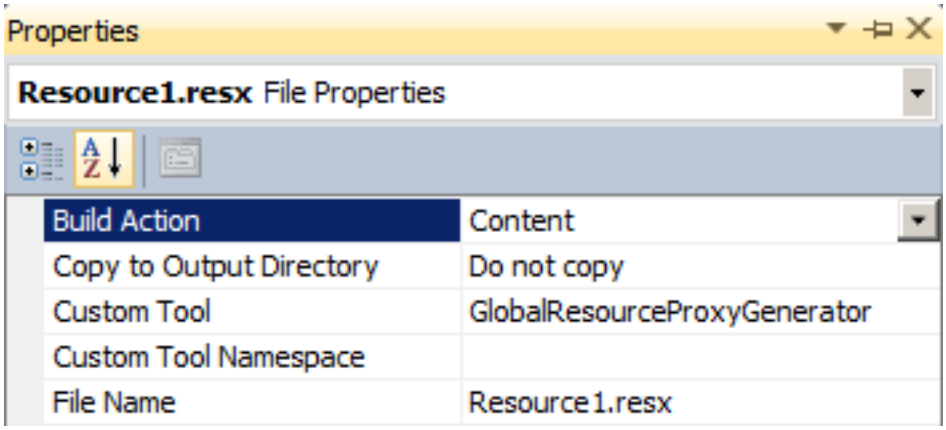
نکته: نحوه برخورد با این نوع از فایل‌های Resource در پروژه‌های Web Site و Web Application کمی باهم فرق می‌کند. موارد اشاره شده در این مطلب بیشتر درباره Web Applicationها صدق می‌کند.

برای آشنایی بیشتر بهتر است یک برنامه **وب اپلیکیشن** جدید ایجاد کرده و همانند تصویر زیر یکسری فایل Resource به فولدرهای اشاره شده در بالا اضافه کنید:



همانطور که مشاهده می‌کنید به صورت پیش‌فرض برای منابع کلی یک فایل cs. تولید می‌شود. اما اثری از این فایل برای منابع محلی نیست.

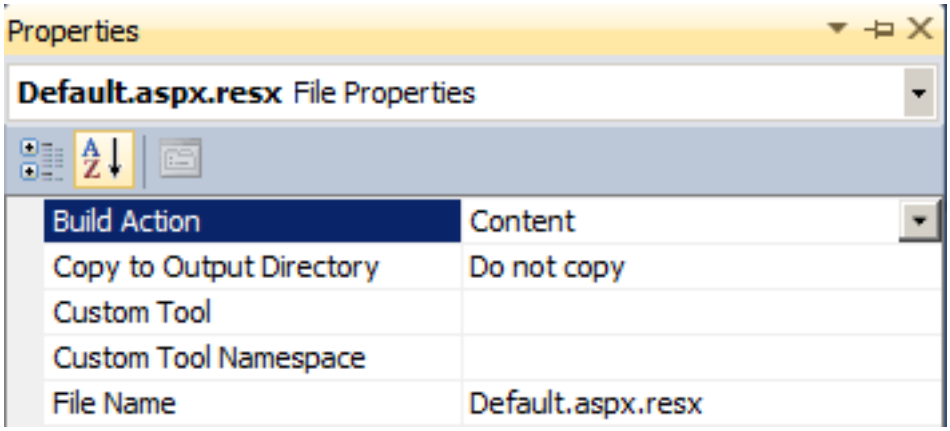
حال اگر پنجره پراپرتی فایل منبع کلی را باز نمایید با چیزی شبیه به تصویر زیر مواجه خواهید شد:



می‌بینید که خاصیت Build Action آن به Content مقداردهی شده است. این مقدار موجب می‌شود تا این فایل به همین صورت و در همین مسیر مستقیما در پابلیش نهایی برنامه ظاهر شود. در قسمت قبل به خاصیت Buil Action و مقادیر مختلف آن اشاره شده است.

هم‌چنین می‌بینید که مقدار پراپرتی Custom Tool به **GlobalResourceProxyGenerator** تنظیم شده است. این ابزار مخصوص تولید کلاس مربوط به منابع کلی در ویژوال استودیو است. با استفاده از این ابزار فایل Resource1.Designer.cs که در تصویر قبلی نیز نشان داده شده، تولید می‌شود.

حالا پنجره پراپرتی‌های منبع محلی را باز کنید:



می‌بینید که همانند منبع کلی خاصیت Build Action آن به Content تنظیم شده است. همچنین مقداری برای پراپرتی Custom Tool تنظیم نشده است. این مقدار پیش فرض را تغییر ندهید، چون با تنظیم مقداری برای آن چیز مفیدی عایدتان نمی‌شود!

نکته: برای به روز رسانی مقادیر کلیدهای منابعی که با توجه به توضیحات بالا به همراه برنامه به صورت فایلهای resx. پابلیش می‌شوند، کافی است تا محتوای فایلهای resx. مربوطه با استفاده از یک ابزار (همانند نمونه ای که در قسمت قبل شرح داده شد) تغییر داده شوند. بقیه عملیات توسط ASP.NET انجام خواهد شد. اما با تغییر محتوای این فایلهای resx. با توجه به رفتار FCN در ASP.NET (که در قسمت قبل نیز توضیح داده شد) سایت Restart خواهد شد. البته این روش تنها برای منابع کلی و محلی درون مسیرهای مخصوص اشاره شده کار خواهد کرد.

استفاده از منابع Local و Global

پس از تولید فایل‌های Resource، می‌توان از آن‌ها در صفحات وب استفاده کرد. معمولا از این نوع منابع برای مقداردهی پراپرتی کنترل‌ها در صفحات وب استفاده می‌شود. برای استفاده از کلیدهای منابع محلی می‌توان از روشی همانند زیر بهره برد:

```
<asp:Label ID="lblLocal" runat="server" meta:resourcekey="lblLocalResources" ></asp:Label>
```

اما برای منابع کلی تنها می‌توان از روش زیر استفاده کرد (یعنی برای منابع محلی نیز می‌توان از این روش استفاده کرد):

```
<asp:Label ID="lblGlobal" runat="server" Text="<%$ Resources:CommonTerms, HelloText %>" ></asp:Label>
```

به این عبارات که با فوت پررنگ مشخص شده اند اصطلاحا «عبارات بومی‌سازی» (Localization Expression) می‌گویند. در ادامه این سری مطالب با نحوه تعریف نمونه‌های سفارشی آن آشنا خواهیم شد.
به نمونه اول که برای منابع محلی استفاده می‌شود نوع ضمنی (Implicit Localization Expression) می‌گویند. زیرا نیازی نیست تا محل کلید موردنظر صراحتا ذکر شود!
به نمونه دوم که برای منابع کلی استفاده می‌شود نوع صریح (Explicit Localization Expression) می‌گویند. زیرا برای یافتن کلید موردنظر باید آدرس دقیق آن ذکر شود!

بومی سازی ضمنی (Implicit Localization) با منابع محلی عنوان کلید مربوطه در این نوع عبارات همانطور که در بالا نشان داده شده است، با استفاده از پراپرتی مخصوص **meta:resoursekey** مشخص می‌شود. در استفاده از منابع محلی تنها یک نام برای کل خواص کنترل مربوطه در صفحات وب کفایت می‌کند. زیرا عنوان کلیدهای این منبع باید از طرح زیر پیروی کند:

ResourceKey.Property

ResourceKey.Property-SubProperty یا ResourceKey.Property.SubProperty

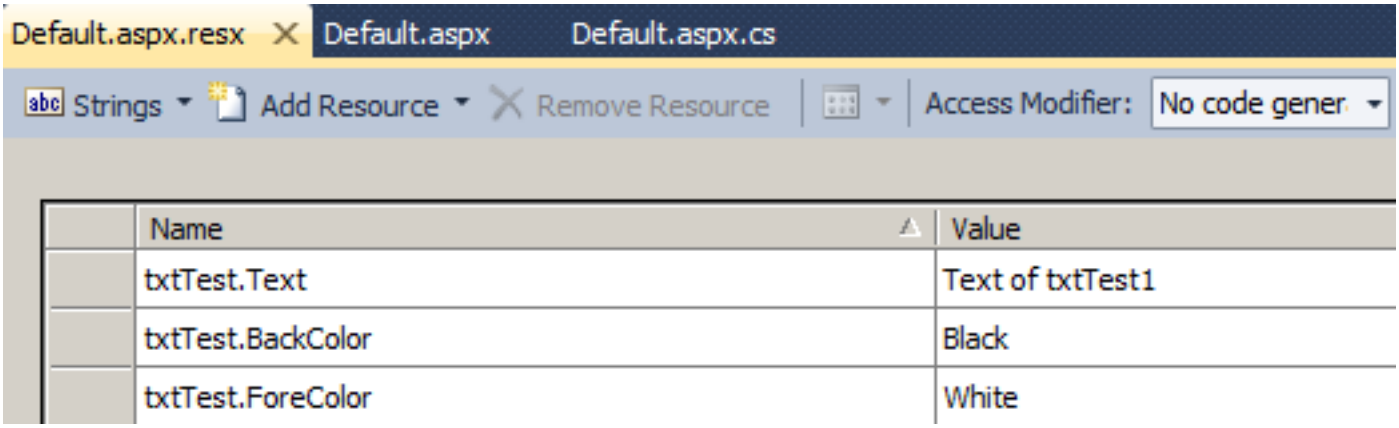
برای مثال در لیبل بالا که نام کلید Resource آن به lblLocalResources تنظیم شده است، اگر نام صفحه وب مربوطه page1.aspx باشد، برای تنظیم خواص آن در فایل page1.aspx.resx مربوطه باید از کلیدهایی با عناوینی مثل عنوان‌های زیر استفاده کرد:

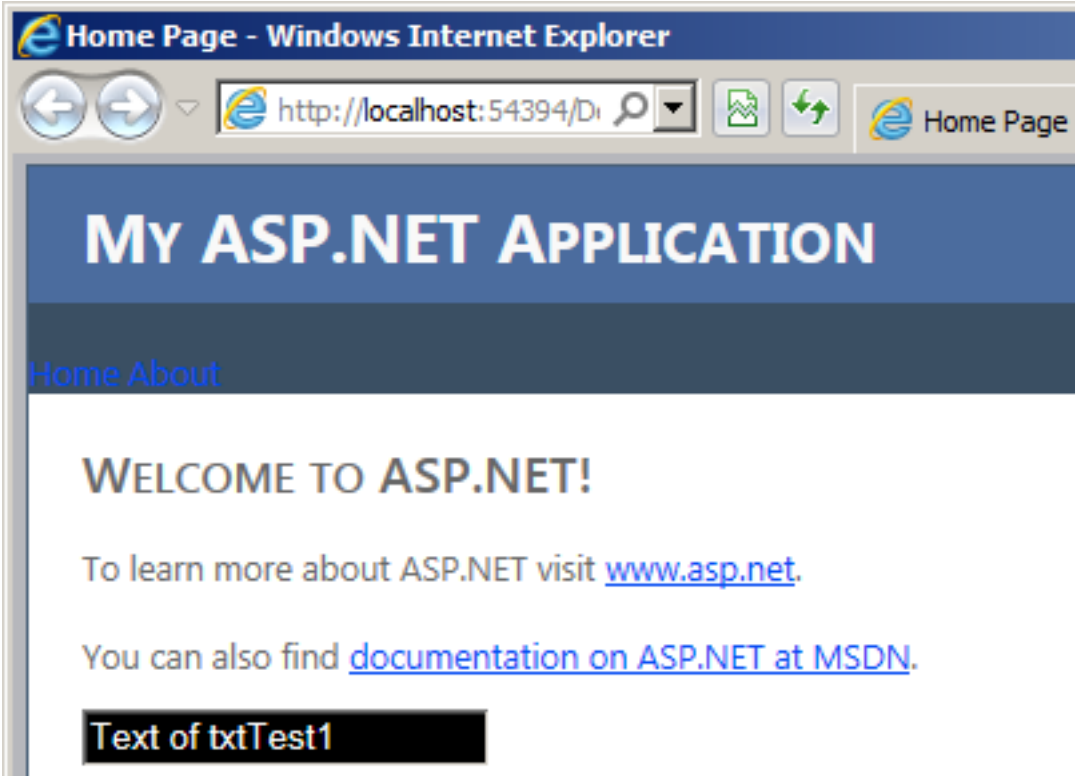
lblLocalResources.Text

lblLocalResources.BackColor

برای نمونه به تصاویر زیر دقت کنید:

```
<asp:TextBox ID="txtTest" runat="server" meta:resourcekey="txtTest" />
```





بومی سازی صریح (Explicit Localization)

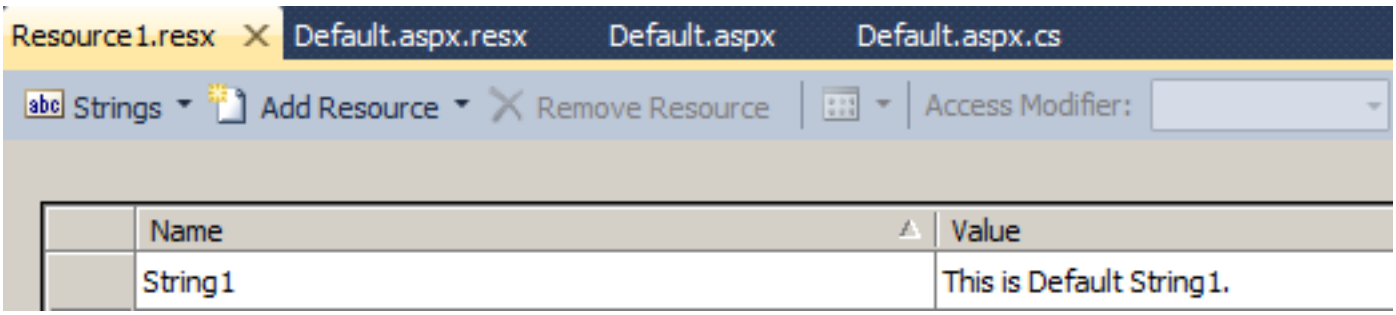
در استفاده از این نوع عبارات، پراپرتی مربوطه و نام فایل منبع صراحتا در تگ کنترل مربوطه آورده می‌شود. بنابراین برای هر خاصیتی که می‌خواهیم مقدار آن از منبعی خاص گرفته شود باید از عبارتی با طرح زیر استفاده کنیم:

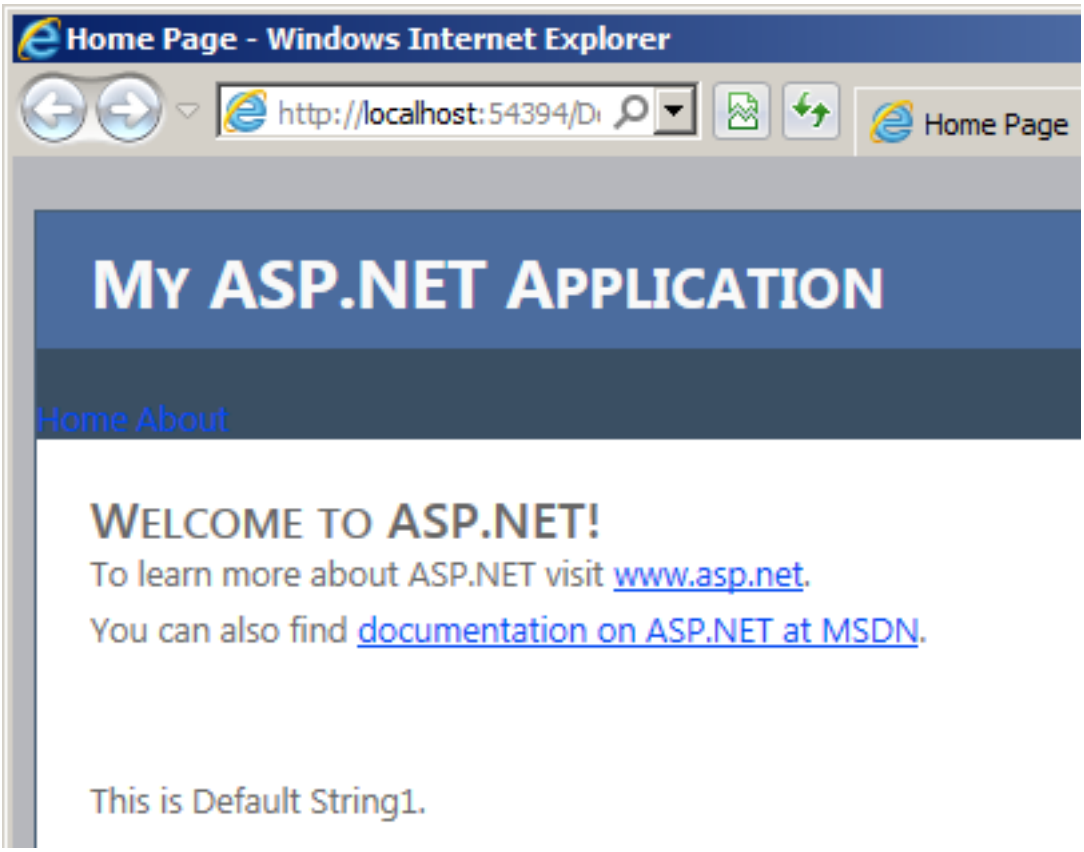
<% **Resources:** Class, ResourceKey $%>

در این عبارت، رشته Resources **پیشوند (Prefix)** نام دارد و مشخص کننده استفاده از نوع صریح عبارات بومی سازی است. Class نام کلاس مربوط به فایل منبع بوده و اختیاری است که تنها برای منابع کلی باید آورده شود. ResourceKey نیز کلید مربوطه را در فایل منبع مشخص می‌کند.

برای نمونه به تصاویر زیر دقت کنید:

```
<asp:Label ID="Label1" runat="server" Text="<%$ Resources: Resource1, String1 %
```





نکته: استفاده همزمان از این دو نوع عبارت بومی سازی در یک کنترل مجاز نیست!

نکته: به دلیل تولید کلاسی مخصوص منابع کلی (با توجه به توضیحات ابتدای این مطلب راجع به پراپرتی Custom Tool)، امکان استفاده مستقیم از آن درون کد نیز وجود دارد. این کلاسها که به صورت خودکار تولید می‌شوند، به صورت مستقیم از کلاس ResourceManager برای یافتن کلیدهای منابع استفاده می‌کنند. اما روش مستقیمی برای استفاده از کلیدهای منابع محلی درون کد وجود ندارد.

نکته: درون کلاس System.Web.UI.TemplateControl و نیز کلاس HttpContext دو متد با نامهای GetGlobalResourceObject و GetLocalResourceObject وجود دارد که برای یافتن کلیدهای منابع به صورت غیرمستقیم استفاده می‌شوند. مقدار برگشتی این دو متد از نوع object است. این دو متد به صورت مستقیم از کلاس ResourceManager استفاده **نمیکنند** ! هم‌چنین ازآنجاکه کلاس Page از کلاس TemplateControl مشتق شده است، بنابراین این دو متد در صفحات وب در دسترس هستند.

دسترسی با برنامه نویسی

همانطور که در بالا اشاره شد امکان دستیابی به کلیدهای منابع محلی و کلی ازطریق دو متد GetGlobalResourceObject و GetLocalResourceObject نیز امکان پذیر است. این دو متد با فراخوانی ResourceProviderFactory جاری سعی در یافتن مقادیر کلیدهای درخواستی در منابع موجود می‌کنند. درباره این فرایند در مطالب بعدی به صورت مفصل بحث خواهد شد.

کلاس TemplateControl

این دو متد در کلاس TemplateControl از نوع Instance (غیر استاتیک) هستند. امضای (Signature) این دو متد در این کلاس به صورت زیر است:

متد GetLocalResourceObject:

```
protected object GetLocalResourceObject(string resourceKey)
protected object GetLocalResourceObject(string resourceKey, Type objType, string propName)
```

در متد اول، پارامتر resourceKey در متد GetLocalResourceObject معرف کلید منبع مربوطه در فایل منبع محلی متناظر با صفحه جاری است. مثلا lblLocalResources.Text. ازآنجاکه به صورت پیش‌فرض موقعیت فایل منبع محلی مرتبط با صفحات وب مشخص است بنابراین تنها ارائه کلید مربوطه برای یافتن مقدار آن کافی است. مثال:

```
txtTest.Text = GetLocalResourceObject("txtTest.Text") as string;
```

متد دوم برای استخراج کلیدهای منبع محلی با مشخص کردن نوع داده محتوا (معمولا برای داده‌های غیر رشته‌ای) و پراپرتی موردنظر به کار می‌رود. در این متد پارامتر objType برای معرفی نوع داده متناظر با داده موجود در کلید resourceKey استفاده می‌شود. از پارامتر propName نیز همانطور که از نامش پیداست برای مشخص کردن پراپرتی موردنظر از این نوع داده معرفی شده استفاده می‌شود.

متد GetGlobalResourceObject:

```
protected object GetGlobalResourceObject(string className, string resourceKey)
protected object GetGlobalResourceObject(string className, string resourceKey, Type objType, string
propName)
```

در این دو متد، پارامتر className مشخص کننده نام کلاس متناظر با فایل منبع اصلی (فایل منبع اصلی که کلاس مربوطه با نام آن ساخته می‌شود) است. سایر پارامترها همانند دو متد قبلی است. مثال:

```
TextBox1.Text = GetGlobalResourceObject("Resource1", "String1") as string;
```

کلاس HttpContext

در این کلاس دو متد موردبحث از نوع استاتیک و به صورت زیر تعریف شده‌اند:

متد GetLocalResourceObject:

```
public static object GetLocalResourceObject(string virtualPath, string resourceKey)
public static object GetLocalResourceObject(string virtualPath, string resourceKey, CultureInfo
culture)
```

در این دو متد، پارامتر virtualPath مشخص کننده مسیر نسبی صفحه وب متناظر با فایل منبع محلی موردنظر است، مثل "~/Default.aspx". پارامتر resourceKey نیز کلید منبع را تعیین می‌کند و پارامتر culture نیز به کالچر موردنظر اشاره دارد. مثال:

```
txtTest.Text = HttpContext.GetLocalResourceObject("~/Default.aspx", "txtTest.Text") as string;
```

متد GetGlobalResourceObject:

```
public static object GetGlobalResourceObject(string classKey, string resourceKey)
public static object GetGlobalResourceObject(string classKey, string resourceKey, CultureInfo culture)
```

در این دو متد، پارامتر className مشخص کننده نام کلاس متناظر با فایل منبع اصلی (فایل منبع بدون نام زبان که کلاس مربوطه با نام آن ساخته می‌شود) است. سایر پارامترها همانند دو متد قبلی است. مثال:

```
TextBox1.Text = HttpContext.GetGlobalResourceObject("Resource1", "String1") as string;
```

نکته: بدیهی است که در MVC تنها می‌توان از متدهای کلاس HttpContext استفاده کرد.

روش دیگری که تنها برای منابع کلی در دسترس است، استفاده مستقیم از کلاسی است که به صورت خودکار توسط ابزارهای Visual Studio برای فایل منبع اصلی تولید می‌شود. نمونه‌ای از این کلاس را که برای یک فایل Resource1.resx (که تنها یک ورودی با نام String1 دارد) در پوشه App_GlobalResources تولید شده است، در زیر مشاهده می‌کنید:

```
//------------------------------------------------------------------------------
// <auto-generated>
//     This code was generated by a tool.
//     Runtime Version:4.0.30319.17626
//
//     Changes to this file may cause incorrect behavior and will be lost if
//     the code is regenerated.
// </auto-generated>
//------------------------------------------------------------------------------

namespace Resources {
    using System;

    /// <summary>
    ///   A strongly-typed resource class, for looking up localized strings, etc.
    /// </summary>
    // This class was auto-generated by the StronglyTypedResourceBuilder
    // class via a tool like ResGen or Visual Studio.
    // To add or remove a member, edit your .ResX file then rerun ResGen
    // with the /str option or rebuild the Visual Studio project.

[global::System.CodeDom.Compiler.GeneratedCodeAttribute("Microsoft.VisualStudio.Web.Application.Strongl
yTypedResourceProxyBuilder", "10.0.0.0")]
    [global::System.Diagnostics.DebuggerNonUserCodeAttribute()]
    [global::System.Runtime.CompilerServices.CompilerGeneratedAttribute()]
    internal class Resource1 {

        private static global::System.Resources.ResourceManager resourceMan;

        private static global::System.Globalization.CultureInfo resourceCulture;

        [global::System.Diagnostics.CodeAnalysis.SuppressMessageAttribute("Microsoft.Performance",
"CA1811:AvoidUncalledPrivateCode")]
        internal Resource1() {
        }

        /// <summary>
        ///   Returns the cached ResourceManager instance used by this class.
        /// </summary>

[global::System.ComponentModel.EditorBrowsableAttribute(global::System.ComponentModel.EditorBrowsableSt
ate.Advanced)]
        internal static global::System.Resources.ResourceManager ResourceManager {
            get {
                if (object.ReferenceEquals(resourceMan, null)) {
                    global::System.Resources.ResourceManager temp = new
global::System.Resources.ResourceManager("Resources.Resource1",
global::System.Reflection.Assembly.Load("App_GlobalResources"));
                    resourceMan = temp;
                }
                return resourceMan;
            }
        }

        /// <summary>
        ///   Overrides the current thread's CurrentUICulture property for all
        ///   resource lookups using this strongly typed resource class.
        /// </summary>

[global::System.ComponentModel.EditorBrowsableAttribute(global::System.ComponentModel.EditorBrowsableSt
ate.Advanced)]
        internal static global::System.Globalization.CultureInfo Culture {
            get {
                return resourceCulture;
            }
            set {
                resourceCulture = value;
            }
        }

        /// <summary>
```

```
        ///   Looks up a localized string similar to String1.
        /// </summary>
        internal static string String1 {
            get {
                return ResourceManager.GetString("String1", resourceCulture);
            }
        }
    }
}
```

نکته: فضای نام پیش‌فرض برای منابع کلی در این کلاس‌ها همیشه Resources است که برابر پیشوند (Prefix) عبارت بومی سازی صریح است.

نکته: در کلاس بالا نحوه نمونه سازی کلاس ResourceManager نشان داده شده است. همانطور که مشاهده می‌کنید تعیین کردن مشخصات فایل اصلی Resource مربوطه که در اسمبلی نهایی تولید و کش می‌شود، اجباری است! در مطلب بعدی با این کلاس بیشتر آشنا خواهیم شد.

نکته: همانطور که قبلا نیز اشاره شد، کار تولید اسمبلی مربوط به فایل‌های منابع کلی و محلی و کش کردن آن‌ها در اسمبلی در زمان اجرا کاملا بر عهده ASP.NET است. مثلا در نمونه کد بالا می‌بینید که کلاس ResourceManager برای استخراج نوع Resources.Resource1 از اسمبلی App_GlobalResources نمونه‌سازی شده است، با اینکه این اسمبلی و نوع مذبور در زمان کامپایل و پابلیش وجود ندارد!

برای استفاده از این کلاس می‌توان به صورت زیر عمل کرد:

```
TextBox1.Text = Resources.Resource1.String1;
```

نکته: همانطور که قبلا هم اشاره شد، متاسفانه روش بالا (برخلاف دو متدی که در قسمت قبل توضیح داده شد) به صورت مستقیم از کلاس ResourceManager استفاده می‌کند، که برای بحث سفارشی سازی پرووایدرهای منابع مشکل‌زاست. در مطالب بعدی با معایب آن و نیز راه حل‌های موجود آشنا خواهیم شد.

نکات نهایی

حال که با مفاهیم کلی بیشتری آشنا شدیم بهتر است کمی هم به نکات ریزتر بپردازیم:

نکته: فایل تولیدی توسط ویژوال استودیو در فرایند مدیریت منابع ASP.NET تاثیرگذار **نیست** ! باز هم تاکید می‌کنم که کار استخراج کلیدهای Resource از درون فایلهای resx. کاملا به صورت جداگانه و خودکار و در زمان اجرا انجام می‌شود (درباره این فرایند در مطالب بعدی شرح مفصلی خواهد آمد). درواقع شما می‌توانید خاصیت Custom Tool مربوط به منابع کلی را نیز همانند منابع محلی به رشته‌ای خالی مقداردهی کنید و ببینید که خللی در فرایند مربوطه رخ نخواهد داد!

نکته: تنها برای حالتی که بخواهید از روش آخری که در بالا اشاره شد برای دسترسی با برنامه‌نویسی به منابع کلی بهره ببرید (روش مستقیم)، به این کلاس تولیدی توسط ویژوال استودیو نیاز خواهید داشت. دقت کنید که در این کلاس نیز کار اصلی برعهده کلاس ResourceManager است. درواقع می‌توان کلا از این فایل خودکارتولیدشده صرفنظر کرد و کار استخراج کلیدهای منابع را به صورت مستقیم به نمونه‌ای از کلاس ResourceManager سپرد. این روش نیز در قسمت‌های بعدی شرح داده خواهد شد.

نکته: اگر فایل‌های Resource درون اسمبلی‌های جداگانه‌ای باشند (مثلا در یک پروژه جداگانه، همانطور که در قسمت اول این سری مطالب پیشنهاد شده است)، موتور پیش فرض منابع در ASP.NET بدرد نخواهد خورد! بنابراین یا باید از نمونه‌های اختصاصی کلاس ResourceManager استفاده کرد (کاری که کلاس‌های خودکار تولیدشده توسط ابزارهای ویژوال استودیو انجام می‌دهند)، یا باید از پرووایدرهای سفارشی استفاده کرد که در مطالب بعدی نحوه تولید آن‌ها شرح داده خواهد شد.

همانطور که در ابتدای این مطلب اشاره شد، این مقدمه در اینجا صرفا برای آشنایی بیشتر با این دونوع Resource آورده شده تا ادامه مطلب روشن‌تر باشد، زیرا با توجه به مطالب ارائه شده در قسمت اول این سری، در پروژه‌های MVC استفاده از یک پروژه جداگانه برای نگهداری این منابع راه حل مناسبتری است.

در مطلب بعدی به شرح نحوه تولید پرووایدرهای سفارشی می‌پردازم.

منابع:

http://msdn.microsoft.com/en-us/library/aa905797.aspx
http://msdn.microsoft.com/en-us/library/ms227427.aspx http://www.west-wind.com/presentations/wwdbresourceprovider
http://msdn.microsoft.com/en-us/library/1ztca10y(v=vs.100).aspx
http://msdn.microsoft.com/en-us/library/ms227982(v=vs.100).aspx
http://msdn.microsoft.com/en-us/library/sb6a8618(v=vs.100).aspx

نظرات خوانندگان

نویسنده: میهمان
تاریخ: ۱۳۹۲/۰۲/۱۸ ۱:۳

ممنون از مطلب بسیار مفیدتان