عنوان: Globalization در ASP.NET MVC - قسمت دوم
نویسنده: یوسف نژاد
تاریخ: ۱۳۹۲/۰۱/۳۱ ۱۳:۵
آدرس: www.dotnettips.info
گروه‌ها: MVC, ASP.Net MVC, Globalization, Resource, Updateable Resource

به‌روزرسانی فایلهای Resource در زمان اجرا یکی از ویژگیهای مهمی که در پیاده سازی محصول با استفاده از فایلهای Resource باید به آن توجه داشت، امکان بروز رسانی محتوای این فایلها در زمان اجراست. از آنجاکه احتمال اینکه کاربران سیستم خواهان تغییر این مقادیر باشند بسیار زیاد است، بنابراین درنظر گرفتن چنین ویژگی‌ای برای محصول نهایی میتواند بسیار تعیین کننده باشد. متاسفانه پیاده سازی چنین امکانی درباره فایلهای Resource چندان آسان نیست. زیرا این فایلها همانطور که در قسمت قبل توضیح داده شد پس از کامپایل به صورت اسمبلی‌های ستلایت (Satellite Assembly) درآمده و دیگر امکان تغییر محتوای آنها بصورت مستقیم و به آسانی وجود ندارد.

نکته: البته نحوه پیاده سازی این فایلها در اسمبلی نهایی (و در حالت کلی نحوه استفاده از هر فایلی در اسمبلی نهایی) در ویژوال استودیو توسط خاصیت Build Action تعیین میشود. برای کسب اطلاعات بیشتر راجع به این خاصیت به اینجا رجوع کنید.

یکی از روشهای نسبتا من‌درآوردی که برای ویرایش و به روزرسانی کلیدهای Resource وجود دارد بدین صورت است:
- ابتدا باید اصل فایلهای Resource به همراه پروژه پابلیش شود. بهترین مکان برای نگهداری این فایلها فولدر App_Data است. زیرا محتویات این فولدر توسط سیستم FCN (همان *File Change Notification*) در ASP.NET رصد نمیشود.
نکته: علت این حساسیت این است که FCN در ASP.NET تقریبا تمام محتویات فولدر سایت در سرور (فولدر App_Data یکی از معدود استثناهاست) را تحت نظر دارد و رفتار پیشفرض این است که با هر تغییری در این محتویات، AppDomain سایت Unload میشود که پس از اولین درخواست دوباره Load میشود. این اتفاق موجب از دست دادن تمام سشن‌ها و محتوای کش‌ها و ... میشود (اطلاعات بیشتر و کاملتر درباره نحوه رفتار FCN در اینجا).
- سپس با استفاده یک مقدار کدنویسی امکاناتی برای ویرایش محتوای این فایلها فراهم شود. ازآنجا که محتوای این فایلها به صورت XML ذخیره میشود بنابراین براحتی میتوان با امکانات موجود این ویژگی را پیاده سازی کرد. اما در فضای نام System.Windows.Forms کلاسهایی وجود دارد که مخصوص کار با این فایلها طراحی شده اند که کار نمایش و ویرایش محتوای فایلهای Resource را ساده‌تر میکند. به این کلاسها در قسمت قبلی اشاره کوتاهی شده بود.
- پس از ویرایش و به روزرسانی محتوای این فایلها باید کاری کنیم تا برنامه از این محتوای تغییر یافته به عنوان منبع جدید بهره بگیرد. اگر از این فایلهای Rsource به صورت embed استفاده شده باشد در هنگام build پروژه محتوای این فایلها به صورت Satellite Assembly در کنار کتابخانه‌های دیگر تولید میشود. اسمبلی مربوط به هر زبان هم در فولدری با عنوان زبان مربوطه ذخیره میشود. مسیر و نام فایل این اسمبلی‌ها مثلا به صورت زیر است:

```
bin\fa\Resources.resources.dll
```

بنابراین در این روش برای استفاده از محتوای به روز رسانی شده باید عملیات Build این کتابخانه دوباره انجام شود و کتابخانه‌های جدیدی تولید شود. راه حل اولی که به ذهن میرسد این است که از ابزارهای پایه و اصلی برای تولید این کتابخانه‌ها استفاده شود. این ابزارها (همانطور که در قسمت قبل نیز توضیح داده شد) عبارتند از Resource Generator و Assembly Linker. اما استفاده از این ابزارها و پیاده سازی روش مربوطه سختتر از آن است که به نظر می‌آید. خوشبختانه درون مجموعه عظیم دات نت ابزار مناسبتری برای این کار نیز وجود دارد که کار تولید کتابخانه‌های موردنظر را به سادگی انجام میدهد. این ابزار با عنوان Microsoft Build شناخته میشود که در اینجا توضیح داده شده است.

خواندن محتویات یک فایل resx.

همانطور که در بالا توضیح داده شد برای راحتی کار میتوان از کلاس زیر که در فایل System.Windows.Forms.dll قرار دارد استفاده کرد:

```
System.Resources.ResXResourceReader
```

این کلاس چندین کانستراکتور دارد که مسیر فایل resx. یا استریم مربوطه به همراه چند گزینه دیگر را به عنوان ورودی میگیرد. این کلاس یک Enumator دارد که یک شی از نوع IDictionaryEnumerator برمیگرداند. هر عضو این enumerator از نوع object است. برای استفاده از این اعضا ابتدا باید آنرا به نوع DictionaryEntry تبدیل کرد. مثلا بصورت زیر:

```
private void TestResXResourceReader()
{
  using (var reader = new ResXResourceReader("Resource1.fa.resx"))
  {
    foreach (var item in reader)
    {
      var resource = (DictionaryEntry)item;
      Console.WriteLine("{0}: {1}", resource.Key, resource.Value);
    }
  }
}
```

همانطور که ملاحظه میکنید استفاده از این کلاس بسیار ساده است. ازآنجاکه DictionaryEntry یک struct است، به عنوان یک راه حل مناسبتر بهتر است ابتدا کلاسی به صورت زیر تعریف شود:

```
public class ResXResourceEntry
{
  public string Key { get; set; }
  public string Value { get; set; }
  public ResXResourceEntry() { }
  public ResXResourceEntry(object key, object value)
  {
    Key = key.ToString();
    Value = value.ToString();
  }
  public ResXResourceEntry(DictionaryEntry dictionaryEntry)
  {
    Key = dictionaryEntry.Key.ToString();
    Value = dictionaryEntry.Value != null ? dictionaryEntry.Value.ToString() : string.Empty;
  }
  public DictionaryEntry ToDictionaryEntry()
  {
    return new DictionaryEntry(Key, Value);
  }
}
```

سپس با استفاده از این کلاس خواهیم داشت:

```
private static List<ResXResourceEntry> Read(string filePath)
{
  using (var reader = new ResXResourceReader(filePath))
  {
    return reader.Cast<object>().Cast<DictionaryEntry>().Select(de => new
ResXResourceEntry(de)).ToList();
  }
}
```

حال این متد برای استفاده‌های آتی آماده است.

نوشتن در فایل resx.

برای نوشتن در یک فایل resx. میتوان از کلاس ResXResourceWriter استفاده کرد. این کلاس نیز در کتابخانه System.Windows.Forms در فایل System.Windows.Forms.dll قرار دارد:

System.Resources.ResXResourceWriter

متاسفانه در این کلاس امکان افزودن یا ویرایش یک کلید به تنهایی وجود ندارد. بنابراین برای ویرایش یا اضافه کردن حتی یک کلید کل فایل باید دوباره تولید شود. برای استفاده از این کلاس نیز میتوان به شکل زیر عمل کرد:

```
private static void Write(IEnumerable<ResXResourceEntry> resources, string filePath)
{
  using (var writer = new ResXResourceWriter(filePath))
  {
```

```
    foreach (var resource in resources)
    {
      writer.AddResource(resource.Key, resource.Value);
    }
  }
}
```

در متد فوق از همان کلاس ResXResourceEntry که در قسمت قبل معرفی شد، استفاده شده است. از متد زیر نیز میتوان برای حالت کلی حذف یا ویرایش استفاده کرد:

```
private static void AddOrUpdate(ResXResourceEntry resource, string filePath)
{
  var list = Read(filePath);
  var entry = list.SingleOrDefault(l => l.Key == resource.Key);
  if (entry == null)
  {
    list.Add(resource);
  }
  else
  {
    entry.Value = resource.Value;
  }
  Write(list, filePath);
}
```

در این متد از متدهای Read و Write که در بالا نشان داده شده‌اند استفاده شده است.

حذف یک کلید در فایل resx.

برای اینکار میتوان از متد زیر استفاده کرد:

```
private static void Remove(string key, string filePath)
{
  var list = Read(filePath);
  list.RemoveAll(l => l.Key == key);
  Write(list, filePath);
}
```

در این متد، از متد Write که در قسمت معرفی شد، استفاده شده است.

راه حل نهایی

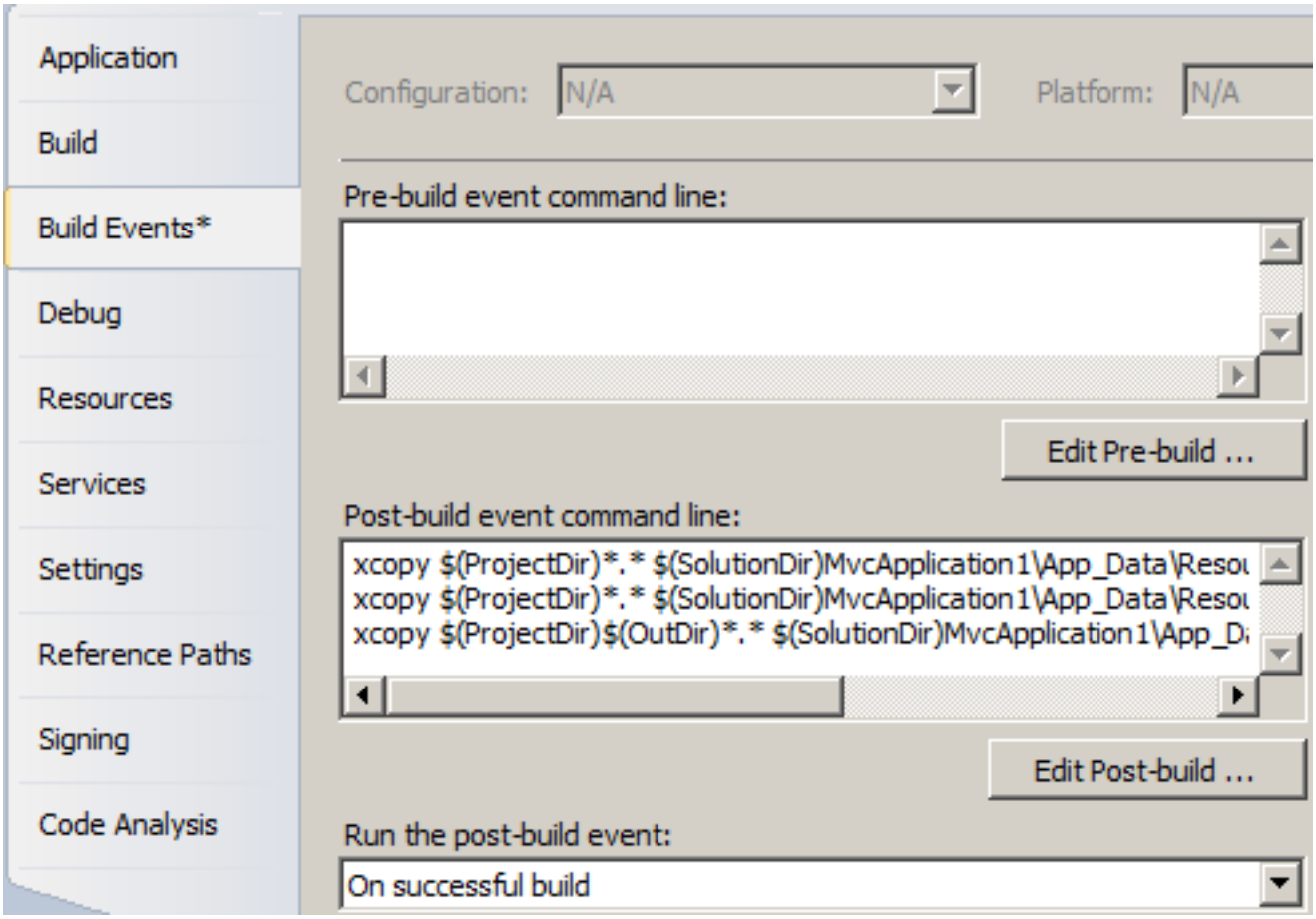
قبل از بکارگیری روشهای معرفی شده در این مطلب بهتر است ابتدا یکسری قرارداد بصورت زیر تعریف شوند:

- طبق راهنماییهای موجود در قسمت قبل یک پروژه جداگانه با عنوان Resources برای نگهداری فایلهای resx. ایجاد شود.

- همواره آخرین نسخه از محتویات موردنیاز از پروژه Resources باید درون فولدری با عنوان Resources در پوشه App_Data قرار داشته باشد.

- آخرین نسخه تولیدی از محتویات موردنیاز پروژه Resource در فولدری با عنوان Defaults در مسیر App_Data\Resources برای فراهم کردن امکان "بازگرداندن به تنظیمات اولیه" وجود داشته باشد.

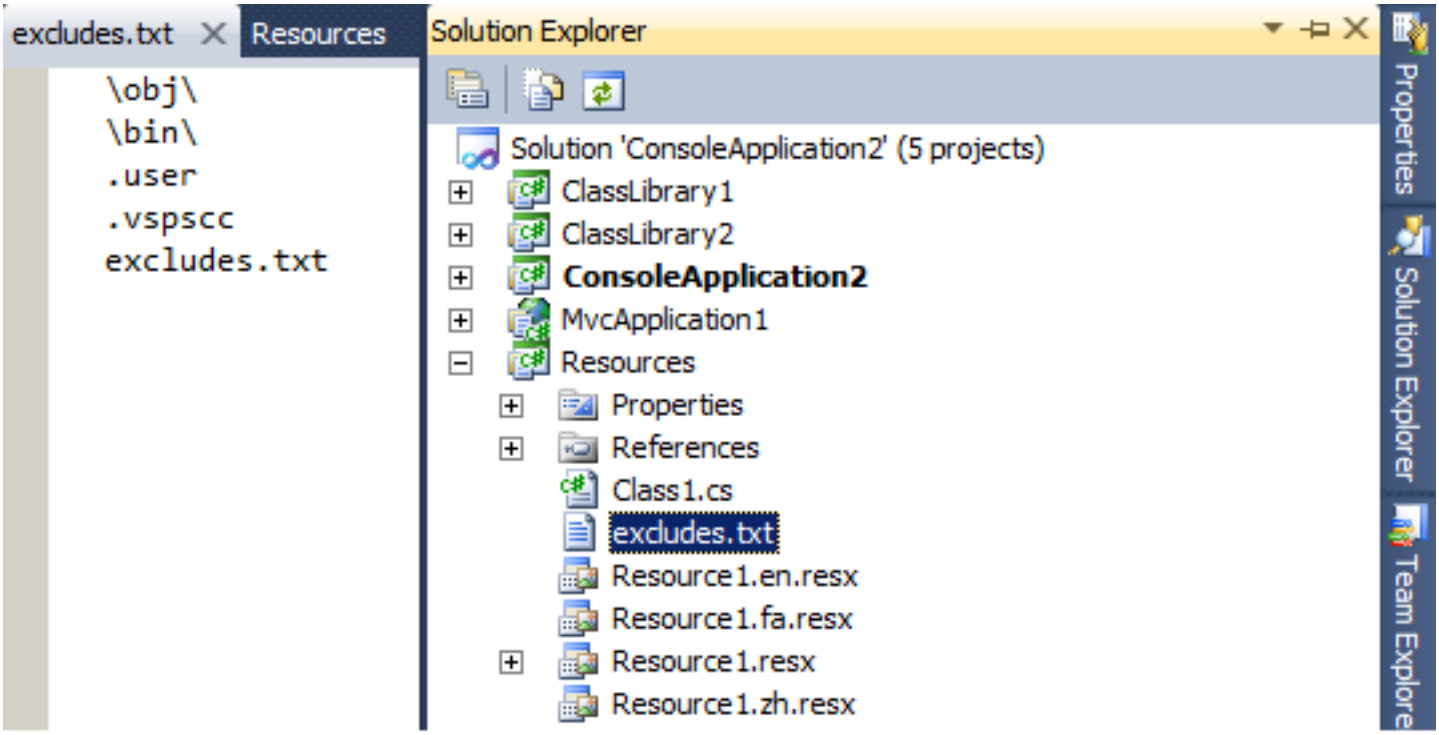
برای فراهم کردن این موارد بهترین راه حل استفاده از تنظیمات Post-build event command line است. اطلاعات بیشتر درباره Build Eventها در اینجا .



برای اینکار من از دستور xcopy استفاده کردم که نسخه توسعه یافته دستور copy است. دستورات استفاده شده در این قسمت عبارتند از:

```
xcopy $(ProjectDir)*.* $(SolutionDir)MvcApplication1\App_Data\Resources /e /y /i
/exclude:$(ProjectDir)excludes.txt
xcopy $(ProjectDir)*.* $(SolutionDir)MvcApplication1\App_Data\Resources\Defaults /e /y /i
/exclude:$(ProjectDir)excludes.txt
xcopy $(ProjectDir)$(OutDir)*.* $(SolutionDir)MvcApplication1\App_Data\Resources\Defaults\bin /e /y /i
```

در دستورات فوق آرگومان e/ برای کپی تمام فولدرها و زیرفولدرها، y/ برای تایید تمام کانفیرم ها، و i/ برای ایجاد خودکار فولدرهای موردنیاز استفاده میشود. آرگومان exclude/ نیز همانطور که از نامش پیداست برای خارج کردن فایلها و فولدرهای موردنظر از لیست کپی استفاده میشود. این آرگومان مسیر یک فایل متنی حاوی لیست این فایلها را دریافت میکند. در تصویر زیر یک نمونه از این فایل و مسیر و محتوای مناسب آن را مشاهده میکنید:



با استفاده از این فایل excludes.txt فولدرهای bin و obj و نیز فایلهای با پسوند user. و vspscc. (مربوط به TFS) و نیز خود فایل excludes.txt از لیست کپی دستور xcopy حذف میشوند و بنابراین کپی نمیشوند. درصورت نیاز میتوانید گزینه‌های دیگری نیز به این فایل اضافه کنید.

همانطور که در اینجا اشاره شده است، در تنظیمات Post-build event command line یکسری متغیرهای ازپیش تعریف شده (Macro) وجود دارند که از برخی از آنها در دستورات فوق استفاده شده است:

$(ProjectDir) : مسیر کامل و مطلق پروژه جاری به همراه یک کاراکتر \ در انتها

$(SolutionDir) : مسیر کامل و مطلق سولوشن به همراه یک کاراکتر \ در انتها

$(OutDir) : مسیر نسبی فولدر Output پروژه جاری به همراه یک کاراکتر \ در انتها

نکته: این دستورات باید در Post-Build Event پروژه Resources افزوده شوند.

با استفاده از این تنظیمات مطمئن میشویم که پس از هر Build آخرین نسخه از فایلهای موردنیاز در مسیرهای تعیین شده کپی میشوند. درنهایت با استفاده از کلاس ResXResourceManager که در زیر آورده شده است، کل عملیات را ساماندهی میکنیم:

```
public class ResXResourceManager
{
  private static readonly object Lock = new object();
  public string ResourcesPath { get; private set; }
  public ResXResourceManager(string resourcesPath)
  {
    ResourcesPath = resourcesPath;
  }
  public IEnumerable<ResXResourceEntry> GetAllResources(string resourceCategory)
  {
    var resourceFilePath = GetResourceFilePath(resourceCategory);
    return Read(resourceFilePath);
  }
  public void AddOrUpdateResource(ResXResourceEntry resource, string resourceCategory)
  {
    var resourceFilePath = GetResourceFilePath(resourceCategory);
    AddOrUpdate(resource, resourceFilePath);
  }
  public void DeleteResource(string key, string resourceCategory)
  {
    var resourceFilePath = GetResourceFilePath(resourceCategory);
    Remove(key, resourceFilePath);
```

```
  }
  private string GetResourceFilePath(string resourceCategory)
  {
    var extension = Thread.CurrentThread.CurrentUICulture.TwoLetterISOLanguageName == "en" ? ".resx" :
".fa.resx";
    var resourceFilePath = Path.Combine(ResourcesPath, resourceCategory.Replace(".", "\\") +
extension);
    return resourceFilePath;
  }
  private static void AddOrUpdate(ResXResourceEntry resource, string filePath)
  {
    var list = Read(filePath);
    var entry = list.SingleOrDefault(l => l.Key == resource.Key);
    if (entry == null)
    {
      list.Add(resource);
    }
    else
    {
      entry.Value = resource.Value;
    }
    Write(list, filePath);
  }
  private static void Remove(string key, string filePath)
  {
    var list = Read(filePath);
    list.RemoveAll(l => l.Key == key);
    Write(list, filePath);
  }
  private static List<ResXResourceEntry> Read(string filePath)
  {
    lock (Lock)
    {
      using (var reader = new ResXResourceReader(filePath))
      {
        var list = reader.Cast<object>().Cast<DictionaryEntry>().ToList();
        return list.Select(l => new ResXResourceEntry(l)).ToList();
      }
    }
  }
  private static void Write(IEnumerable<ResXResourceEntry> resources, string filePath)
  {
    lock (Lock)
    {
      using (var writer = new ResXResourceWriter(filePath))
      {
        foreach (var resource in resources)
        {
          writer.AddResource(resource.Key, resource.Value);
        }
      }
    }
  }
}
```

در این کلاس تغییراتی در متدهای معرفی شده در قسمتهای بالا برای مدیریت دسترسی همزمان با استفاده از بلاک lock ایجاد شده است.

با استفاده از کلاس BuildManager عملیات تولید کتابخانه‌ها مدیریت میشود. (در مورد نحوه استفاده از MSBuild در اینجا توضیحات کافی آورده شده است):

```
public class BuildManager
{
  public string ProjectPath { get; private set; }
  public BuildManager(string projectPath)
  {
    ProjectPath = projectPath;
  }
  public void Build()
  {
    var regKey = Registry.LocalMachine.OpenSubKey(@"SOFTWARE\Microsoft\MSBuild\ToolsVersions\4.0");
    if (regKey == null) return;
    var msBuildExeFilePath = Path.Combine(regKey.GetValue("MSBuildToolsPath").ToString(),
"MSBuild.exe");
    var startInfo = new ProcessStartInfo
    {
      FileName = msBuildExeFilePath,
```

```
      Arguments = ProjectPath,
      WindowStyle = ProcessWindowStyle.Hidden
    };
    var process = Process.Start(startInfo);
    process.WaitForExit();
  }
}
```

درنهایت مثلا با استفاده از کلاس ResXResourceFileManager مدیریت فایلهای این کتابخانه‌ها صورت میپذیرد:

```
public class ResXResourceFileManager
{
  public static readonly string BinPath =
Path.GetDirectoryName(Assembly.GetExecutingAssembly().GetName().CodeBase.Replace("file:///", ""));
  public static readonly string ResourcesPath = Path.Combine(BinPath, @"..\App_Data\Resources");
  public static readonly string ResourceProjectPath = Path.Combine(ResourcesPath, "Resources.csproj");
  public static readonly string DefaultsPath = Path.Combine(ResourcesPath, "Defaults");
  public static void CopyDlls()
  {
    File.Copy(Path.Combine(ResourcesPath, @"bin\debug\Resources.dll"), Path.Combine(BinPath,
"Resources.dll"), true);
    File.Copy(Path.Combine(ResourcesPath, @"bin\debug\fa\Resources.resources.dll"),
Path.Combine(BinPath, @"fa\Resources.resources.dll"), true);
    Directory.Delete(Path.Combine(ResourcesPath, "bin"), true);
    Directory.Delete(Path.Combine(ResourcesPath, "obj"), true);
  }
  public static void RestoreAll()
  {
    RestoreDlls();
    RestoreResourceFiles();
  }
  public static void RestoreDlls()
  {
    File.Copy(Path.Combine(DefaultsPath, @"bin\Resources.dll"), Path.Combine(BinPath, "Resources.dll"),
true);
    File.Copy(Path.Combine(DefaultsPath, @"bin\fa\Resources.resources.dll"), Path.Combine(BinPath,
@"fa\Resources.resources.dll"), true);
  }
  public static void RestoreResourceFiles(string resourceCategory)
  {
    RestoreFile(resourceCategory.Replace(".", "\\"));
  }
  public static void RestoreResourceFiles()
  {
    RestoreFile(@"Global\Configs");
    RestoreFile(@"Global\Exceptions");
    RestoreFile(@"Global\Paths");
    RestoreFile(@"Global\Texts");

    RestoreFile(@"ViewModels\Employees");
    RestoreFile(@"ViewModels\LogOn");
    RestoreFile(@"ViewModels\Settings");

    RestoreFile(@"Views\Employees");
    RestoreFile(@"Views\LogOn");
    RestoreFile(@"Views\Settings");
  }

  private static void RestoreFile(string subPath)
  {
    File.Copy(Path.Combine(DefaultsPath, subPath + ".resx"), Path.Combine(ResourcesPath, subPath +
".resx"), true);
    File.Copy(Path.Combine(DefaultsPath, subPath + ".fa.resx"), Path.Combine(ResourcesPath, subPath +
".fa.resx"), true);
  }
}
```

در این کلاس از مفهومی با عنوان resourceCategory برای استفاده راحتتر در ویوها استفاده شده است که بیانگر فضای نام نسبی فایلهای Resource و کلاسهای متناظر با آنهاست که براساس استانداردها باید برطبق مسیر فیزیکی آنها در پروژه باشد مثل Global.Texts یا Views.LogOn. همچنین در متد RestoreResourceFiles نمونه هایی از مسیرهای این فایلها آورده شده است.

پس از اجرای متد Build از کلاس BuildManager، یعنی پس از build پروژه Resource در زمان اجرا، باید ابتدا فایلهای تولیدی به

مسیرهای مربوطه در فولدر bin برنامه کپی شده سپس فولدرهای تولیدشده توسط msbuild، حذف شوند. این کار در متد CopyDlls از کلاسResXResourceFileManager انجام میشود. هرچند در این قسمت فرض شده است که فایل csprj. موجود برای حالت debug تنظیم شده است.

نکته: دقت کنید که در این قسمت بلافاصله پس از کپی فایلها در مقصد با توجه به توضیحات ابتدای این مطلب سایت Restart خواهد شد که یکی از ضعفهای عمده این روش به شمار میرود.

سایر متدهای موجود نیز برای برگرداندن تنظیمات اولیه بکار میروند. در این متدها از محتویات فولدر Defaults استفاده میشود.

نکته : درصورت ساخت دوباره اسمبلی و یا بازگرداندن اسمبلی‌های اولیه، از آنجاکه وب‌سایت Restart خواهد شد، بنابراین بهتر است تا صفحه جاری بلافاصله پس از اتمام عملیات،دوباره بارگذاری شود. مثلا اگر از ajax برای اعمال این دستورات استفاده شده باشد میتوان با استفاده از کدی مشابه زیر در پایان فرایند صفحه را دوباره بارگذاری کرد:

```
window.location.reload();
```

در قسمت بعدی راه حل بهتری با استفاده از فراهم کردن پرووایدر سفارشی برای مدیریت فایلهای Resource ارائه میشود.

نظرات خوانندگان

نویسنده: بهمن خلفی
تاریخ: ۱۳۹۲/۰۲/۰۱ ۹:۲۲

با سلام خدمت شما
مطلب بسیار مفیدی است و امیدوارم ادامه دهید...

نویسنده: امیرمحسن یک
تاریخ: ۱۳۹۴/۰۲/۰۴ ۲۳:۵۰

با سلام و با تشکر؛ پس از پیاده سازی مطالب در محیط لوکال، جواب صحیح را گرفتم ولی پس از آپلود نمودن سایت در محیط وب در هنگام Build نمودن پروژه پوشه fa_ir در پوشه bin ایجاد نمی‌گردد بنابراین قابلیت تغییر عناوین فارسی را ندارم. لازم به عرض که دسترسی write برای پوشه bin و تمامی زیر پوشه‌ها فعال می‌باشد. نکته بعدی اینکه فایل dll اصلی ساخته می‌شود و فقط در ساخت پوشه fa-ir که فایل dll زبان فارسی در آن می‌باشد اشکال وجود دارد.

نویسنده: محسن خان
تاریخ: ۱۳۹۴/۰۲/۰۵ ۹:۳۷

کلاس BuildManager ارائه شده در این مطلب بعیده که روی یک هاست با دسترسی کم اجرا شه. عملا دسترسی اجرای یک فایل exe رو روی هاست ندارید.

نویسنده: امیرمحسن یک
تاریخ: ۱۳۹۴/۰۲/۰۵ ۱۴:۳۹

کلاس BuildManager مگه اجرایی هست؟
موضوع اینه که مشکل من با Build پروژه نیست (Build با موفقیت انجام میشود)
موضوع عدم ایجاد پوشه fa-ir و بالطبع فایل dll داخل آن می‌باشد.
خواهشمند است اگر سوال بنده جاییش نامفهوم است بفرمائید که اصلاح نمایم تا به پاسخ برسم.

نویسنده: محسن خان
تاریخ: ۱۳۹۴/۰۲/۰۵ ۱۵:۶

کلاس BuildManager فایل **exe** .MSBuild رو اجرا می‌کنه (مطابق سورس مطلب فوق). بنابراین دو سؤال هست: آیا این فایل exe روی سرور هست؟ اگر هست آیا دسترسی Process.Start دارید؟ یعنی دسترسی دارید فایل exe اجرا کنید با برنامه‌ی ASP.NET؟

نویسنده: امیرمحسن یک
تاریخ: ۱۳۹۴/۰۲/۰۵ ۱۷:۲۴

بلی MSBuild قابلیت اجرا برروی IIS سرورهای arvixe را دارند (علت را نمی‌دانم)

مشکل بنده با Build پروژه نیست (Build با موفقیت انجام میشود و فایل dll زبان اصلی با موفقیت ساخته می‌شود)
موضوع عدم ایجاد پوشه fa-ir و بالطبع فایل dll داخل آن می‌باشد.