

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

مدیریت منابع در ASP.NET

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

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

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



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

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

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

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

چرا پرووایدر سفارشی؟

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

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

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

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

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

مدل پرووایدر منابع

همانطور که قبلاً هم اشاره شد، وظیفه استخراج داده‌ها از Resourceها به صورت پیش‌فرض، در نهایت بر عهده نمونه‌ای از کلاس ResourceManager است. در واقع این کلاس کل فرایند انتخاب مناسب‌ترین کلید از منابع موجود را با توجه به کالچر رابط کاربری (UI Culture) در ثرد جاری کپسوله می‌کند. درباره این کلاس در ادامه بیشتر بحث خواهد شد.

هم‌چنین بازهم همانطور که قبلاً توضیح داده شد، استفاده از ورودی‌های منابع موجود به دو روش انجام می‌شود. استفاده از عبارات بومی‌سازی و نیز با استفاده از برنامه‌نویسی که از طریق دومتد GetGlobalResourceObject و GetLocalResourceObject انجام می‌شود. در ضمن کلیه عبارات بومی‌سازی در زمان رندر صفحات وب در نهایت تبدیل به فراخوانی‌هایی از این دو متد در کلاس TemplateControl خواهند شد.

عملیات پس از فراخوانی این دو متد جایی است که مدل Resource Provider پیش‌فرض ASP.NET وارد کار می‌شود. این فرایند ابتدا با فراخوانی نمونه‌ای از کلاس ResourceProviderFactory آغاز می‌شود که پیاده‌سازی پیش‌فرض آن در کلاس ResXResourceProviderFactory قرار دارد.

این کلاس سپس با توجه به نوع منبع درخواستی (Global یا Local) نمونه‌ای از پرووایدر مربوطه (که باید اینترفیس IResourceProvider را پیاده‌سازی کرده باشند) را تولید می‌کند. پیاده‌سازی پیش‌فرض این پرووایدرها در ASP.NET در کلاس‌های GlobalResXResourceProvider و LocalResXResourceProvider قرار دارد.

این پروایدرها در نهایت باتوجه به محل ورودی درخواستی، نمونه مناسب از کلاس ResourceManager را تولید و استفاده می‌کنند. هم‌چنین در پروایدرهای محلی، برای استفاده از عبارات بومی‌سازی ضمنی، نمونه‌ای از کلاس ResourceReader مورد استفاده قرار می‌گیرد. در زمان تجزیه و تحلیل صفحه وب درخواستی در سرور، با استفاده از این کلاس کلیدهای موردنظر یافته می‌شوند. این کلاس درواقع پیاده‌سازی اینترفیس IResourceReader بوده که حاوی یک Enumerator که جفت داده‌های Key-Value از کلیدهای Resource را برمی‌گرداند، است.

تصویر زیر نمایی کلی از فرایند پیش‌فرض موردبحث را نشان می‌دهد:



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

پیاده‌سازی‌ها

کلاس `ResourceProviderFactory` به صورت زیر تعریف شده است:

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

همانطور که مشاهده می‌کنید دو متد برای تولید پرووایدرهای مخصوص منابع کلی و محلی در این کلاس وجود دارد. پرووایدر کلی تنها نیاز به نام کلید `Resource` برای یافتن داده موردنظر دارد. اما پرووایدر محلی به مسیر صفحه درخواستی برای اینکار نیاز دارد که با توجه به توضیحات ابتدای این مطلب کاملاً بدیهی است.

پس از تولید پرووایدر موردنظر با استفاده از متد مناسب با توجه به شرایط شرح داده شده در بالا، نمونه تولیدشده از کلاس پرووایدر موردنظر وظیفه فراهم کردن کلیدهای `Resource` را برعهده دارد. پرووایدرهای موردبحث باید اینترفیس `IResourceProvider` را که به صورت زیر تعریف شده است، پیاده سازی کنند:

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

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

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

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

کلاس ResXResourceProviderFactory

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

```
// Type: System.Web.Compilation.ResXResourceProviderFactory
// Assembly: System.Web, Version=4.0.0.0, Culture=neutral, PublicKeyToken=b03f5f7f11d50a3a
// Assembly location:
C:\Windows\Microsoft.NET\assembly\GAC_32\System.Web\v4.0.4.0.0__b03f5f7f11d50a3a\System.Web.dll

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

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

کلاس GlobalResXResourceProvider

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

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

```

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

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

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

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

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

کلاس LocalResXResourceProvider

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

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

```

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

```

عملیات موجود در این کلاس باتوجه به فرایندهای مربوط به یافتن اسمبلی مربوطه با استفاده از مسیر ارائه شده، کمی پیچیده تر از کلاس قبلی است.

در متد `GetLocalResourceAssembly` عملیات یافتن اسمبلی متناظر با درخواست جاری انجام می شود. اینکار باتوجه به نحوه نامگذاری اسمبلی منابع محلی که در ابتدای این مطلب اشاره شد انجام می شود. مثلاً اگر صفحه درخواستی در مسیر `~/SubDir1/Page1.aspx` باشد، در این متد با استفاده از ابزارهای موجود عنوان اسمبلی نهایی برای این مسیر که به صورت `App_LocalResources.SubDir1.XXXXX` است تولید و در نهایت اسمبلی مربوطه استخراج می شود. در ضمن در اینجا هم کلاس `ResourceManager` برای نوع داده متناظر با نام فایل اصلی منبع محلی تولید می شود. مثلاً برای مسیر مجازی `~/SubDir1/Page1.aspx` نوع داده ای با نام `Page1.aspx` در نظر گرفته خواهد شد (با توجه به نام فایل منبع محلی که باید به صورت `Page1.aspx.resx` باشد. در [قسمت قبل](#) در این باره شرح داده شده است).

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

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

نکته: باتوجه به استفاده از عبارات بومی سازی ضمنی در استفاده از ورودی های منابع محلی، خاصیت `ResourceReader` در این کلاس نمونه ای متناظر برای درخواست جاری از کلاس `ResourceReader` با استفاده از `Stream` استخراج شده از اسمبلی یافته شده، تولید می کند.

کلاس پایه `BaseResXResourceProvider`

کلاس پایه `BaseResXResourceProvider` که در دو پیاده سازی نشان داده شده در بالا استفاده شده است (هر دو کلاس از این کلاس مشتق شده اند)، به صورت زیر است:

```

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

```



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

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

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

کلاس ResourceManager

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

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

کلاس ResourceManager به صورت زیر نمونه سازی می‌شود:

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

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

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

یا

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


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

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

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

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

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

نمونه‌سازی مناسب از ResourceManager

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

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

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

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

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

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

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

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

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

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

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

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

از جمله نکات مهم این متد تولید یک نمونه از کلاس VirtualPath برای Parse کردن مسیر مجازی وارد شده برای صفحه درخواستی است. از این کلاس برای بدست آوردن نام فایل منبع محلی به همراه مسیر فولدر مربوطه جهت استخراج اسمبلی متناظر استفاده میشود.

نکته مهم دیگر این کد دسترسی به متد GetLocalResourcesAssemblyName از کلاس BuildManager است که با استفاده از مسیر فولدر مربوط به صفحه درخواستی نام اسمبلی منبع محلی مربوطه را برمی گرداند. در نهایت با استفاده از متد GetBuildResultFromCache از کلاس BuildManager اسمبلی مورد نظر بدست می آید. همانطور که از نام این متد برمی آید این اسمبلی از کش خوانده می شود. البته مدیریت این اسمبلی ها کاملاً توسط BuildManager و سایر ابزارهای موجود در ASP.NET انجام خواهد شد.

نحوه استفاده از متد فوق نیز به صورت زیر است:

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

نکته: ارائه و شرح کدهای پیاده سازی های پیش فرض برای آشنایی با نحوه صحیح سفارشی سازی این کلاس ها آورده شده است. پس با دقت بیشتر بر روی این کدها سعی کنید نحوه پیاده سازی مناسب را برای سفارشی سازی مورد نظر خود پیدا کنید.

تا اینجا با مقدمات فرایند تولید پرووایدرهای سفارشی برای استفاده در فرایند بارگذاری ورودی های Resource ها آشنا شدیم. در ادامه به بحث تولید پرووایدرهای سفارشی برای استفاده از دیگر انواع منابع (به غیر از فایل های .resx) خواهیم پرداخت.

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

<http://msdn.microsoft.com/en-us/library/ms227427.aspx> <http://www.westwind.com/presentations/wfdbresourceprovider>

<http://www.codeproject.com/Articles/104667/Under-the-Hood-of-BuildManager-and-Resource-Handli>

<http://www.onpreinit.com/2009/06/updatable-aspnet-resx-resource-provider.html> [http://msdn.microsoft.com/en-us/library/h6270d0z\(v=vs.100\).aspx](http://msdn.microsoft.com/en-us/library/h6270d0z(v=vs.100).aspx)

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