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

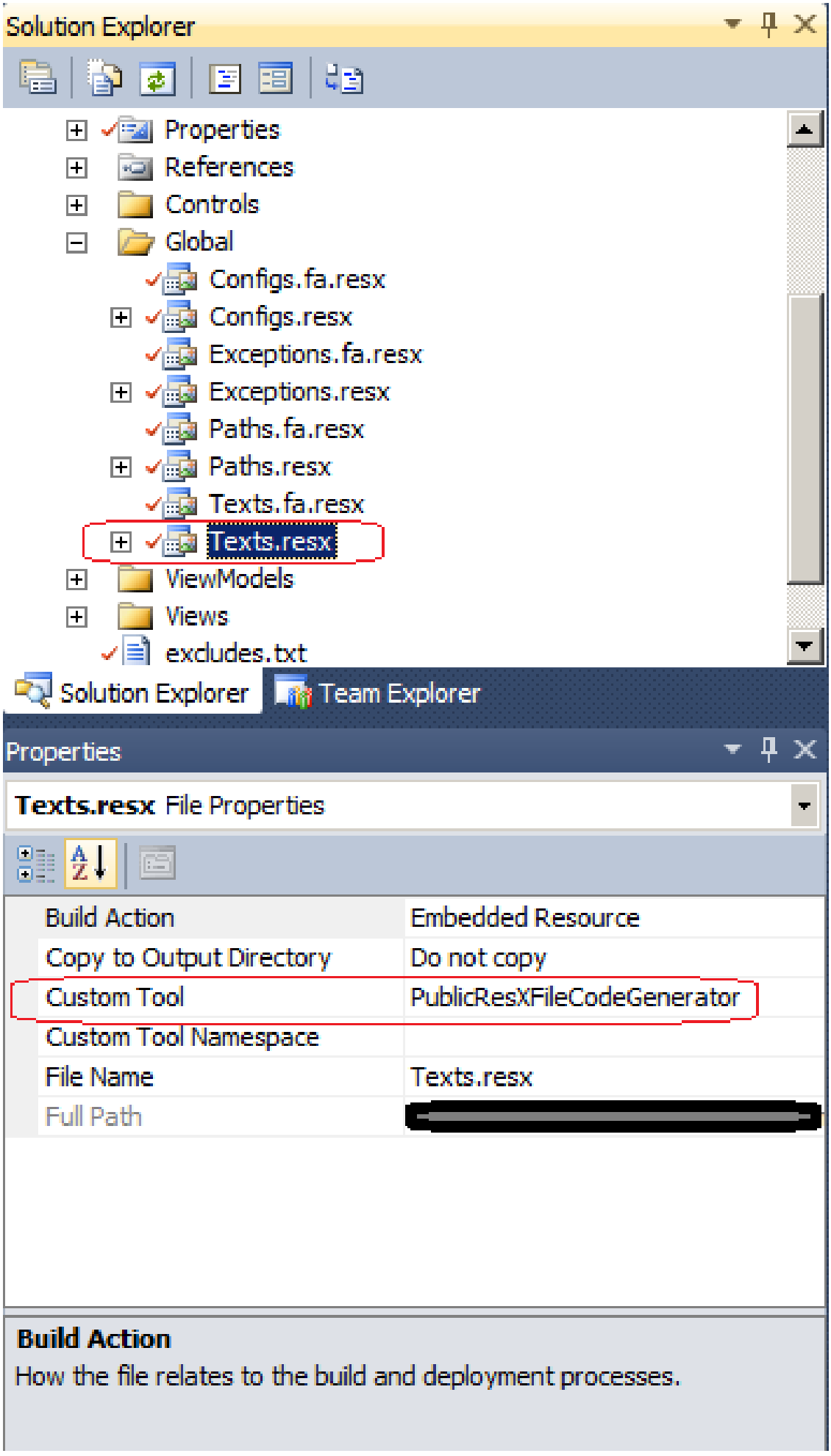
البته روش سومی هم که از ترکیب این دو روش استفاده میکند نیز وجود دارد. انتخاب روش مناسب کمی به سلیقه‌ها و عادات برنامه نویسی بستگی دارد. اگر فکر میکنید که استفاده از ویوهای مختلف به دلیل جداسازی مفاهیم درگیر در کالچرها (مثل جانمایی اجزای مختلف ویوها یا بحث Direction) باعث مدیریت بهتر و کاهش هزینه‌های 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

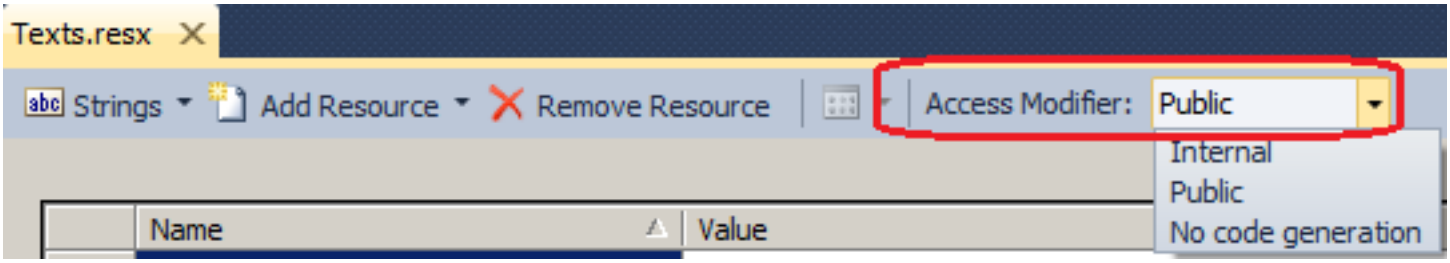
بهترین مکان برای نگهداری فایلهای Resource در یک پروژه جداگانه است. در پروژه‌های از نوع وب‌سایت پوشه‌هایی با نام App_GlobalResources یا App_LocalResources وجود دارد که میتوان از آنها برای نگهداری و مدیریت این نوع فایلها استفاده کرد. اما همانطور که در اینجا توضیح داده شده است این روش مناسب نیست. بنابراین 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 مخنلف 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

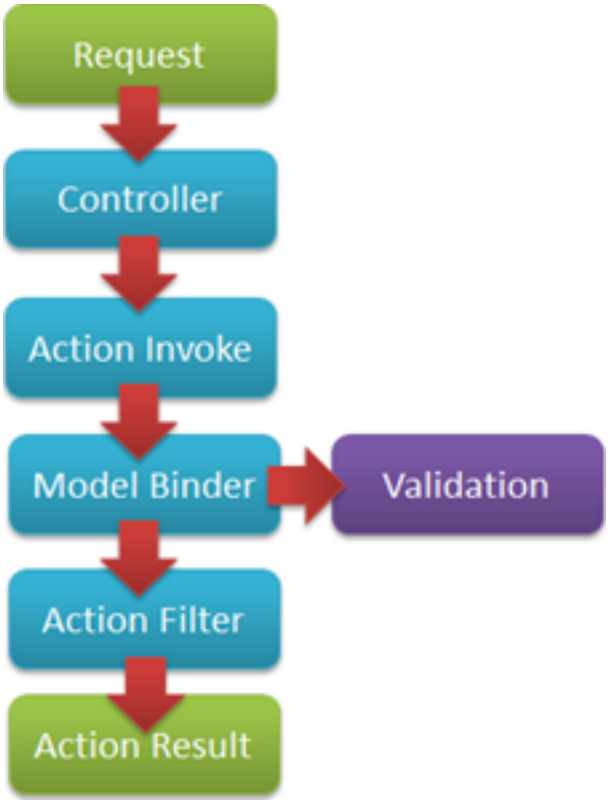
اینکار باید خاصیت CurrentUICulture در ثرد جاری کنترل و مدیریت شود. یکی از مکانهایی که برای نگهداری زبان جاری استفاده میشود کوکی است. معمولا برای اینکار از کوکی‌های دارای تاریخ انقضای طولانی استفاده میشود. میتوان از تنظیمات موجود در فایل کانفیگ برای ذخیره زبان پیش فرض سیستم نیز استفاه کرد.
روشی که معمولا 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:

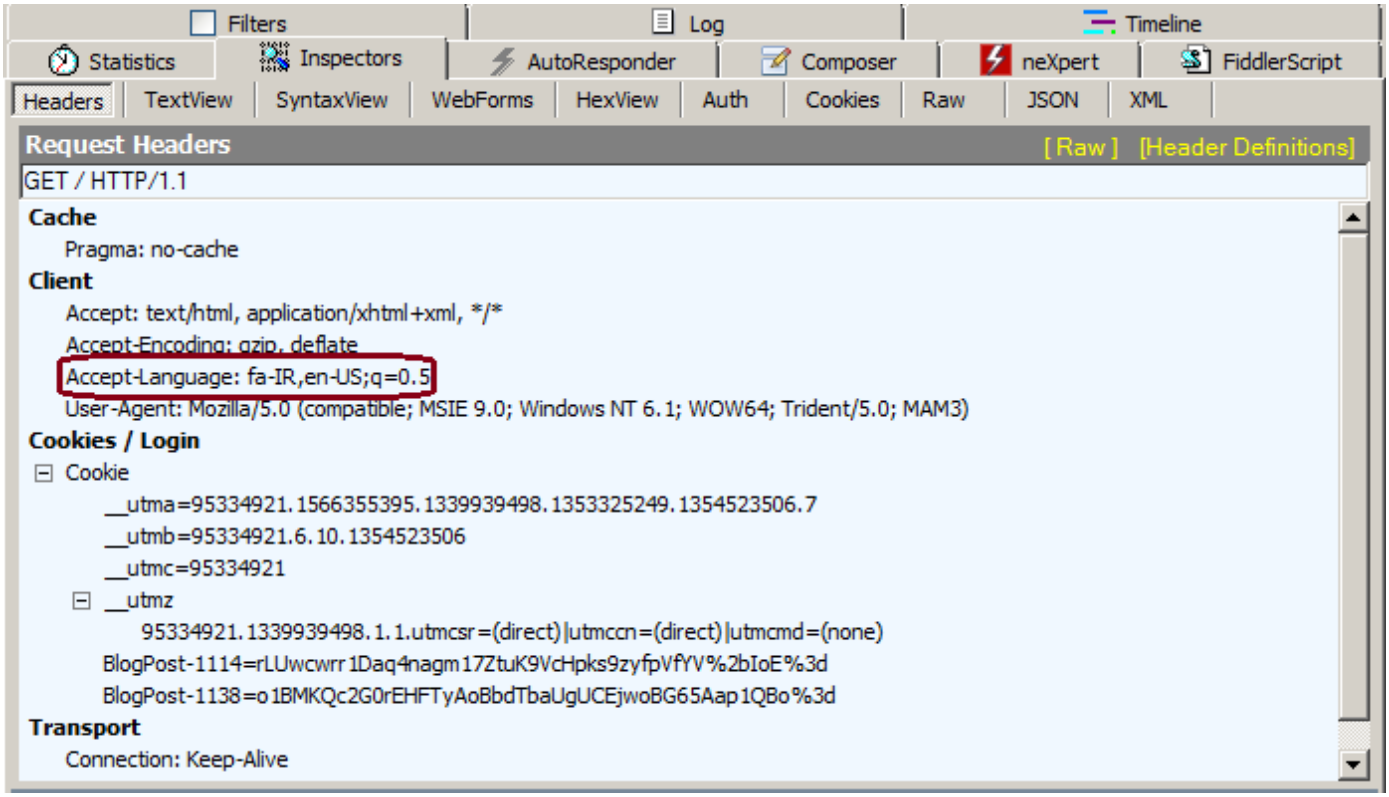
```
<select id="langs" onchange="languageChanged()">
  <option value="fa-IR">فارسی</option>
  <option value="en-US">انگلیسی</option>
</select>
<script type="text/javascript">
  function languageChanged() {
    setCookie("MyLanguageCookieName", $('#langs').val(), 365);
    window.location.reload();
  }
  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:

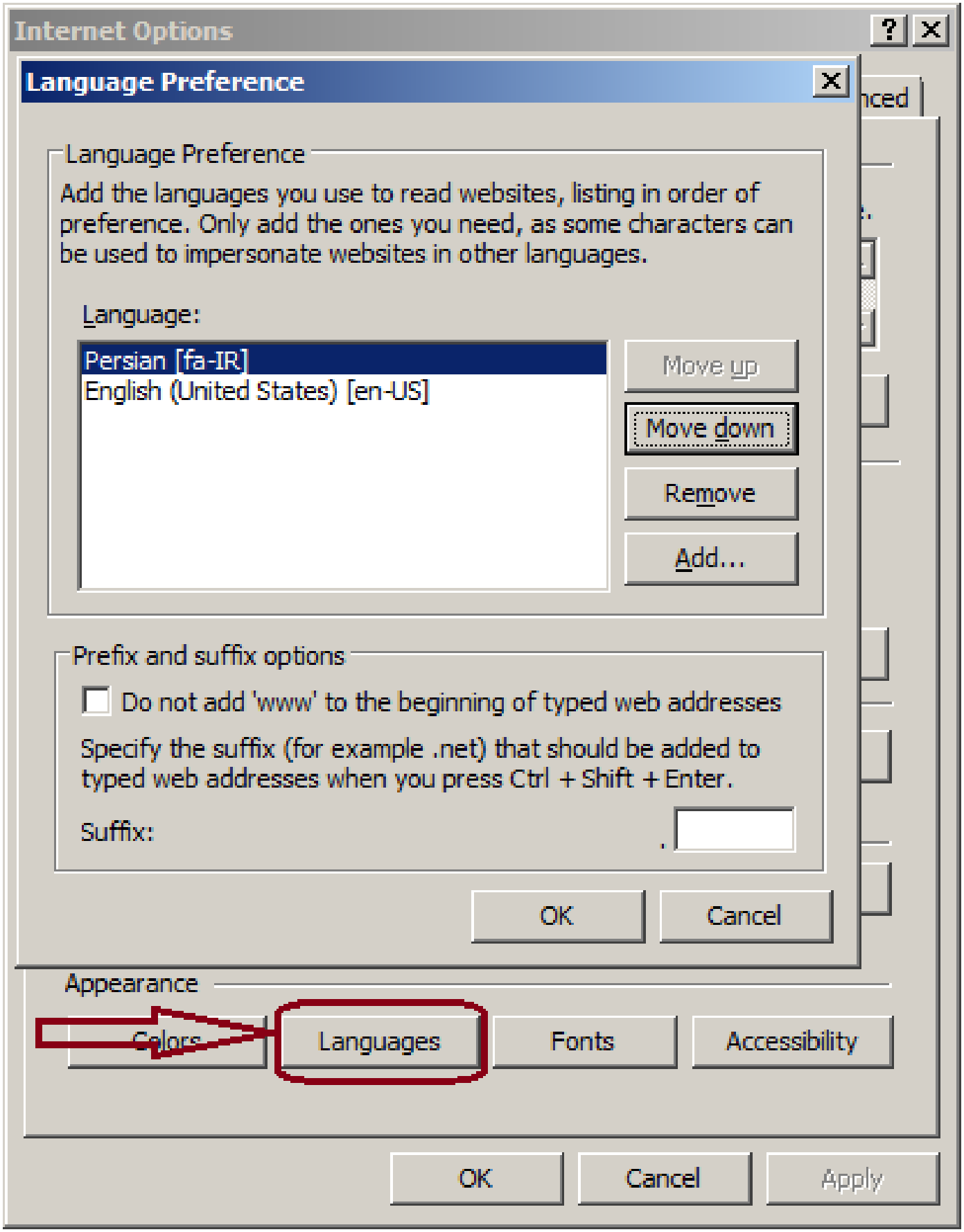



**نکته:** پارامتر q در عبارت مشخص شده در تصویر فوق relative quality factor نام دارد و به نوعی مشخص کننده اولویت زبان مربوطه است. مقدار آن بین 0 و 1 است و مقدار پیش فرض آن 1 است. هرچه مقدار این پارامتر بیشتر باشد زبان مربوطه اولویت

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();
  }
}
```

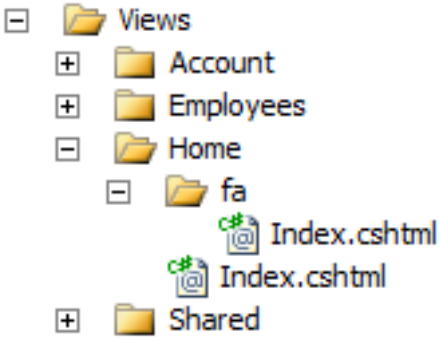
این کار را در یک اکشن فیلتر هم میتوان انجام داد اما با توجه به توضیحاتی که در قسمت قبل داده شد استفاده از اکشن فیلتر برای تعیین زبان جاری کار مناسبی نیست.

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

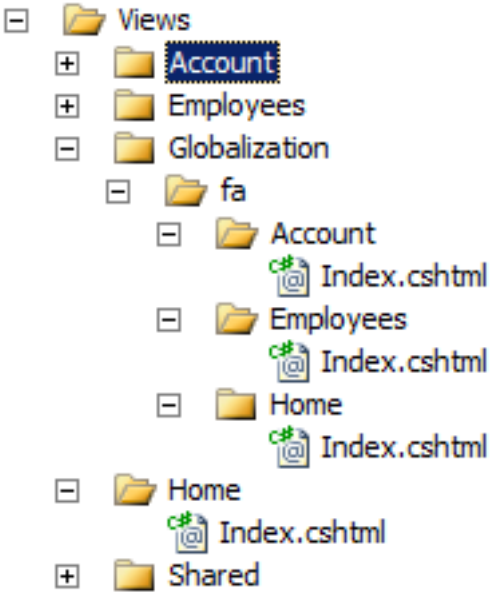
**استفاده از ویوهای جداگانه برای زبانهای مختلف**

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

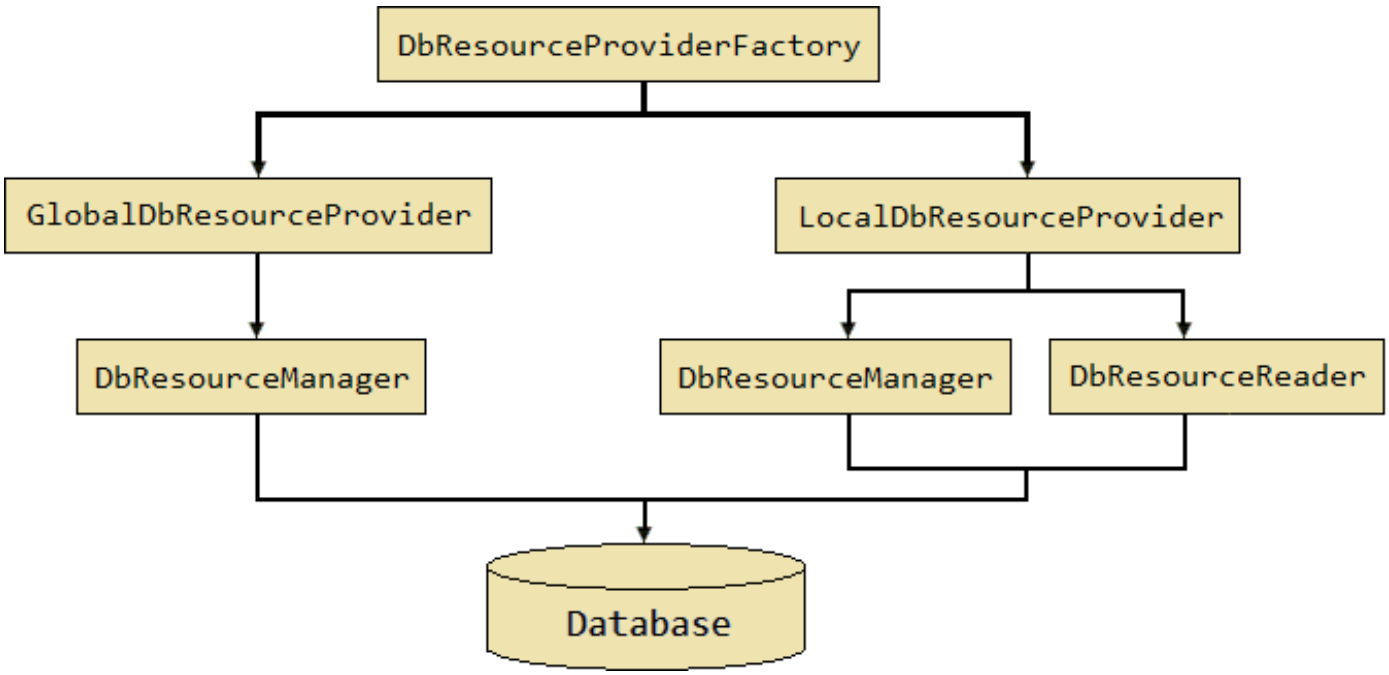
با سلام
ضمن تشکر از مطلب بسیار خوبتون
خاصیت DisplayFormat قابلیت استفاده از کلیدهای Resource را ندارد !
لطفا راهنمایی فرمایید که چطور میشه از این خاصیت برای DisplayFormat استفاده کرد؟
من می‌خوام برای تاریخ در زبانه فارسی از فرمت {yyyy-MM-dd} و در زبانه انگلیسی از {yyyy-dd-MM} استفاده کنم.
با سپاس فراوان

عنوان: **Globalization در ASP.NET MVC - قسمت پنجم**
نویسنده: یوسف نژاد
تاریخ: ۱۳۹۲/۰۳/۰۴ ۱۳:۵۰
آدرس: www.dotnettips.info
برچسب‌ها: Localization, Globalization, Internationalization, Resource, ResourceProviderFactory, Custom Resource Provider, Database Resource Provider

در قسمت قبل راجع به مدل پیش‌فرض پرووایدر منابع در ASP.NET بحث نسبتا مفصلی شد. در این قسمت تولید یک پرووایدر سفارشی برای استفاده از دیتابیس به جای فایل‌های resx. به عنوان منبع نگهداری داده‌ها بحث می‌شود.
قبلا هم اشاره شده بود که در پروژه‌های بزرگ ذخیره تمام ورودی‌های منابع درون فایل‌های resx. بازدهی مناسبی نخواهد داشت. هم‌چنین به مرور زمان و با افزایش تعداد این فایل‌ها، کار مدیریت آن‌ها بسیار دشوار و طاقت‌فرسا خواهد شد. درضمن به‌دلیل رفتار سیستم کشینگ این منابع در ASP.NET، که محتویات کل یک فایل را بلافاصله پس از اولین درخواست یکی از ورودی‌های آن در حافظه سرور کش می‌کند، در صورت وجود تعداد زیادی فایل منبع و با ورودی‌های بسیار، با گذشت زمان بازدهی کلی سایت به شدت تحت تاثیر قرار خواهد گرفت.
بنابراین استفاده از یک منبع مثل دیتابیس برای چنین شرایطی و نیز کنترل مدیریت دسترسی به ورودی‌های آن به صورت سفارشی، می‌تواند به بازدهی بهتر برنامه کمک زیادی کند. درضمن فرایند به‌روزرسانی مقادیر این ورودی‌ها در صورت استفاده از یک دیتابیس می‌تواند ساده‌تر از حالت استفاده از فایل‌های resx. انجام شود.

### تولید یک پرووایدر منابع دیتابیسی - بخش اول

در بخش اول این مطلب با نحوه پیاده‌سازی کلاس‌های اصلی و اولیه موردنیاز آشنا خواهیم شد. مفاهیم پیشرفته‌تر (مثل کش‌کردن ورودی‌ها و عملیات fallback) و نیز ساختار مناسب جدول یا جداول موردنیاز در دیتابیس و نحوه ذخیره ورودی‌ها برای انواع منابع در دیتابیس در مطلب بعدی آورده می‌شود.
با توجه به توضیحاتی که در قسمت قبل داده شد، می‌توان از طرح اولیه‌ای به صورت زیر برای سفارشی‌سازی یک پرووایدر منابع دیتابیسی استفاده کرد:




اگر مطالب قسمت قبل را خوب مطالعه کرده باشید، پیاده سازی اولیه طرح بالا نباید کار سختی باشد. در ادامه یک نمونه از

پیاده‌سازی‌های ممکن نشان داده شده است.
برای آغاز کار ابتدا یک پروژه ClassLibrary جدید مثلا با نام DbResourceProvider ایجاد کنید و ریفرنسی از اسمبلی System.Web به این پروژه اضافه کنید. سپس کلاس‌هایی که در ادامه شرح داده شده‌اند را به آن اضافه کنید.

**کلاس DbResourceProviderFactory**

همه چیز از یک ResourceProviderFactory شروع می‌شود. نسخه سفارشی نشان داده شده در زیر برای منابع محلی و کلی از کلاس‌های پرووایدر سفارشی استفاده می‌کند که در ادامه آورده شده‌اند.

```
using System.Web.Compilation;
namespace DbResourceProvider
{
  public class DbResourceProviderFactory : ResourceProviderFactory
  {
    #region Overrides of ResourceProviderFactory
    public override IResourceProvider CreateGlobalResourceProvider(string classKey)
    {
      return new GlobalDbResourceProvider(classKey);
    }
    public override IResourceProvider CreateLocalResourceProvider(string virtualPath)
    {
      return new LocalDbResourceProvider(virtualPath);
    }
    #endregion
  }
}
```

درباره اعضای کلاس ResourceProviderFactory در قسمت قبل توضیحاتی داده شد. در نمونه سفارشی بالا دو متد این کلاس برای برگرداندن پرووایدرهای سفارشی منابع محلی و کلی بازنویسی شده‌اند. سعی شده است تا نمونه‌های سفارشی در اینجا رفتاری همانند نمونه‌های پیش‌فرض در ASP.NET داشته باشند، بنابراین برای پرووایدر منابع کلی (GlobalDbResourceProvider) نام منبع درخواستی (className) و برای پرووایدر منابع محلی (LocalDbResourceProvider) مسیر مجازی درخواستی (virtualPath) به عنوان پارامتر کانستراکتور ارسال می‌شود.
**نکته:** برای استفاده از این کلاس به جای کلاس پیش‌فرض ASP.NET باید یکسری تنظیمات در فایل کانفیگ برنامه مقصد اعمال کرد که در ادامه آورده شده است.

**کلاس BaseDbResourceProvider**

برای پیاده‌سازی راحت‌تر کلاس‌های موردنظر، بخش‌های مشترک بین دو پرووایدر محلی و کلی در یک کلاس پایه به صورت زیر قرار داده شده است. این طرح دقیقا مشابه نمونه پیش‌فرض ASP.NET است.

```
using System.Globalization;
using System.Resources;
using System.Web.Compilation;
namespace DbResourceProvider
{
  public abstract class BaseDbResourceProvider : IResourceProvider
  {
    private DbResourceManager _resourceManager;
    protected abstract DbResourceManager CreateResourceManager();
    private void EnsureResourceManager()
    {
      if (_resourceManager != null) return;
      _resourceManager = CreateResourceManager();
    }
    #region Implementation of IResourceProvider
    public object GetObject(string resourceKey, CultureInfo culture)
    {
      EnsureResourceManager();
      if (_resourceManager == null) return null;
      if (culture == null) culture = CultureInfo.CurrentUICulture;
      return _resourceManager.GetObject(resourceKey, culture);
    }
    public virtual IResourceReader ResourceReader { get { return null; } }
    #endregion
  }
}
```

کلاس بالا چون یک کلاس صرفا پایه است بنابراین به صورت abstract تعریف شده است. در این کلاس، از نمونه سفارشی DbResourceManager برای بازیابی داده‌ها از دیتابیس استفاده شده است که در ادامه شرح داده شده است.
در اینجا، از متد CreateResourceManager برای تولید نمونه مناسب از کلاس DbResourceManager استفاده می‌شود. این متد به صورت abstract و protected تعریف شده است بنابراین پیاده‌سازی آن باید در کلاس‌های مشتق شده که در ادامه آورده شده‌اند انجام شود.
در متد EnsureResourceManager کار بررسی نال نبودن resouceManager_ انجام می‌شود تا درصورت نال بودن آن، بلافاصله نمونه‌ای تولید شود.

**نکته:** ازآنجاکه نقطه آغازین فرایند یعنی تولید نمونه‌ای از کلاس DbResourceProviderFactory توسط خود ASP.NET انجام خواهد شد، بنابراین مدیریت تمام نمونه‌های ساخته شده از کلاس‌هایی که در این مطلب شرح داده می‌شوند درنهایت عملا برعهده ASP.NET است. در ASP.NET درطول عمر یک برنامه تنها یک نمونه از کلاس Factory تولید خواهد شد، و متدهای موجود در آن در حالت عادی تنها یکبار به ازای هر منبع درخواستی (کلی یا محلی) فراخوانی می‌شوند. درنتیجه به ازای هر منبع درخواستی (کلی یا محلی) هر یک از کلاس‌های پرووایدر منابع تنها یک‌بار نمونه‌سازی خواهد شد. بنابراین بررسی نال نبودن این متغیر و تولید نمونه‌ای جدید تنها در صورت نال بودن آن، کاری منطقی است. این نمونه بعدا توسط ASP.NET به ازای هر منبع یا صفحه درخواستی کش می‌شود تا در درخواست‌های بعدی تنها از این نسخه کش‌شده استفاده شود.

در متد GetObject نیز کار استخراج ورودی منابع انجام می‌شود. ابتدا با استفاده از متد EnsureResourceManager از وجود نمونه‌ای از کلاس DbResourceManager اطمینان حاصل می‌شود. سپس درصورتی‌که مقدار این کلاس همچنان نال باشد مقدار نال برگشت داده می‌شود. این حالت وقتی پیش می‌آید که نتوان با استفاده از داده‌های موجود نمونه‌ای مناسب از کلاس DbResourceManager تولید کرد.
سپس مقدار کالچر ورودی بررسی می‌شود و درصورتی‌که نال باشد مقدار کالچر UI ثرد جاری که در CultureInfo.CurrentUICulture قرار دارد برای آن درنظر گرفته می‌شود. درنهایت با فراخوانی متد GetObject از DbResourceManager تولیدی برای کلید و کالچر مربوطه کار استخراج ورودی درخواستی پایان می‌پذیرد.
پراپرتی ResourceReader در این کلاس به صورت virtual تعریف شده است تا بتوان پیاده‌سازی مناسب آن را در هر یک از کلاس‌های مشتق‌شده اعمال کرد. فعلا برای این کلاس پایه مقدار نال برگشت داده می‌شود.

**کلاس GlobalDbResourceProvider**

برای پرووایدر منابع کلی از این کلاس استفاده می‌شود. نحوه پیاده‌سازی آن نیز دقیقا همانند طرح نمونه پیش‌فرض ASP.NET است.

```
using System;
using System.Resources;
namespace DbResourceProvider
{
  public class GlobalDbResourceProvider : BaseDbResourceProvider
  {
    private readonly string _classKey;
    public GlobalDbResourceProvider(string classKey)
    {
      _classKey = classKey;
    }
    #region Implementation of BaseDbResourceProvider
    protected override DbResourceManager CreateResourceManager()
    {
      return new DbResourceManager(_classKey);
    }
    public override IResourceReader ResourceReader
    {
      get { throw new NotSupportedException(); }
    }
    #endregion
  }
}
```

GlobalDbResourceProvider از کلاس پایه‌ای که در بالا شرح داده شد مشتق شده است. بنابراین تنها بخش‌های موردنیاز یعنی متد CreateResourceManager و پراپرتی ResourceReader در این کلاس پیاده‌سازی شده است.

در اینجا نمونه مخصوص کلاس ResourceManager (همان DbResourceManager) با توجه به نام فایل مربوط به منبع کلی تولید می‌شود. نام فایل در اینجا همان چیزی است که در دیتابیس برای نام منبع مربوطه ذخیره می‌شود. ساختار آن بعدا بحث می‌شود.

همان‌طور که می‌بینید برای پراپرتی ResourceReader خطای عدم پشتیبانی صادر می‌شود. دلیل آن در قسمت قبل و نیز به‌صورت کمی دقیق‌تر در ادامه آورده شده است.

**کلاس LocalDbResourceProvider**

برای منابع محلی نیز از طرحی مشابه نمونه پیش‌فرض ASP.NET که در قسمت قبل نشان داده شد، استفاده شده است.

```
using System.Resources;
namespace DbResourceProvider
{
  public class LocalDbResourceProvider : BaseDbResourceProvider
  {
    private readonly string _virtualPath;
    public LocalDbResourceProvider(string virtualPath)
    {
      _virtualPath = virtualPath;
    }
    #region Implementation of BaseDbResourceProvider
    protected override DbResourceManager CreateResourceManager()
    {
      return new DbResourceManager(_virtualPath);
    }
    public override IResourceReader ResourceReader
    {
      get { return new DbResourceReader(_virtualPath); }
    }
    #endregion
  }
}
```

این کلاس نیز از کلاس پایه‌ای BaseDbResourceProvider مشتق شده و پیاده‌سازی‌های مخصوص منابع محلی برای متد CreateResourceManager و پراپرتی ResourceReader در آن انجام شده است.
در متد CreateResourceManager کار تولید نمونه‌ای از DbResourceManager با استفاده از مسیر مجازی صفحه درخواستی انجام می‌شود. این فرایند شبیه به پیاده‌سازی پیش‌فرض ASP.NET است. در واقع در پیاده‌سازی جاری، نام منابع محلی همنام با مسیر مجازی متناظر آن‌ها در دیتابیس ذخیره می‌شود. درباره ساختار جدول دیتابیس بعدا بحث می‌شود.
در این کلاس کار بازخوانی کلیدهای موجود برای پراپرتی‌های موجود در یک صفحه از طریق نمونه‌ای از کلاس DbResourceReader انجام شده است. شرح این کلاس در ادامه آمده است.

**نکته:** همانطور که در قسمت قبل هم اشاره کوتاهی شده بود، از خاصیت ResourceReader در پرووایدر منابع برای تعیین تمام پراپرتی‌های موجود در منبع استفاده می‌شود تا کار جستجوی کلیدهای موردنیاز در عبارات بومی‌سازی **ضمنی** برای رندر صفحه وب راحت‌تر انجام شود. بنابراین از این پراپرتی تنها در پرووایدر منابع **محلی** استفاده می‌شود. ازآنجاکه در عبارات بومی‌سازی ضمنی تنها قسمت اول نام کلید ورودی منبع آورده می‌شود، بنابراین قسمت دوم (و یا قسمت‌های بعدی) کلید موردنظر که همان نام پراپرتی کنترل متناظر است از جستجو میان ورودی‌های یافته شده توسط این پراپرتی بدست می‌آید تا ASP.NET بداند که برای رندر صفحه چه پراپرتی‌هایی نیاز به رجوع به پرووایدر منبع محلی مربوطه دارد (برای آشنایی بیشتر با عبارت بومی‌سازی ضمنی رجوع شود به قسمت قبل ).

**نکته:** دقت کنید که پس از اولین درخواست، خروجی حاصل از enumerator این ResourceReader کش می‌شود تا در درخواست‌های بعدی از آن استفاده شود. بنابراین در حالت عادی، به ازای هر صفحه تنها یکبار این پراپرتی فراخوانده می‌شود. درباره این enumerator در ادامه بحث شده است.

**کلاس DbResourceManager**

کار اصلی مدیریت و بازیابی ورودی‌های منابع از دیتابیس از طریق کلاس DbResourceManager انجام می‌شود. نمونه‌ای بسیار ساده

و اولیه از این کلاس را در زیر مشاهده می‌کنید:

```
using System.Globalization;
using DbResourceProvider.Data;
namespace DbResourceProvider
{
  public class DbResourceManager
  {
    private readonly string _resourceName;
    public DbResourceManager(string resourceName)
    {
      _resourceName = resourceName;
    }
    public object GetObject(string resourceKey, CultureInfo culture)
    {
      var data = new ResourceData();
      return data.GetResource(_resourceName, resourceKey, culture.Name).Value;
    }
  }
}
```

کار استخراج ورودی‌های منابع با استفاده از نام منبع درخواستی در این کلاس مدیریت خواهد شد. این کلاس با استفاده نام منبع درخواستی به عنوان پارامتر کانستراکتور ساخته می‌شود. با استفاده از متد GetObject که نام کلید ورودی موردنظر و کالچر مربوطه را به عنوان پارامتر ورودی دریافت می‌کند فرایند استخراج انجام می‌شود. برای کپسوله‌سازی عملیات از کلاس جداگانه‌ای (ResourceData) برای تبادل با دیتابیس استفاده شده است. شرح بیشتر درباره این کلاس و نیز پیاده سازی کامل‌تر کلاس DbResourceManager به همراه مدیریت کش ورودی‌های منابع و نیز عملیات fallback در مطلب بعدی آورده می‌شود.

**کلاس DbResourceReader**

این کلاس که درواقع پیاده‌سازی اینترفیس IResourceReader است برای یافتن تمام کلیدهای تعریف شده برای یک منبع به‌کار می‌رود، پیاده‌سازی آن نیز به صورت زیر است:

```
using System.Collections;
using System.Resources;
using System.Security;
using DbResourceProvider.Data;
namespace DbResourceProvider
{
  public class DbResourceReader : IResourceReader
  {
    private readonly string _resourceName;
    private readonly string _culture;
    public DbResourceReader(string resourceName, string culture = "")
    {
      _resourceName = resourceName;
      _culture = culture;
    }
    #region Implementation of IResourceReader
    public void Close() { }
    public IDictionaryEnumerator GetEnumerator()
    {
      return new DbResourceEnumerator(new ResourceData().GetResources(_resourceName, _culture));
    }
    #endregion
    #region Implementation of IEnumerable
    IEnumerator IEnumerable.GetEnumerator()
    {
      return GetEnumerator();
    }
    #endregion
    #region Implementation of IDisposable
    public void Dispose()
    {
      Close();
    }
    #endregion
  }
}
```

این کلاس تنها با استفاده از نام منبع و عنوان کالچر موردنظر کار بازخوانی ورودی‌های موجود را انجام می‌دهد.

تنها نکته مهم در کد بالا متد GetEnumerator است که نمونه‌ای از اینترفیس IDictionaryEnumerator را برمی‌گرداند. در اینجا از کلاس DbResourceEnumerator که برای کار با دیتابیس طراحی شده، استفاده شده است. همانطور که قبلا هم اشاره شده بود، هر یک از اعضای این enumerator از نوع DictionaryEntry هستند که یک struct است. این کلاس در ادامه شرح داده شده است.
متد Close برای بستن و از بین بردن منابعی است که در تهیه enumerator موردبحث نقش داشته‌اند. مثل منابع شبکه‌ای یا فایلی که باید قبل از اتمام کار با این کلاس به صورت کامل بسته شوند. هرچند در نمونه جاری چنین موردی وجود ندارد و بنابراین این متد بلااستفاده است. در کلاس فوق نیز برای دریافت اطلاعات از ResourceData استفاده شده است که بعدا به همراه ساختار مناسب جدول دیتابیس شرح داده می‌شود.
**نکته:** دقت کنید که در پیاده‌سازی نشان داده شده برای کلاس LocalDbResourceProvider برای یافتن ورودی‌های موجود از مقدار پیش‌فرض (یعنی رشته خالی) برای کالچر استفاده شده است تا از ورودی‌های پیش‌فرض که در حالت عادی باید شامل تمام موارد تعریف شده موجود هستند استفاده شود (قبلا هم شرح داده شد که منبع اصلی و پیش‌فرض یعنی همانی که برای زبان پیش‌فرض برنامه درنظر گرفته می‌شود و بدون نام کالچر مربوطه است، باید شامل حداکثر ورودی‌های تعریف شده باشد. منابع مربوطه به سایر کالچرها می‌توانند همه این ورودی‌های تعریف‌شده در منبع اصلی و یا قسمتی از آن را شامل شوند. عملیات fallback تضمین می‌دهد که درنهایت نزدیک‌ترین گزینه متناظر با درخواست جاری را برگشت دهد).

**کلاس DbResourceEnumerator**

کلاس دیگری که در اینجا استفاده شده است، DbResourceEnumerator است. این کلاس در واقع پیاده سازی اینترفیس IDictionaryEnumerator است. محتوای این کلاس در زیر آورده شده است:

```
using System.Collections;
using System.Collections.Generic;
using DbResourceProvider.Models;
namespace DbResourceProvider
{
  public sealed class DbResourceEnumerator : IDictionaryEnumerator
  {
    private readonly List<Resource> _resources;
    private int _dataPosition;
    public DbResourceEnumerator(List<Resource> resources)
    {
      _resources = resources;
      Reset();
    }
    public DictionaryEntry Entry
    {
      get
      {
        var resource = _resources[_dataPosition];
        return new DictionaryEntry(resource.Key, resource.Value);
      }
    }
    public object Key { get { return Entry.Key; } }
    public object Value { get { return Entry.Value; } }
    public object Current { get { return Entry; } }
    public bool MoveNext()
    {
      if (_dataPosition >= _resources.Count - 1) return false;
      ++_dataPosition;
      return true;
    }
    public void Reset()
    {
      _dataPosition = -1;
    }
  }
}
```

تفاوت این اینترفیس با اینترفیس IEnumerable در سه عضو اضافی است که برای استفاده در سیستم مدیریت منابع ASP.NET نیاز است. همان‌طور که در کد بالا مشاهده می‌کنید این سه عضو عبارتند از پراپرتی‌های Entry و Key و Value. پراپرتی Entry که ورودی جاری در enumerator را مشخص می‌کند از نوع DictionaryEntry است. پراپرتی‌های Key و Value هم که از نوع object تعریف شده‌اند برای کلید و مقدار ورودی جاری استفاده می‌شوند.
این کلاس لیستی از Resource به عنوان پارامتر کانستراکتور برای تولید enumerator دریافت می‌کند. کلاس Resource مدل تولیدی از ساختار جدول دیتابیس برای ذخیره ورودی‌های منابع است که در مطلب بعدی شرح داده می‌شود. بقیه قسمت‌های کد فوق هم پیاده‌سازی معمولی یک enumerator است.

**نکته:** به جای تعریف کلاس جداگانه‌ای برای enumerator اینترفیس IResourceProvider می‌توان از enumerator کلاس‌هایی که IDictionary را پیاده‌سازی کرده‌اند نیز استفاده کرد، مانند کلاس Dictionary<object,object> یا ListDictionary.

**تنظیمات فایل کانفیگ**

برای اجبار کردن ASP.NET به استفاده از Factory موردنظر باید تنظیمات زیر را در فایل web.config اعمال کرد:

```
<system.web>
    ...
    <globalization resourceProviderFactoryType=" نام کامل اسمبلی مربوطه ,نام پرووایدر فکتوری به همراه
فضای نام آن " />
    ...
</system.web>
```

روش نشان داده شده در بالا حالت کلی تعریف و تنظیم یک نوع داده در فایل کانفیگ را نشان می‌دهد. درباره نام کامل اسمبلی در اینجا شرح داده شده است. مثلا برای پیاده‌سازی نشان داده شده در این مطلب خواهیم داشت:

```
<globalization resourceProviderFactoryType="DbResourceProvider.DbResourceProviderFactory,
DbResourceProvider" />
```

در مطلب بعدی درباره ساختار مناسب جدول یا جداول دیتابیس برای ذخیره ورودهای منابع و نیز پیاده‌سازی کامل‌تر کلاس‌های مورداستفاده بحث خواهد شد.

**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.onpreinit.com/2009/06/updatable-aspnet-resx-resource-provider.html
http://msdn.microsoft.com/en-us/library/system.web.compilation.resourceproviderfactory.aspx
http://www.codeproject.com/Articles/14190/ASP-NET-2-0-Custom-SQL-Server-ResourceProvider
http://www.codeproject.com/Articles/104667/Under-the-Hood-of-BuildManager-and-Resource-Handli

## نظرات خوانندگان

نویسنده: ابوالفضل رجب پور
تاریخ: ۱۳۹۲/۰۳/۰۶ ۱۱:۲۲

سلام جناب یوسف نژاد
برای پروژه م می‌خوام از روند شما استفاده کنم. بی صبرانه منتظر قسمت بعدی هستم
تشکر

عنوان: **Globalization در ASP.NET MVC - قسمت ششم**
نویسنده: یوسف نژاد
تاریخ: ۱۳۹۲/۰۳/۰۷ ۲۳:۴۰
آدرس: www.dotnettips.info
گروه‌ها: Localization, Globalization, Internationalization, ResourceProviderFactory, Custom Resource Provider, Database Resource Provider, Resource

در قسمت قبل ساختار اصلی و پیاده‌سازی ابتدایی یک پرووایدر سفارشی دیتابیسی شرح داده شد. در این قسمت ادامه بحث و مطالب پیشرفته‌تر آورده شده است.

## تولید یک پرووایدر منابع دیتابیسی - بخش دوم

در بخش دوم این سری مطلب، ساختار دیتابیس و مباحث پیشرفته پیاده‌سازی کلاس‌های نشان داده‌شده در بخش اول در قسمت قبل شرح داده می‌شود. این مباحث شامل نحوه کش صحیح و بهینه داده‌های دریافتی از دیتابیس، پیاده‌سازی فرایند fallback، و پیاده‌سازی مناسب کلاس DbResourceManager برای مدیریت کل عملیات است.

### ساختار دیتابیس

برای پیاده‌سازی منابع دیتابیسی روش‌های مختلفی برای آرایش جداول جهت ذخیره انواع ورودی‌ها می‌توان درنظر گرفت.
مثلا درصورتی‌که حجم و تعداد منابع بسیار باشد و نیز منابع دیتابیسی به اندازه کافی در دسترس باشد، می‌توان به ازای هر منبع یک جدول درنظر گرفت.
یا درصورتیکه منابع داده‌ای محدودتر باشند می‌توان به ازای هر کالچر یک جدول درنظر گرفت و تمام منابع مربوط به یک کالچر را درون یک جدول ذخیره کرد. درهرصورت نحوه انتخاب آرایش جداول منابع کاملا بستگی به شرایط کاری و سلایق برنامه‌نویسی دارد.
برای مطلب جاری به عنوان یک راه‌حل ساده و کارآمد برای پروژه‌های کوچک و متوسط، تمام ورودی‌های منابع درون یک جدول با ساختاری مانند زیر ذخیره می‌شود:

| | Column Name | Data Type | Allow Nulls |
|---|---|---|---|
| 🔑 | Id | bigint | ☐ |
| | Name | nvarchar(200) | ☐ |
| | [Key] | nvarchar(200) | ☐ |
| | Culture | nvarchar(6) | ☐ |
| | Value | nvarchar(MAX) | ☐ |

نام این جدول را با درنظر گرفتن شرایط موجود می‌توان Resources گذاشت.

ستون Name برای ذخیره نام منبع درنظر گرفته شده است. این نام برابر نام منابع درخواستی در سیستم مدیریت منابع ASP.NET است که درواقع برابر همان نام فایل منبع اما بدون پسوند resx. است.

ستون Key برای نگهداری کلید ورودی منبع استفاده می‌شود که دقیقا برابر همان مقداری است که درون فایلهای resx. ذخیره می‌شود.

ستون Culture برای ذخیره کالچر ورودی منبع به کار می‌رود. این مقدار می‌تواند برای کالچر پیش‌فرض برنامه برابر رشته خالی باشد.

ستون Value نیز برای نگهداری مقدار ورودی منبع استفاده می‌شود.

برای ستون Id می‌توان از GUID نیز استفاده کرد. در اینجا برای راحتی کار از نوع داده bigint و خاصیت Identity برای تولید خودکار آن در Sql Server استفاده شده است.

**نکته:** برای امنیت بیشتر می‌توان یک Unique Constraint بر روی سه فیلد Name و Key و Culture اعمال کرد.

برای نمونه به تصویر زیر که ذخیره تعدای ورودی منبع را درون جدول Resources نمایش می‌دهد دقت کنید:

| Id | Name | Key | Culture | Value |
|---|---|---|---|---|
| 1 | GlobalTexts | Yes | | yesssss |
| 2 | GlobalTexts | Yes | fa | بله |
| 3 | GlobalTexts | Yes | fr | oui |
| 4 | GlobalTexts | No | | no |
| 5 | GlobalTexts | No | fa | خیر |
| 6 | GlobalTexts | No | fr | pas |
| 7 | Default.aspx | Label1.Text | | Hello |
| 10 | Default.aspx | Label1.ForeColor | | red |
| 11 | Default.aspx | Label1.Text | en-US | hello |
| 13 | Default.aspx | Label1.ForeColor | en-US | blue |
| 14 | Default.aspx | Label1.Text | fa | درود |
| 16 | Default.aspx | Label1.ForeColor | fa | red |
| 17 | Default.aspx | Label2.Text | | GoodBye |
| 18 | Default.aspx | Label2.ForeColor | | orange |
| 19 | Default.aspx | Label2.Text | en-US | goodbye |
| 20 | Default.aspx | Label2.ForeColor | en-US | green |
| 21 | dir1/page1.aspx | Label1.Text | | sssss |
| 22 | dir1/page1.aspx | Label2.Text | | aaaaa |
| 23 | dir1/page1.aspx | Label1.Text | en-US | String 1 |
| 24 | dir1/page1.aspx | Label2.Text | en-US | String 2 |
| 25 | dir1/page1.aspx | Label1.Text | fa | رشته 1 |
| 26 | dir1/page1.aspx | Label2.Text | fa | رشته 2 |

### اصلاح کلاس DbResourceProviderFactory

برای ذخیره منابع محلی، جهت اطمینان از یکسان بودن نام منبع، متد مربوطه در کلاس DbResourceProviderFactory باید به‌صورت زیر تغییر کند:

```
public override IResourceProvider CreateLocalResourceProvider(string virtualPath)
{
  if (!string.IsNullOrEmpty(virtualPath))
  {
    virtualPath = virtualPath.Remove(0, virtualPath.IndexOf('/') + 1); // removes everything from start
to the first '/'
  }
  return new LocalDbResourceProvider(virtualPath);
}
```

با این تغییر مسیرهای درخواستی چون "~/Default.aspx" و یا "/Default.aspx" هر دو به صورت "Default.aspx" در می‌آیند تا با نام ذخیره شده در دیتابیس یکسان شوند.

### ارتباط با دیتابیس

خوشبختانه برای تبادل اطلاعات با جدول بالا امروزه راه‌های زیادی وجود دارد. برای پیاده‌سازی آن مثلا می‌توان از یک اینترفیس استفاده کرد. سپس با استفاده از سازوکارهای موجود مثلا به‌کارگیری IoC ، نمونه مناسبی از پیاده‌سازی اینترفیس مذبور را در اختیار برنامه قرار داد.
اما برای جلوگیری از پیچیدگی بیش از حد و دور شدن از مبحث اصلی، برای پیاده‌سازی فعلی از EF Code First به صورت مستقیم در پروژه استفاده شده است که سری آموزشی کاملی از آن در همین سایت وجود دارد.

پس از پیاده‌سازی کلاس‌های مرتبط برای استفاده از EF Code First، از کلاس ResourceData که در بخش اول نیز نشان داده شده بود، برای کپسوله کردن ارتباط با داده‌ها استفاده می‌شود که نمونه‌ای ابتدایی از آن در زیر آورده شده است:

```
using System.Collections.Generic;
using System.Linq;
using DbResourceProvider.Models;

namespace DbResourceProvider.Data
{
  public class ResourceData
  {
    private readonly string _resourceName;
    public ResourceData(string resourceName)
    {
      _resourceName = resourceName;
    }
    public Resource GetResource(string resourceKey, string culture)
    {
      using (var data = new TestContext())
      {
        return data.Resources.SingleOrDefault(r => r.Name == _resourceName && r.Key == resourceKey &&
r.Culture == culture);
      }
    }
    public List<Resource> GetResources(string culture)
    {
      using (var data = new TestContext())
      {
        return data.Resources.Where(r => r.Name == _resourceName && r.Culture == culture).ToList();
      }
    }
  }
}
```

کلاس فوق نسبت به نمونه‌ای که در قسمت قبل نشان داده شد کمی فرق دارد. بدین صورت که برای راحتی بیشتر نام منبع

درخواستی به جای پارامتر متدها، در اینجا به عنوان پارامتر کانستراکتور وارد می‌شود.

**نکته:** درصورتی‌که این کلاس‌ها در پروژه‌ای جداگانه قرار دارند، باید ConnectionString مربوطه در فایل کانفیگ برنامه مقصد نیز تنظیم شود.

**کش کردن ورودی‌ها**

برای کش کردن ورودی‌ها این نکته را که قبلا هم به آن اشاره شده بود باید درنظر داشت:

**پس از اولین درخواست برای هر منبع، نمونه تولیدشده از پرووایدر مربوطه در حافظه سرور کش خواهد شد.**

یعنی متدهای کلاس DbResourceProviderFactory **به‌ازای هر منبع** تنها یکبار فراخوانی می‌شود. نمونه‌های کش‌شده از پرووایدرهای کلی و محلی به همراه تمام محتویاتشان (مثلا نمونه تولیدی از کلاس DbResourceManager) تا زمان Unload شدن سایت در حافظه سرور باقی می‌مانند. بنابراین عملیات کشینگ ورودی‌ها را می‌توان درون خود کلاس DbResourceManager به ازای هر منبع انجام داد.

برای کش کردن ورودی‌های هر منبع می‌توان چند روش را درپیش گرفت. روش اول این است که به ازای هر کلید درخواستی تنها ورودی مربوطه از دیتابیس فراخوانی شده و در برنامه کش شود. این روش برای حالاتی که تعداد ورودی‌ها یا تعداد درخواست‌های کلیدهای هر منبع کم باشد مناسب خواهد بود.

یکی از پیاده‌سازی این روش این است که ورودی‌ها به ازای هر کالچر ذخیره شوند. پیاده‌سازی اولیه این نوع فرایند کشینگ در کلاس DbResourceManager به صورت زیر است:

```
using System.Collections.Generic;
using System.Globalization;
using DbResourceProvider.Data;
namespace DbResourceProvider
{
  public class DbResourceManager
  {
    private readonly string _resourceName;
    private readonly Dictionary<string, Dictionary<string, object>> _resourceCacheByCulture;
    public DbResourceManager(string resourceName)
    {
      _resourceName = resourceName;
      _resourceCacheByCulture = new Dictionary<string, Dictionary<string, object>>();
    }
    public object GetObject(string resourceKey, CultureInfo culture)
    {
      return GetCachedObject(resourceKey, culture.Name);
    }
    private object GetCachedObject(string resourceKey, string cultureName)
    {
      if (!_resourceCacheByCulture.ContainsKey(cultureName))
        _resourceCacheByCulture.Add(cultureName, new Dictionary<string, object>());
      var cachedResource = _resourceCacheByCulture[cultureName];
      lock (this)
      {
        if (!cachedResource.ContainsKey(resourceKey))
        {
          var data = new ResourceData(_resourceName);
          var dbResource = data.GetResource(resourceKey, cultureName);
          if (dbResource == null) return null;
          var cachedResources = _resourceCacheByCulture[cultureName];
          cachedResources.Add(dbResource.Key, dbResource.Value);
        }
      }
      return cachedResource[resourceKey];
    }
  }
}
```

همانطور که قبلا توضیح داده شد کشِ پرووایدرهای منابع به ازای هر منبع درخواستی (و به تبع آن نمونه‌های موجود در آن مثل DbResourceManager) برعهده خود ASP.NET است. بنابراین برای کش کردن ورودی‌های درخواستی هر منبع در کلاس DbResourceManager تنها کافی است آن‌ها را درون یک متغیر محلی در سطح کلاس (فیلد) ذخیره کرد. کاری که در کد بالا در متغیر _resourceCacheByCulture انجام شده است. در این متغیر که از نوع دیکشنری تعریف شده است کلیدهای هر عضو آن برابر نام کالچر مربوطه است. مقادیر هر عضو این دیکشنری نیز خود یک دیکشنری است که ورودی‌های منابع مربوط به کالچر مربوطه در

آن ذخیره می‌شوند.
عملیات در متد GetCachedObject انجام می‌شود. همان‌طور که می‌بینید ابتدا وجود ورودی موردنظر در متغیر کشینگ بررسی می‌شود و درصورت عدم وجود، مقدار آن مستقیما از دیتابیس درخواست می‌شود. سپس این مقدار درخواستی ابتدا درون متغیر کشینگ ذخیره شده (به همراه بلاک lock) و درنهایت برگشت داده می‌شود.

**نکته:** کل فرایند بررسی وجود کلید در متغیر کشینگ (شرط دوم در متد GetCachedObject) درون بلاک lock قرار داده شده است تا در درخواست‌های همزمان احتمال افزودن چندباره یک کلید ازبین برود.

پیاده‌سازی دیگر این فرایند کشینگ، ذخیره ورودی‌ها براساس نام کلید به جای نام کالچر است. یعنی کلید دیکشنری اصلی نام کلید و کلید دیکشنری داخلی نام کالچر است که این روش زیاد جالب نیست.
روش دوم که بیشتر برای برنامه‌های بزرگ با ورودی‌ها و درخواست‌های زیاد به‌کار می‌رود این است که درهر بار درخواست به دیتابیس به جای دریافت تنها همان ورودی درخواستی، تمام ورودی‌های منبع و کالچر درخواستی استخراج شده و کش می‌شود تا تعداد درخواست‌های به سمت دیتابیس کاهش یابد. برای پیاده‌سازی این روش کافی است تغییرات زیر در متد GetCachedObject اعمال شود:

```
private object GetCachedObject(string resourceKey, string cultureName)
{
  lock (this)
  {
    if (!_resourceCacheByCulture.ContainsKey(cultureName))
    {
      _resourceCacheByCulture.Add(cultureName, new Dictionary<string, object>());
      var cachedResources = _resourceCacheByCulture[cultureName];
      var data = new ResourceData(_resourceName);
      var dbResources = data.GetResources(cultureName);
      foreach (var dbResource in dbResources)
      {
        cachedResources.Add(dbResource.Key, dbResource.Value);
      }
    }
  }
  var cachedResource = _resourceCacheByCulture[cultureName];
  return !cachedResource.ContainsKey(resourceKey) ? null : cachedResource[resourceKey];
}
```

دراینجا هم می‌توان به جای استفاده از نام کالچر برای کلید دیکشنری اصلی از نام کلید ورودی منبع استفاده کرد که چندان توصیه نمی‌شود.

**نکته:** انتخاب یکی از دو روش فوق برای فرایند کشینگ کاملا به **شرایط موجود** و سلیقه برنامه نویس بستگی دارد.

**فرایند Fallback**

درباره فرایند fallback به اندازه کافی در قسمت‌های قبلی توضیح داده شده است. برای پیاده‌سازی این فرایند ابتدا باید به نوعی به سلسله مراتب کالچرهای موجود از کالچر جاری تا کالچر اصلی و پیش فرض سیستم دسترسی پیدا کرد. برای اینکار ابتدا باید با استفاده از روشی کالچر والد یک کالچر را بدست آورد. کالچر والد کالچری است که عمومیت بیشتری نسبت به کالچر موردنظر دارد. مثلا کالچر fa، کالچر والد fa-IR است. همچنین کالچر Invariant به عنوان والد تمام کالچرها شناخته می‌شود.
خوشبختانه در کلاس CultureInfo (که در قسمت‌های قبلی شرح داده شده است) یک پراپرتی با عنوان Parent وجود دارد که کالچر والد را برمی‌گرداند.
برای رسیدن به سلسله مراتب مذبور در کلاس ResourceManager دات نت، از کلاسی با عنوان ResourceFallbackManager استفاده می‌شود. هرچند این کلاس با سطح دسترسی internal تعریف شده است اما نام‌گذاری نامناسبی دارد زیرا کاری که می‌کند به عنوان Manager هیچ ربطی ندارد. این کلاس با استفاده از یک کالچر ورودی، یک enumerator از سلسله مراتب کالچرها که در بالا صحبت شد تهیه می‌کند.
با استفاده پیاده‌سازی موجود در کلاس ResourceFallbackManager کلاسی با عنوان CultureFallbackProvider تهیه کردم که به صورت زیر است:

```
using System.Collections;
using System.Collections.Generic;
using System.Globalization;
namespace DbResourceProvider
{
  public class CultureFallbackProvider : IEnumerable<CultureInfo>
  {
    private readonly CultureInfo _startingCulture;
    private readonly CultureInfo _neutralCulture;
    private readonly bool _tryParentCulture;
    public CultureFallbackProvider(CultureInfo startingCulture = null,
                                   CultureInfo neutralCulture = null,
                                   bool tryParentCulture = true)
    {
      _startingCulture = startingCulture ?? CultureInfo.CurrentUICulture;
      _neutralCulture = neutralCulture;
      _tryParentCulture = tryParentCulture;
    }
    #region Implementation of IEnumerable<CultureInfo>
    public IEnumerator<CultureInfo> GetEnumerator()
    {
      var reachedNeutralCulture = false;
      var currentCulture = _startingCulture;
      do
      {
        if (_neutralCulture != null && currentCulture.Name == _neutralCulture.Name)
        {
          yield return CultureInfo.InvariantCulture;
          reachedNeutralCulture = true;
          break;
        }
        yield return currentCulture;
        currentCulture = currentCulture.Parent;
      } while (_tryParentCulture && !HasInvariantCultureName(currentCulture));
      if (!_tryParentCulture || HasInvariantCultureName(_startingCulture) || reachedNeutralCulture)
        yield break;
      yield return CultureInfo.InvariantCulture;
    }
    #endregion
    #region Implementation of IEnumerable
    IEnumerator IEnumerable.GetEnumerator()
    {
      return GetEnumerator();
    }
    #endregion
    private bool HasInvariantCultureName(CultureInfo culture)
    {
      return culture.Name == CultureInfo.InvariantCulture.Name;
    }
  }
}
```

این کلاس که اینترفیس IEnumerable<CultureInfo> را پیاده‌سازی کرده است، سه پارامتر کانستراکتور دارد.

اولین پارامتر، کالچر جاری یا آغازین را مشخص می‌کند. این کالچری است که تولید enumerator مربوطه از آن آغاز می‌شود. درصورتی‌که این پارامتر نال باشد مقدار کالچر UI در ثرد جاری برای آن درنظر گرفته می‌شود. مقدار پیش‌فرضی که برای این پارامتر درنظر گرفته شده است، null است.

پارامتر بعدی کالچر خنثی موردنظر کاربر است. این کالچری است که درصورت رسیدن enumerator به آن کار پایان خواهد یافت. درواقع کالچر پایانی enumerator است. این پارامتر می‌تواند نال باشد. مقدار پیش‌فرضی که برای این پارامتر درنظر گرفته شده است، null است.

پارمتر آخر هم تعیین می‌کند که آیا enumerator از کالچرهای والد استفاده بکند یا خیر. مقدار پیش‌فرضی که برای این پارامتر درنظر گرفته شده است، true است.

کار اصلی کلاس فوق در متد GetEnumerator انجام می‌شود. در این کلاس یک حلقه do-while وجود دارد که enumerator را با استفاده از کلمه کلیدی yield تولید می‌کند. در این متد ابتدا درصورت نال نبودن کالچر خنثی ورودی، بررسی می‌شود که آیا نام کالچر جاری حلقه (که در متغیر محلی currentCulture ذخیره شده است) برابر نام کالچر خنثی است یا خیر. درصورت برقراری شرط، کار این حلقه با برگشت CultureInfo.InvariantCulture پایان می‌یابد. InvariantCulture کالچر بدون زبان و فرهنگ و موقعیت مکانی است که درواقع به عنوان کالچر والد تمام کالچرها درنظر گرفته می‌شود. پراپرتی Name این کالچر برابر string.Empty است.

کار حلقه با برگشت مقدار کالچر جاری enumerator ادامه می‌یابد. سپس کالچر جاری با کالچر والدش مقداردهی می‌شود. شرط قسمت while حلقه تعیین می‌کند که درصورتی‌که کلاس برای استفاده از کالچرهای والد تنظیم شده باشد، تا زمانی که نام کالچر جاری برابر نام کالچر Invariant **نباشد** ، تولید اعضای enumerator ادامه یابد.
درانتها نیز درصورتی‌که با شرایط موجود، قبلا کالچر Invariant برگشت داده نشده باشد این کالچر نیز yield می‌شود. درواقع درصورتی‌که استفاده از کالچرهای والد اجازه داده **نشده** باشد **یا** کالچر آغازین برابر کالچر Invariant باشد و **یا** قبلا به دلیل رسیدن به کالچر خنثی ورودی، مقدار کالچر Invariant برگشت داده شده باشد، enumerator قطع شده و عملیات پایان می‌یابد. در غیر اینصورت کالچر Invariant به عنوان کالچر پایانی برگشت داده می‌شود.

**استفاده از CultureFallbackProvider**

با استفاده از کلاس CultureFallbackProvider می‌توان عملیات جستجوی ورودی‌های درخواستی را با ترتیبی مناسب بین تمام کالچرهای موجود به انجام رسانید.
برای استفاده از این کلاس باید تغییراتی در متد GetObject کلاس DbResourceManager به صورت زیر اعمال کرد:

```
public object GetObject(string resourceKey, CultureInfo culture)
{
  foreach (var currentCulture in new CultureFallbackProvider(culture))
  {
    var value = GetCachedObject(resourceKey, currentCulture.Name);
    if (value != null) return value;
  }
  throw new KeyNotFoundException("The specified 'resourceKey' not found.");
}
```

با استفاده از یک حلقه foreach درون enumerator کلاس CultureFallbackProvider، کالچرهای موردنیاز برای fallback یافته می‌شوند. در اینجا از مقادیر پیش‌فرض دو پارامتر دیگر کانستراکتور کلاس CultureFallbackProvider استفاده شده است.
سپس به ازای هر کالچر یافته شده مقدار ورودی درخواستی بدست آمده و درصورتی‌که نال نباشد (یعنی ورودی موردنظر برای کالچر جاری یافته شود) آن مقدار برگشت داده می‌شود و درصورتی‌که نال باشد عملیات برای کالچر بعدی ادامه می‌یابد.
درصورتی‌که ورودی درخواستی یافته نشود (خروج از حلقه بدون برگشت مقداری برای ورودی منبع درخواستی) استثنای KeyNotFoundException صادر می‌شود تا کاربر را از اشتباه رخداده مطلع سازد.

**آزمایش پرووایدر سفارشی**

ابتدا تنظیمات موردنیاز فایل کانفیگ را که در قسمت قبل نشان داده شد، در برنامه خود اعمال کنید.

داده‌های نمونه نشان داده شده در ابتدای این مطلب را درنظر بگیرید. حال اگر در یک برنامه وب اپلیکیشن، صفحه Default.aspx در ریشه سایت حاوی دو کنترل زیر باشد:

```
<asp:Label ID="Label1" runat="server" meta:resourcekey="Label1" />
<asp:Label ID="Label2" runat="server" meta:resourcekey="Label2" />
```

خروجی برای کالچر "en-US" (معمولا پیش‌فرض، اگر تنظیمات سیستم عامل تغییر نکرده باشد) چیزی شبیه تصویر زیر خواهد بود:

hello goodbye

سپس تغییر زیر را در فایل web.config اعمال کنید تا کالچر UI سایت به fa تغییر یابد (به بخش uiCulture="fa" دقت کنید):

```
<globalization uiCulture="fa" resourceProviderFactoryType =
```

```
"DbResourceProvider.DbResourceProviderFactory, DbResourceProvider" />
```

بنابراین صفحه Default.aspx با همان داده‌های نشان داده شده در بالا به صورت زیر تغییر خواهد کرد:


می‌بینید که با توجه به عدم وجود مقداری برای Label2.Text برای کالچر fa، عملیات fallback اتفاق افتاده است.

**بحث و نتیجه‌گیری**

کار تولید یک پرووایدر منابع سفارشی دیتابیسی به اتمام رسید. تا اینجا اصول کلی تولید یک پرووایدر سفارشی شرح داده شد. بدین ترتیب می‌توان برای هر حالت خاص دیگری نیز پرووایدرهای سفارشی مخصوص ساخت تا مدیریت منابع به آسانی تحت کنترل برنامه نویس قرار گیرد.

اما نکته‌ای را که باید به آن توجه کنید این است که در پیاده‌سازی‌های نشان داده شده با توجه به نحوه کش‌شدن مقادیر ورودی‌ها، اگر این مقادیر در دیتابیس تغییر کنند، تا زمانیکه سایت ریست نشود این تغییرات در برنامه اعمال نخواهد شد. زیرا همانطور که اشاره شد، مدیریت نمونه‌های تولیدشده از پرووایدرهای منابع برای هر منبع درخواستی درنهایت برعهده ASP.NET است. بنابراین باید مکانیزمی پیاده شود تا کلاس DbResourceManager از به‌روزرسانی ورودی‌های کش‌شده اطلاع یابد تا آنها را ریفرش کند.

در ادامه درباره روش‌های مختلف نحوه پیاده‌سازی قابلیت به‌روزرسانی ورودی‌های منابع در زمان اجرا با استفاده از پرووایدرهای منابع سفارشی بحث خواهد شد. همچنین راه‌حل‌های مختلف استفاده از این پرووایدرهای سفارشی در جاهای مختلف پروژه‌های MVC شرح داده می‌شود.

البته مباحث پیشرفته‌تری چون تزریق وابستگی برای پیاده‌سازی لایه ارتباط با دیتابیس در بیرون و یا تولید یک Factory برای تزریق کامل پرووایدر منابع از بیرون نیز جای بحث و بررسی دارد.

**منابع**

http://weblogs.asp.net/thangchung/archive/2010/06/25/extending-resource-provider-for-soring-resources-in-the-database.aspx

http://msdn.microsoft.com/en-us/library/aa905797.aspx

http://msdn.microsoft.com/en-us/library/system.web.compilation.resourceproviderfactory.aspx

http://www.dotnetframework.org/default.aspx/.../ResourceFallbackManager@cs

http://www.codeproject.com/Articles/14190/ASP-NET-2-0-Custom-SQL-Server-ResourceProvider

http://www.west-wind.com/presentations/wwdbresourceprovider

**نظرات خوانندگان**

نویسنده: صابر فتح الهی
تاریخ: ۱۳۹۲/۰۳/۰۸ ۰:۴۲

با تشکر از کار زیبای شما
لطفا برچسپ resource را اضافه کنید تا پیوستگی مطالب حفظ شود.

نویسنده: یوسف نژاد
تاریخ: ۱۳۹۲/۰۳/۰۸ ۱:۴۰

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

نویسنده: صابر فتح الهی
تاریخ: ۱۳۹۲/۰۳/۰۸ ۳:۱۵

مهندس بک سوال؟
مشکلی نداره ما سه جدول:
-1 جدولی برای ذخیره نام کالچرها
-2جدولی برای ذخیره عنوان کلیدهای اصلی
-3 جدولی برای ذخیره مقادیر یک کالچر برای یک کلید خاص

تعریف کنیم؟
اگر درست فهمیده باشم فقط باید بخش بازیابی کلیدها تغییر کنه درسته؟

نویسنده: محسن خان
تاریخ: ۱۳۹۲/۰۳/۰۸ ۸:۴۱

اون وقت حداقل 2 تا join باید بنویسید و وجود هر join یعنی کم‌تر شدن سرعت دسترسی به اطلاعات. چرا؟ چه تکرار اطلاعاتی رو مشاهده می‌کنید که قصد دارید تا این حد نرمالش کنید؟ نام و کلید و فرهنگ یک موجودیت هستند.

نویسنده: یوسف نژاد
تاریخ: ۱۳۹۲/۰۳/۰۸ ۹:۱۱

دلیل خاصی برای تفکیک این چنینی وجود نداره و همونطور که دوستمون گفتن این روشی که شما اشاره کردین مشکلات و معایبی هم به همراه داره.
روش اشاره شده تو این مطلب تو بیش از 99 درصد پروژه‌ها کفایت میکنه. فقط تو پروژه‌های بسیار بسیار بزرگ با ورودی‌های منابع بسیار بسیار زیاد (چند صد هزار و یا بیشتر) تغییر این ساختار برای رسیدن به کارایی مناسب میتونه مفید باشه.
درهرصورت اگر نیاز به تغییر ساختار جدول دارین فقط لایه دسترسی به بانک باید تغییر بکنه و فرایند کلی دسترسی به ورودی‌های منابع ذخیره شده در دیتابیس باید به همون صورتی باشه که در اینجا آورده شده. یعنی درنهایت با استفاده از سه پارامتر نام منبع، نام کالچر و عنوان کلید درخواستی کار استخراج مقدار ورودی باید انجام بشه.

نویسنده: صابر فتح الهی
تاریخ: ۱۳۹۲/۰۳/۰۸ ۱۰:۱۴

برای طراحی یک سامانه مدیریت محتوا با کلی ماژول فکر می‌کنم حرفم منطقی باشه مهندس، در ضمن همونجوری که مهندس یوسف نژاد فرمودن اطلاعات در بازیابی اولیه کش میشه و تا ری ستارت شدن سایت در حافظه می‌مونه، فکر می‌کنم چندان تاثیری

بروی کارایی داشته باشه با توجه به فرضیات، فرض کن من 10000 عنوان دارم، 30 تا زبان دارم در این صورت توی یک جدول زبان انگلیسی (en-کالچر انگلیسی) 10000 بار تکرار میشه علاوه بر اون عنوان مثلا "نام کاربری" به ازای 30 زبان 30 بار تکرار میشه زیادم حرف من غیر منطقی نیست و الا حرف شما درسته بله join سرعت پایین میاره اما ما که قرار نیست زیادی دسترسی به این جداول داشته باشیم.

**"پس از اولین درخواست برای هر منبع، نمونه تولیدشده از پرووایدر مربوطه در حافظه سرور کش خواهد شد.** " سخن مهندس یوسف نژاد

نویسنده: محسن خان
تاریخ: ۱۳۹۲/۰۳/۰۸ ۱۲:۱۰

یک سری از برآوردها تخیلی هستند. حتی مایکروسافت هم با لشگر مترجم‌هایی که داره مثلا برای شیرپوینت تجاری خودش زیر 10 تا زبان رو تونسته ارائه بده.

نویسنده: بهنام حقی
تاریخ: ۱۳۹۳/۰۱/۳۱ ۱۷:۹

با سلام
من این حالت رو میخوام با uow میخوام پیاده سازی کنم. میخوام یک سری تغییرات تو ساختار جدول بدم.
یک جدول برای مدیریت اضافه و حذف زبان (نام، Culture، ISO، RTL و ...) و جدول دیگم برای ریسورس ها(کلید، اسم، مقدار)
در واقع میخوام مقادیر ریسورس‌ها با اضافه و حذف شدن یک زبان به سیستم مدیریت بشه.
میخواستم ببینم که چه پیشنهادی برای این حالت دارید؟

عنوان: Globalization در ASP.NET MVC - قسمت هفتم
نویسنده: یوسف نژاد
تاریخ: ۱۳۹۲/۰۳/۱۲ ۱۳:۳۰
آدرس: www.dotnettips.info
گروه‌ها: Localization, Globalization, Internationalization, Resource, Custom Resource Provider, Database Resource Provider

در قسمت قبل مطالب تکمیلی تولید پرووایدر سفارشی منابع دیتابیسی ارائه شد. در این قسمت نحوه بروزرسانی ورودی‌های منابع در زمان اجرا بحث می‌شود.

## تولید یک پرووایدر منابع دیتابیسی - بخش سوم

برای پیاده‌سازی ویژگی به‌روزرسانی ورودی‌های منابع در زمان اجرا راه‌حل‌های مختلفی ممکن است به ذهن برنامه‌نویس خطور کند که هر کدام معایب و مزایای خودش را دارد. اما درنهایت بسته به شرایط موجود انتخاب روش مناسب برعهده خود برنامه‌نویس است.

مثلا برای پرووایدر سفارشی دیتابیسی تهیه‌شده در مطالب قبلی، تنها کافی است ابزاری تهیه شود تا به کاربران اجازه به‌روزرسانی مقادیر موردنظرشان در دیتابیس را بدهد که کاری بسیار ساده است. بدین ترتیب به‌روزرسانی این مقادیر در زمان اجرا کاری بسیار ابتدایی به نظر می‌رسد. اما در قسمت قبل نشان داده شد که برای بالا بردن بازدهی بهتر است که مقادیر موجود در دیتابیس در حافظه سرور کش شوند. استراتژی اولیه و ساده‌ای نیز برای نحوه پیاده‌سازی این فرایند کشینگ ارائه شد. بنابراین باید امکاناتی فراهم شود تا درصورت تغییر مقادیر کش‌شده در سمت دیتابیس، برنامه از این تغییرات آگاه شده و نسبت به به‌روزرسانی این مقادیر در متغیر کشینگ اقدامات لازم را انجام دهد.

اما همان‌طور که در قسمت قبل نیز اشاره شد، نکته‌ای که باید درنظر داشت این است که مدیریت تمامی نمونه‌های تولیدشده از کلاس‌های موردبحث کاملا برعهده ASP.NET است، بنابراین دسترسی مستقیمی به این نمونه‌ها در بیرون و در زمان اجرا وجود ندارد تا این ویژگی را بتوان در مورد آن‌ها پیاده کرد.

یکی از روش‌های موجود برای حل این مشکل این است که مکانیزمی پیاده شود تا بتوان به تمامی نمونه‌های تولیدی از کلاس DbResourceManager در بیرون از محیط سیستم مدیریت منابع ASP.NET دسترسی داشت. مثلا یک کلاس حاول متغیری استاتیک جهت ذخیره نمونه‌های تولیدی از کلاس DbResourceManager، به کتابخانه خود اضافه کرد تا با استفاده از یکسری امکانات بتوان این نمونه‌های تولیدی را از تغییرات رخداده در سمت دیتابیس آگاه کرد. در این قسمت پیاده‌سازی این راه‌حل شرح داده می‌شود.

.

**نکته:** قبل از هرچیز برای مناسب شدن طراحی کتابخانه تولیدی و افزایش امنیت آن بهتر است تا سطح دسترسی تمامی کلاس‌های پیاده‌سازی شده تا این مرحله به internal تغییر کند. ازآنجاکه سیستم مدیریت منابع ASP.NET از ریفلکشن برای تولید نمونه‌های موردنیاز خود استفاده می‌کند، بنابراین این تغییر تاثیری بر روند کاری آن نخواهد گذاشت.

.

**نکته:** با توجه به شرایط خاص موجود، ممکن است نام‌های استفاده شده برای کلاس‌های این کتابخانه کمی گیج‌کننده باشد. پس با دقت بیشتری به مطلب توجه کنید.

.

### پیاده‌سازی امکان پاک‌سازی مقادیر کش‌شده

برای این‌کار باید تغییراتی در کلاس DbResourceManager داده شود تا بتوان این کلاس را از تغییرات بوجود آمده آگاه ساخت. روشی که من برای این کار درنظر گرفتم استفاده از یک اینترفیس حاوی اعضای موردنیاز برای پیاده‌سازی این امکان است تا مدیریت این ویژگی در ادامه راحت‌تر شود.

.

### اینترفیس IDbCachedResourceManager

این اینترفیس به صورت زیر تعریف شده است:

```
namespace DbResourceProvider
{
  internal interface IDbCachedResourceManager
  {
    string ResourceName { get; }

    void ClearAll();
    void Clear(string culture);
    void Clear(string culture, string resourceKey);
  }
}
```

در پراپرتی فقط خواندنی ResourceName نام منبع کش شده ذخیره خواهد شد.

متد ClearAll برای پاک‌سازی تمامی ورودی‌های کش‌شده استفاده می‌شود.

متدهای Clear برای پاک‌سازی ورودی‌های کش‌شده یک کالچر به خصوص و یا یک ورودی خاص استفاده می‌شود.

با استفاده از این اینترفیس، پیاده‌سازی کلاس DbResourceManager به صورت زیر تغییر می‌کند:

```
using System.Collections.Generic;
using System.Globalization;
using DbResourceProvider.Data;
namespace DbResourceProvider
{
  internal class DbResourceManager : IDbCachedResourceManager
  {
    private readonly string _resourceName;
    private readonly Dictionary<string, Dictionary<string, object>> _resourceCacheByCulture;
    public DbResourceManager(string resourceName)
    {
      _resourceName = resourceName;
      _resourceCacheByCulture = new Dictionary<string, Dictionary<string, object>>();
    }
    public object GetObject(string resourceKey, CultureInfo culture) { ... }
    private object GetCachedObject(string resourceKey, string cultureName) { ... }

    #region Implementation of IDbCachedResourceManager
    public string ResourceName
    {
      get { return _resourceName; }
    }
    public void ClearAll()
    {
      lock (this)
      {
        _resourceCacheByCulture.Clear();
      }
    }
    public void Clear(string culture)
```

```
    {
      lock (this)
      {
        if (!_resourceCacheByCulture.ContainsKey(culture)) return;
        _resourceCacheByCulture[culture].Clear();
      }
    }
    public void Clear(string culture, string resourceKey)
    {
      lock (this)
      {
        if (!_resourceCacheByCulture.ContainsKey(culture)) return;
        _resourceCacheByCulture[culture].Remove(resourceKey);
      }
    }
    #endregion
  }
}
```

اعضای اینترفیس IDbCachedResourceManager به صورت مناسبی در کد بالا پیاده‌سازی شدند. در تمام این پیاده‌سازی‌ها مقادیر مربوطه از درون متغیر کشینگ پاک می‌شوند تا پس از اولین درخواست، بلافاصله از دیتابیس خوانده شوند. برای جلوگیری از دسترسی هم‌زمان نیز از بلاک lock استفاده شده است.

برای استفاده از این امکانات جدید همان‌طور که در بالا نیز اشاره شد باید بتوان نمونه‌های تولیدی از کلاس DbResourceManager توسط ASP.NET درون متغیری استاتیک ذخیره شوند. برای اینکار از کلاس جدیدی با عنوان DbResourceCacheManager استفاده می‌شود که برخلاف تمام کلاس‌های تعریف‌شده تا اینجا با سطح دسترسی public تعریف می‌شود.

.

### کلاس DbResourceCacheManager

مدیریت نمونه‌های تولیدی از کلاس DbResourceManager در این کلاس انجام می‌شود. این کلاس پیاده‌سازی ساده‌ای به‌صورت زیر دارد:

```
using System.Collections.Generic;
using System.Linq;
namespace DbResourceProvider
{
  public static class DbResourceCacheManager
  {
    internal static List<IDbCachedResourceManager> ResourceManagers { get; private set; }
    static DbResourceCacheManager()
    {
      ResourceManagers = new List<IDbCachedResourceManager>();
    }
    public static void ClearAll()
    {
      ResourceManagers.ForEach(r => r.ClearAll());
    }
    public static void Clear(string resourceName)
    {
      GetResouceManagers(resourceName).ForEach(r => r.ClearAll());
    }
    public static void Clear(string resourceName, string culture)
    {
      GetResouceManagers(resourceName).ForEach(r => r.Clear(culture));
    }
    public static void Clear(string resourceName, string culture, string resourceKey)
    {
      GetResouceManagers(resourceName).ForEach(r => r.Clear(culture, resourceKey));
    }

    private static List<IDbCachedResourceManager> GetResouceManagers(string resourceName)
    {
      return ResourceManagers.Where(r => r.ResourceName.ToLower() == resourceName.ToLower()).ToList();
    }
```

```
  }
}
```

ازآنجاکه نیازی به تولید نمونه ای از این کلاس وجود ندارد، این کلاس به صورت استاتیک تعریف شده است. بنابراین تمام اعضای درون آن نیز استاتیک هستند.

از پراپرتی ResourceManagers برای نگهداری لیستی از نمونه‌های تولیدی از کلاس DbResourceManager استفاده می‌شود. این پراپرتی از نوع List<IDbCachedResourceManager> تعریف شده است و برای جلوگیری از دسترسی بیرونی، سطح دسترسی آن internal درنظر گرفته شده است.

در کانستراکتور استاتیک این کلاس (اطلاعات بیشتر درباره static constructor در اینجا ) این پراپرتی با مقداردهی به یک نمونه تازه از لیست، اصطلاحا initialize می‌شود.

سایر متدها نیز برای فراخوانی متدهای موجود در اینترفیس IDbCachedResourceManager پیاده‌سازی شده‌اند. تمامی این متدها دارای سطح دسترسی public هستند. همان‌طور که می‌بینید از خاصیت ResourceName برای مشخص‌کردن نمونه موردنظر استفاده شده است که دلیل آن در قسمت قبل شرح داده شده است.

دقت کنید که برای اطمینان از انتخاب درست همه موارد موجود در شرط انتخاب نمونه موردنظر در متد GetResouceManagers از متد ToLower برای هر دو سمت شرط استفاده شده است.

.

**نکته مهم:** درباره علت برگشت یک لیست از متد انتخاب نمونه موردنظر از کلاس DbResourceManager در کد بالا (یعنی متد GetResouceManagers) باید نکته‌ای اشاره شود. در قسمت قبل عنوان شد که سیستم مدیریت منابع ASP.NET نمونه‌های تولیدی از پرووایدرهای منابع را به ازای هر منبع کش می‌کند. اما یک نکته بسیار مهم که باید به آن توجه کرد این است که این کش برای «عبارات بومی‌سازی **ضمنی** » و نیز «متد مربوط به **منابع محلی** » موجود در کلاس HttpContext و یا نمونه مشابه آن در کلاس TemplateControl (همان متد GetLocalResourceObject که درباره این متدها در قسمت سوم این سری شرح داده شده است) از یکدیگر جدا هستند و استفاده از هریک از این دو روش موجب تولید یک نمونه مجزا از پرووایدر مربوطه می‌شود که متاسفانه کنترل آن از دست برنامه نویس خارج است. دقت کنید که این اتفاق برای منابع کلی رخ نمی‌دهد.

بنابراین برای پاک کردن مناسب ورودی‌های کش‌شده در کلاس فوق به جای استفاده از متد Single در انتخاب نمونه موردنظر از کلاس DbResourceManager (در متد GetResouceManagers) از متد Where استفاده شده و یک لیست برگشت داده می‌شود. چون با توجه به توضیح بالا امکان وجود دو نمونه DbResourceManager از یک منبع درخواستی **محلی** در لیست نمونه‌های نگهداری شده در این کلاس وجود دارد.

.

### افزودن نمونه‌ها به کلاس DbResourceCacheManager

برای نگهداری نمونه‌های تولید شده از DbResourceManager، باید در یک قسمت مناسب این نمونه‌ها را به لیست مربوطه در کلاس DbResourceCacheManager اضافه کرد. بهترین مکان برای انجام این عمل در کلاس پایه BaseDbResourceProvider است که درخواست تولید نمونه را در متد EnsureResourceManager درصورت نال بودن آن می‌دهد. بنابراین این متد را به صورت زیر تغییر می‌دهیم:

```
private void EnsureResourceManager()
{
  if (_resourceManager != null) return;
  {
    _resourceManager = CreateResourceManager();
    DbResourceCacheManager.ResourceManagers.Add(_resourceManager);
  }
}
```

تا اینجا کار پیاده‌سازی امکان مدیریت مقادیر کش‌شده در کتابخانه تولیدی به پایان رسیده است.

.

**استفاده از کلاس DbResourceCacheManager**

پس از پیاده‌سازی تمامی موارد لازم، حالتی را درنظر بگیرید که مقادیر ورودی‌های تعریف شده در منبع "dir1/page1.aspx" تغییر کرده است. بنابراین برای بروزرسانی مقادیر کش‌شده کافی است تا از کدی مثل کد زیر استفاده شود:

```
DbResourceCacheManager.Clear("dir1/page1.aspx");
```

کد بالا کل ورودی‌های کش‌شده برای منبع "dir1/page1.aspx" را پاک می‌کند. برای پاک کردن کالچر یا یک ورودی خاص نیز می‌توان از کدهایی مشابه زیر استفاده کرد:

```
DbResourceCacheManager.Clear("Default.aspx", "en-US");
DbResourceCacheManager.Clear("GlobalTexts", "en-US", "Yes");
```

.

**دریافت کد پروژه**

کد کامل پروژه DbResourceProvider به همراه مثال و اسکریپت‌های دیتابیسی مربوطه از لینک زیر قابل دریافت است:

DbResourceProvider.rar

برای استفاده از این مثال ابتدا باید کتابخانه Entity Framework (با نام EntityFramework.dll) را مثلا از طریق نوگت دریافت کنید. نسخه‌ای که من در این مثال استفاده کردم نسخه 4.4 با حجم حدود 1 مگابایت است.

**نکته:** در این کد یک بهبود جزئی اما مهم در کلاس ResourceData اعمال شده است. در قسمت سوم این سری، اشاره شد که نام ورودی‌های منابع Case Sensitive **نیست** . بنابراین برای پیاده‌سازی این ویژگی، متدهای این کلاس باید به صورت زیر تغییر کنند:

```
public Resource GetResource(string resourceKey, string culture)
{
  using (var data = new TestContext())
  {
    return data.Resources.SingleOrDefault(r => r.Name.ToLower() == _resourceName.ToLower() &&
r.Key.ToLower() == resourceKey.ToLower() && r.Culture == culture);
  }
}
```

```
public List<Resource> GetResources(string culture)
{
  using (var data = new TestContext())
  {
    return data.Resources.Where(r => r.Name.ToLower() == _resourceName.ToLower() && r.Culture ==
culture).ToList();
  }
}
```

تغییرات اعمال شده همان استفاده از متد ToLower در دو طرف شرط مربوط به نام منابع و کلید ورودی‌هاست.

.

**در آینده...**

در ادامه مطالب، بحث تهیه پرووایدر سفارشی فایلهای resx. برای پیاده‌سازی امکان به‌روزرسانی در زمان اجرا ارائه خواهد شد. بعد از پایان تهیه این پرووایدر سفارشی، این سری مطالب با ارائه نکات استفاده از این پرووایدرها در ASP.NET MVC پایان خواهد یافت.

.

**منابع**

http://msdn.microsoft.com/en-us/library/aa905797.aspx

http://www.west-wind.com/presentations/wwdbresourceprovider

## نظرات خوانندگان

نویسنده: محسن خان
تاریخ: ۱۳۹۲/۰۳/۱۲ ۱۴:۲۳

با تشکر از زحمات شما.

یک بهبود جزئی: مطابق Managed Threading Best Practices بهتره از lock this استفاده نشه و از یک شیء object خصوصی استفاده شود.

.Use caution when locking on instances, for example lock(this) in C# or SyncLock(Me) in Visual Basic
.If other code in your application, external to the type, takes a lock on the object, deadlocks could occur

نویسنده: یوسف نژاد
تاریخ: ۱۳۹۲/۰۳/۱۲ ۱۴:۳۶

مطلب شما کاملا صحیح است.
ممنون بابت یادآوری.

نویسنده: علیرضا همتی
تاریخ: ۱۳۹۲/۰۳/۲۹ ۲۳:۵۷

سلام و تشکر از زحمات شما.
من نتوانستم از این پروایدر در displayAttribute و بقیه اتریبیوتها استفاده کنم. لطفا من و راهنمایی کنید.

نویسنده: یوسف نژاد
تاریخ: ۱۳۹۲/۰۳/۳۰ ۱۰:۳۰

متاسفانه امکان استفاده مستقیم از این پرووایدرهای سفارشی در این attributeها در MVC میسر نیست. این attributeها به جای استفاده از پرووایدر منابع برای استخراج مقادیر ورودی‌ها طوری طراحی شده اند که با استفاده از Reflection از داده‌های ارائه شده مقادیر را از کلاس و پراپرتی مربوطه استخراج کنند. بنابراین در این attributeها نمیتوان جایی برای استفاده از پرووایدرهای منابع یافت.

برای حل این مشکل چندین راه حل وجود دارد:
مثلا attributeهای موردنیاز توسط خود برنامه نویس پیاده‌سازی شوند.
یا اینکه یک کلاس مخصوص ایجاد کرد و استخراج مقادیر ورودی‌های منابع را در آن پیاده‌سازی کرد و در attributeهای موردنیاز از نام این کلاس و پراپرتی‌های درون آن استفاده کرد.
یا اگر از فایلهای resx. استفاده می‌شود یک ابزار سفارشی برای تولید کلاس مرتبط با منبع اصلی مثل ابزار توکار ویژوال استودیو (PublicResXFileCodeGenerator) تولید کرد تا کلاس‌های تولیدی به جای استفاده از ResourceManager از پرووایدر منابع استفاده کند (با استفاده از متدهای موجود در HttpContext).
البته این روش‌ها برای حل مشکلات مربوطه در MVC در ادامه این سری شرح داده می‌شوند.

نویسنده: علیرضا همتی
تاریخ: ۱۳۹۲/۰۳/۳۰ ۱۳:۱۱

ممنون از شما.

نویسنده: محسن موسوی
تاریخ: ۱۳۹۲/۰۶/۰۳ ۱۷:۳۹

با تشکر از زحمات شما
اینجا بیان شده زمانیکه از اسمبلی دیگری برای resourceها استفاده میکنید فقط میتوان **global resources** را پوشش داد.
بنابراین برای استفاده از کلاس LocalDbResourceProvider بایستی تغییراتی صورت بگیره.
چونکه همیشه این متد

```
using System.Web.Compilation;

namespace DbResourceProvider
{
  internal class DbResourceProviderFactory : ResourceProviderFactory
  {
    #region Overrides of ResourceProviderFactory

    public override IResourceProvider CreateGlobalResourceProvider(string classKey)
    {
      return new GlobalDbResourceProvider(classKey);
    }

    ...
  }
}
```

اجرا میشود.

نویسنده: صابر فتح الهی
تاریخ: ۱۳۹۲/۰۷/۰۸ ۹:۵۸

مهندس عزیز با تشکر از کار گرانقدر شما
یک سوال؟
چگونه می‌توان الگوی کار را در این پروایدر گنجاند؟
آیا اصلا چنین امکانی دارد یا خیر؟