عنوان: **آشنایی با CLR: قسمت ششم**
نویسنده: علی یگانه مقدم
تاریخ: ۱۳۹۴/۰۳/۲۷ ۰:۵۵
آدرس: www.dotnettips.info
گروه‌ها: CLR, .NET Framework

در مقاله قبلی مبحث کامپایلر JIT را آغاز کردیم. در این قسمت قصد داریم مبحث کارآیی CLR و مباحث دیباگینگ را پیش بکشیم.
از آنجا که یک کد مدیریت نشده، مبحث کارهای JIT را ندارد، ولی CLR مجبور است وقتی را برای آن بگذارد، به نظر می‌رسد ما با یک نقص کوچک در کارآیی روبرو هستیم. گفتیم که جیت کدها را در حافظه‌ی پویا ذخیره می‌کند. به همین خاطر با terminate شدن یا خاتمه دادن به برنامه، این کدها از بین می‌روند یا اینکه اگر دو نمونه از برنامه را اجرا کنیم، هر کدام جداگانه کد را تولید می‌کنند و هر کدام برای خودشان حافظه‌ای بر خواهند داشت و اگر مقایسه‌ای با کدهای مدیریت نشده داشته باشید، در مورد مصرف حافظه یک مشکل ایجاد می‌کند. همچنین JIT در حین تبدیل به کدهای بومی یک بهینه سازی روی کد هم انجام میدهد که این بهینه سازی وقتی را به خود اختصاص می‌دهد ولی همین بهینه سازی کد موجب کارآیی بهتر برنامه می‌گردد.
در زبان سی شارپ دو سوئیچ وجود دارند که بر بهینه سازی کد تاثیر گذار هستند؛ سوئیچ‌های debug و optimize. در جدول زیر تاثیر هر یک از سوئیچ‌ها را بر کیفیت کد IL و JIT در تبدیل به کد بومی را نشان میدهد.

Compiler Switch Settings	C# IL Code Quality	JIT Native Code Quality
/optimize- /debug- (this is the default)	Unoptimized	Optimized
/optimize- /debug(+/full/pdbonly)	Unoptimized	Unoptimized
/optimize+ /debug(-/+/full/pdbonly)	Optimized	Optimized

موقعیکه از دستور optimize- استفاده می‌شود، کد IL تولید شده شامل تعداد زیادی از دستورات بدون دستورالعمل No Operation یا به اختصار NOP و پرش‌های شاخه‌ای به خط کد بعدی می‌باشد. این دستور العمل‌ها ما را قادر میسازند تا ویژگی edit & Continue را برای دیباگ کردن و یک سری دستورالعمل‌ها را برای کدنویسی راحت‌تر برای دیباگ کردن و ایجاد break pointها داشته باشیم.

موقعی که کد IL بهینه شده تولید شود، این خصوصیات اضافه حذف خواهند شد و دنبال کردن خط به خط کد، کار سختی می‌شود. ولی در عوض فایل نهایی exe یا dll، کوچکتر خواهد شد. بهینه سازی IL توسط JIT حذف خواهد شد و برای کسانی که دوست دارند کدهای IL را تحلیل و آنالیز کنند، خواندنش ساده‌تر و آسان‌تر خواهد بود.

نکته‌ی بعدی اینکه موقعیکه شما از سوئیچ debug(+/full/pdbonly/) استفاده می‌کنید، یک فایل PDB یا Program Database ایجاد می‌شود. این فایل به دیباگرها کمک می‌کند تا متغیرهای محلی را شناسایی و به کدهای IL متصل شوند. کلمه‌ی full بدین معنی است که JIT می‌تواند دستورات بومی را ردیابی کند تا مبداء آن کد را پیدا کند. سبب می‌شود که ویژوال استودیو به یک دیباگر متصل شده تا در حین اجرای پروسه، آن را دیباگ کند. در صورتی که این سوئیچ را استفاده نکنید، به طور پیش فرض پروسه اجرا و مصرف حافظه کمتر می‌شود. اگر شما پروسه‌ای را اجرا کنید که دیباگر به آن متصل شود، به طور اجباری JIT مجبور به انجام عملیات ردیابی خواهد شد؛ مگر اینکه گزینه‌ی suppress jit optimization on module load را غیرفعال کرده باشید.
موقعیکه در ویژوال استودیو دو حالت دیباگ و ریلیز را انتخاب می‌کنید، در واقع تنظیمات زیر را اجرا می‌کنید:

```
//debug

/optimize-
/debug:full

//=======================
```

```
//Release

/optimize+
/debug:pdbonly
```

احتمالا موارد بالا به شما می‌گویند که یک سیستم مبتنی بر CLR مشکلات زیادی دارد که یکی از آن‌ها، زمان‌بر بودن انجام عملیات فرآیند پردازش است و دیگری مصرف زیاد حافظه و عدم اشترک حافظه که در مورد کامپایل جیت به آن اشاره کردیم. ولی در بند بعدی قصد داریم نظرتان را عوض کنم.

اگر خیلی شک دارید که واقعا یک برنامه‌ی CLR کارآیی یک برنامه را پایین می‌آورد، بهتر هست به بررسی کارآیی چند برنامه غیر آزمایشی noTrial که حتی خود مایکروسافت آن برنامه‌ها را ایجاد کرده است بپردازید و آن‌ها را با یک برنامه‌ی unmanaged مقایسه کنید. قطعا باعث تعجب شما خواهد شد. این نکته دلایل زیادی دارد که در زیر تعدادی از آن‌ها را بررسی می‌کنیم.
اینکه CLR در محیط اجرا قصد کمپایل دارد، باعث آشنایی کامپایلر با محیط اجرا می‌گردد. از این رو تصمیماتی را که می‌گیرد، می‌تواند به کارآیی یک برنامه کمک کند. در صورتیکه یک برنامه‌ی unmanaged که قبلا کمپایل شده و با محیط‌های متفاوتی که روی آن‌ها اجرا میشود، هیچ آشنایی ندارد و نمیتواند از آن محیطها حداکثر بهره‌وری لازم را به عمل آورد.
برای آشنایی با این ویژگی‌ها توجه شما را به نکات ذیل جلب می‌کنم:

یک. JIT می‌تواند با نوع پردازنده آشنا شود که آیا این پردازنده از نسل پنتیوم 4 است یا نسل Core i. به همین علت می‌تواند از این مزیت استفاده کرده و دستورات اختصاصی آن‌ها را به کار گیرد، تا برنامه با performance بالاتری اجرا گردد. در صورتی که unmanaged باید حتما دستورات را در پایین‌ترین سطح ممکن و عمومی اجرا کند؛ در صورتیکه شاید یک دستور اختصاصی در یک سی پی یو خاص، در یک عملیات موجب 4 برابر، اجرای سریعتر شود.

دو. JIT میتواند بررسی هایی را که برابر false هستند، تشخیص دهد. برای فهم بهتر، کد زیر را در نظر بگیرید:

```
if (numberOfCPUs > 1) {
...
}
```

کد بالا در صورتیکه پردازنده تک هسته‌ای باشد یک کد بلا استفاده است که جیت باید وقتی را برای کامپایل آن اختصاص دهد؛ در صورتیکه JIT باهوش‌تر از این حرفاست و در کدی که تولید می‌کند، این دستورات حذف خواهند شد و باعث کوچکتر شدن کد و اجرای سریعتر می‌گردد.

سه. مورد بعدی که هنوز پیاده سازی نشده، ولی احتمال اجرای آن در آینده است، این است که یک کد می‌تواند جهت تصحیح بعضی موارد چون مسائل مربوط به دیباگ کردن و مرتب سازی‌های مجدد، عمل کامپایل را مجددا برای یک کد اعمال نماید.
دلایل بالا تنها قسمت کوچکی است که به ما اثبات می‌کند که چرا CLR می‌تواند کارآیی بهتری را نسبت به زبان‌های unmanaged امروزی داشته باشد. همچنین قول‌هایی از سازندگان برای بهبود کیفیت هر چه بیشتر این سیستم‌ها به گوش می‌رسد.

کارآیی بالاتر
اگر برنامه‌ای توسط شما بررسی شد و دیدید که نتایج مورد نیاز در مورد performance را نشان نمی‌دهد، می‌توانید از ابزار کمکی که مایکروسافت در بسته‌های فریمورک دات نت قرار داده است استفاده کنید. نام این ابزار Ngen.exe است و وظیفه‌ی آن این است که وقتی برنامه بر روی یک سیستم برای اولین مرتبه اجرا می‌گردد، کد همه‌ی اسمبلی‌ها را تبدیل کرده و آن‌ها روی دیسک ذخیره می‌کند. بدین ترتیب در دفعات بعدی اجرا، JIT بررسی می‌کند که آیا کد کامپایل شده‌ی اسمبلی از قبل موجود است یا خیر. در صورت وجود، عملیات کامپایل به کد بومی لغو شده و از کد ذخیره شده استفاده خواهد کرد.
نکته‌ای که باید در حین استفاده از این ابزار به آن دقت کنید این است که کد در محیط‌های واقعی اجرا چندان بهینه نیست. بعدا در مورد این ابزار به تفصیل صحبت می‌کنیم.

system.runtime.profileoptimization

کلاس بالا سبب می‌شود که CLR در یک فایل ثبت کند که چه متدهایی در حین اجرای برنامه کمپایل شوند تا در آینده در حین آغاز اجرای برنامه کامپایلر JIT بتواند همزمان این متدها را در ترد دیگری کامپایل کند. اگر برنامه‌ی شما روی یک پردازنده‌ی چند هسته‌ای اجرا می‌شود، در نتیجه اجرای سریعتری خواهید داشت. به این دلیل که چندین متد به طور همزمان در حال کمپایل شدن هستند و همزمان با آماده سازی برنامه برای اجرا اتفاق می‌افتد؛ به جای اینکه عمل کمپایل همزمان با تعامل کاربر با برنامه باشد.