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؟