عنوان: **Globalization در ASP.NET MVC**
نویسنده: یوسف نژاد
تاریخ: ۱۳۹۱/۱۰/۲۱ ۱۶:۲۵
آدرس: www.dotnettips.info
گروه‌ها: ASP.Net MVC, .NET, Localization, Globalization, Internationalization, Resource, .resx, MVC

اگر بازار هدف یک محصول شامل چندین کشور، منطقه یا زبان مختلف باشد، طراحی و پیاده سازی آن برای پشتیبانی از ویژگی‌های چندزبانه یک فاکتور مهم به حساب می‌آید. یکی از بهترین روشهای پیاده سازی این ویژگی در دات نت استفاده از فایلهای Resource است. درواقع هدف اصلی استفاده از فایلهای Resource نیز Globalization است. Globalization برابر است با Internationalization + Localization که به اختصار به آن g11n میگویند. در تعریف، Internationalization (یا به اختصار i18n) به فرایند طراحی یک محصول برای پشتیبانی از فرهنگ(culture)ها و زبانهای مختلف و Localization (یا L10n) یا بومی‌سازی به شخصی‌سازی یک برنامه برای یک فرهنگ یا زبان خاص گفته میشود. (اطلاعات بیشتر در اینجا).
استفاده از این فایلها محدود به پیاده سازی ویژگی چندزبانه نیست. شما میتوانید از این فایلها برای نگهداری تمام رشته‌های موردنیاز خود استفاده کنید. نکته دیگری که باید بدان اشاره کرد این است که تقریبا تمامی منابع مورد استفاده در یک محصول را میتوان درون این فایلها ذخیره کرد. این منابع در حالت کلی شامل موارد زیر است:
- انواع رشته‌های مورد استفاده در برنامه چون لیبل‌ها و پیغام‌ها و یا مسیرها (مثلا نشانی تصاویر یا نام کنترلرها و اکشنها) و یا حتی برخی تنظیمات ویژه برنامه (که نمیخواهیم براحتی قابل نمایش یا تغییر باشد و یا اینکه بخواهیم با تغییر زبان تغییر کنند مثل direction و امثال آن)
- تصاویر و آیکونها و یا فایلهای صوتی و انواع دیگر فایل ها
- و ...
نحوه بهره برداری از فایلهای Resource در دات نت، پیاده سازی نسبتا آسانی را در اختیار برنامه نویس قرار میدهد. برای استفاده از این فایلها نیز روشهای متنوعی وجود دارد که در مطلب جاری به چگونگی استفاده از آنها در پروژه‌های ASP.NET MVC پرداخته میشود.

Globalization در دات نت

فرمت نام یک culture دات نت (که در کلاس CultureInfo پیاده شده است) بر اساس استاندارد RFC 4646 (^ و ^) است. (در اینجا اطلاعاتی راجع به RFC یا Request for Comments آورده شده است). در این استاندارد نام یک فرهنگ (کالچر) ترکیبی از نام زبان به همراه نام کشور یا منطقه مربوطه است. نام زبان برپایه استاندارد ISO 639 که یک عبارت دوحرفی با حروف کوچک برای معرفی زبان است مثل fa برای فارسی و en برای انگلیسی و نام کشور یا منطقه نیز برپایه استاندارد ISO 3166 که به عبارت دوحرفی با حروف بزرگ برای معرفی یک کشور یا یک منطقه است مثل IR برای ایران یا US برای آمریکاست. برای نمونه میتوان به fa-IR برای زبان فارسی کشور ایران و یا en-US برای زبان انگلیسی آمریکایی اشاره کرد. البته در این روش نامگذاری یکی دو مورد استثنا هم وجود دارد (اطلاعات کامل کلیه زبانها: National Language Support (NLS) API Reference). یک فرهنگ خنثی (Neutral Culture) نیز تنها با استفاده از دو حرف نام زبان و بدون نام کشور یا منطقه معرفی میشود. مثل fa برای فارسی یا de برای آلمانی. در این بخش نیز دو استثنا وجود دارد (^).
در دات نت دو نوع culture وجود دارد: **Culture** و **UICulture** . هر دوی این مقادیر در هر Thread مقداری منحصربه فرد دارند. مقدار Culture بر روی توابع وابسته به فرهنگ (مثل فرمت رشته‌های تاریخ و اعداد و پول) تاثیر میگذارد. اما مقدار UICulture تعیین میکند که سیستم مدیریت منابع دات نت (Resource Manager) از کدام فایل Resource برای بارگذاری داده‌ها استفاده کند. درواقع در دات نت با استفاده از پراپرتی‌های موجود در کلاس استاتیک Thread برای ثرد جاری (که عبارتند از CurrentCulture و CurrentUICulture) برای فرمت کردن و یا انتخاب Resource مناسب تصمیم گیری میشود. برای تعیین کالچر جاری به صورت دستی میتوان بصورت زیر عمل کرد:

```
Thread.CurrentThread.CurrentUICulture = new CultureInfo("fa-IR");
Thread.CurrentThread.CurrentUICulture = CultureInfo.CreateSpecificCulture("fa-IR");
```

دراینجا باید اشاره کنم که کار انتخاب Resource مناسب با توجه به کالچر ثرد جاری توسط ResourceProviderFactory پیشفرض دات نت انجام میشود. در مطالب بعدی به نحوه تعریف یک پرووایدر شخصی سازی شده هم خواهم پرداخت.

پشتیبانی از زبانهای مختلف در MVC

برای استفاده از ویژگی چندزبانه در MVC دو روش کلی وجود دارد.

1. استفاده از فایلهای Resource برای تمامی رشته‌های موجود
2. استفاده از Viewهای مختلف برای هر زبان

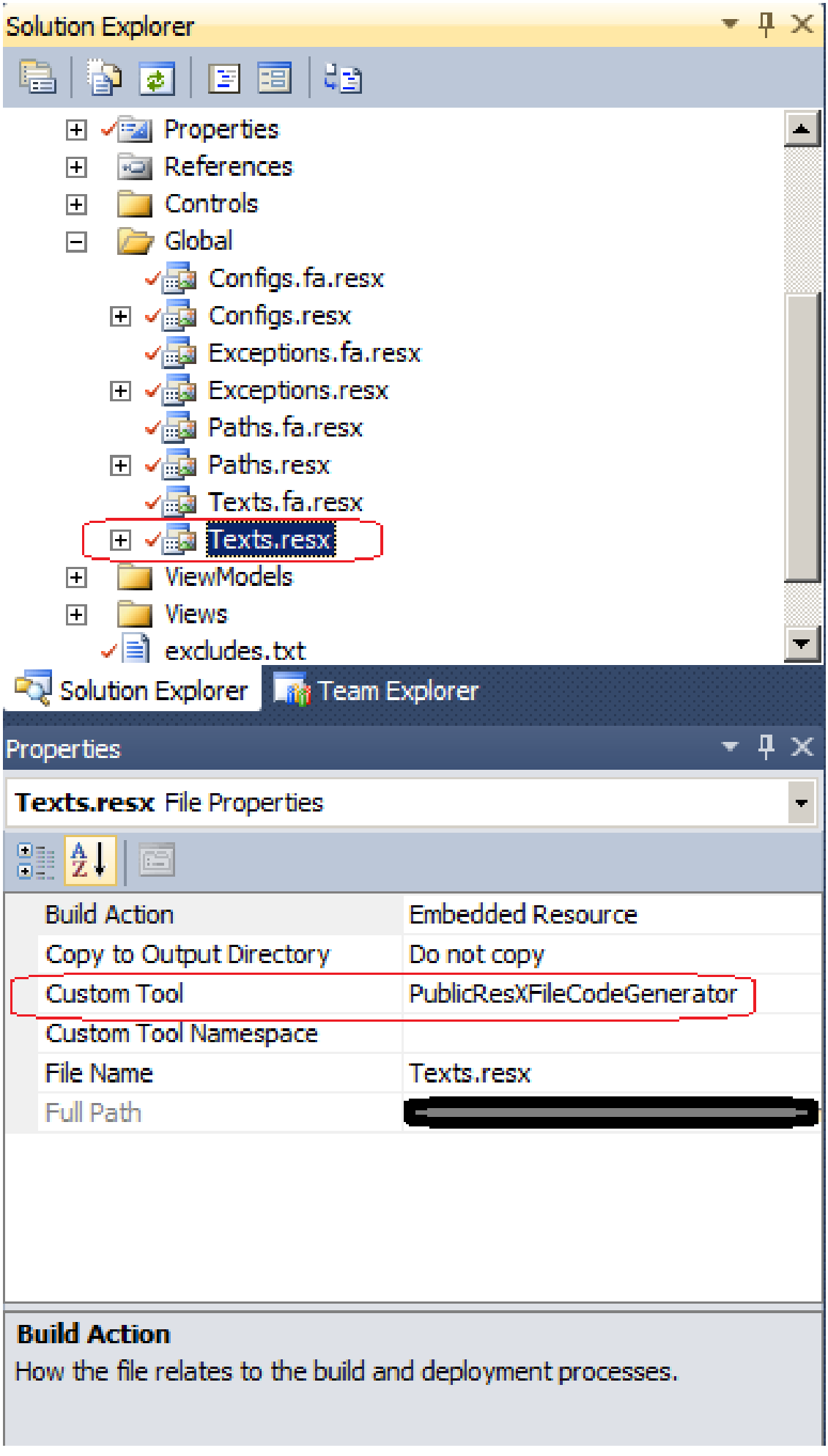
البته روش سومی هم که از ترکیب این دو روش استفاده میکند نیز وجود دارد. انتخاب روش مناسب کمی به سلیقه‌ها و عادات برنامه نویسی بستگی دارد. اگر فکر میکنید که استفاده از ویوهای مختلف به دلیل جداسازی مفاهیم درگیر در کالچرها (مثل جانمایی اجزای مختلف ویوها یا بحث Direction) باعث مدیریت بهتر و کاهش هزینه‌های پشتیبانی میشود بهتر است از روش دوم یا ترکیبی از این دو روش استفاده کنید. خودم به شخصه سعی میکنم از روش اول استفاده کنم. چون معتقدم استفاده از ویوهای مختلف باعث افزایش بیش از اندازه حجم کار میشود. اما در برخی موارد استفاده از روش دوم یا ترکیبی از دو روش میتواند بهتر باشد.

تولید فایلهای Resource

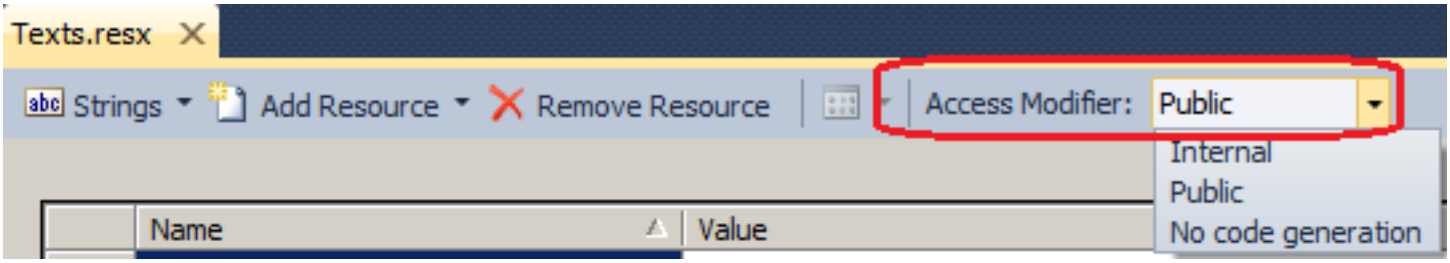
بهترین مکان برای نگهداری فایلهای Resource در یک پروژه جداگانه است. در پروژه‌های از نوع وب‌سایت پوشه‌هایی با نام App_GlobalResources یا App_LocalResources وجود دارد که میتوان از آنها برای نگهداری و مدیریت این نوع فایلها استفاده کرد. اما همانطور که در اینجا توضیح داده شده است این روش مناسب نیست. بنابراین ابتدا یک پروژه مخصوص نگهداری فایلهای Resource ایجاد کنید و سپس اقدام به تهیه این فایلها نمایید. سعی کنید که عنوان این پروژه به صورت زیر باشد. برای کسب اطلاعات بیشتر درباره نحوه نامگذاری اشیای مختلف در دات نت به این مطلب رجوع کنید.

SolutionName>.Resources<

برای افزودن فایلهای Resource به این پروژه ابتدا برای انتخاب زبان پیش فرض محصول خود تصمیم بگیرید. پیشنهاد میکنم که از زبان انگلیسی (en-US) برای اینکار استفاده کنید. ابتدا یک فایل Resource (با پسوند resx.) مثلا با نام Texts.resx به این پروژه اضافه کنید. با افزودن این فایل به پروژه، ویژوال استودیو به صورت خودکار یک فایل cs. حاوی کلاس متناظر با این فایل را به پروژه اضافه میکند. این کار توسط ابزار توکاری به نام ResXFileCodeGenerator انجام میشود. اگر به پراپرتی‌های این فایل resx. رجوع کنید میتوانید این عنوان را در پراپرتی Custom Tool ببینید. البته ابزار دیگری برای تولید این کلاسها نیز وجود دارد. این ابزارهای توکار برای سطوح دسترسی مختلف استفاده میشوند. ابزار پیش فرض در ویژوال استودیو یعنی همان ResXFileCodeGenerator، این کلاسها را با دسترسی internal تولید میکند که مناسب کار ما نیست. ابزار دیگری که برای اینکار درون ویژوال استودیو وجود دارد PublicResXFileCodeGenerator است و همانطور که از نامش پیداست از سطح دسترسی public استفاده میکند. برای تغییر این ابزار کافی است تا عنوان آن را دقیقا در پراپرتی Custom Tool تایپ کنید.


Solution Explorer
Properties
References
Controls
Global
Configs.fa.resx
Configs.resx
Exceptions.fa.resx
Exceptions.resx
Paths.fa.resx
Paths.resx
Texts.fa.resx
Texts.resx
ViewModels
Views
excludes.txt
Solution Explorer
Team Explorer
Properties
Texts.resx File Properties
Build Action
Embedded Resource
Copy to Output Directory
Do not copy
Custom Tool
PublicResXFileCodeGenerator
Custom Tool Namespace
File Name
Texts.resx
Full Path
Build Action
How the file relates to the build and deployment processes.

نکته: درباره پراپرتی مهم Build Action این فایلها در مطالب بعدی بیشتر بحث میشود.
برای تعیین سطح دسترسی Resource موردنظر به روشی دیگر، میتوانید فایل Resource را باز کرده و Access Modifier آن را به Public تغییر دهید.



سپس برای پشتیبانی از زبانی دیگر، یک فایل دیگر Resource به پروژه اضافه کنید. نام این فایل باید همنام فایل اصلی به همراه نام کالچر موردنظر باشد. مثلا برای زبان فارسی عنوان فایل باید Texts.fa-IR.resx یا به صورت ساده‌تر برای کالچر خنثی (بدون نام کشور) Texts.fa.resx باشد. دقت کنید اگر نام فایل را در همان پنجره افزودن فایل وارد کنید ویژوال استودیو این همنامی را به صورت هوشمند تشخیص داده و تغییراتی را در پراپرتی‌های پیش فرض فایل Resource ایجاد میکند.
نکته: این هوشمندی مرتبه نسبتا بالایی دارد. بدین صورت که تنها درصورتیکه عبارت بعد از نام فایل اصلی Resource (رشته بعد از نقطه مثلا fa در اینجا) متعلق به یک کالچر معتبر باشد این تغییرات اعمال خواهد شد.
مهمترین این تغییرات این است که ابزاری را برای پراپرتی Custom Tool این فایلها انتخاب نمیکند! اگر به پراپرتی فایل Texts.fa.resx مراجعه کنید این مورد کاملا مشخص است. در نتیجه دیگر فایل cs. حاوی کلاسی جداگانه برای این فایل ساخته نمیشود. همچنین اگر فایل Resource جدید را باز کنید میبینید که برای Access Modifier آن گزینه No Code Generation انتخاب شده است.

در ادامه شروع به افزودن عناوین موردنظر در این دو فایل کنید. در اولی (بدون نام زبان) رشته‌های مربوط به زبان انگلیسی و در دومی رشته‌های مربوط به زبان فارسی را وارد کنید. سپس در هرجایی که یک لیبل یا یک رشته برای نمایش وجود دارد از این کلیدهای Resource استفاده کنید مثل:

SolutionName>.Resources.Texts.Save>

SolutionName>.Resources.Texts.Cancel>

استفاده از Resource در ویومدل ها

دو خاصیت معروفی که در ویومدلها استفاده میشوند عبارتند از: DisplayName و Required. پشتیبانی از کلیدهای Resource به صورت توکار در خاصیت Required وجود دارد. برای استفاده از آنها باید به صورت زیر عمل کرد:

```
[Required(ErrorMessageResourceName = "ResourceKeyName", ErrorMessageResourceType =
typeof(<SolutionName>.Resources.<ResourceClassName>))]
```

در کد بالا باید از نام فایل Resource اصلی (فایل اول که بدون نام کالچر بوده و به عنوان منبع پیشفرض به همراه یک فایل cs. حاوی کلاس مربوطه نیز هست) برای معرفی ErrorMessageResourceType استفاده کرد. چون ابزار توکار ویژوال استودیو از نام این فایل برای تولید کلاس مربوطه استفاده میکند.
متاسفانه خاصیت DisplayName که در فضای نام System.ComponentModel (در فایل System.dll) قرار دارد قابلیت استفاده از کلیدهای Resource را به صورت توکار ندارد. در دات نت 4 خاصیت دیگری در فضای نام System.ComponentModel.DataAnnotations به نام Display (در فایل System.ComponentModel.DataAnnotations.dll) وجود دارد که این امکان را به صورت توکار دارد. اما قابلیت استفاده از این خاصیت تنها در MVC 3 وجود دارد. برای نسخه‌های قدیمیتر

MVC امکان استفاده از این خاصیت حتی اگر نسخه فریمورک هدف 4 باشد وجود ندارد، چون هسته این نسخه‌های قدیمی امکان استفاده از ویژگی‌های جدید فریمورک با نسخه بالاتر را ندارد. برای رفع این مشکل میتوان کلاس خاصیت DisplayName را برای استفاده از خاصیت Display به صورت زیر توسعه داد:

```
public class LocalizationDisplayNameAttribute : DisplayNameAttribute
  {
    private readonly DisplayAttribute _display;
    public LocalizationDisplayNameAttribute(string resourceName, Type resourceType)
    {
      _display = new DisplayAttribute { ResourceType = resourceType, Name = resourceName };
    }
    public override string DisplayName
    {
      get
      {
        try
        {
          return _display.GetName();
        }
        catch (Exception)
        {
          return _display.Name;
        }
      }
    }
  }
```

در این کلاس با ترکیب دو خاصیت نامبرده امکان استفاده از کلیدهای Resource فراهم شده است. در پیاده سازی این کلاس فرض شده است که نسخه فریمورک هدف حداقل برابر 4 است. اگر از نسخه‌های پایین‌تر استفاده میکنید در پیاده سازی این کلاس باید کاملا به صورت دستی کلید موردنظر را از Resource معرفی شده بدست آورید. مثلا به صورت زیر:

```
public class LocalizationDisplayNameAttribute : DisplayNameAttribute
{
    private readonly PropertyInfo nameProperty;
    public LocalizationDisplayNameAttribute(string displayNameKey, Type resourceType = null)
        : base(displayNameKey)
    {
        if (resourceType != null)
            nameProperty = resourceType.GetProperty(base.DisplayName, BindingFlags.Static |
BindingFlags.Public);
    }
    public override string DisplayName
    {
        get
        {
            if (nameProperty == null) base.DisplayName;
            return (string)nameProperty.GetValue(nameProperty.DeclaringType, null);
        }
    }
}
```

برای استفاده از این خاصیت جدید میتوان به صورت زیر عمل کرد:

```
[LocalizationDisplayName("ResourceKeyName", typeof(<SolutionName>.Resources.<ResourceClassName>))]
```

البته بیشتر خواص متداول در ویومدلها از ویژگی موردبحث پشتیبانی میکنند.

نکته: به کار گیری این روش ممکن است در پروژه‌های بزرگ کمی گیج کننده و دردسرساز بوده و باعث پیچیدگی بی‌مورد کد و نیز افزایش بیش از حد حجم کدنویسی شود. در مقاله آقای فیل هک (Model Metadata and Validation Localization using Conventions) روش بهتر و تمیزتری برای مدیریت پیامهای این خاصیت‌ها آورده شده است.

پشتیبانی از ویژگی چند زبانه

مرحله بعدی برای چندزبانه کردن پروژه‌های MVC تغییراتی است که برای مدیریت Culture جاری برنامه باید پیاده شوند. برای

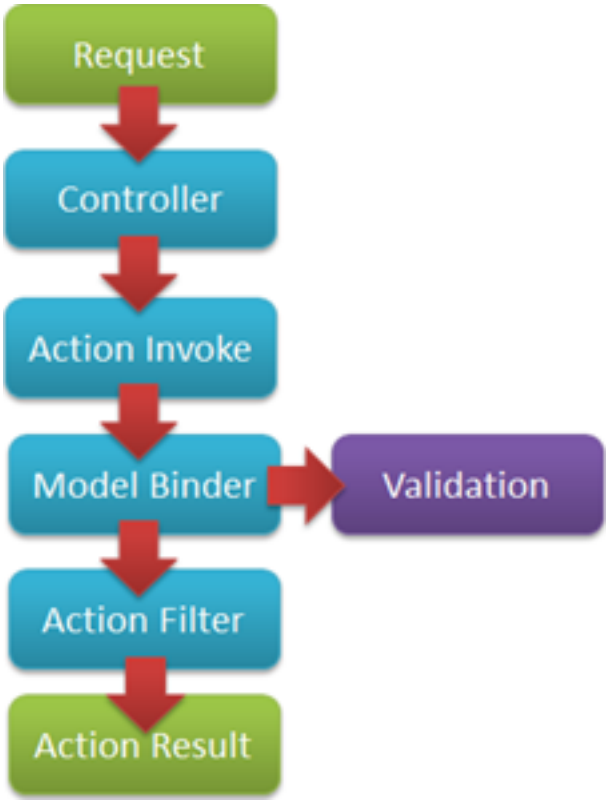
اینکار باید خاصیت CurrentUICulture در ثرد جاری کنترل و مدیریت شود. یکی از مکانهایی که برای نگهداری زبان جاری استفاده میشود کوکی است. معمولا برای اینکار از کوکی‌های دارای تاریخ انقضای طولانی استفاده میشود. میتوان از تنظیمات موجود در فایل کانفیگ برای ذخیره زبان پیش فرض سیستم نیز استفاه کرد.
روشی که معمولا برای مدیریت زبان جاری میتوان از آن استفاده کرد پیاده سازی یک کلاس پایه برای تمام کنترلرها است. کد زیر راه حل نهایی را نشان میدهد:

```
public class BaseController : Controller
  {
    private const string LanguageCookieName = "MyLanguageCookieName";
    protected override void ExecuteCore()
    {
      var cookie = HttpContext.Request.Cookies[LanguageCookieName];
      string lang;
      if (cookie != null)
      {
        lang = cookie.Value;
      }
      else
      {
        lang = ConfigurationManager.AppSettings["DefaultCulture"] ?? "fa-IR";
        var httpCookie = new HttpCookie(LanguageCookieName, lang) { Expires = DateTime.Now.AddYears(1)
};
        HttpContext.Response.SetCookie(httpCookie);
      }
      Thread.CurrentThread.CurrentUICulture = CultureInfo.CreateSpecificCulture(lang);
      base.ExecuteCore();
    }
  }
```

راه حل دیگر استفاده از یک ActionFilter است که نحوه پیاده سازی یک نمونه از آن در زیر آورده شده است:

```
public class LocalizationActionFilterAttribute : ActionFilterAttribute
  {
    private const string LanguageCookieName = "MyLanguageCookieName";
    public override void OnActionExecuting(ActionExecutingContext filterContext)
    {
      var cookie = filterContext.HttpContext.Request.Cookies[LanguageCookieName];
      string lang;
      if (cookie != null)
      {
        lang = cookie.Value;
      }
      else
      {
        lang = ConfigurationManager.AppSettings["DefaultCulture"] ?? "fa-IR";
        var httpCookie = new HttpCookie(LanguageCookieName, lang) { Expires = DateTime.Now.AddYears(1)
};
        filterContext.HttpContext.Response.SetCookie(httpCookie);
      }
      Thread.CurrentThread.CurrentUICulture = CultureInfo.CreateSpecificCulture(lang);
      base.OnActionExecuting(filterContext);
    }
  }
```

نکته مهم: تعیین زبان جاری (یعنی همان مقداردهی پراپرتی CurrentCulture ثرد جاری) در یک اکشن فیلتر بدرستی عمل نمیکند. برای بررسی بیشتر این مسئله ابتدا به تصویر زیر که ترتیب رخدادن رویدادهای مهم در ASP.NET MVC را نشان میدهد دقت کنید:



همانطور که در تصویر فوق مشاهده میکنید رویداد OnActionExecuting که در یک اکشن فیلتر به کار میرود بعد از عملیات مدل بایندینگ رخ میدهد. بنابراین قبل از تعیین کالچر جاری، عملیات validation و یافتن متن خطاها از فایلهای Resource انجام میشود که منجر به انتخاب کلیدهای مربوط به کالچر پیشفرض سرور (و نه آنچه که کاربر تنظیم کرده) خواهد شد. بنابراین استفاده از یک اکشن فیلتر برای تعیین کالچر جاری مناسب نیست. راه حل مناسب استفاده از همان کنترلر پایه است، زیرا متد ExecuteCore قبل از تمامی این عملیات صدا زده میشود. بنابراین همیشه کالچر تنظیم شده توسط کاربر به عنوان مقدار جاری آن در ثرد ثبت میشود.

امکان تعیین/تغییر زبان توسط کاربر

برای تعیین یا تغییر زبان جاری سیستم نیز روشهای گوناگونی وجود دارد. استفاده از زبان تنظیم شده در مرورگر کاربر، استفاده از عنوان زبان در آدرس صفحات درخواستی و یا تعیین زبان توسط کاربر در تنظیمات برنامه/سایت و ذخیره آن در کوکی یا دیتابیس و مواردی از این دست روشهایی است که معمولا برای تعیین زبان جاری از آن استفاده میشود. در کدهای نمونه ای که در بخشهای قبل آورده شده است فرض شده است که زبان جاری سیستم درون یک کوکی ذخیره میشود بنابراین برای استفاده از این روش میتوان از قطعه کدی مشابه زیر (مثلا در فایل _Layout.cshtml) برای تعیین و تغییر زبان استفاه کرد:

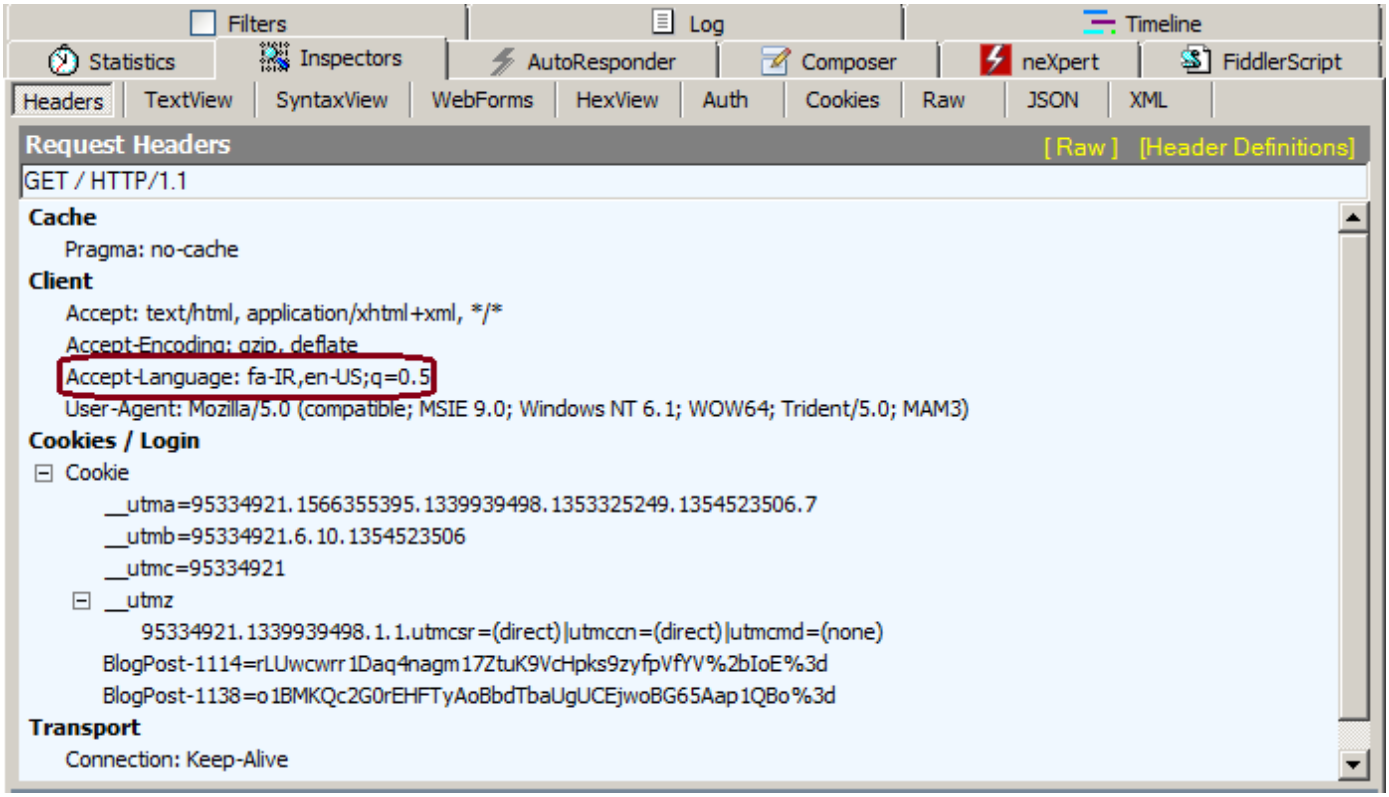
```
<select id="langs" onchange="languageChanged()">
  <option value="fa-IR">فارسی</option>
  <option value="en-US">انگلیسی</option>
</select>
<script type="text/javascript">
  function languageChanged() {
    setCookie("MyLanguageCookieName", $('#langs').val(), 365);
    window.location.reload();
  }
  document.ready = function () {
    $('#langs').val(getCookie("MyLanguageCookieName"));
  };
  function setCookie(name, value, exdays, path) {
    var exdate = new Date();
    exdate.setDate(exdate.getDate() + exdays);
    var newValue = escape(value) + ((exdays == null) ? "" : "; expires=" + exdate.toUTCString()) +
((path == null) ? "" : "; path=" + path) ;
    document.cookie = name + "=" + newValue;
  }
```

```
  function getCookie(name) {
    var i, x, y, cookies = document.cookie.split(";");
    for (i = 0; i < cookies.length; i++) {
      x = cookies[i].substr(0, cookies[i].indexOf("="));
      y = cookies[i].substr(cookies[i].indexOf("=") + 1);
      x = x.replace(/^\s+|\s+$/g, "");
      if (x == name) {
        return unescape(y);
      }
    }
  }
</script>
```

متدهای setCookie و getCookie جاوا اسکریپتی در کد بالا از اینجا گرفته شده اند البته پس از کمی تغییر.
نکته : مطلب Cookieها بحثی نسبتا مفصل است که در جای خودش باید به صورت کامل آورده شود. اما در اینجا تنها به همین نکته اشاره کنم که عدم توجه به پراپرتی path کوکی‌ها در این مورد خاص برای خود من بسیار گیج‌کننده و دردسرساز بود.
به عنوان راهی دیگر میتوان به جای روش ساده استفاده از کوکی، تنظیماتی در اختیار کاربر قرار داد تا بتواند زبان تنظیم شده را درون یک فایل یا دیتابیس ذخیره کرد البته با درنظر گرفتن مسائل مربوط به کش کردن این تنظیمات.
راه حل بعدی میتواند استفاده از تنظیمات مرورگر کاربر برای دریافت زبان جاری تنظیم شده است. مرورگرها تنظیمات مربوط به زبان را در قسمت Accept-Languages در HTTP Header درخواست ارسالی به سمت سرور قرار میدهند. بصورت زیر:

```
GET http://www.dotnettips.info HTTP/1.1
...
Accept-Language: fa-IR,en-US;q=0.5
...
```

این هم تصویر مربوط به Fiddler آن:

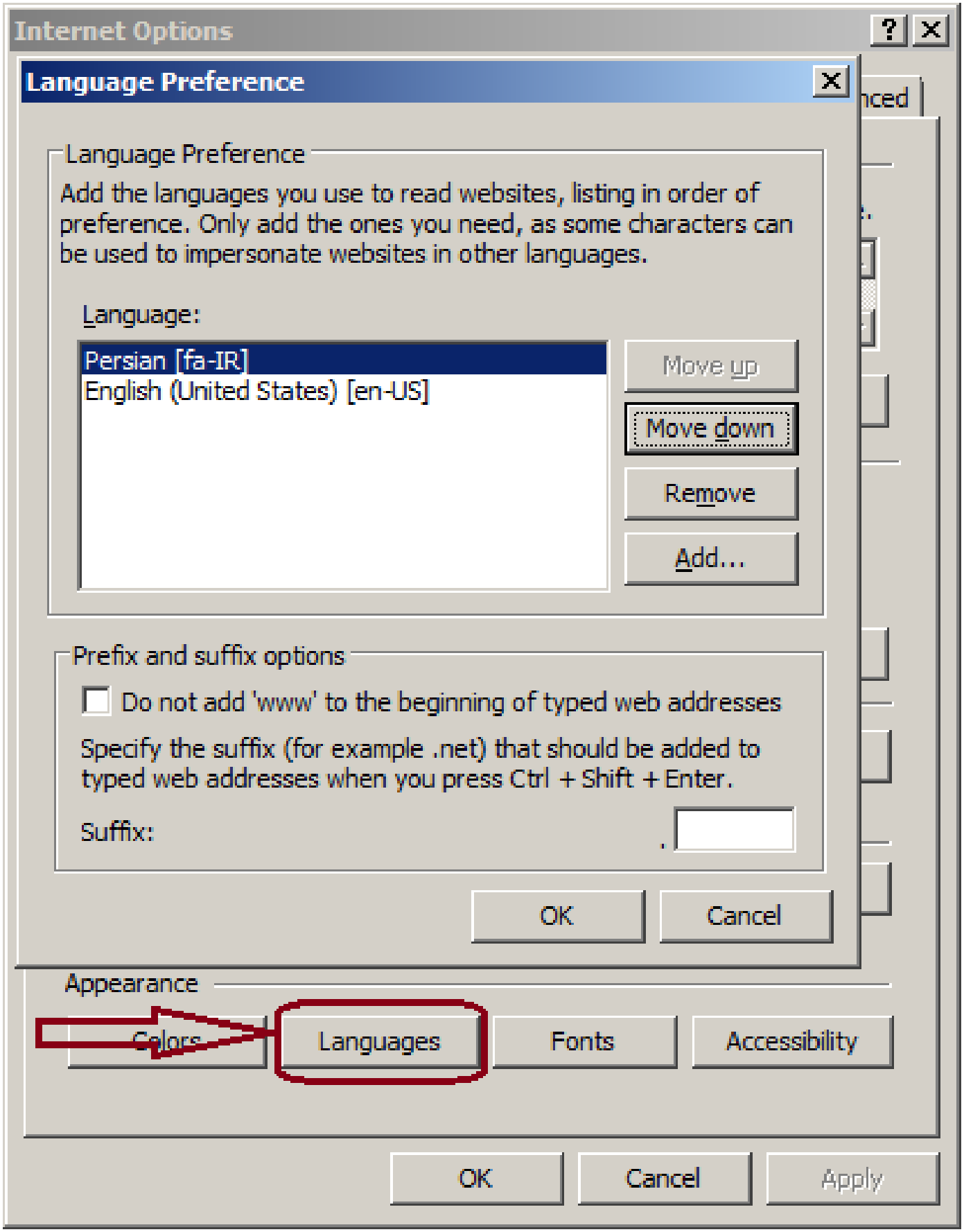


نکته: پارامتر q در عبارت مشخص شده در تصویر فوق relative quality factor نام دارد و به نوعی مشخص کننده اولویت زبان مربوطه است. مقدار آن بین 0 و 1 است و مقدار پیش فرض آن 1 است. هرچه مقدار این پارامتر بیشتر باشد زبان مربوطه اولویت

بالاتری دارد. مثلا عبارت زیر را درنظر بگیرید:

```
Accept-Language: fa-IR,fa;q=0.8,en-US;q=0.5,ar-BH;q=0.3
```

در این حالت اولویت زبان fa-IR برابر 1 و fa برابر 0.8 (fa;q=0.8) است. اولویت دیگر زبانهای تنظیم شده نیز همانطور که نشان داده شده است در مراتب بعدی قرار دارند. در تنظیم نمایش داده شده برای تغییر این تنظیمات در IE میتوان همانند تصویر زیر اقدام کرد:



در تصویر بالا زبان فارسی اولویت بالاتری نسبت به انگلیسی دارد. برای اینکه سیستم g11n دات نت به صورت خودکار از این مقادیر جهت زبان ثرد جاری استفاده کند میتوان از تنظیم زیر در فایل کانفیگ استفاده کرد:

```
<system.web>
    <globalization enableClientBasedCulture="true" uiCulture="auto" culture="auto"></globalization>
</system.web>
```

در سمت سرور نیز برای دریافت این مقادیر تنظیم شده در مرورگر کاربر میتوان از کدهای زیر استفاه کرد. مثلا در یک اکشن فیلتر:

```
var langs = filterContext.HttpContext.Request.UserLanguages;
```

پراپرتی UserLanguages از کلاس Request حاوی آرایه‌ای از استرینگ است. این آرایه درواقع از Split کردن مقدار Accept-Languages با کاراکتر ',' بدست می‌آید. بنابراین اعضای این آرایه رشته‌ای از نام زبان به همراه پارامتر q مربوطه خواهند بود (مثل "fa;q=0.8").

راه دیگر مدیریت زبانها استفاده از عنوان زبان در مسیر درخواستی صفحات است. مثلا آدرسی شبیه به www.MySite.com/fa/Employees نشان میدهد کاربر درخواست نسخه فارسی از صفحه Employees را دارد. نحوه استفاده از این عناوین و نیز موقعیت فیزیکی این عناوین در مسیر صفحات درخواستی کاملا به سلیقه برنامه نویس و یا کارفرما بستگی دارد. روش کلی بهره برداری از این روش در تمام موارد تقریبا یکسان است.

برای پیاده سازی این روش ابتدا باید یک route جدید در فایل Global.asax.cs اضافه کرد:

```
routes.MapRoute(
    "Localization", // Route name
    "{lang}/{controller}/{action}/{id}", // URL with parameters
    new { controller = "Home", action = "Index", id = UrlParameter.Optional } // Parameter defaults
);
```

دقت کنید که این route باید قبل از تمام routeهای دیگر ثبت شود. سپس باید کلاس پایه کنترلر را به صورت زیر پیاده سازی کرد:

```
public class BaseController : Controller
{
  protected override void ExecuteCore()
  {
    var lang = RouteData.Values["lang"];
    if (lang != null && !string.IsNullOrWhiteSpace(lang.ToString()))
    {
      Thread.CurrentThread.CurrentUICulture = CultureInfo.CreateSpecificCulture(lang.ToString());
    }
    base.ExecuteCore();
  }
}
```

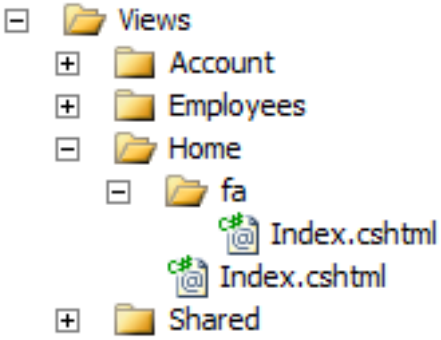
این کار را در یک اکشن فیلتر هم میتوان انجام داد اما با توجه به توضیحاتی که در قسمت قبل داده شد استفاده از اکشن فیلتر برای تعیین زبان جاری کار مناسبی نیست.

نکته: به دلیل آوردن عنوان زبان در مسیر درخواستها باید کنترل دقیقتری بر کلیه مسیرهای موجود داشت!

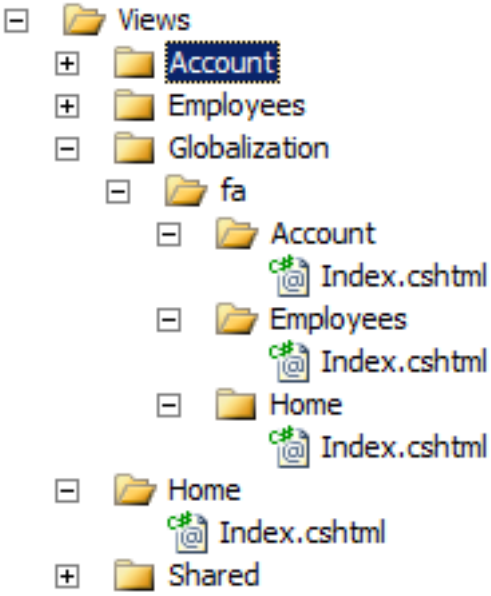
استفاده از ویوهای جداگانه برای زبانهای مختلف

برای اینکار ابتدا ساختار مناسبی را برای نگهداری از ویوهای مختلف خود درنظر بگیرید. مثلا میتوانید همانند نامگذاری فایلهای Resource از نام زبان یا کالچر به عنوان بخشی از نام فایلهای ویو استفاده کنید و تمام ویوها را در یک مسیر ذخیره کنید. همانند تصویر زیر:

البته اینکار ممکن است به مدیریت این فایلها را کمی مشکل کند چون به مرور زمان تعداد فایلهای ویو در یک فولدر زیاد خواهد شد. روش دیگری که برای نگهداری این ویوها میتوان به کار برد استفاده از فولدرهای جداگانه با عناوین زبانهای موردنظر است. مانند تصویر زیر:



روش دیگری که برای نگهداری و مدیریت بهتر ویوهای زبانهای مختلف از آن استفاده میشود به شکل زیر است:



استفاه از هرکدام از این روشها کاملا به سلیقه و راحتی مدیریت فایلها برای برنامه نویس بستگی دارد. درهر صورت پس از

انتخاب یکی از این روشها باید اپلیکشن خود را طوری تنظیم کنیم که با توجه به زبان جاری سیستم، ویوی مربوطه را جهت نمایش انتخاب کند.
مثلا برای روش اول نامگذاری ویوها میتوان از روش دستکاری متد OnActionExecuted در کلاس پایه کنترلر استفاده کرد:

```
public class BaseController : Controller
{
  protected override void OnActionExecuted(ActionExecutedContext context)
  {
    var view = context.Result as ViewResultBase;
    if (view == null) return; // not a view
    var viewName = view.ViewName;
    view.ViewName = GetGlobalizationViewName(viewName, context);
    base.OnActionExecuted(context);
  }
  private static string GetGlobalizationViewName(string viewName, ControllerContext context)
  {
    var cultureName = Thread.CurrentThread.CurrentUICulture.Name;
    if (cultureName == "en-US") return viewName; // default culture
    if (string.IsNullOrEmpty(viewName))
      return context.RouteData.Values["action"] + "." + cultureName; // "Index.fa"
    int i;
    if ((i = viewName.IndexOf('.')) > 0) // ex: Index.cshtml
      return viewName.Substring(0, i + 1) + cultureName + viewName.Substring(i); // "Index.fa.cshtml"
    return viewName + "." + cultureName; // "Index" ==> "Index.fa"
  }
}
```

همانطور که قبلا نیز شرح داده شد، چون متد ExecuteCore قبل از OnActionExecuted صدا زده میشود بنابراین از تنظیم درست مقدار کالچر در ثرد جاری اطمینان داریم.
روش دیگری که برای مدیریت انتخاب ویوهای مناسب استفاده از یک ویوانجین شخصی سازی شده است. مثلا برای روش سوم نامگذاری ویوها میتوان از کد زیر استفاده کرد:

```
public sealed class RazorGlobalizationViewEngine : RazorViewEngine
  {
    protected override IView CreatePartialView(ControllerContext controllerContext, string partialPath)
    {
      return base.CreatePartialView(controllerContext, GetGlobalizationViewPath(controllerContext,
partialPath));
    }
    protected override IView CreateView(ControllerContext controllerContext, string viewPath, string
masterPath)
    {
      return base.CreateView(controllerContext, GetGlobalizationViewPath(controllerContext, viewPath),
masterPath);
    }
    private static string GetGlobalizationViewPath(ControllerContext controllerContext, string
viewPath)
    {
      //var controllerName = controllerContext.RouteData.GetRequiredString("controller");
      var request = controllerContext.HttpContext.Request;
      var lang = request.Cookies["MyLanguageCookie"];
      if (lang != null && !string.IsNullOrEmpty(lang.Value) && lang.Value != "en-US")
      {
        var localizedViewPath = Regex.Replace(viewPath, "^~/Views/",
string.Format("~/Views/Globalization/{0}/", lang.Value));
        if (File.Exists(request.MapPath(localizedViewPath))) viewPath = localizedViewPath;
      }
      return viewPath;
    }
```

و برای ثبت این ViewEngine در فایل Global.asax.cs خواهیم داشت:

```
protected void Application_Start()
{
  ViewEngines.Engines.Clear();
  ViewEngines.Engines.Add(new RazorGlobalizationViewEngine());
}
```

محتوای یک فایل Resource

ساختار یک فایل resx. به صورت XML استاندارد است. در زیر محتوای یک نمونه فایل Resource با پسوند resx. را مشاهده میکنید:

```
<?xml version="1.0" encoding="utf-8"?>
<root>
  <!--
    Microsoft ResX Schema ...
    -->
  <xsd:schema id="root" xmlns="" xmlns:xsd="http://www.w3.org/2001/XMLSchema"
xmlns:msdata="urn:schemas-microsoft-com:xml-msdata">
   ...
  </xsd:schema>
  <resheader name="resmimetype">
    <value>text/microsoft-resx</value>
  </resheader>
  <resheader name="version">
    <value>2.0</value>
  </resheader>
  <resheader name="reader">
    <value>System.Resources.ResXResourceReader, System.Windows.Forms, Version=4.0.0.0, Culture=neutral,
PublicKeyToken=b77a5c561934e089</value>
  </resheader>
  <resheader name="writer">
    <value>System.Resources.ResXResourceWriter, System.Windows.Forms, Version=4.0.0.0, Culture=neutral,
PublicKeyToken=b77a5c561934e089</value>
  </resheader>
  <data name="RightToLeft" xml:space="preserve">
    <value>false</value>
    <comment>RightToleft is false in English!</comment>
  </data>
</root>
```

در قسمت ابتدایی تمام فایلهای resx. که توسط ویژوال استودیو تولید میشود کامنتی طولانی وجود دارد که به صورت خلاصه به شرح محتوا و ساختار یک فایل Resource میپردازد. در ادامه تگ نسبتا طولانی xsd:schema قرار دارد. از این قسمت برای معرفی ساختار داده ای فایلهای XML استفاده میشود. برای آشنایی بیشتر با XSD (یا XML Schema) به اینجا مراجعه کنید. به صورت خلاصه میتوان گفت که XSD برای تعیین ساختار دادهها یا تعیین نوع داده ای اطلاعات موجود در یک فایل XML به کار میرود. درواقع تگهای XSD به نوعی فایل XML ما را Strongly Typed میکند. با توجه به اطلاعات این قسمت، فایلهای resx. شامل 4 نوع گره اصلی هستند که عبارتند از: metadata و assembly و data و resheader. در تعریف هر یک از گرهها در این قسمت مشخصاتی چون نام زیر گرههای قابل تعریف در هر گره و نام و نوع خاصیتهای هر یک معرفی شده است.

بخش موردنظر ما در این مطلب قسمت انتهایی این فایلهاست (تگهای resheader و data). همانطور در بالا مشاهده میکنید تگهای reheader شامل تنظیمات مربوط به فایل resx. با ساختاری ساده به صورت name/value است. یکی از این تنظیمات resmimetype فایل resource را معرفی میکند که درواقع مشخص کننده نوع محتوای (Content Type) فایل XML است(^). برای فایلهای resx این مقدار برابر text/microsoft-resx است. تنظیم بعدی نسخه مربوط به فایل resx (یا Microsoft ResX Schema) را نشان میدهد. در حال حاضر نسخه جاری (در VS 2010) برابر 2.0 است. تنظیم بعدی مربوط به کلاسهای reader و writer تعریف شده برای استفاده از این فایلهاست. به نوع این کلاسهای خواننده و نویسنده فایلهای resx. و مکان فیزیکی و فضای نام آنها دقت کنید که در مطالب بعدی از آنها برای ویرایش و بروزرسانی فایلهای resource در زمان اجرا استفاده خواهیم کرد.

در پایان نیز تگهای data که برای نگهداری دادهها از آنها استفاده میشود. هر گره data شامل یک خاصیت نام (name) و یک زیرگره مقدار (value) است. البته امکان تعیین یک کامنت در زیرگره comment نیز وجود دارد که اختیاری است. هر گره data مینواند شامل خاصیت type و یا mimetype نیز باشد. خاصیت type مشخص کننده نوعی است که تبدیل text/value را با استفاده از ساختار TypeConverter پشتیبانی میکند. البته اگر در نوع مشخص شده این پشتیبانی وجود نداشته باشد، داده موردنظر پس از سریالایز شدن با فرمت مشخص شده در خاصیت mimetype ذخیره میشود. این mimetype اطلاعات موردنیاز را برای کلاس خواننده این فایلها (ResXResourceReader به صورت پیشفرض) جهت چگونگی بازیابی آبجکت موردنظر فراهم میکند. مشخص کردن این دو خاصیت برای انواع رشته ای نیاز نیست. انواع mimetype قابل استفاده عبارتند از:

- application/x-microsoft.net.object.binary.base64: آبجکت موردنظر باید با استفاده از کلاس System.Runtime.Serialization.Formatters.Binary.BinaryFormatter سریالایز شده و سپس با فرمت base64 به یک رشته انکد شود (راجع به انکدینگ base64 ^ و ^).

- application/x-microsoft.net.object.soap.base64: آبجکت موردنظر باید با استفاده از کلاس

System.Runtime.Serialization.Formatters.Soap.SoapFormatter سریالایز شده و سپس با فرمت base64 به یک رشته انکد شود.

- application/x-microsoft.net.object.bytearray.base64: آبجکت ابتدا باید با استفاده از یک System.ComponentModel.TypeConverter به آرایه ای از بایت سریالایز شده و سپس با فرمت base64 به یک رشته انکد شود.

نکته : امکان جاسازی کردن (embed) فایلهای resx. در یک اسمبلی یا کامپایل مستقیم آن به یک سَتِلایت اسمبلی (ترجمه مناسبی برای satellite assembly پیدا نکردم، چیزی شبیه به اسمبلی قمری یا وابسته و از این قبیل ...) وجود ندارد. ابتدا باید این فایلهای resx. به فایلهای resources. تبدیل شوند. اینکار با استفاده از ابزار Resource File Generator (نام فایل اجرایی آن resgen.exe است) انجام میشود (^ و ^). سپس میتوان با استفاده از Assembly Linker ستلایت اسمبلی مربوطه را تولید کرد (^). کل این عملیات در ویژوال استودیو با استفاده از ابزار msbuild به صورت خودکار انجام میشود!

نحوه یافتن کلیدهای Resource در بین فایلهای مختلف Resx توسط پرووایدر پیش فرض در دات نت

عملیات ابتدا با بررسی خاصیت CurrentUICulture از ثرد جاری آغاز میشود. سپس با استفاده از عنوان استاندارد کالچر جاری، فایل مناسب Resource یافته میشود. در نهایت بهترین گزینه موجود برای کلید درخواستی از منابع موجود انتخاب میشود. مثلا اگر کالچر جاری fa-IR و کلید درخواستی از کلاس Texts باشد ابتدا جستجو برای یافتن فایل Texts.fa-IR.resx آغاز میشود و اگر فایل موردنظر یا کلید درخواستی در این فایل یافته نشد جستجو در فایل Texts.fa.resx ادامه می‌یابد. اگر باز هم یافته نشد درنهایت این عملیات جستجو در فایل resource اصلی خاتمه می‌یابد و مقدار کلید منبع پیش فرض به عنوان نتیجه برگشت داده میشود. یعنی در تمامی حالات سعی میشود تا دقیقترین و بهترین و نزدیکترین نتیجه انتخاب شود. البته درصورتیکه از یک پرووایدر شخصی سازی شده برای کار خود استفاده میکنید باید چنین الگوریتمی را جهت یافتن کلیدهای منابع خود از فایلهای Resource (یا هرمنبع دیگر مثل دیتابیس یا حتی یک وب سرویس) درنظر بگیرید.

Globalization در کلاینت (javascript g11n)

یکی دیگر از موارد استفاده g11n در برنامه نویسی سمت کلاینت است. با وجود استفاده گسترده از جاوا اسکریپت در برنامه نویسی سمت کلاینت در وب اپلیکیشنها، متاسفانه تا همین اواخر عملا ابزار یا کتابخانه مناسبی برای مدیریت g11n در این زمینه وجود نداشته است. یکی از اولین کتابخانه‌های تولید شده در این زمینه کتابخانه jQuery Globalization است که توسط مایکروسافت توسعه داده شده است (برای آشنایی بیشتر با این کتابخانه به ^ و ^ مراجعه کنید). این کتابخانه بعدا تغییر نام داده و اکنون با عنوان Globalize شناخته میشود. Globalize یک کتابخانه کاملا مستقل است که وابستگی به هیچ کتابخانه دیگر ندارد (یعنی برای استفاده از آن نیازی به jQuery نیست). این کتابخانه حاوی کالچرهای بسیاری است که عملیات مختلفی چون فرمت و parse انواع داده‌ها را نیز در سمت کلاینت مدیریت میکند. همچنین با فراهم کردن منابعی حاوی جفتهای key/culture میتوان از مزایایی مشابه مواردی که در این مطلب بحث شد در سمت کلاینت نیز بهره برد. نشانی این کتابخانه در github اینجا است. با اینکه خود این کتابخانه ابزار کاملی است اما در بین کالچرهای موجود در فایلهای آن متاسفانه پشتیبانی کاملی از زبان فارسی نشده است. ابزار دیگری که برای اینکار وجود دارد پلاگین jquery localize است که برای بحث g11n رشته‌ها پیاده‌سازی بهتر و کاملتری دارد.

در مطالب بعدی به مباحث تغییر مقادیر کلیدهای فایلهای resource در هنگام اجرا با استفاده از روش مستقیم تغییر محتوای فایلها و کامپایل دوباره توسط ابزار msbuild و نیز استفاده از یک ResourceProvider شخصی سازی شده به عنوان یک راه حل بهتر برای اینکار میپردازم.

در تهیه این مطلب از منابع زیر استفاده شده است: Localization in ASP.NET MVC – 3 Days Investigation, 1 Day Job

ASP.NET MVC 3 Internationalization

Localization and skinning in ASP.NET MVC 3 web applications Simple ASP.Net MVC Globalization with Graceful Fallback

Globalization, Internationalization and Localization in ASP.NET MVC 3, JavaScript and jQuery - Part 1

نظرات خوانندگان

نویسنده: امیرحسین مرجانی
تاریخ: ۱۳۹۱/۱۰/۲۱ ۲۳:۵

سلام آقای یوسف نژاد
من بعد از تلاش‌های زیاد توی پروژه‌های مختلف این مطالبی که شما نوشته اید رو پیاده سازی کردم ، ولی خیلی پراکنده.
ولی حالا می‌بینم شما به زیبایی این مطالب رو کنار هم قرار دادید.
می خواستم بابت مطلب خوب و مفیدتون و همچنین وقتی که گذاشتید تشکر کنم.
ممنونم بابت زحمات شما

اگر ممکنه برچسب MVC رو هم به مطلبتون اضافه کنید.

نویسنده: یوسف نژاد
تاریخ: ۱۳۹۱/۱۰/۲۱ ۲۳:۲۴

با سلام و تشکر بابت نظر لطف شما.
البته باید بگم که همه دوستانی که اینجا به عنوان نویسنده کمک میکنند هدفشون اشتراک مطالبی هست که یاد گرفته اند تا سایر دوستان هم استفاده کنند.

برچسب MVC هم اضافه شد. با تشکر از دقت نظر شما.

نویسنده: امیرحسین جلوداری
تاریخ: ۱۳۹۱/۱۰/۲۲ ۱:۳

کاملا مشخصه که مطلب از روی تجربه‌ی کاریه و بسیار عالی جمع آوری شده ... ممنون ... به طرز عجیبی منتظر قسمت بعدم :دی

نویسنده: پندار
تاریخ: ۱۳۹۱/۱۲/۰۸ ۲۱:۳۸

گویا در MVC 4 این روش پاسخ نمیدهد. لطفا در این مورد برای MVC 4 راه حلی بدهید

نویسنده: محسن
تاریخ: ۱۳۹۱/۱۲/۰۸ ۲۳:۶

MVC 4 فقط یک سری افزونه بیشتر از MVC3 داره. مثلا razor آن بهبود پیدا کرده، فشرده سازی فایل‌های CSS به اون اضافه شده یا Web API رو به صورت یکپارچه داره. از لحاظ کار با فایل‌های منبع فرقی نکرده.

نویسنده: پندار
تاریخ: ۱۳۹۱/۱۲/۰۹ ۹:۲۱

متن نشانی زیر را مطالعه کنید

http://geekswithblogs.net/shaunxu/archive/2012/09/04/localization-in-asp.net-mvc-ndash-upgraded.aspx

نویسنده: محسن
تاریخ: ۱۳۹۱/۱۲/۰۹ ۹:۴۳

مطلبی که لینک دادی در مورد آپدیت یک helper شخصی توسعه داده شده توسط شخص ثالث است از MVC2 به MVC4. اگر کسی

از این راه حل شخصی و خاص استفاده نکرده باشه، اصول فوق فرقی نکرده.

نویسنده: صابر فتح الهی
تاریخ: ۱۳۹۱/۱۲/۱۴ ۱۶:۵

مطلب خیلی خوبی بود کلی استفاده کردیم.
مهندس کالچر زبان کردی چی میشه؟ توی لیست منابعی که دادین گیر نیاوردم

نویسنده: وحید نصیری
تاریخ: ۱۳۹۱/۱۲/۱۴ ۱۷:۲۲

kur هست مطابق استاندارد .

نویسنده: صابر فتح الهی
تاریخ: ۱۳۹۱/۱۲/۱۷ ۲:۱۱

سلام
اما مهندس کلاس Culture Info این مقدار قبول نمی‌کنه

نویسنده: وحید نصیری
تاریخ: ۱۳۹۱/۱۲/۱۷ ۹:۳

می‌تونید کلاس فرهنگ سفارشی را ایجاد و استفاده کنید.

نویسنده: صابر فتح الهی
تاریخ: ۱۳۹۱/۱۲/۱۷ ۱۰:۱۲

اما روش گفته شده نیاز به دسترسی مدیریت دارد که روی سرورهای اشتراکی ممکن نیست

نویسنده: وحید نصیری
تاریخ: ۱۳۹۱/۱۲/۱۷ ۱۱:۱۹

نحوه توسعه اکثر برنامه‌ها و کتابخانه‌ها در طول زمان، بر اساس تقاضا و پیگیری مصرف کننده است. اگر بعد از بیش از 10 سال، چنین فرهنگی اضافه نشده یعنی درخواستی نداشته. مراجعه کنید به محل پیگیری این نوع مسایل .

نویسنده: صابر فتح الهی
تاریخ: ۱۳۹۱/۱۲/۱۹ ۱۰:۱۴

سلام مهندس یوسف نژاد (ابتدا ممنونم از پست خوب شما)
با پیروی از پست شما
ابتدا فایل‌های ریسورس در پروژه جاری فولدر App_GlobalResources گذاشتم و پروژه در صفحات aspx با قالب زیر به راحتی تغییر زبان داده میشد:

```
<asp:Literal ID="Literal1" Text='<%$ Resources:resource, Title %>' runat="server" />
```

اما بعدش فایل هارو توی یک پروژه کتابخانه ای جدید گذاشتم و Build Action فایل‌های ریسورس روی Embedded Resource تنظیم کردم، پروژه با موفقیت اجرا شد و در سمت سرور با کد زیر راحت به مقادیر دسترسی دارم:

```
Literal1.Text=ResourcesManager.Resource.Title;
```

اما در سمت صفحات aspx با کد قبلی به شکل زیر نمایش نمیده و خطا صادر میشه:

```
<asp:Literal ID="Literal1"  runat="server" Text='<%$ ResourcesManager.Resource:resource, Title %>'/>
```

و خطای زیر صادر میشه:

```
Parser Error
Description: An error occurred during the parsing of a resource required to service this request.
Please review the following specific parse error details and modify your source file appropriately.

Parser Error Message: The expression prefix 'ResourcesManager.Resource' was not recognized.  Please
correct the prefix or register the prefix in the <expressionBuilders> section of configuration.

Source Error:
```

مراحل این پست روی هم دنبال کردم اما بازم نشد.
چه تنظیماتی ست نکردم ؟

نویسنده: یوسف نژاد
تاریخ: ۱۳۹۲/۰۱/۳۱ ۱۲:۴۴

ببخشید یه چند وقتی فعال نبودم و پاسخ این سوال رو دیر دارم میدم.
امکان استفاده از کلیدهای Resource برای مقداردهی خواص سمت سرور کنترلها در صفحات aspx به صورت مستقیم وجود ندارد. بنابراین برای استفاده از این کلیدها همانند روش پیش فرض موجود در ASP.NET باید از یکسری ExpressionBuilder استفاده شود که کار Parse عبارت وارده برای این خواص را در سمت سرور انجام میدهد. کلاس پیش فرض برای اینکار در ASP.NET Web Form که از پیشوند Resources استفاده میکند تنها برای Resourceهای محلی (Local) موجود در فولدرهای پیش فرض (App_GlobalResources و App_LocalResources) کاربرد دارد و برای استفاده از Resourceهای موجود در منابع ریفرنس داده شده به پروژه باید از روشی مثل اونچه که خود شما لینکش رو دادین استفاده کرد.
من این روش رو استفاده کردم و پیاده سازی موفقی داشتم. نمیدونم مشکل شما چیه...

نویسنده: یوسف نژاد
تاریخ: ۱۳۹۲/۰۱/۳۱ ۱۲:۵۲

اگر مشکلی در پیاده سازی روش بالا دارین، تمام مراحلی که من طی کردم دقیقا اینجا میارم:
ابتدا کلاس ExpressionBuilder رو به صورت زیر مثلا در خود پروژه Resources اضافه میکنیم:

```
using System.Web.Compilation;
using System.CodeDom;
namespace Resources
{
  [ExpressionPrefix("MyResource")]
  public class ResourceExpressionBuilder : ExpressionBuilder
  {
    public override System.CodeDom.CodeExpression GetCodeExpression(System.Web.UI.BoundPropertyEntry
entry, object parsedData, System.Web.Compilation.ExpressionBuilderContext context)
    {
      return new CodeSnippetExpression(entry.Expression);
    }
  }
}
```

سپس تنظیمات زیر رو به Web.config اضافه میکنیم:

```
<compilation debug="true" targetFramework="4.0">
    <expressionBuilders>
        <add expressionPrefix="MyResource" type="Resources.ResourceExpressionBuilder, Resources" />
```

```
    </expressionBuilders>
</compilation>
```

درنهایت به صورت زیر میتوان از این کلاس استفاده کرد:

```
<asp:Literal ID="Literal1"  runat="server" Text="<%$ MyResource: Resources.Resource1.String2 %>" />
```

هرچند ظاهرا مقدار پیشوند معرفی شده در Attribute کلاس ResourceExpressionBuilder اهمیت چندانی ندارد! امیدوارم مشکلتون حل بشه.

نویسنده: صابر فتح الهی
تاریخ: ۱۳۹۲/۰۲/۰۱ ۲:۲۹

ممنونم از پاسخ شما
همون روش شمارو دنبال کردم پاسخ گرفتم، اشکال از خودم بود
با تشکر از شما

نویسنده: صادق نجاتی
تاریخ: ۱۳۹۲/۱۲/۲۷ ۱۲:۰

با سلام
ضمن تشکر از مطلب بسیار خوبتون
خاصیت DisplayFormat قابلیت استفاده از کلیدهای Resource را ندارد !
لطفا راهنمایی فرمایید که چطور میشه از این خاصیت برای DisplayFormat استفاده کرد؟
من می‌خوام برای تاریخ در زبانه فارسی از فرمت {yyyy-MM-dd} و در زبانه انگلیسی از {yyyy-dd-MM} استفاده کنم.
با سپاس فراوان

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).

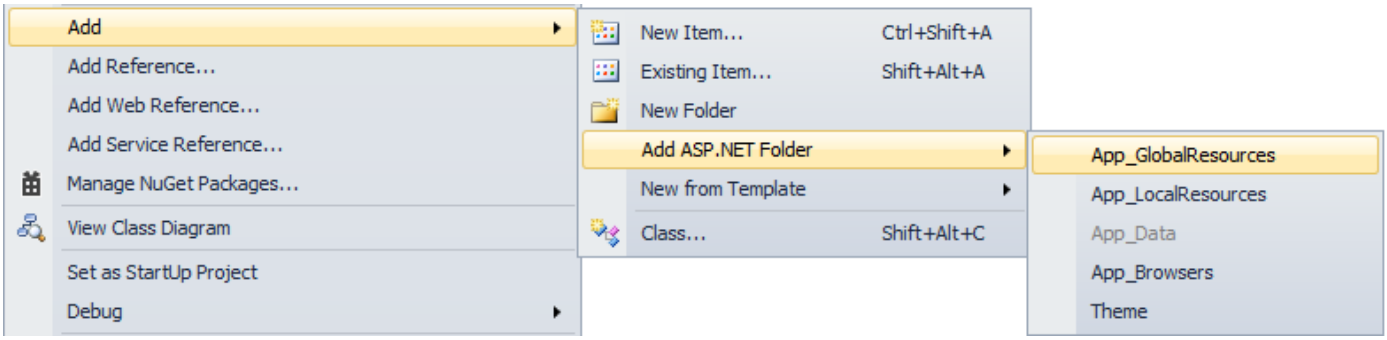
- در زمان اجرا موتور پیش فرض مدیریت منابع دات نت با توجه به کالچر UI در ثرد جاری اقدام به انتخاب مقدار مناسب برای کلیدهای درخواستی (به همراه فرایند fallback) می‌کند. فرایند نسبتا پیچیده fallback در اینجا شرح داده شده است.

منابع Global و Local

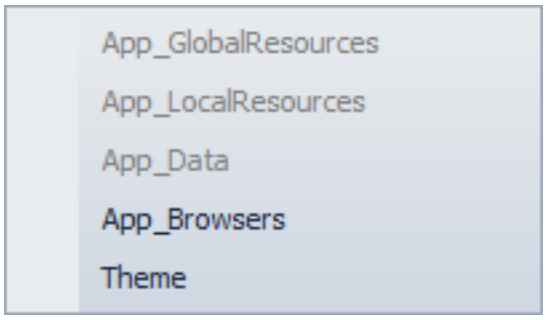
در ASP.NET دو نوع کلی Resource وجود دارد که هر کدام برای موقعیت‌های خاصی مورد استفاده قرار می‌گیرند:

- 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:



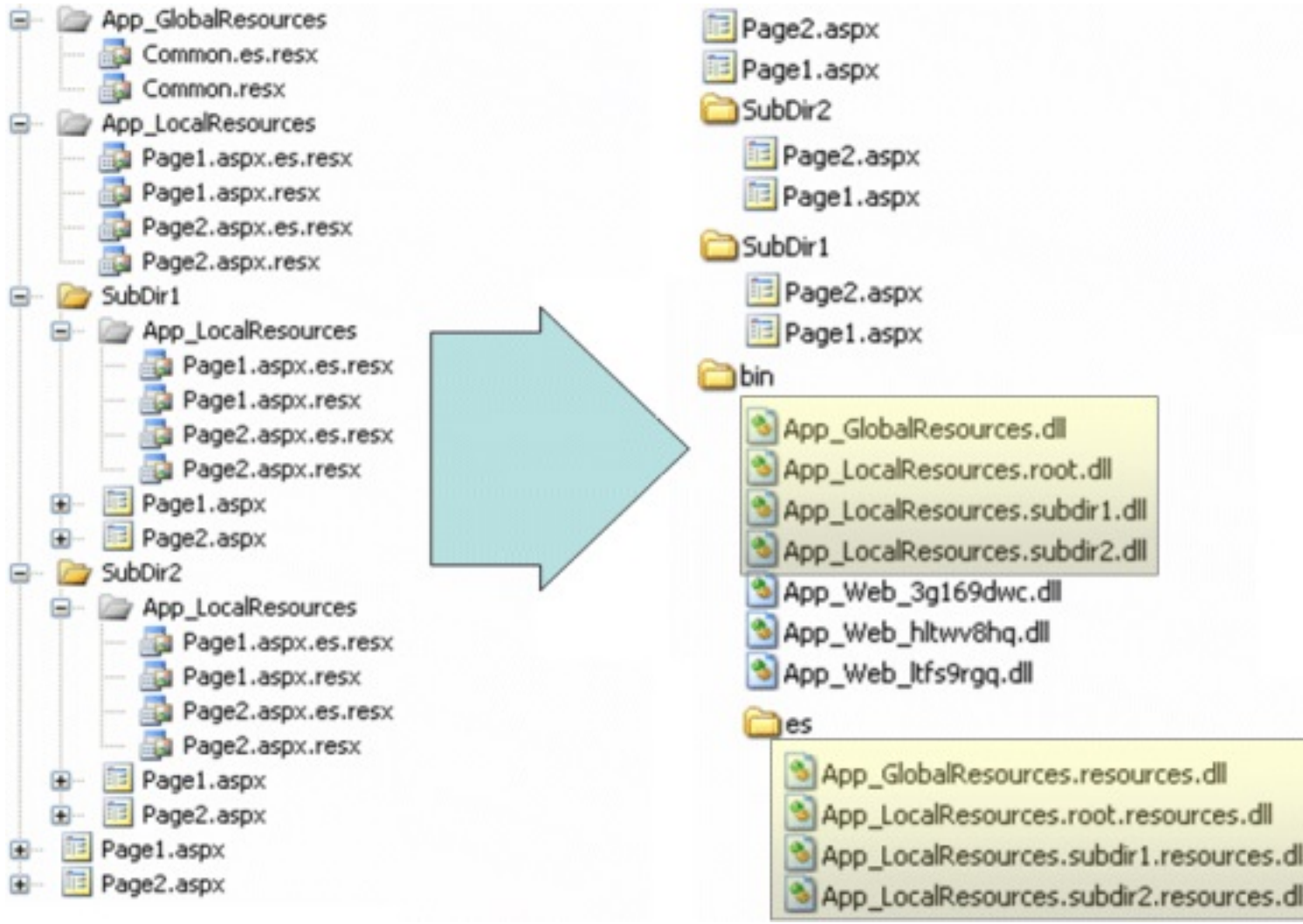
نکته: دقت کنید که تنها یک فولدر App_GlobalResources به هر پروژه می‌توان افزود. همچنین در ریشه هر مسیر موجود در پروژه تنها می‌توان یک فولدر Appp_LocalResources داشت. پس از افزودن هر یک از این فولدرهای مخصوص، منوی فوق به صورت زیر در خواهد آمد:



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.

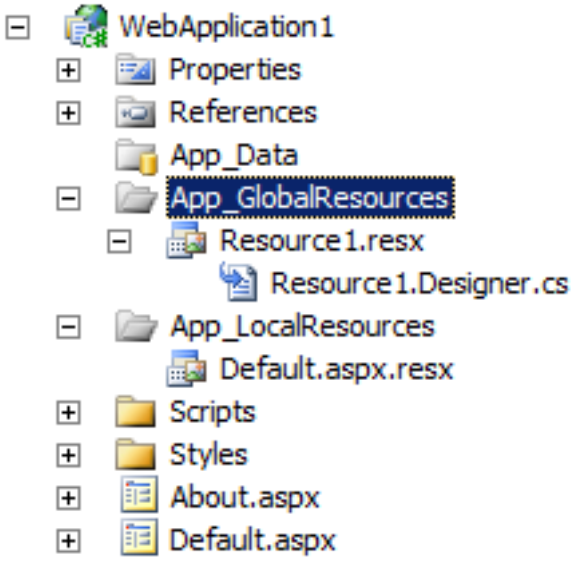
برای روشن‌تر شدن مطالب اشاره شده در بالا به تصویر فرضی! زیر توجه کنید (اسمبلی‌های تولید شده برای منابع کلی و محلی فرضی است):



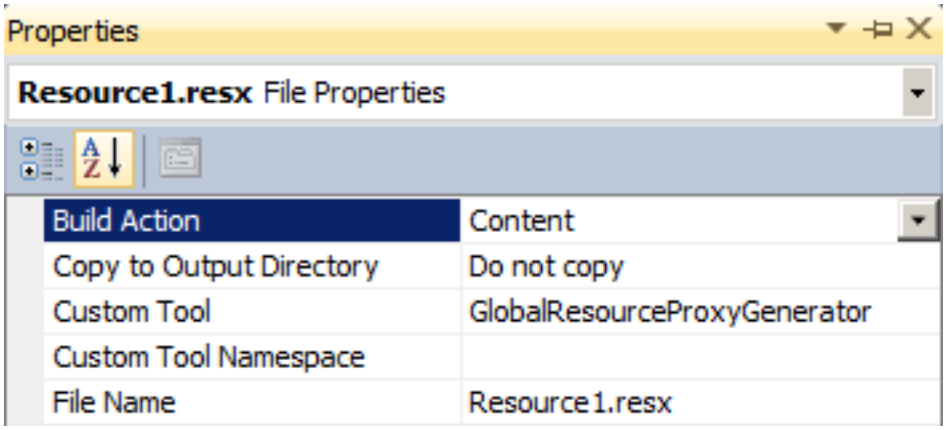
در تصویر بالا محل قرارگیری انواع مختلف فایلهای Resource و نیز محل نهایی فرضی اسمبلی‌های ستلایت تولید شده، برای حداقل یک زبان غیر از زبان پیش فرض برنامه، نشان داده شده است.

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.

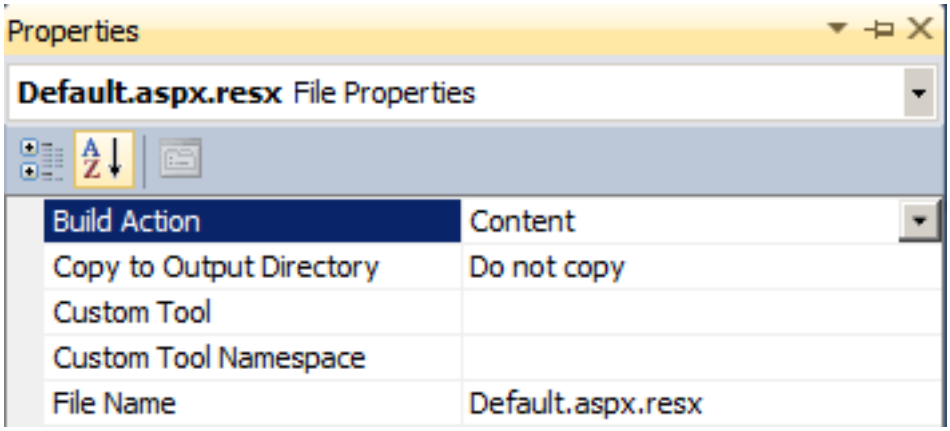
برای آشنایی بیشتر بهتر است یک برنامه **وب اپلیکیشن** جدید ایجاد کرده و همانند تصویر زیر یکسری فایل Resource به فولدرهای اشاره شده در بالا اضافه کنید:



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 استفاه 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

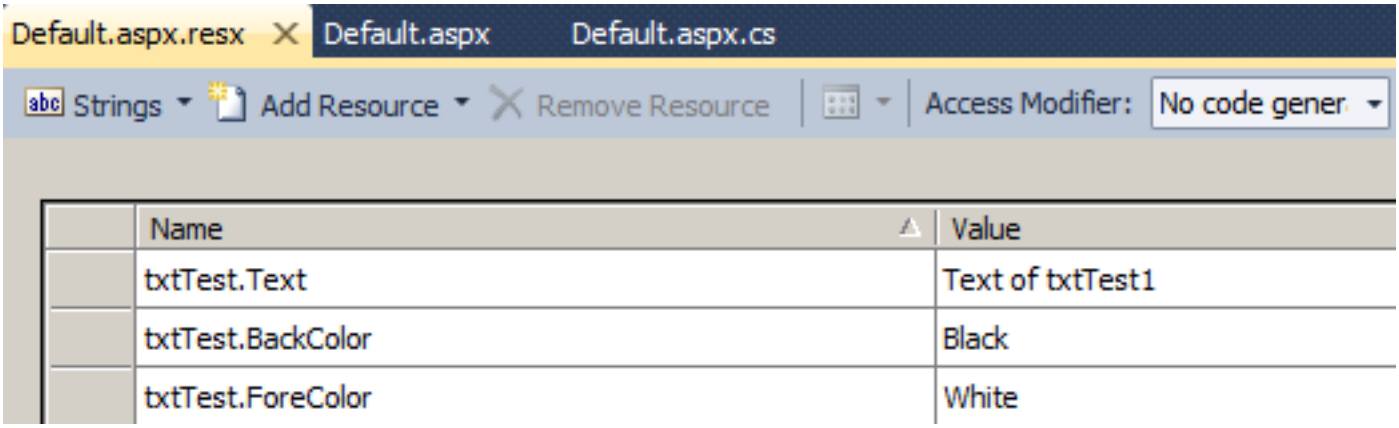
برای مثال در لیبل بالا که نام کلید Resource آن به lblLocalResources تنظیم شده است، اگر نام صفحه وب مربوطه page1.aspx باشد، برای تنظیم خواص آن در فایل page1.aspx.resx مربوطه باید از کلیدهایی با عناوینی مثل عنوان‌های زیر استفاه کرد:

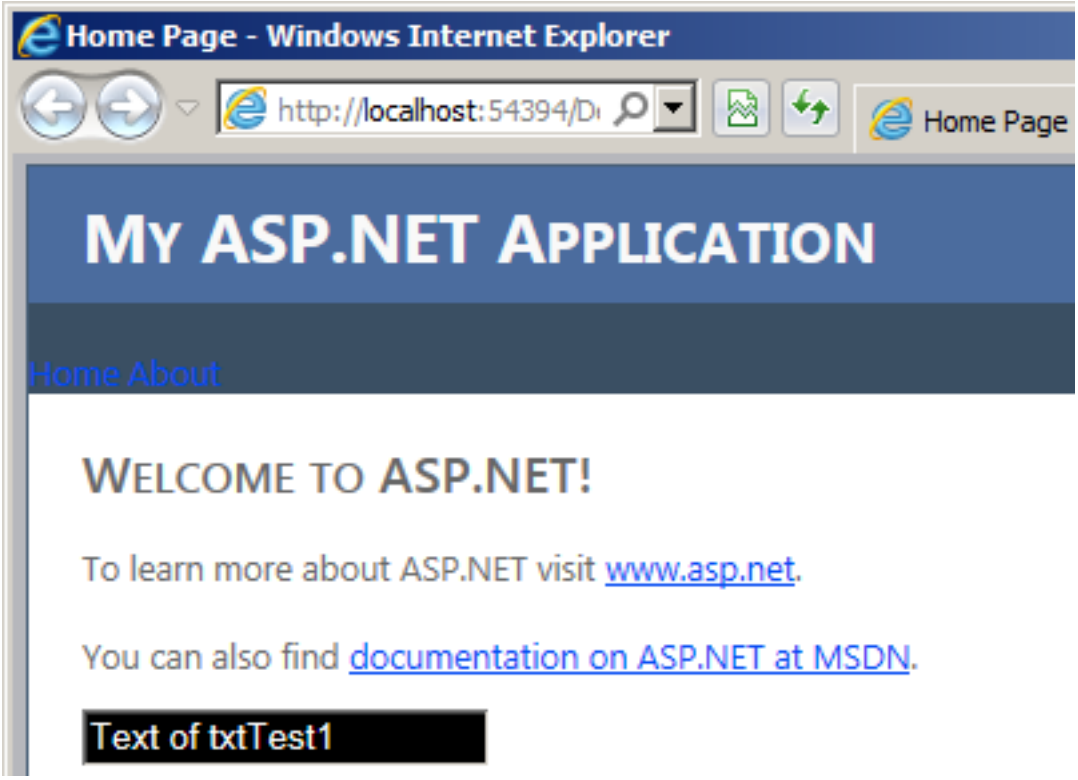
lblLocalResources.Text

lblLocalResources.BackColor

برای نمونه به تصاویر زیر دقت کنید:

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





بومی سازی صریح (Explicit Localization)

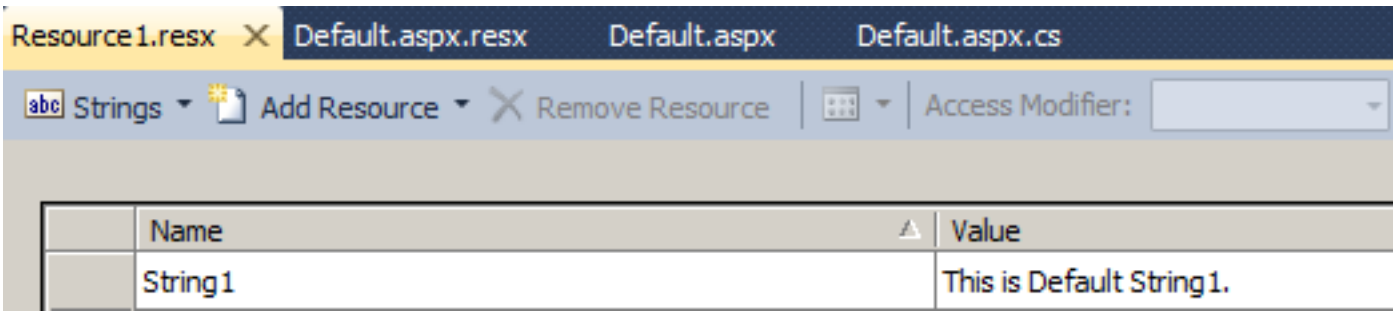
در استفاده از این نوع عبارات، پراپرتی مربوطه و نام فایل منبع صراحتا در تگ کنترل مربوطه آورده می‌شود. بنابراین برای هر خاصیتی که می‌خواهیم مقدار آن از منبعی خاص گرفته شود باید از عبارتی با طرح زیر استفاده کنیم:

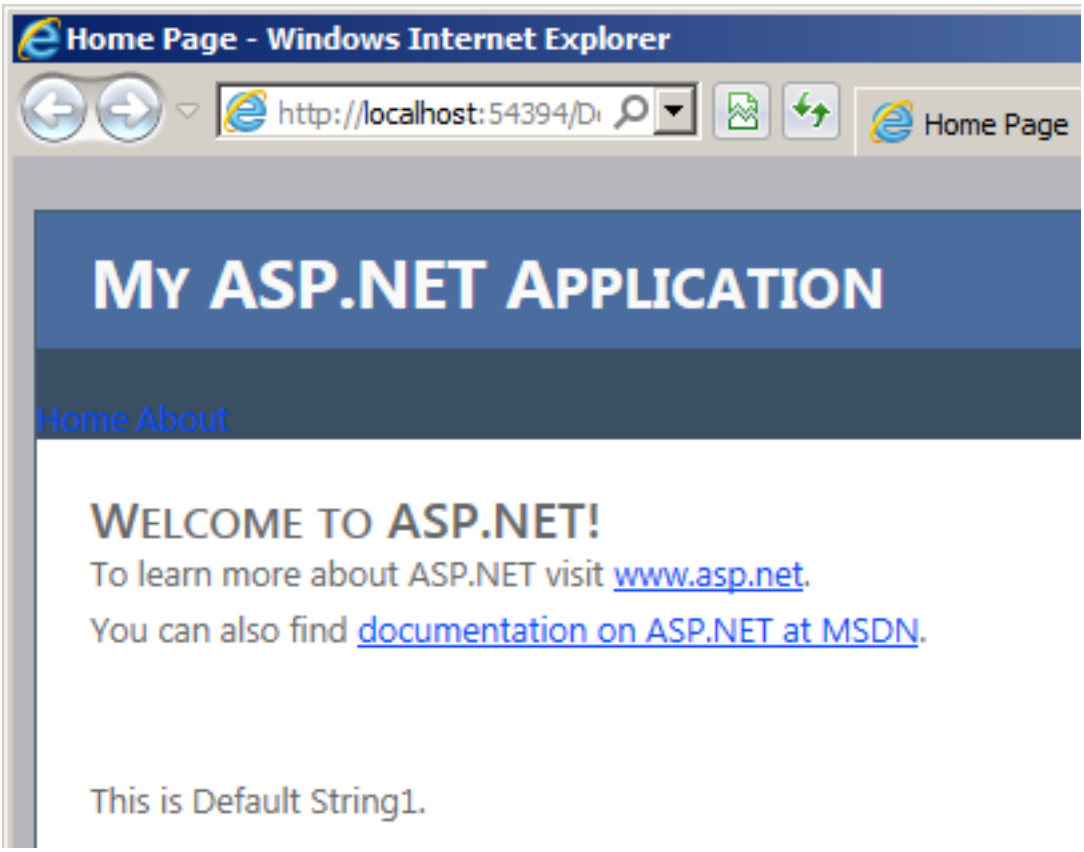
```
<% Resources: Class, ResourceKey $%>
```

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

عنوان: **Globalization در ASP.NET MVC - قسمت چهارم**
نویسنده: یوسف نژاد
تاریخ: ۱۳۹۲/۰۲/۲۴ ۲۱:۲۵
آدرس: www.dotnettips.info
برچسب‌ها: Globalization, Resource, .resx, ResourceProviderFactory, ResourceManager

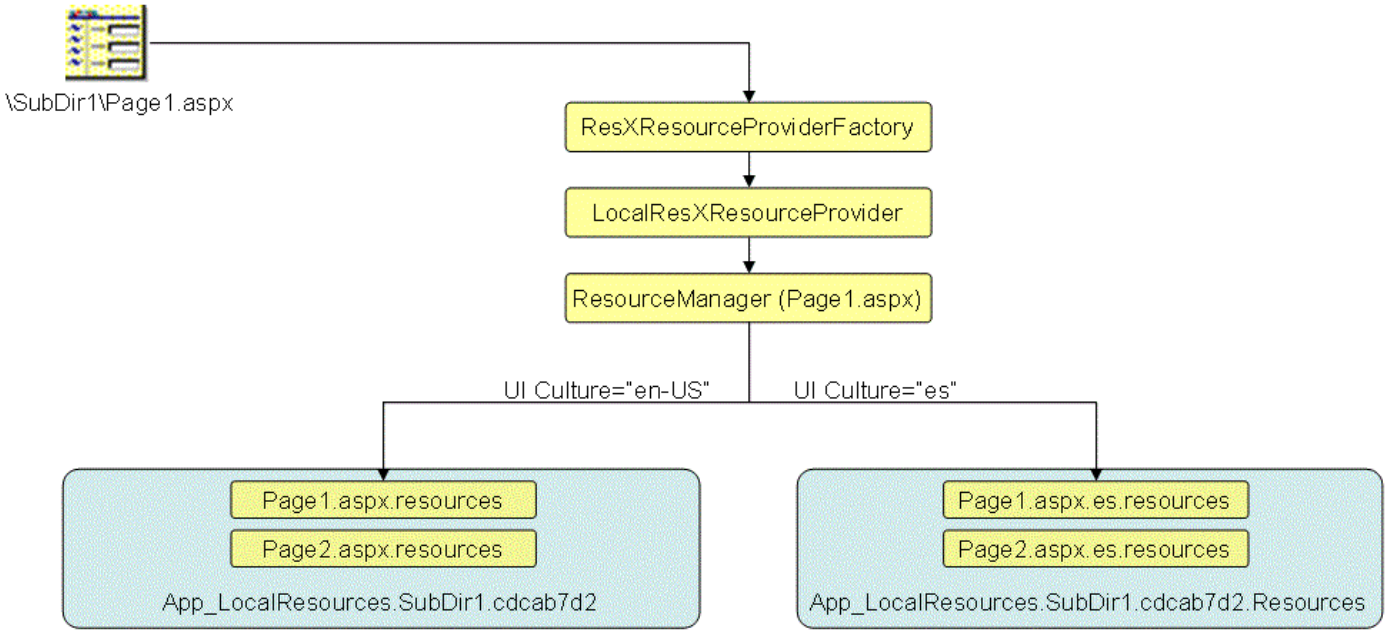
در قسمت قبل مقدمه ای راجع به انواع منابع موجود در ASP.NET و برخی مسائل پیرامون آن ارائه شد. در این قسمت راجع به نحوه رفتار ASP.NET در برخورد با انواع منابع بحث می‌شود.

مدیریت منابع در ASP.NET

در مدل پرووایدر منابع در ASP.NET کار مدیریت منابع از کلاس **ResourceProviderFactory** شروع می‌شود. این کلاس که از نوع abstract تعریف شده است، دو متد برای فراهم کردن پرووایدرهای کلی و محلی دارد.
کلاس پیش‌فرض در ASP.NET برای پیاده‌سازی ResourceProviderFactory در اسمبلی System.Web قرار دارد. این کلاس که **ResXResourceProviderFactory** نام دارد نمونه‌هایی از کلاس‌های **LocalResxResourceProvider** و **GlobalResxResourceProvider** را برمی‌گرداند. درباره این کلاس‌ها در ادامه بیشتر بحث خواهد شد.

نکته: هر سه کلاس پیش‌فرض اشاره شده در بالا و نیز سایر کلاس‌های مربوط به عملیات مدیریت منابع در آن‌ها، همگی در فضای نام System.Web.Compilation قرار دارند و متاسفانه دارای سطح دسترسی internal هستند. بنابراین به صورت مستقیم در دسترس نیستند.

برای نمونه با توجه به تصویر فرضی نشان داده شده در قسمت قبل ، در اولین بارگذاری صفحه SubDir1\Page1.aspx عبارات ضمنی بکاربرده شده در این صفحه برای منابع محلی (در قسمت قبل شرح داده شده است) باعث فراخوانی متد مربوط به Local Resources در کلاس **ResXResourceProviderFactory** می‌شود. این متد نمونه‌ای از کلاس **LocalResXResourceProvider** برمی‌گرداند. (در ادامه با نحوه سفارشی‌سازی این کلاس‌ها نیز آشنا خواهیم شد).
رفتار پیش‌فرض این پرووایدر این است که نمونه‌ای از کلاس **ResourceManager** با توجه به کلید درخواستی برای صفحه موردنظر (مثلا نوع Page1.aspx در اسمبلی App_LocalResources.subdir1.XXXXXX که در تصویر موجود در قسمت قبل نشان داده شده است) تولید می‌کند. حال این کلاس با استفاده از کالچر مربوط به درخواست موردنظر، ورودی موردنظر را از منبع مربوطه استخراج می‌کند. مثلا اگر کالچر موردبحث es (اسپانیایی) باشد، اسمبلی ستلایت موجود در مسیر نسبی \es\ انتخاب می‌شود.
برای روشن‌تر شدن بحث به تصویر زیر که عملیات مدیریت منابع پیش فرض در ASP.NET در درخواست صفحه Page1.aspx از پوشه SubDir1 را نشان می‌دهد، دقت کنید:



همانطور که در قسمت اول این سری مطالب عنوان شد، رفتار کلاس ResourceManager برای یافتن کلیدهای Resource، استخراج آن از نزدیکترین گزینه موجود است. یعنی مثلا برای یافتن کلیدی در کالچر es در مثال بالا، ابتدا اسمبلی‌های مربوط به این کالچر جستجو می‌شود و اگر ورودی موردنظر یافته نشد، جستجو در اسمبلی‌های ستلایت پیش‌فرض سیستم موجود در ریشه فولدر bin برنامه ادامه می‌یابد، تا درنهایت نزدیک‌ترین گزینه پیدا شود (فرایند fallback).

نکته: همانطور که در تصویر بالا نیز مشخص است، نحوه نامگذاری اسمبلی منابع محلی به صورت App_LocalResources.<SubDirectory>.<A random code> است.

نکته: پس از اولین بارگذاری هر اسمبلی، آن اسمبلی به همراه خود نمونه کلاس ResourceManager که مثلا توسط کلاس LocalResXResourceProvider تولید شده است در حافظه سرور کش می‌شوند تا در استفاده‌های بعدی به کار روند.

نکته: فرایند مشابه‌ای برای یافتن کلیدها در منابع کلی (Global Resources) به انجام می‌رسد. تنها تفاوت آن این است که کلاس ResXResourceProviderFactory نمونه‌ای از کلاس **GlobalResXResourceProvider** تولید می‌کند.

چرا پرووایدر سفارشی؟

تا اینجا بالا با کلیات عملیاتی که ASP.NET برای بارگذاری منابع محلی و کلی به انجام می‌رساند، آشنا شدیم. حالا باید به این پرسش پاسخ داد که چرا پرووایدری سفارشی نیاز است؟ علاوه بر دلایلی که در قسمت‌های قبلی به آنها اشاره شد، می‌توان دلایل زیر را نیز برشمرد:

- استفاده از منابع و یا اسمبلی‌های ستلایت موجود - اگر بخواهید در برنامه خود از اسمبلی‌هایی مشترک، بین برنامه‌های ویندوزی و وبی استفاده کنید، و یا بخواهید به هردلیلی از اسمبلی‌های جداگانه‌ای برای این منابع استفاده کنید، مدل پیش‌فرض موجود در ASP.NET جوابگو نخواهد بود.

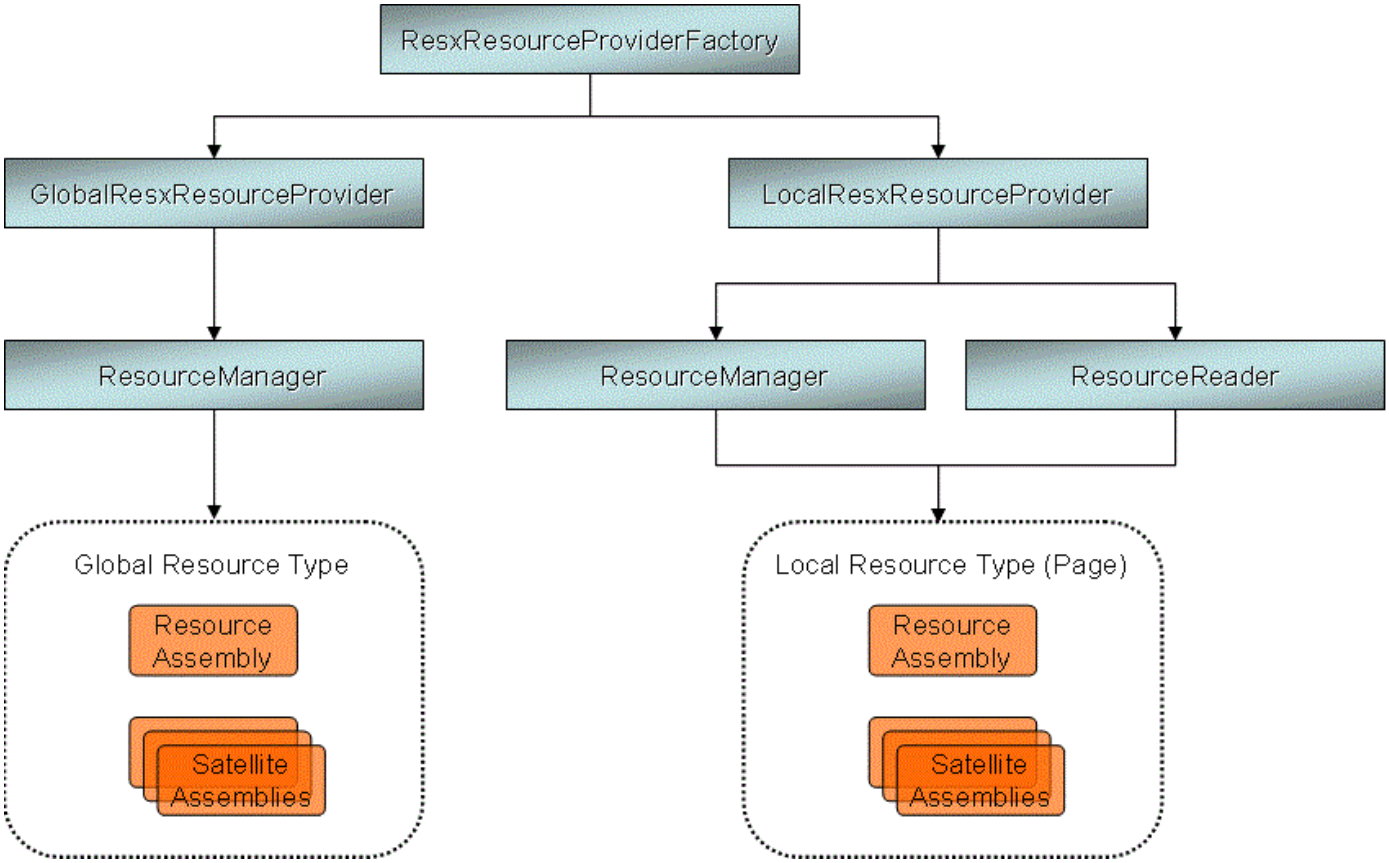
- استفاده از منابع دیگری به غیر از فایلهای resx. مثل دیتابیس - برای برنامه‌های تحت وب که صفحات بسیار زیاد به همراه ورودی‌های بیشماری از Resourceها دارند، استفاده از مدل پرووایدر منابع پیش‌فرض در ASP.NET و ذخیره تمامی این ورودی‌ها درون فایل‌های resx. بار نسبتا زیادی روی حافظه سرور خواهد گذاشت. درصورت مدیریت بهینه فراخوانی‌های سمت دیتابیس می‌توان با بهره‌برداری از جداول یک دیتابیس به عنوان منبع، کمک زیادی به وب سرور کرد! هم‌چنین با استفاده از دیتابیس می‌توان

مدیریت بهتری بر ورودی‌ها داشت و نیز امکان ذخیره‌سازی حجم بیشتری از داده‌ها در اختیار توسعه دهنده قرار خواهد گرفت.
البته به غیر از دیتابیس و فایل‌های resx. نیز گزینه‌های دیگری برای ذخیره‌سازی ورودی‌های این منابع وجود دارند. به عنوان مثال می‌توان مدیریت این منابع را کلا به سیستم دیگری سپرد و درخواست ورودی‌های موردنیاز را به یکسری وب‌سرویس سپرد. برای پیاده سازی چنین سیستمی نیاز است تا مدلی سفارشی تهیه و استفاده شود.

- پیاده سازی امکان به روزرسانی منابع در زمان اجرا - درصورتی‌که بخواهیم امکان بروزرسانی ورودی‌ها را در زمان اجرا در استفاده از فایلهای resx. داشته باشیم، یکی از راه‌حل‌ها، سفارشی سازی این پرووایدرهاست.

مدل پرووایدر منابع

همانطور که قبلا هم اشاره شد، وظیفه استخراج داده‌ها از Resourceها به صورت پیش‌فرض، درنهایت بر عهده نمونه‌ای از کلاس ResourceManager است. در واقع این کلاس کل فرایند انتخاب مناسب‌ترین کلید از منابع موجود را با توجه به کالچر رابط کاربری (UI Culture) در ثرد جاری کپسوله می‌کند. درباره این کلاس در ادامه بیشتر بحث خواهد شد.
هم‌چنین بازهم همانطور که قبلا توضیح داده شد، استفاده از ورودی‌های منابع موجود به دو روش انجام می‌شود. استفاده از عبارات بومی‌سازی و نیز با استفاده از برنامه‌نویسی که ازطریق دومتد GetLocalResourceObject و GetGlobalResourceObject انجام می‌شود. درضمن کلیه عبارات بومی‌سازی در زمان رندر صفحات وب درنهایت تبدیل به فراخوانی‌هایی از این دو متد در کلاس TemplateControl خواهند شد.
عملیات پس از فراخوانی این دو متد جایی است که مدل Resource Provider پیش‌فرض ASP.NET وارد کار می‌شود. این فرایند ابتدا با فراخوانی نمونه‌ای از کلاس ResourceProviderFactory آغاز می‌شود که پیاده‌سازی پیش‌فرض آن در کلاس ResXResourceProviderFactory قرار دارد.
این کلاس سپس با توجه به نوع منبع درخواستی (Global یا Local) نمونه‌ای از پرووایدر مربوطه (که باید اینترفیس IResourceProvider را پیاده‌سازی کرده باشند) را تولید می‌کند. پیاده‌سازی پیش‌فرض این پرووایدرها در ASP.NET در کلاس‌های GlobalResXResourceProvider و LocalResXResourceProvider قرار دارد.
این پروایدرها درنهایت باتوجه به محل ورودی درخواستی، نمونه مناسب از کلاس RsourceManager را تولید و استفاده می‌کنند.
هم‌چنین در پروایدرهای محلی، برای استفاده از عبارات بومی‌سازی ضمنی، نمونه‌ای از کلاس **ResourceReader** مورد استفاده قرار می‌گیرد. در زمان تجزیه و تحلیل صفحه وب درخواستی در سرور، با استفاده از این کلاس کلیدهای موردنظر یافته می‌شوند. این کلاس درواقع پیاده‌سازی اینترفیس IResourceReader بوده که حاوی یک Enumerator که جفت داده‌های Key-Value از کلیدهای Resource را برمی‌گرداند، است.
تصویر زیر نمایی کلی از فرایند پیش‌فرض موردبحث را نشان می‌دهد:



این فرایند باتوجه به پیاده سازی نسبتا جامع آن، قابلیت بسیاری برای توسعه و سفارشی سازی دارد. بنابراین قبل از ادامه مبحث بهتر است، کلاس‌های اصلی این مدل بیشتر شرح داده شوند.

پیاده‌سازی‌ها

کلاس ResourceProviderFactory به صورت زیر تعریف شده است:

```
public abstract class ResourceProviderFactory
{
    public abstract IResourceProvider CreateGlobalResourceProvider(string classKey);
    public abstract IResourceProvider CreateLocalResourceProvider(string virtualPath);
}
```

همانطور که مشاهده می‌کنید دو متد برای تولید پرووایدرهای مخصوص منابع کلی و محلی در این کلاس وجود دارد. پرووایدر کلی تنها نیاز به نام کلید Resource برای یافتن داده موردنظر دارد. اما پرووایدر محلی به مسیر صفحه درخواستی برای اینکار نیاز دارد که با توجه به توضیحات ابتدای این مطلب کاملا بدیهی است.
پس از تولید پرووایدر موردنظر با استفاده از متد مناسب با توجه به شرایط شرح داده شده در بالا، نمونه تولیدشده از کلاس پرووایدر موردنظر وظیفه فراهم‌کردن کلیدهای Resource را برعهده دارد. پرووایدرهای موردبحث باید اینترفیس IResourceProvider را که به صورت زیر تعریف شده است، پیاده سازی کنند:

```
public interface IResourceProvider
{
    IResourceReader ResourceReader { get; }
    object GetObject(string resourceKey, CultureInfo culture);
}
```

همانطور که می‌بینید این پرووایدرها باید یک RsourceReader برای خواندن کلیدهای Resource فراهم کنند. همچنین یک متد با عنوان GetObject که کار اصلی برگرداندن داده ذخیره‌شده در ورودی موردنظر را برعهده دارد باید در این پرووایدرها پیاده‌سازی

شود. همانطور که قبلا اشاره شد، پیاده‌سازی پیش‌فرض این کلاس‌ها درنهایت نمونه‌ای از کلاس ResourceManager را برای یافتن مناسب‌ترین گزینه از بین کلیدهای موجود تولید می‌کند. این نمونه مورد بحث در متد GetObject مورد استفاده قرار می‌گیرد.

نکته: کدهای نشان‌داده‌شده در ادامه مطلب با استفاده از ابزار محبوب ReSharper استخراج شده‌اند. این ابزار برای دریافت این کدها معمولا از APIهای سایت SymbolSource.org استفاده می‌کند. البته منبع اصلی تمام کدهای دات نت فریمورک همان referencesource.microsoft.com است.

کلاس ResXResourceProviderFactory

پیاده‌سازی پیش‌فرض کلاس ResourceProviderFactory در ASP.NET که در کلاس ResXResourceProviderFactory قرار دارد، به صورت زیر است:

```
// Type: System.Web.Compilation.ResXResourceProviderFactory
// Assembly: System.Web, Version=4.0.0.0, Culture=neutral, PublicKeyToken=b03f5f7f11d50a3a
// Assembly location:
C:\Windows\Microsoft.NET\assembly\GAC_32\System.Web\v4.0_4.0.0.0__b03f5f7f11d50a3a\System.Web.dll

using System.Runtime;
using System.Web;
namespace System.Web.Compilation
{
  internal class ResXResourceProviderFactory : ResourceProviderFactory
  {
    [TargetedPatchingOptOut("Performance critical to inline this type of method across NGen image
boundaries")]
    public ResXResourceProviderFactory()    {    }
    public override IResourceProvider CreateGlobalResourceProvider(string classKey)
    {
      return (IResourceProvider) new GlobalResXResourceProvider(classKey);
    }
    public override IResourceProvider CreateLocalResourceProvider(string virtualPath)
    {
      return (IResourceProvider) new LocalResXResourceProvider(VirtualPath.Create(virtualPath));
    }
  }
}
```

در این کلاس برای تولید پرووایدر منابع محلی از کلاس VirtualPath استفاده شده است که امکاناتی جهت استخراج مسیرهای موردنظر با توجه به مسیر نسبی و مجازی ارائه‌شده فراهم می‌کند. متاسفانه این کلاس نیز با سطح دسترسی internal تعریف شده است و امکان استفاده مستقیم از آن وجود ندارد.

کلاس GlobalResXResourceProvider

پیاده‌سازی پیش‌فرض اینترفیس IResourceProvider در ASP.NET برای منابع کلی که در کلاس GlobalResXResourceProvider قرار دارد، به صورت زیر است:

```
internal class GlobalResXResourceProvider : BaseResXResourceProvider
{
  private string _classKey;
  internal GlobalResXResourceProvider(string classKey)
  {
    _classKey = classKey;
  }
  protected override ResourceManager CreateResourceManager()
  {
    string fullClassName = BaseResourcesBuildProvider.DefaultResourcesNamespace + "." + _classKey;
    // If there is no app resource assembly, return null
    if (BuildManager.AppResourcesAssembly == null)
      return null;
    ResourceManager resourceManager = new ResourceManager(fullClassName,
BuildManager.AppResourcesAssembly);
    resourceManager.IgnoreCase = true;
    return resourceManager;
  }
  public override IResourceReader ResourceReader
  {
    get
    {
```

```
      // App resources don't support implicit resources, so the IResourceReader should never be needed
      throw new NotSupportedException();
    }
  }
}
```

در این کلاس عملیات تولید نمونه مناسب از کلاس ResourceManager انجام می‌شود. مقدار BaseResourcesBuildProvider.DefaultResourcesNamespace به صورت زیر تعریف شده است:

```
internal const string DefaultResourcesNamespace = "Resources";
```

که قبلا هم درباره این مقدار پیش فرض اشاره‌ای شده بود.
پارامتر classKey درواقع اشاره به نام فایل اصلی منبع کلی دارد. مثلا اگر این مقدار برابر Resource1 باشد، کلاس ResourceManager برای نوع داده Resources.Resource1 تولید خواهد شد.
هم‌چنین اسمبلی موردنظر برای یافتن ورودی‌های منابع کلی که از BuildManager.AppResourcesAssembly دریافت شده است، به صورت پیش فرض هم‌نام با مسیر منابع کلی و با عنوان App_GlobalResources تولید می‌شود.
کلاس **BuildManager** فرایندهای کامپایل کدها و صفحات برای تولید اسمبلی‌ها و نگهداری از آن‌ها در حافظه را مدیریت می‌کند. این کلاس که محتوای نسبتا مفصلی دارد (نزدیک به 2000 خط کد) به صورت public و sealed تعریف شده است. بنابراین با ریفرنس دادن اسمبلی System.Web در فضای نام System.Web.Compilation در دسترس است، اما نمی‌توان کلاسی از آن مشتق کرد. BuildManager حاوی تعداد زیادی اعضای استاتیک برای دسترسی به اطلاعات اسمبلی‌هاست. اما متاسفانه بیشتر آن‌ها سطح دسترسی عمومی ندارند.

نکته: همانطور که در بالا نیز اشاره شد، ازآنجاکه کلاس ResourceReader در اینجا تنها برای عبارات بومی سازی ضمنی کاربرد دارد، و نیز عبارات بومی‌سازی ضمنی تنها برای منابع محلی کاربرد دارند، در این کلاس برای خاصیت مربوطه در پیاده سازی اینترفیس IResourceProvider یک خطای عدم پشتیبانی (NotSupportedException) صادر شده است.

کلاس LocalResXResourceProvider

پیاده‌سازی پیش‌فرض اینترفیس IResourceProvider در ASP.NET برای منابع محلی که در کلاس LocalResXResourceProvider قرار دارد، به صورت زیر است:

```
internal class LocalResXResourceProvider : BaseResXResourceProvider
{
  private VirtualPath _virtualPath;
  internal LocalResXResourceProvider(VirtualPath virtualPath)
  {
    _virtualPath = virtualPath;
  }
  protected override ResourceManager CreateResourceManager()
  {
    ResourceManager resourceManager = null;
    Assembly pageResAssembly = GetLocalResourceAssembly();
    if (pageResAssembly != null)
    {
      string fileName = _virtualPath.FileName;
      resourceManager = new ResourceManager(fileName, pageResAssembly);
      resourceManager.IgnoreCase = true;
    }
    else
    {
      throw new
InvalidOperationException(SR.GetString(SR.ResourceExpresionBuilder_PageResourceNotFound));
    }
    return resourceManager;
  }
  public override IResourceReader ResourceReader
  {
    get
    {
      // Get the local resource assembly for this page
      Assembly pageResAssembly = GetLocalResourceAssembly();
```

```
      if (pageResAssembly == null) return null;
      // Get the name of the embedded .resource file for this page
      string resourceFileName = _virtualPath.FileName + ".resources";
      // Make it lower case, since GetManifestResourceStream is case sensitive
      resourceFileName = resourceFileName.ToLower(CultureInfo.InvariantCulture);
      // Get the resource stream from the resource assembly
      Stream resourceStream = pageResAssembly.GetManifestResourceStream(resourceFileName);
      // If this page has no resources, return null
      if (resourceStream == null) return null;
      return new ResourceReader(resourceStream);
    }
  }
  [PermissionSet(SecurityAction.Assert, Unrestricted = true)]
  private Assembly GetLocalResourceAssembly()
  {
    // Remove the page file name to get its directory
    VirtualPath virtualDir = _virtualPath.Parent;
    // Get the name of the local resource assembly
    string cacheKey = BuildManager.GetLocalResourcesAssemblyName(virtualDir);
    BuildResult result = BuildManager.GetBuildResultFromCache(cacheKey);
    if (result != null)
    {
      return ((BuildResultCompiledAssembly)result).ResultAssembly;
    }
    return null;
  }
}
```

عملیات موجود در این کلاس باتوجه به فرایندهای مربوط به یافتن اسمبلی مربوطه با استفاده از مسیر ارائه‌شده، کمی پیچیده‌تر از کلاس قبلی است.

در متد GetLocalResourceAssembly عملیات یافتن اسمبلی متناظر با درخواست جاری انجام می‌شود. اینکار باتوجه به نحوه نامگذاری اسمبلی منابع محلی که در ابتدای این مطلب اشاره شد انجام می‌شود. مثلا اگر صفحه درخواستی در مسیر ~/SubDir1/Page1.aspx باشد، در این متد با استفاده از ابزارهای موجود عنوان اسمبلی نهایی برای این مسیر که به صورت App_LocalResources.SubDir1.XXXXX است تولید و درنهایت اسمبلی مربوطه استخراج می‌شود.

درضمن در اینجا هم کلاس ResourceManager برای نوع داده متناظر با نام فایل اصلی منبع محلی تولید می‌شود. مثلا برای مسیر مجازی ~/SubDir1/Page1.aspx نوع داده‌ای با نام Page1.aspx درنظر گرفته خواهد شد (با توجه به نام فایل منبع محلی که باید به صورت Page1.aspx.resx باشد. در قسمت قبل در این باره شرح داده شده است).

نکته: کلاس **SR** (مخفف String Resources) که در فضای نام System.Web قرار دارد، حاوی عناوین کلیدهای Resourceهای مورداستفاده در اسمبلی System.Web است. این کلاس با سطح دسترسی internal و به صورت sealed تعریف شده است. عنوان تمامی کلیدها به صورت ثوابتی از نوع رشته تعریف شده‌اند.

SR درواقع یک Wrapper بر روی کلاس ResourceManager است تا از تکرار عناوین کلیدهای منابع که از نوع رشته هستند، در جاهای مختلف برنامه جلوگیری شود. کار این کلاس مشابه کاری است که کتابخانه T4MVC برای نگهداری عناوین کنترلرها و اکشن‌ها به صورت رشته‌های ثابت انجام می‌دهد. از این روش در جای جای دات نت فریمورک برای نگهداری رشته‌های ثابت استفاده شده است!

نکته: باتوجه به استفاده از عبارات بومی‌سازی ضمنی در استفاده از ورودی‌های منابع محلی، خاصیت ResourceReader در این کلاس نمونه‌ای متناظر برای درخواست جاری از کلاس ResourceReader با استفاده از Stream استخراج شده از اسمبلی یافته شده، تولید می‌کند.

کلاس پایه BaseResXResourceProvider

کلاس پایه BaseResXResourceProvider که در دو پیاده‌سازی نشان داده شده در بالا استفاده شده است (هر دو کلاس از این کلاس مشتق شده‌اند)، به صورت زیر است:

```
internal abstract class BaseResXResourceProvider : IResourceProvider
{
  private ResourceManager _resourceManager;
  ///// IResourceProvider implementation
  public virtual object GetObject(string resourceKey, CultureInfo culture)
  {
```

```
    // Attempt to get the resource manager
    EnsureResourceManager();
    // If we couldn't get a resource manager, return null
    if (_resourceManager == null) return null;
    if (culture == null) culture = CultureInfo.CurrentUICulture;
    return _resourceManager.GetObject(resourceKey, culture);
  }
  public virtual IResourceReader ResourceReader { get { return null; } }
  ///// End of IResourceProvider implementation
  protected abstract ResourceManager CreateResourceManager();
  private void EnsureResourceManager()
  {
    if (_resourceManager != null)  return;
    _resourceManager = CreateResourceManager();
  }
}
```

در این کلاس پیاده‌سازی اصلی اینترفیس IResourceProvider انجام شده است. همانطور که می‌بینید کار نهایی استخراج ورودی‌های منابع در متد GetObject با استفاده از نمونه فراهم شده از کلاس ResourceManager انجام می‌شود.

نکته: دقت کنید که در کد بالا درصورت فراهم نکردن مقداری برای کالچر، از کالچر UI در ثرد جاری (CultureInfo.CurrentUICulture) به عنوان مقدار پیش‌فرض استفاده می‌شود.

کلاس ResourceManager

در زمان اجرا ASP.NET کلید مربوط به منبع موردنظر را با استفاده از کالچر جاری UI انتخاب می‌کند. در قسمت اول این سری مطالب شرح کوتاهی بابت انواع کالچرها داده شد، اما برای توضیحات کاملتر به اینجا مراجعه کنید.
در ASP.NET به صورت پیش‌فرض تمام منابع در زمان اجرا از طریق نمونه‌ای از کلاس ResourceManager در دسترس خواهند بود. به ازای هر نوع Resource که درخواستی برای یک کلید آن ارسال می‌شود یک نمونه از کلاس ResourceManager ساخته می‌شود. در این هنگام (یعنی پس از اولین درخواست به کلیدهای یک منبع) اسمبلی ستلایت مناسب آن پس از یافته شدن (یا تولیدشدن در زمان اجرا) به دامین ASP.NET جاری بارگذاری می‌شود و تا زمانیکه این دامین Unload نشود در حافظه سرور باقی خواهد ماند.

نکته: کلاس ResourceManager **تنها** توانایی استخراج کلیدهای Resource از اسمبلی‌های ستلایتی (فایل‌های resources. که در قسمت اول به آن‌ها اشاره شد) که در AppDomain جاری بارگذاری شده‌اند را دارد.

کلاس ResourceManager به صورت زیر نمونه سازی می‌شود:

```
System.Resources.ResourceManager(string baseName, Assembly assemblyName)
```

پارامتر baseName به نام کامل ریشه اسمبلی اصلی موردنظر(با فضای نام و ...) اما بدون پسوند اسمبلی مربوطه (.resources) اشاره دارد. این نام که برابر نام کلاس نهایی تولیدشده برای منبع موردنظر است همنام با فایل اصلی و پیش‌فرض منبع (فایلی که حاوی عنوان هیچ زبان و کالچری نیست) تولید می‌شود. مثلا برای اسمبلی ستلایت با عنوان MyApplication.MyResource.fa-IR.resources باید از عبارت MyApplication.MyResource استفاده شود.
پارامتر assemblyName نیز به اسمبلی حاوی اسمبلی ستلایت اصلی اشاره دارد. درواقع همان اسمبلی اصلی که نوع داده مربوط به فایل منبع اصلی درون آن embed شده است.
مثلا:

```
var manager = new System.Resources.ResourceManager("Resources.Resource1", typeof(Resource1).Assembly)
```

یا

```
var manager = new System.Resources.ResourceManager("Resources.Resource1",
Assembly.LoadFile(@"c:\MyResources\MyGlobalResources.dll"))
```

روش دیگری نیز برای تولید نمونه‌ای از این کلاس وجود دارد که با استفاده از متد استاتیک زیر که در خود کلاس ResourceManager تعریف شده است انجام می‌شود:

```
public static ResourceManager CreateFileBasedResourceManager(string baseName, string resourceDir, Type
usingResourceSet)
```

در این متد کار استخراج ورودی‌های منابع مستقیما از فایل‌های resources. انجام می‌شود. در اینجا baseName نام فایل اصلی منبع بدون پیشوند resources. است. resourceDir نیز مسیری است که فایل‌های resources. در آن قرار دارند. usingResourceSet نیز نوع کلاس سفارشی سازی شده از ResourceSet برای استفاده به جای کلاس پیش‌فرض است که معمولا مقدار null برای آن وارد می‌شود تا از همان کلاس پیش‌فرض استفاده شود (چون برای بیشتر نیازها همین کلاس پیش‌فرض کفایت می‌کند).
نکته: برای تولید فایل resources. از یک فایل resx. میتوان از ابزار resgen همانند زیر استفاده کرد:

```
resgen d:\MyResources\MyResource.fa.resx
```

نکته: عملیاتی که درون کلاس ResourceManager انجام می‌شود پیچیده‌تر از آن است که به نظر می‌آید. این عملیات شامل فرایندهای بسیاری شامل بارگذاری کلیدهای مختلف یافته شده و مدیریت ذخیره موقت آن‌ها در حافظه (کش)، کنترل و مدیریت انواع Resource Setها، و مهمتر از همه مدیریت عملیات Fallback و ... که در نهایت شامل هزاران خط کد است که با یک جستجوی ساده قابل مشاهده و بررسی است (^).

نمونه‌سازی مناسب از ResourceManager

در کدهای نشان داده شده در بالا برای پیاده‌سازی پیش‌فرض در ASP.NET، مهمترین نکته همان تولید نمونه مناسب از کلاس ResourceManager است. پس از آماده شدن این کلاس عملیات استخراج ورودی‌های منابع براحتی و با مدیریت کامل انجام می‌شود. اما ازآنجاکه تقریبا تمامی APIهای موردنیاز با سطح دسترسی internal تعریف شده‌اند، متاسفانه تهیه و تولید این نمونه مناسب خارج از اسمبلی System.Web به صورت مستقیم وجود ندارد.
درهرصورت، برای آشنایی بیشتر با فرایند نشان داده شده، تولید این نمونه مناسب و استفاده مستقیم از آن می‌تواند مفید و نیز جالب باشد. پس از کمی تحقیق و با استفاده از Reflection به کدهای زیر رسیدم:

```
private ResourceManager CreateGlobalResourceManager(string classKey)
{
  var baseName = "Resources." + classKey;
  var buildManagerType = typeof(BuildManager);
  var property = buildManagerType.GetProperty("AppResourcesAssembly", BindingFlags.Static |
BindingFlags.NonPublic | BindingFlags.GetField);
  var appResourcesAssembly = (Assembly)property.GetValue(null, null);
  return new ResourceManager(baseName, appResourcesAssembly) { IgnoreCase = true };
}
```

تنها نکته کد فوق دسترسی به اسمبلی منابع کلی در خاصیت AppResourcesAssembly از کلاس BuildManager با استفاده از BindingFlagهای نشان داده شده است. نحوه استفاده از این متد هم به صورت زیر است:

```
var manager = CreateGlobalResourceManager("Resource1");
Label1.Text = manager.GetString("String1");
```

اما برای منابع محلی کار کمی پیچیده‌تر است. کد مربوط به تولید نمونه مناسب از ResourceManager برای منابع محلی به صورت زیر خواهد بود:

```
private ResourceManager CreateLocalResourceManager(string virtualPath)
{
  var virtualPathType = typeof(BuildManager).Assembly.GetType("System.Web.VirtualPath", true);
  var virtualPathInstance = Activator.CreateInstance(virtualPathType, BindingFlags.NonPublic |
BindingFlags.Instance, null, new object[] { virtualPath }, CultureInfo.InvariantCulture);
  var buildResultCompiledAssemblyType =
```

```
typeof(BuildManager).Assembly.GetType("System.Web.Compilation.BuildResultCompiledAssembly", true);
  var propertyResultAssembly = buildResultCompiledAssemblyType.GetProperty("ResultAssembly",
BindingFlags.NonPublic | BindingFlags.Instance);
  var methodGetLocalResourcesAssemblyName =
typeof(BuildManager).GetMethod("GetLocalResourcesAssemblyName", BindingFlags.NonPublic |
BindingFlags.Static);
  var methodGetBuildResultFromCache = typeof(BuildManager).GetMethod("GetBuildResultFromCache",
BindingFlags.NonPublic | BindingFlags.Static, null, new Type[] { typeof(string) }, null);

  var fileNameProperty = virtualPathType.GetProperty("FileName");
  var virtualPathFileName = (string)fileNameProperty.GetValue(virtualPathInstance, null);

  var parentProperty = virtualPathType.GetProperty("Parent");
  var virtualPathParent = parentProperty.GetValue(virtualPathInstance, null);

  var localResourceAssemblyName = (string)methodGetLocalResourcesAssemblyName.Invoke(null, new object[]
{ virtualPathParent });
  var buildResultFromCache = methodGetBuildResultFromCache.Invoke(null, new object[] {
localResourceAssemblyName });
  Assembly localResourceAssembly = null;
  if (buildResultFromCache != null)
    localResourceAssembly = (Assembly)propertyResultAssembly.GetValue(buildResultFromCache, null);

  if (localResourceAssembly == null)
    throw new InvalidOperationException("Unable to find the matching resource file.");

  return new ResourceManager(virtualPathFileName, localResourceAssembly) { IgnoreCase = true };
}
```

ازجمله نکات مهم این متد تولید یک نمونه از کلاس VirtualPath برای Parse کردن مسیر مجازی واردشده برای صفحه درخواستی است. از این کلاس برای بدست آوردن نام فایل منبع محلی به همراه مسیر فولدر مربوطه جهت استخراج اسمبلی متناظر استفاده میشود.
نکته مهم دیگر این کد دسترسی به متد GetLocalResourcesAssemblyName از کلاس BuildManager است که با استفاده از مسیر فولدر مربوط به صفحه درخواستی نام اسمبلی منبع محلی مربوطه را برمی‌گرداند.
درنهایت با استفاده از متد GetBuildResultFromCache از کلاس BuildManager اسمبلی موردنظر بدست می‌آید. همانطور که از نام این متد برمی‌آید این اسمبلی از کش خوانده می‌شود. البته مدیریت این اسمبلی‌ها کاملا توسط BuildManager و سایر ابزارهای موجود در ASP.NET انجام خواهد شد.

نحوه استفاده از متد فوق نیز به صورت زیر است:

```
var manager = CreateLocalResourceManager("~/Default.aspx");
Label1.Text = manager.GetString("Label1.Text");
```

نکته: ارائه و شرح کدهای پیاده‌سازی‌های پیش‌فرض برای آشنایی با نحوه صحیح سفارشی سازی این کلاس‌ها آورده شده است. پس با دقت بیشتر بر روی این کدها سعی کنید نحوه پیاده‌سازی مناسب را برای سفارشی‌سازی موردنظر خود پیدا کنید.

تا اینجا با مقدمات فرایند تولید پرووایدرهای سفارشی برای استفاده در فرایند بارگذاری ورودی‌های Resourceها آشنا شدیم. در ادامه به بحث تولید پرووایدرهای سفارشی برای استفاده از دیگر انواع منابع (به غیر از فایل‌های resx.) خواهم پرداخت.

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://www.codeproject.com/Articles/104667/Under-the-Hood-of-BuildManager-and-Resource-Handli
http://www.onpreinit.com/2009/06/updatable-aspnet-resx-resource-provider.html http://msdn.microsoft.com/en-us/library/h6270d0z(v=vs.100).aspx
http://msdn.microsoft.com/en-us/library/system.web.compilation.resourceproviderfactory.aspx