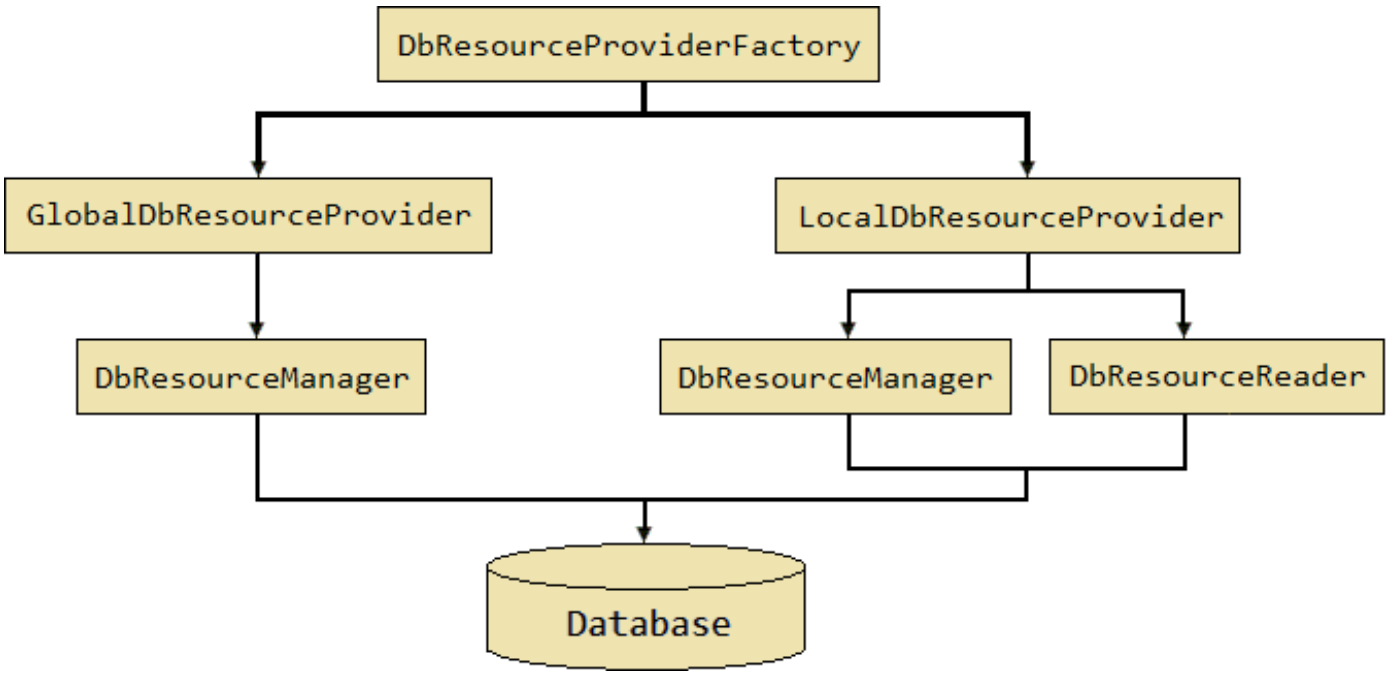
عنوان: **Globalization در ASP.NET MVC - قسمت پنجم**
نویسنده: یوسف نژاد
تاریخ: ۱۳۹۲/۰۳/۰۴ ۱۳:۵۰
آدرس: www.dotnettips.info
برچسب‌ها: Localization, Globalization, Internationalization, Resource, ResourceProviderFactory, Custom Resource Provider, Database Resource Provider

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