عنوان: **آشنایی با CLR: قسمت سیزدهم**
نویسنده: علی یگانه مقدم
تاریخ: ۱۳۹۴/۰۵/۲۸ ۱:۳۵
آدرس: www.dotnettips.info
گروه‌ها: CLR, .NET Framework

ترکیب ماژول‌ها به قالب یک اسمبلی

فایل Program.exe یک فایل PE با جداول متادیتا است که همچنین یک اسمبلی هم می‌باشد. یک اسمبلی مجموعه‌ای از یک یا چند فایل، شامل تعاریف نوع و منابع (ریسورس) می‌باشد و یکی از فایل‌های اسمبلی، برای نگهداری manifest انتخاب می‌شود. این جدول مجموعه‌ای است از جداول متادیتا که به طور کلی شامل نام فایل‌هایی است که قسمتی از اسمبلی را تشکیل می‌دهند. برای همین گفتیم که CLR با اسمبلی‌ها کار می‌کند. ابتدا جداول manifest را خوانده تا نام فایل‌ها را شناسایی کرده تا از آن‌ها را به حافظه بارگزاری کند. اسمبلی‌ها چند خصوصیت دارند که باید آن‌ها را بدانید:

- نوع‌های با قابلیت استفاده‌ی مجدد را تعریف می‌کنند.
- داری شماره‌ی نسخه version هستند.
- می‌توانند شامل اطلاعات امنیتی باشند.

این خواصی است که یک اسمبلی به همراه دارد و فایل‌هایی که شامل می‌شود، نمی‌توانند چنین خاصیتی را داشته باشند؛ مگر اینکه آن فایل‌ها در متای خود جدول manifest داشته باشند.

شما برای بسته بندی، شماره نسخه، مباحث امنیتی و استفاده از نوع‌ها، باید آن‌ها را داخل ماژولی قرار دهید که جزئی از اسمبلی است. یک فایل اسمبلی همانند program.exe به عنوان یک فایل واحد شناخته می‌شود. با اینکه یک اسمبلی از چند فایل تشکیل می‌شود، فایل‌های PE به همراه جداول متادیتای آن و تعدادی ریسورس مثل فایل‌های gif و jpg است که به شما کمک می‌کند به همه‌ی آن‌ها به عنوان یک فایل منطقی EXE یا dll نگاه کنید.

یکی از دلایلی که در قسمت سوم گفتیم این بود که می‌توانیم فایل‌هایی را که به ندرت استفاده می‌شوند، از طریق اینترنت مورد استفاده قرار دهیم. در حالتیکه نیاز به دسترسی به اسمبلی‌های روی اینترنت دارید، CLR ابتدا کش را بررسی می‌کند تا آیا فایل حاضر است یا خیر؟ اگر پاسخ مثبت بود، در حافظه قرار می‌گیرد. ولی اگر پاسخ منفی بود، CLR به آدرسی که اسمبلی در آن قرار دارد، رجوع کرده و آن را دانلود می‌کند و اگر فایل مد نظر یافت نشد، استثنای FileNotFound را در حین اجرا صادر خواهد کرد.

آقای جفری ریچر در کتاب خود سه تا از دلایل استفاده‌ی از اسمبلی‌های چند فایله را بر می‌شمارد:
جداسازی نوع‌ها در فایل‌های جداگانه که باعث کاهش حجم فایل از طریق اینترنت و بارگزاری حجم کمتر در حافظه می‌شوند.

استفاده از فایل‌های منبع و داده‌ها در اسمبلی: فرض کنید نیاز به محاسبه‌ی اطلاعات بیمه دارید و برای این کار به اطلاعات داخل یک جدول آماری احتیاج دارید. این جدول آماری می‌تواند یک فایل متنی ساده یا یک صفحه‌ی گسترده مثل اکسل یا در قالب ورد و هر چیز دیگری باشد که به جای embed شدن این جداول در سورس کد برنامه، آن‌ها را با استفاده از ابزاری مثل Assembly Linker -AL.exe می‌توانید جزئی از اسمبلی کنید و فقط نیاز است که بدانید چگونه آن فایل را پارس یا تبدیل کنید.

استفاده از انواع ایجاد شده در زبان‌های مختلف. در این حالت شما مقداری از کد را با استفاده از C# نوشته اید و مقداری از آن را با Visual Basic می‌نویسید و هر کدام در نهایت به یک ماژول جداگانه کامپایل خواهند شد. ولی تبدیل آن به یک واحد منطقی مثل اسمبلی ممکن است و از این نظر می‌توانید روی ماژول‌های یک دسته کنترل داشته باشید.

اگر چندین نوع دارید که شامل نسخه بندی و تنظیمات امنیتی مشترک هستند، بهتر است در یک اسمبلی قرار گیرند تا اینکه در اسمبلی‌های جداگانه‌ایی قرار بگیرند. دلیل این کار هم ایجاد performance یا کارآیی بهتر است. بارگذاری یک اسمبلی در حافظه زمانی را برای یافتن آن از CLR و ویندوز می‌گیرد و سپس وارد بارگیری آن‌ها در حافظه و آماده سازی می‌شود. پس هر چه تعداد اسمبلی‌ها کمتر باشد، کارآیی بهتری خواهید داشت، چون کمتر شدن بارگیری برابر با کاهش صفحات کاری است و پراکندگی fragmentation فضای آدرس دهی آن فرایند را کاهش خواهد داد. نهایتا Ngen می‌تواند در بهینه سازی فایل‌های بزرگتر موفق باشد.

برای ساخت اسمبلی، باید یکی از فایل‌های PE را برای نگهداری جدول manifest انتخاب کنید؛ یا خودتان یک فایل PE جدا درست کنید که تنها شامل جدول مانیفست شود. جدول زیر قالبی از جداول مانیفست هست که بابت ماژول‌های اضافه شده به یک

اسمبلی ایجاد می‌شوند.

AssemblyDef	شامل مدخل ورودی (آدرس شروع حافظه) برای اسمبلی‌هایی است که ماژول عضو آن است. این مدخل شامل نام اسمبلی (بدون مسیر و پسوند)، شماره نسخه یا ورژن، culture، فلگ، الگوریتم هش و کلید عمومی ناشر، که می‌تواند نال باشد، هست.
FileDef	شامل یک مدخل ورودی برای هر فایل PE و فایل‌های ریسورسی است که قسمتی از اسمبلی را تشکیل می‌دهند. این مدخل ورودی شامل نام و پسوند فایل (بدون ذکر مسیر)، فلگ و مقدار هش می‌شود. اگر تنها یک اسمبلی وجود داشته باشد، این جدول هیچ مدخلی نخواهد داشت.
ManifestResourceDef	شامل یک مدخل ورودی برای هر فایل ریسورس است. این مدخل شامل نام فایل ریسورس، فلگ و یک اندیس به جدول FileDef است که در آن اشاره‌ای به آن فایل ریسورس یا استریم است.
ExportedTypesDef	شامل یک مدخل ورودی برای هر نوع عمومی است که از همه ماژول‌های PE استخراج شده است. هر مدخل شامل نام نوع و اندیسی به جدول FileDef و یک اندیس دیگر به جدول TypeDef است. نکته: برای ذخیره سازی حافظه و کم حجم شدن فایل‌ها، نوع‌های استخراج شده از فایلی که شامل مانیفست است دیگر در جدول جاری نام نوع‌ها ذکر نمی‌گردد؛ چرا که این اطلاعات در جدول TypeDef اسمبلی جاری موجود است.

نکته: اسمبلی که شامل مانیفست است، شامل یک جدول AssemblyRef نیز می‌گردد که به تمام اسمبلی‌های ارجاع شده در آن اسمبلی اشاره می‌کند. با استفاده از ابزارهای موجود می‌توان اسمبلی مدنظر را باز کرده و به این ترتیب لیستی از اسمبلی‌های ارجاع شده را خواهید دید و بدین صورت این اسمبلی یک اسمبلی خود تعریف می‌شود.

کامپایلر سی شارپ با استفاده از سوئیچ‌های زیر یک اسمبلی را تولید می‌کند:

```
/t[arget]:exe, /t[arget]:winexe, /t[arget]: appcontainerexe, /t[arget]: library, or /t[arget]:winmdobj
```

سوئیچ‌های بالا باعث می‌شود که یک فایل PE با جدول مانیفست تولید گردد. در صورتیکه سوئیچ زیر را به کار ببرید، فایل تولید شده شامل جدول مانیفست نمی‌شود.

```
/t[arget]:module
```

این فایل PE تولید شده در قالب یک dll است که باید قبل از اینکه CLR به نوع‌های داخل آن دسترسی پیدا کند، به یک اسمبلی اضافه گردد. موقعی‌که شما از سوئیچ بالا استفاده می‌کنید، کامپایلر سی شارپ به طور پیش فرض از پسوند netmodule برای فایل خروجی استفاده می‌کند.

نکته‌ی پایانی: محیط توسعه ویژوال استادیو به طور پیش فرض از اسمبلی‌های چند فایل پشتیبانی نمی‌کند، اگر میخواهید که اسمبلی‌های چند فایله تولید کنید باید در سوئیچ‌های مورد استفاده آن تجدید نظری داشته باشید.

در مقاله آینده این روش‌ها را بررسی خواهیم کرد...