آشنایی با کلاس System.GC در دات نت (نسخه PDF)

در مقـاله ای در بـاره مـدیریت حافظه و نحوه کار Garbage Collector صـحبت کردیم. اما مـدیریت حافظه تنها اختصاص به CLR ندارد و برنامه نویسان هم می توانند در زبان های برنامه نویسی مانند سی شارپ و Visual Basic به سرویس های GC دسترسی داشته باشند. این امر بوسیله کلاس System.GC امکـان پـذیر است. این کلاس در فایل mscorlib.dll قرار گرفته است. معمولاً تنها زمانی بایـد از کلاس GC استفاده کنید از در داخل برنامه و کلاس ها از منابع Unmanaged استفاده کرده اید. برای مثال، استفاده از قابلیت COM Interop یا استفاده از قابلیت Platform Invoke در زبان سی شارپ. در ابتدا برخی از اعضاء کلاس GC را بررسی می کنیم:

۱. متد AddMemoryPressure و RemoveMemoryPressure: زمان فراخوانی GC کاملاً بسـتگی به مقدار حافظه اختصاص داده شده در Heap دارد. فرض کنیـد شـئ ای داریم که مقدار کمی از حافظه Heap را در اختیار گرفته، اما داخل آن شـئ مقـدار زیادی از حافظه Unmanaged را اسـتفاده کرده ایم. GC اطلاعی از حافظه Unmanaged تخصـیص داده شـده ندارد و تنها بر اساس حافظه Managed وارد عمل می شود. بوسـیله این دو متد می تواند به CLR در مورد استفاده از حافظه Unmanaged اطلاع داد. به طور کلی زمانی که شئ ای در فضای Unmanaged قرار میگیرد، سایز آن را بوسیله این متد اطلاع رسانی کرده و بعد معمولاً در متد Finalize حذف آن را نیز مجدد بوسیله RemoveMemoryPressure اطلاع می دهند.
۲. متد Collect: بوسـیله این متد می توان GC را به صورت دستی فراخوانی کرد. امکـان آنتخاب Generation مورد نظر برای فراخوانی GC و همچنین حالت فراخوانی توسط GCCollectionMode که یک enum است وجود دارد.
۳. متد CollectionCount: می توان تعداد دفعات فراخوانی و مدیریت یک Generation توسط GC را توسط این متد بدست آورد.
۴. متد GetGeneration: شماره Generation مربوط به یک شئ را برای ما بر میگرداند.
۵. متد GetTotalMemory: فضای تخمینی اختصاص داده شـده در حافظه Heap را برای ما بر میگردانـد. می توان بوسـیله یک پارامتر boolean مشخص کرد که محاسبه فضای اختصاص داده شده باید بعد از یکبار فراخوانی GC انجام شود یا خیر.
۶. خصوصیت MaxGeneration: تعداد Generation ها را بر می گرداند که در محیط دات نت تعداد آن ها ۳ می باشد.
۷. متد SuppressFinalize: بوسیله این متد می توان یک شئ را برای جلوگیری از متد Finalize مشخص کرد.
۸. متد WaitForPendingFinalizers: اجرای Thread جاری را تا زمان اجرای متد Finalize تمام اشیاء Finalizable متوقف می کند. معمولاً این متد بعد از فراخوانی متد Collect فراخوانی می شود.

نمونه کد زیر، اطلاعاتی در مورد وضعیت حافظه Heap و همچنین اطلاعات مربوط به یک شئ به ما نمایش می دهد:

```
Console.WriteLine("Bytes on heap: {0}", GC.GetTotalMemory(false));
Console.WriteLine("Object Generations: {0}", (GC.MaxGeneration + 1));
Person person = new Person();
Console.WriteLine("Generation of person: {0}", GC.GetGeneration(person));
```

خروجی کد بالا به صورت زیر است:

```
Bytes on heap: 29868
Object Generations: 3
Generation of person: 0
```

همـانطور که گفتیم بعضـی از مواقع نیاز است که عملیات GC به صورت دستی فراخوانی شود. بوسـیله متد Collect این کـار را می توانیم انجام دهیم. البته فراخوانی متد GC.Collect تضـمینی بر فراخوانی بلافاصله عملیات GC نیست، بلکه سـرویس آماده می شود تا در اولین فرصت عملیات پاک سازی و مدیریت حافظه را اجرا کند. معمولاً در مواقع زیر عملیات پاک سازی حافظه به صورت دستی انجام می شود:

۱. برنامه شما قصد اجرای بخشی را دارد که نباید توسط اجرای Garbage Collection وقفه ای در آن ایجاد شود.
۲. شما عملیات تخصیص حافظه برای تعداد زیادی شئ را انجام داده و تصمیم دارید در اولین فرصت عملیات حذف سایر اشیاء ای که مورد استفاده واقع نمی شوند را انجام دهید.

برای اجرای GC به صورت زیر عمل می کنیم:

```
GC.Collect();
GC.WaitForPendingFinalizers();
```

بعـد از فراخوانی GC.Collect به یاد داشـته باشـید که حتماً متد WaitForPendingFinalizers رو اجرا کنیـد تا مطمئن شویـد متد Finalizer تمامی اشـیاء حـذف شـده از حافظه اجرا شده اند و با اجرای Finalizer ها وقفه ای در اجرای کـدهای شـما بوجود نمی آید. بوسـیله متد Collect می توان شماره Generation را برای اجرا مشخص کرد:

```
GC.Collect(0);
GC.WaitForPendingFinalizers();
```

بوسیله کد بالا، عملیات پاک سازی بر روی Generation ۰ انجام می شود. همچنین بوسیله Enum ای با نام GCCollectionMode نحوه اجرای GC را مشخص کنید که حاوی موارد زیر است:

۱. حالت Default: حالت Forced حالت پیش فرض است
۲. حالت Forced: به CLR می گوید که هم اکنون باید عملیات Collect اجرا شود.
۳. حالت Optimized: انتخاب زمان برای اجرا را به عهده CLR می گذارد تا بهرین زمان را برای اجرای Collect انتخاب کند.

در ادامه با یک مثال کاملتر مطلب خود را به پایان می رسانیم:

```
Console.WriteLine("Bytes on heap: {0}", GC.GetTotalMemory(false));
Console.WriteLine("Object Generations: {0}.", (GC.MaxGeneration + 1));
Person person = new Person();
Console.WriteLine("Generation of Person is: {0}", GC.GetGeneration(person));
object[] persons = new object[50000];
for (int i = 0; i < 50000; i++)
    persons[i] = new object();
GC.Collect(0, GCCollectionMode.Forced);
GC.WaitForPendingFinalizers();
Console.WriteLine("Generation of person is: {0}",
GC.GetGeneration(person));
if (persons[9000] != null)
{
    Console.WriteLine("Generation of persons[9000] is: {0}",
    GC.GetGeneration(persons[9000]));
}
else
    Console.WriteLine("persons[9000] is no longer alive.");
Console.WriteLine("Gen 0 has been swept {0} times", GC.CollectionCount(0));
Console.WriteLine("Gen 1 has been swept {0} times", GC.CollectionCount(1));
Console.WriteLine("Gen 2 has been swept {0} times", GC.CollectionCount(2));
Console.ReadLine();
```

با اجرای کد بالا، خروجی زیر نمایش داده می شود، البته با هر بار اجرا احتمال تغییر در خروجی ممکن است:

```
Bytes on heap: 38060
Object Generations: 3.
Generation of Person is: 0
Generation of person is: 1
Generation of persons[9000] is: 1
Gen 0 has been swept 1 times
Gen 1 has been swept 0 times
Gen 2 has been swept 0 times
```

در این مطلب با نحوه کار با کلاس GC و مدیریت حافظه در دات نت بشتر آشنا شدیم. امیدوارم این مطلب مورد توجه دوستان قرار گرفته باشد. **ITPRO باشید**

نویسنده: حسین احمدی

انجمن تخصصی فناوری اطلاعات ایران

مطلب اصلی