عنوان: **آشنایی با CLR: قسمت هشتم**
نویسنده: علی یگانه مقدم
تاریخ: ۱۳۹۴/۰۳/۲۹ ۹:۱۰
آدرس: www.dotnettips.info
گروه‌ها: CLR, .NET Framework

در قسمت پنجم در مورد ابزار Ngen کمی صحبت کردیم و در این قسمت هم در مورد آن صحبت هایی خواهیم کرد. گفتیم که این ابزار در زمان نصب، اسمبلی‌ها را کامپایل می‌کند تا در زمان اجرا JIT وقتی برای آن نگذارد. این کار دو مزیت به همراه دارد:

بهینه سازی زمان آغاز به کار برنامه
کاهش صفحات کاری برنامه: از آنجا که برنامه از قبل کامپایل شده، فراهم کردن صفحه بندی از ابتدای کار امر چندان دشواری نخواهد بود؛ لذا در این حالت صفحه بندی حافظه به صورت پویاتری انجام می‌گردد. شیوه‌ی کار به این صورت است که اسمبلی‌ها به چندین پروسه‌ی کاری کوچک‌تر تبدیل شده تا صفحه بندی هر کدام جدا صورت گیرد و محدوده‌ی صفحه بندی کوچکتر می‌شود. در نتیجه کمتر نقصی در صفحه بندی دیده شده یا کلا دیده نخواهد شد. نتیجه‌ی کار هم در یک فایل ذخیره می‌گردد که این فایل می‌تواند نگاشت به حافظه شود تا این قسمت از حافظه به طور اشتراکی مورد استفاده قرار گیرد و بدین صورت نیست که هر پروسه‌ای برای خودش قسمتی را گرفته باشد.
موقعی که اسمبلی، کد IL آن به کد بومی تبدیل می‌شود، یک اسمبلی جدید ایجاد شده که این فایل جدید در مسیر زیر قرار می‌گیرد:

```
%SystemRoot%\Assembly\NativeImages_v4.0.#####_64
```

نام دایرکتوری اطلاعاتی شامل نسخه CLR و اطلاعاتی مثل اینکه برنامه بر اساس چه نسخه‌ای 32 یا 64 بیت کامپایل شده است.

**معایب**
احتمالا شما پیش خود می‌گویید این مورد فوق العاده امکان جالبی هست. کدها از قبل تبدیل شده‌اند و دیگر فرآیند جیت صورت نمی‌گیرد. در صورتیکه ما تمامی امکانات یک CLR مثل مدیریت استثناءها و GC و ... را داریم، ولی غیر از این یک مشکلاتی هم به کارمان اضافه می‌شود که در زیر به آنها اشاره می‌کنیم:

*عدم محافظت از کد در برابر بیگانگان:* بعضی‌ها تصور می‌کنند که این کد را می‌توانند روی ماشین شخصی خود کامپایل کرده و فایل ngen را همراه با آن ارسال کنند. در این صورت کد IL نخواهد بود ولی موضوع این هست اینکار غیر ممکن است و هنوز استفاده از اطلاعات متادیتاها پابرجاست به خصوص در مورد اطلاعات چون reflection و serializationها . پس کد IL کماکان همراهش هست. نکته‌ی بعدی اینکه انتقال هم ممکن نیست؛ بنا به شرایطی که در مورد بعدی دلیل آن را متوجه خواهید شد.

*از سینک با سیستم خارج میشوند:* موقعیکه CLR، اسمبلی‌ها را به داخل حافظه بار می‌کند، یک سری خصوصیات محیط فعلی را با زمانیکه عملیات تبدیل IL به کد ماشین صورت گرفته است، چک می‌کند. اگر این خصوصیات هیچ تطابقی نداشته باشند، عملیات JIT همانند سابق انجام می‌گردد. خصوصیات و ویژگی‌هایی که چک می‌شوند به شرح زیر هستند:
ورژن CLR: در صورت تغییر، حتی با پچ‌ها و سرویس پک ها.
نوع پردازنده: در صورت تغییر پردازنده یا ارتقا سخت افزاری.
نسخه سیستم عامل : ارتقاء با سرویس پک ها.
MVID یا Assemblies Identity module Version Id: در صورت کامپایل مجدد تغییر می‌کند.
Referenced Assembly's version ID: در صورت کامپایل مجدد اسمبلی ارجاع شده.
تغییر مجوزها: در صورتی که تغییری نسبت به اولین بار رخ دهد؛ مثلا در قسمت قبلی در مورد اجازه نامه اجرای کدهای ناامن صحبت کردیم. برای نمونه اگر در همین اجازه نامه تغییری رخ دهد، یا هر نوع اجازه نامه دیگری، برنامه مثل سابق (جیت) اجرا خواهد شد.

پی نوشت: در آپدیت‌های دات نت فریم ورک به طور خودکار ابزار ngen صدا زده شده و اسمبلی‌ها مجددا کمپایل و دخیره میشوند و برنامه سینک و آپدیت باقی خواهد ماند.

*کارایی پایین کد در زمان اجرا:* استفاده از ngen از ابتدا قرار بود کارآیی را با حذف جیت بالا ببرد، ولی گاهی اوقات در بعضی شرایط ممکن نیست. کدهایی که ngen تولید می‌کند به اندازه‌ی جیت بهینه نیستند. برای مثال ngen نمی‌تواند بسیاری از دستورات خاص پردازنده را جز در زمان runtime مشخص کند. همچنین فیلدهایی چون static را از آنجا که نیاز است آدرس واقعی آن‌ها در زمان اجرا به دست بیاید، مجبور به تکنیک و ترفند میشود و موارد دیگری از این قبیل.
پس حتما نسخه‌ی ngen شده و غیر ngen را بررسی کنید و کارآیی هر دو را با هم مقایسه کنید. برای بسیاری از برنامه‌ها کاهش صفحه بندی یک مزیت و باعث بهبود کارآیی می‌شود. در نتیجه در این قسمت ngen برنده اعلام می‌شود.

توجه کنید برای سیستم‌هایی که در سمت سرور به فعالیت می‌پردازند، از آنجا که تنها اولین درخواست برای اولین کاربر کمی زمان می‌برد و برای باقی کاربران درخواست با سرعت بالاتری اجرا می‌گردد و اینکه برای بیشتر برنامه‌های تحت سرور از آنجا که تنها یک نسخه در حال اجراست، هیچ مزیت صفحه بندی را ngen ایجاد نمی‌کند.

برای بسیاری از برنامه‌های کلاینت که تجربه‌ی startup طولانی دارند، مایکروسافت ابزاری را به نام Managed Profile Guided Optimization Tool یا MPGO.exe دارد. این ابزار به تحلیل اجرای برنامه شما پرداخته و بررسی می‌کند که در زمان آغازین برنامه چه چیزهایی نیاز است. اطلاعات به دست آمده از تحلیل به سمت ngen فرستاده شده تا کد بومی بهینه‌تری تولید گردد. موقعیکه شما آماده ارائه برنامه خود هستید، برنامه را از طریق این تحلیل و اجرا کرده و با قسمت‌های اساسی برنامه کار کنید. با این کار اطلاعاتی در مورد اجرای برنامه در داخل یک پروفایل embed شده در اسمبلی، قرار گرفته و ngen موقع تولید کد، این پروفایل را جهت تولید کد بهینه مطالعه خواهد کرد.

در مقاله‌ی بعدی در مورد FCL صحبت‌هایی خواهیم کرد.