

عنوان: مقایسه بین حلقه های تکرار (foreach و for و Lambda ForEach)

نویسنده: مسعود پاکدل

تاریخ: ۲۳:۴۰ ۱۳۹۲/۰۳/۰۵

آدرس: www.dotnettips.info

برچسب‌ها: C#, foreach, Performance

به حلقه‌های تکرار زیر دقت کنید.

#1 حلقه for با استفاده از متغیر Count لیست

```
var ListOfNumber = new List<int>() { 100, 200, 300 , 400 , 500 };
for ( int i = 0 ; i < ListOfNumber.Count ; i++ )
{
    Console.WriteLine( ListOfNumber[i] );
}
```

#2 حلقه for با استفاده از متغیر یا مقدار صریح

```
var ListOfNumber = new List<int>() { 100, 200, 300 , 400 , 500 };
for ( int i = 0 ; i < 5 ; i++ )
{
    Console.WriteLine( ListOfNumber[i] );
}
```

#3 foreach ساده که احتمالا خیلی از شماها از اون استفاده می‌کنید.

```
var ListOfNumber = new List<int>() { 100, 200, 300 , 400 , 500 };
foreach ( var number in ListOfNumber )
{
    Console.WriteLine( number );
}
```

#4 Lambda ForEach که مورد علاقه بعضی‌ها از جمله خود من است.

```
var ListOfNumber = new List<int>() { 100, 200, 300 , 400 , 500 };
ListOfNumber.ForEach( number =>
{
    Console.WriteLine( number );
});
```

به نظر شما حلقه‌های بالا از نظر کارایی چه تفاوتی با هم دارند؟

تمام حلقه‌های بالا یک خروجی رو چاپ خواهند کرد ولی اگر فکر می‌کنید که هیچ تفاوتی ندارند سخت در اشتباه هستید.

هر 4 حلقه تکرار بالا رو در 21 حالت مختلف با شریط یکسان در یک سیستم تست کردیم و نتایج زیر حاصل شد.(منظور از نتایج مدت زمان اجرای هر حلقه است)

تعداد تکرار	#1 for با استفاده از متغیر Count لیست	#2 for-استفاده از متغیر	#3 foreach	#4 Lambda ForEach
1000	0.000008	0.000007	0.000014	0.000012
2000	0.000014	0.000013	0.000026	0.000022
3000	0.000019	0.000016	0.000036	0.000028
4000	0.000024	0.000022	0.000047	0.000035
5000	0.000029	0.000025	0.000058	0.000043
10,000	0.000059	0.000047	0.000117	0.000081

#4 Lambda ForEach	#3 foreach	#2-for استفاده از متغیر	#1 for با استفاده از متغیر Count لیست	تعداد تکرار
0.000161	0.000225	0.000093	0.000128	20,000
0.000233	0.000336	0.000141	0.000157	30,000
0.000310	0.000442	0.000180	0.000221	40,000
0.000307	0.000553	0.000236	0.000263	50,000
0.000773	0.001103	0.000443	0.000530	100,000
0.001531	0.002194	0.000879	0.001070	200,000
0.002308	0.003281	0.001345	0.001641	300,000
0.003083	0.004388	0.001783	0.002233	400,000
0.003873	0.005521	0.002244	0.002615	500,000
0.007767	0.011072	0.004520	0.005303	1,000,000
0.015536	0.022127	0.009074	0.010543	2,000,000
0.023268	0.033186	0.013569	0.015738	3,000,000
0.031188	0.044335	0.018113	0.021039	4000,000
0.038793	0.055521	0.022593	0.026280	5000,000
0.078482	0.111517	0.046090	0.052528	10,000,000

بررسی نتایج :

سریع ترین حلقه تکرار حلقه for با استفاده از متغیر معمولی به عنوان تعداد تکرار حلقه است.
رتبه دوم برای حلقه for همراه با استفاده از خاصیت Count لیست مورد نظر بوده است. دلیلش هم اینه که سرعت دستیابی کامپایلر به متغیرهای معمولی حتی تا 3 برابر سریع تر از دسترسی به متد get خاصیت هاست.
مهم ترین نکته این است که Lambda ForEach عملکردی بسیار بهتری نسبت به foreach معمولی داره.

پس هر گاه قصد اجرای حلقه ForEach رو برای لیست دارید و سرعت اجرا هم براتون اهمیت داره بهتره که از Lambda ForEach استفاده کنید. حالا به کد زیر دقت کنید:

```
int[] arrayOfNumbers = new int[] { 100 , 200 , 300 , 400 , 500 };
Array.ForEach<int>( arrayOfNumbers, ( int counter ) => { Console.WriteLine( counter ); } );
```

من همون حلقه بالا رو به صورت آرایه پیاده سازی کردم و برای اجرای حلقه از دستور Array.ForEach که عملکردی مشابه با List.ForEach داره استفاده کردم که نتیجه به دست اومده نشون داد که Array.ForEach از نظر سرعت به مراتب از foreach معمولی کندتر عمل می کنه. دلیلش هم اینه که کامپایلر هنگام کار با آرایه ها و اجرای اون ها به صورت حلقه، کد IL خاصی رو تولید می کنه که مخصوص کار با آرایه هاست و سرعت اون به مراتب از سرعت کد IL تولید شده برای IEnumerator ها پایین تره.

نظرات خوانندگان

نویسنده: قاسم کشاورز حداد
تاریخ: ۱۹:۲۹ ۱۳۹۲/۰۳/۰۶

خیلی برام جالب بود، ممنون از مطلب

نویسنده: محمد رعیت پیشه
تاریخ: ۲۳:۵۱ ۱۳۹۲/۰۳/۰۶

ممنون از مطلبتون.
فقط در صورت امکان توضیحی هم درباره نحوه تست کردن چنین دستوراتی بدید.

نویسنده: شاهین کیاست
تاریخ: ۸:۴۸ ۱۳۹۲/۰۳/۰۷

یک روش ساده :

```
Stopwatch sw = Stopwatch.StartNew();  
var ListOfNumber = new List<int>() { 100, 200, 300 , 400 , 500 };  
for ( int i = 0 ; i < ListOfNumber.Count ; i++ )  
{  
    Console.WriteLine( ListOfNumber[i] );  
}  
sw.Stop();  
Console.WriteLine("Total time (ms): {0}", (long) sw.ElapsedMilliseconds);
```

نویسنده: مصطفی عسگری
تاریخ: ۲۳:۳ ۱۳۹۲/۰۳/۰۷

جالب بود که این روش از foreach سریعتر عمل میکنه

نویسنده: یوسف نژاد
تاریخ: ۰:۳۳ ۱۳۹۲/۰۳/۰۸

یا استفاده از [Microbenchmark](#) برای دریافت نتایج دقیقتر.

نویسنده: یوسف نژاد
تاریخ: ۰:۴۱ ۱۳۹۲/۰۳/۰۸

متد ForEach در کلاس List از حلقه for معمولی استفاده میکنه و نه foreach:

```
public void ForEach(Action<T> action)  
{  
    if (action == null)  
        ThrowHelper.ThrowArgumentNullException(ExceptionArgument.match);  
    for (int index = 0; index < this._size; ++index)  
        action(this._items[index]);  
}
```

متد Array.ForEach هم از روشی مشابه استفاده کرده:

```
public static void ForEach<T>(T[] array, Action<T> action)
```

```
{
    if (array == null)
        throw new ArgumentNullException("array");
    if (action == null)
        throw new ArgumentNullException("action");
    for (int index = 0; index < array.Length; ++index)
        action(array[index]);
}
```

foreach به دلیل استفاده از اشیای درون IEnumerable و در نتیجه اجرای دستورات بیشتر در هر حلقه کندتر عمل میکند. اما! اگر هدف تنها بررسی سرعت اجرای حلقه‌های اشاره شده باشد متدهای بالا نتیجه درستی نشان نخواهد داد، چون عملیات انجام شده در حلقه‌های نشان داده شده با هم دقیقاً یکسان نیست. بهتره که به عملیات ثابت و مستقل از متغیرهای درگیر استفاده بشه تا نتایج دقیقتری بدست بیاد. مثلاً به چیزی مثل اکشن زیر:

```
() => { int a = 1; }
```

بهتره تو این تستها مشخصات دقیق سخت افزاری هم ارائه بشه تا مقایسه‌ها بهتر انجام بگیره. با این شرح با روشی که در مطلب [Microbenchmark](#) آورده شده آزمایشات رو دوباره انجام دادم و برای تعداد تکرار 100 میلیون اختلاف تمام حلقه‌ها در حد چند میلی ثانیه بود که کاملاً قابل صرفنظره! نتایج برای حالات مختلف موجود تفاوت‌های زیادی داشت اما در نسخه ریلیز نهایتاً نتایج کلی این بود که حلقه for معمولی از همه سریعتر، سپس Array.ForEach و بعد متد ForEach در کلاس List و در نهایت از همه کندتر حلقه foreach بود. من آزمایشات روی یک سیستم با پردازنده 4 هسته ای با کلاک 3.4 گیگاهرتز (AMD Phenom II 965) با ویندوز 7 و 32 بیتی با رم 4 گیگ (3.25 گیگ قابل استفاده) انجام دادم. متأسفانه تعداد تکرار بیشتر خطای OutOfMemory میداد. **نکته:** اجرای تستهای این چینی برای آزمایش کارایی و سرعت به شدت تحت تاثیر عوامل جانبی هستند. مثل میزان منابع در دسترس سخت افزاری، نوع سیستم عامل، برنامه‌ها و سرویس‌های در حال اجرا، و مهمتر از همه نوع نسخه بیلد شده از برنامه تستر (دیبگ یا ریلیز) و محل اجرای تست (منظور اجرا در محیط دیباگ ویزوال استودیو یا اجرای مستقل برنامه) و ... (همونطور که آقای نصیری هم مطلبی مرتبط رو به اشتراک گذاشتند [^](#))

نویسنده: مسعود م. پاکدل
تاریخ: ۱۴۰۲/۰۳/۰۸

در ابتدا بهتر عنوان کنم که در کل 2 نوع برنامه نویسی وجود داره. برنامه نویسی که می‌خواه برنامه درست کار کنه و برنامه نویسی که می‌خواه برنامه درست نوشته بشه. در این جا هدف اصلی ما نوشتن برنامه به صورت درست هستش. دلیل اینکه foreach کندتر از Lambda ForEach عمل می‌کنه همان طور که جناب یوسف نژاد عنوان کردند به خاطر اجرای دستورات بیشتر در هر تکرار است. مثل کد زیر:

```
long Sum(List<int> intList)
{
    long result = 0;
    foreach (int i in intList)
        result += i;
    return result;
}
```

کامپایلر برای انجام کامپایل، کدهای بالا رو تبدیل به کدهای قابل فهم زیر می‌کنه:

```
long Sum(List<int> intList)
{
    long result = 0;
    List<T>.Enumerator enumerator = intList.GetEnumerator();
    try
    {
        while (enumerator.MoveNext())
        {
            int i = enumerator.Current;
            result += i;
        }
    }
}
```

```

}
finally
{
    enumerator.Dispose();
}
return result;
}

```

همانطور که می بینید از دو دستور enumerator.MoveNext و enumerator.Current در هر تکرار داره استفاده می شه در حالی که List.ForEach فقط نیاز به یک فراخوانی در هر تکرار دارد.

در مورد Array.ForEach هم این نکته رو اضافه کنم که Array.ForEach فقط برای آرایه های یک بعدی استفاده میشه و کامپایلر هنگام کار با آرایه ها کد IEnumerator رو که در بالا توضیح دادم تولید نمی کنه در نتیجه در حلقه foreach برای آرایه ها هیچ فراخوانی متدی صورت نمی گیرد در حالی Array.ForEach نیاز به فراخوانی delegate تعریف شده در ForEach به ازای هر تکرار دارد.

آزمایشات بالا هم در یک سیستم DELL Inspiron 9400 با Core Duo T2400 و 2 GB RAM انجام شده است . این آزمایشات رو اگر در هر سیستم دیگر با هر Config اجرا کنید نتیجه کلی تغییر نخواهد کرد و فقط از نظر زمان اجرا تفاوت خواهیم داشت نه در نتیجه کلی.

نویسنده:

یوسف نژاد

تاریخ:

۱۳۹۲/۰۳/۱۲ ۱۲:۳۳

"این آزمایشات رو اگر در هر سیستم دیگر با هر Config اجرا کنید نتیجه کلی تغییر نخواهد کرد و فقط از نظر زمان اجرا تفاوت خواهیم داشت نه در نتیجه کلی."

این مطلب لزوما صحیح نیست. یک بنچمارک میتونه تو مجموعه سخت افزارهای مختلف، نتایج کاملا متفاوتی داشته باشه. مثلا سوالی در همین زمینه آقای [شهرز جعفری](#) تو [StackOverflow](#) پرسیدن که در جوابش دو نفر نتایج متفاوتی ارائه دادن. معمولا برای بیان نتایج تستهای بنچمارک ابتدا مشخصات سخت افزاری ارائه میشه مخصوصا وقتی که نتایج دقیق (و نه کلی) نشون داده میشه. مثل همین نتایج دقیق زمانهای اجرای حلقه ها.

نکته ای که من درکامنتم اشاره کردم صرفا درباره تست "سرعت اجرای" انواع حلقه ها بود که ممکنه با تست کارایی حلقه ها در اجرای یک کد خاص فرق داشته باشه.

نکته دیگه هم اینکه نمیدونم که آیا شما از همون متد Console.WriteLine در حلقه ها برای اجرای تستون استفاده کردین یا نه. فقط باید بگم که به خاطر مسائل و مشکلات مختلفی که استفاده از این متد به همراه داره، به نظر من بکارگیری اون تو این جور تست ها اصلا مناسب نیست و باعث دور شدن زیاد نتایج از واقعیت میشه. مثلا من تست کردم و هر دفعه یه نتیجه ای می داد که نمیشه بر اساس اون نتیجه گیری کرد.

مورد دیگه ای هم که باید اضافه کنم اینه که بهتر بود شما کد کامل تست خودتون رو هم برای داندلود میذاشتین تا دیگران هم بتونن استفاده کنن. اینجوری خیلی بهتر میشه نتایج مختلف رو با هم مقایسه کرد. این مسئله برای تستای بنچمارک نسبتا رایج هست. مثل کد زیر که من آماده کردم:

```

static void Main(string[] args)
{
    //Action<int> func = Console.WriteLine;
    Action<int> func = number => number++;
    do
    {
        try
        {
            Console.Write("Iteration: ");
            var iterations = Convert.ToInt32(Console.ReadLine());
            Console.Write("Loop Type (for:0, foreach:1, List.ForEach:2, Array.ForEach:3): ");
            var loopType = Console.ReadLine();
            switch (loopType)
            {
                case "0":
                    Console.WriteLine("FOR loop test for {0} iterations", iterations.ToString("0,0"));

```

```

        TestFor(iterations, func);
        break;
    case "1":
        Console.WriteLine("FOREACH loop test for {0} iterations", iterations.ToString("0,0"));
        TestForEach(iterations, func);
        break;
    case "2":
        Console.WriteLine("LIST.FOREACH test for {0} iterations", iterations.ToString("0,0"));
        TestListForEach(iterations, func);
        break;
    case "3":
        Console.WriteLine("ARRAY.FOREACH test for {0} iterations", iterations.ToString("0,0"));
        TestArrayForEach(iterations, func);
        break;
    }
}
catch (Exception ex)
{
    Console.WriteLine(ex);
}
Console.Write("Continue?(Y/N)");
Console.WriteLine("");
} while (Console.ReadKey(true).Key != ConsoleKey.N);

Console.WriteLine("Press any key to exit");
Console.ReadKey();
}

static void TestFor(int iterations, Action<int> func)
{
    StartupTest(func);

    var watch = Stopwatch.StartNew();
    for (int i = 0; i < iterations; i++)
    {
        func(i);
    }
    watch.Stop();
    ShowResults("for loop test: ", watch);
}

static void TestForEach(int iterations, Action<int> func)
{
    StartupTest(func);
    var list = Enumerable.Range(0, iterations);

    var watch = Stopwatch.StartNew();
    foreach (var item in list)
    {
        func(item);
    }
    watch.Stop();
    ShowResults("foreach loop test: ", watch);
}

static void TestListForEach(int iterations, Action<int> func)
{
    StartupTest(func);
    var list = Enumerable.Range(0, iterations).ToList();

    var watch = Stopwatch.StartNew();
    list.ForEach(func);
    watch.Stop();
    ShowResults("list.ForEach test: ", watch);
}

static void TestArrayForEach(int iterations, Action<int> func)
{
    StartupTest(func);
    var array = Enumerable.Range(0, iterations).ToArray();

    var watch = Stopwatch.StartNew();
    Array.ForEach(array, func);
    watch.Stop();
    ShowResults("Array.ForEach test: ", watch);
}

static void StartupTest(Action<int> func)
{
    // clean up

```

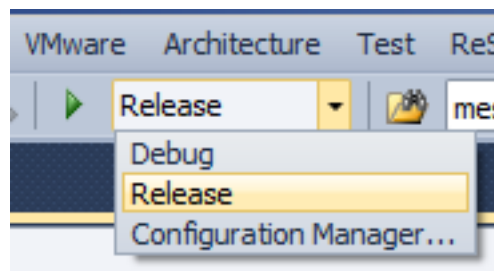
```

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

// warm up
func(0);
}
static void ShowResults(string description, Stopwatch watch)
{
    Console.WriteLine(description);
    Console.WriteLine(" Time Elapsed {0} ms", watch.ElapsedMilliseconds);
}

```

قبل از اجرای تست بهتره برنامه رو برای نسخه Release بیلد کنیم. ساده ترین روشش در تصویر زیر نشون داده شده:



پس از این تغییر و بیلد پروژه نتایج رو مقایسه میکنیم. نتایج اجرای این تست در همون سیستمی که قبلا تستای [StringBuilder](#) و [Microbenchmark](#) رو انجام دادم (یعنی لپ تاپ msi GE 620 با Core i7-2630QM) بصورت زیر:

```

C:\windows\system32\cmd.exe
Iteration: 100000000
Loop Type (for:0, foreach:1, List.ForEach:2, Array.ForEach:3): 0
FOR loop test for 100,000,000 iterations
for loop test: Time Elapsed 415 ms
Continue?(Y/N)
Iteration: 100000000
Loop Type (for:0, foreach:1, List.ForEach:2, Array.ForEach:3): 1
FOREACH loop test for 100,000,000 iterations
foreach loop test: Time Elapsed 1136 ms
Continue?(Y/N)
Iteration: 100000000
Loop Type (for:0, foreach:1, List.ForEach:2, Array.ForEach:3): 2
LIST.FOREACH test for 100,000,000 iterations
list.ForEach test: Time Elapsed 650 ms
Continue?(Y/N)
Iteration: 100000000
Loop Type (for:0, foreach:1, List.ForEach:2, Array.ForEach:3): 3
ARRAY.FOREACH test for 100,000,000 iterations
Array.ForEach test: Time Elapsed 460 ms

```

البته نتایج این تستها مطلق نیستن. نکاتی که در کامنت قبلی اشاره کردم از عوامل تاثیرگذار هستن. موفق باشین.

تاریخ: ۱۳:۳۸ ۱۳۹۲/۰۳/۱۲

شما هم در کل به این نتیجه رسیدید که list.ForEach از foreach loop سریعتر است. حلقه for معمولی نیز از تمام اینها سریعتر. بنابراین کار شما ناقض مطلب آقای پاکدل «نتیجه کلی تغییر نخواهد کرد و فقط از نظر زمان اجرا تفاوت خواهیم داشت نه در نتیجه کلی» نیست و مطلب ایشان برقرار است.

نویسنده: یوسف نژاد

تاریخ: ۱۳:۴۷ ۱۳۹۲/۰۳/۱۲

من نمیخواستم مطلبی رو نقض کنم فقط میخواستم بگم بهتره برای مقایسه نتایج اینجوری عمل بشه. درضمن نتایج بدست اومده من برای متد Array.ForEach با نتایج آقای پاکدل فرق میکنه. اما بحثی که اشاره کردم درست است و "یکسان بودن نتایج کلی با تغییر سخت افزار" همیشه برقرار نیست و برخی مواقع میتونه تفاوتهایی هم وجود داشته باشه. اما شاید تو این مثال کوچیک بهش برنخوریم اما در کل اینطوریست.

نویسنده: وحید نصیری

تاریخ: ۱۴:۰۰ ۱۳۹۲/۰۳/۱۲

در مورد تفاوت نتایج حاصل از بررسی کارآیی Array.ForEach، مطالبی در اینجا هست که علت رو بیشتر باز کرده (و دقیقاً در مثالهای جاری صادق هست؛ یکی با lambda است و دیگری بدون lambda):

[تفاوت کارآیی در حین استفاده از Lambdas و Method groups](#)

دسترسی به داده‌ها پیش شرط انجام همه‌ی منطق‌های اکثر نرم افزارهای تجاری می‌باشد. داده‌های ممکن در حافظه ، پایگاه داده ، فایل‌های فیزیکی و هر منبع دیگری قرار گرفته باشند. هنگامی که حجم داده‌ها کم باشد شاید روش دسترسی و الگوریتم مورد استفاده اهمیتی نداشته باشد اما با افزایش حجم داده‌ها روش‌های بهینه‌تر تاثیر مستقیم در کارایی برنامه دارند. در این مثال سعی بر این است که در یک سناریوی خاص تفاوت بین Dictionary و List را بررسی کنیم : فرض کنید 2 کلاس Student و Grade موجود است که وظیفه‌ی نگهداری اطلاعات دانش آموز و نمره را بر عهده دارند.

```
public class Grade
{
    public Guid StudentId { get; set; }
    public string Value { get; set; }

    public static IEnumerable<Grade> GetData()
    {
        for (int i = 0; i < 10000; i++)
        {
            yield return new Grade
            {
                StudentId = GuidHelper.ListOfIds[i], Value = "Value " + i
            };
        }
    }
}

public class Student
{
    public Guid Id { get; set; }
    public string Name { get; set; }
    public string Grade { get; set; }

    public static IEnumerable<Student> GetStudents()
    {
        for (int i = 0; i < 10000; i++)
        {
            yield return new Student
            {
                Id = GuidHelper.ListOfIds[i],
                Name = "Name " + i
            };
        }
    }
}
```

از کلاس GuidHelper برای تولید و نگهداری شناسه‌های یکتا برای دانش آموز کمک گرفته شده است :

```
public class GuidHelper
{
    public static List<Guid> ListOfIds=new List<Guid>();

    static GuidHelper()
    {
        for (int i = 0; i < 10000; i++)
        {
            ListOfIds.Add(Guid.NewGuid());
        }
    }
}
```

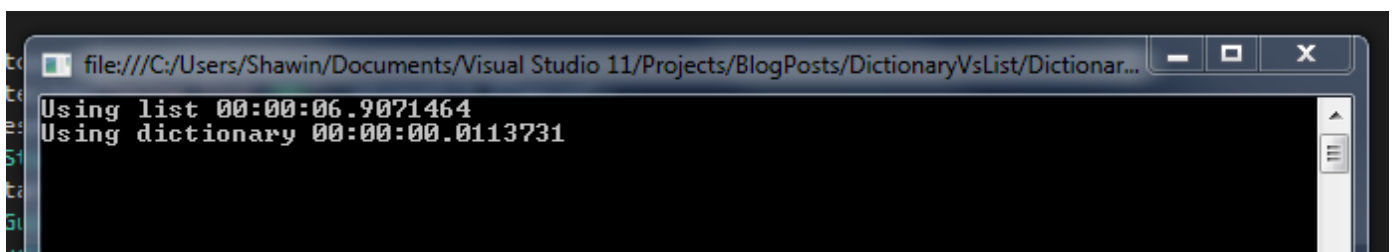
سپس لیستی از دانش آموزان و نمرات را درون حافظه ایجاد کرده و با یک حلقه نمره‌ی هر دانش آموز به Property مورد نظر مقدار داده می‌شود.

ابتدا از LINQ روی لیست برای پیدا کردن نمره‌ی مورد نظر استفاده کرده و در روش دوم برای پیدا کردن نمره‌ی هر دانش آموز از Dictionary استفاده شده :

```
internal class Program
{
    private static void Main(string[] args)
    {
        var stopwatch = new Stopwatch();
        List<Grade> grades = Grade.GetData().ToList();
        List<Student> students = Student.GetStudents().ToList();

        stopwatch.Start();
        foreach (Student student in students)
        {
            student.Grade = grades.Single(x => x.StudentId == student.Id).Value;
        }
        stopwatch.Stop();
        Console.WriteLine("Using list {0}", stopwatch.Elapsed);
        stopwatch.Reset();
        students = Student.GetStudents().ToList();
        stopwatch.Start();
        Dictionary<Guid, string> dictionary = Grade.GetData().ToDictionary(x => x.StudentId, x =>
x.Value);
        foreach (Student student in students)
        {
            student.Grade = dictionary[student.Id];
        }
        stopwatch.Stop();
        Console.WriteLine("Using dictionary {0}", stopwatch.Elapsed);
        Console.ReadKey();
    }
}
```

نتیجه‌ی مقایسه در سیستم من اینگونه می‌باشد :



همانگونه که مشاهده می‌شود در این سناریو خواندن نمره از روی Dictionary بر اساس 'کلید' بسیار سریع‌تر از انجام یک پرس و جوی LINQ روی لیست است.

زمانی که از LINQ on list

```
student.Grade = grades.Single(x => x.StudentId == student.Id).Value;
```

برای پیدا کردن مقدار مورد نظر یک به یک روی اعضا لیست حرکت می‌کند تا به مقدار مورد نظر برسد در نتیجه پیچیدگی زمانی آن $O(n)$ هست. پس هر چه میزان داده‌ها بیشتر باشد این روش کندتر می‌شود.

زمانی که از Dictionary

```
student.Grade = dictionary[student.Id];
```

برای پیدا کردن مقدار استفاده می‌شود با اولین تلاش مقدار مورد نظر یافت می‌شود پس پیچیدگی زمانی آن 1 0 می‌باشد.

در نتیجه اگر نیاز به پیدا کردن اطلاعات بر اساس یک مقدار یکتا یا کلید باشد تبدیل اطلاعات به Dictionary و خواندن از آن بسیار به صرفه‌تر است.

تفاوت این 2 روش وقتی مشخص می‌شود که میزان داده‌ها زیاد باشد.

در همین رابطه ([1](#) ، [2](#))

[DictionaryVsList.zip](#)

نظرات خوانندگان

نویسنده: حسین مرادی نیا
تاریخ: ۲۱:۳۵ ۱۳۹۲/۰۳/۱۷

یه نگاهی هم به این بندازید. جالبه: <http://stackoverflow.com/questions/1009107/what-net-collection-provides-the-fastest-search>

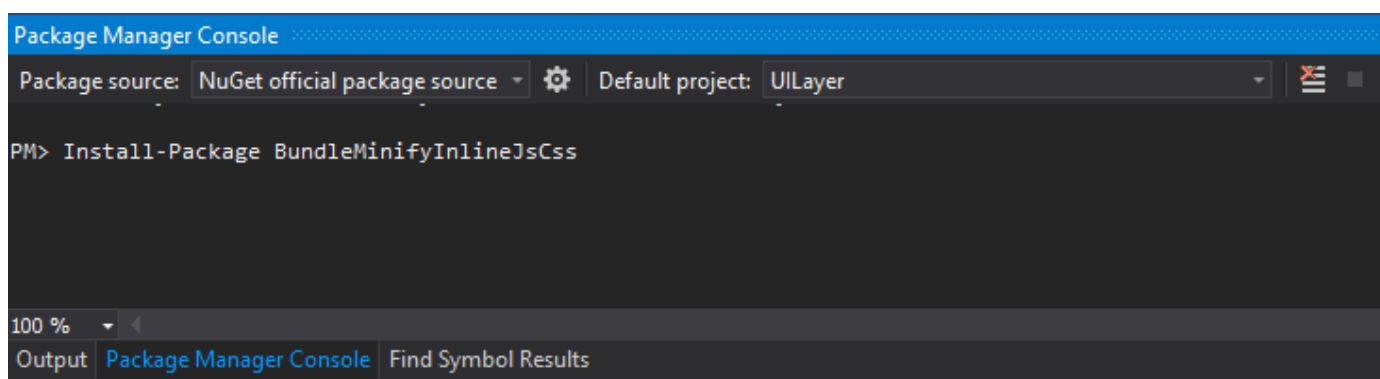
نویسنده: مهدی فرزاد
تاریخ: ۰:۲ ۱۳۹۲/۰۳/۱۸

با تشکر از دوست خوبم ، یک سؤال مطرح میشه شما این نتیجه رو از روی داده‌های موجود در حافظه انجام دادید ، اگر این داده‌ها در دیتا بیس باشه و با استفاده از یک ORM مثل EF به داده‌ها دسترسی داشته باشیم برای استفاده از Dictionary ابتدا تمام داده‌ها یک بار واکنشی شده و در نتیجه جستجو میشه؟ آیا این مطلب درسته؟ اگر آره پس نتیجه به نفع Linq تغییر میکنه

نویسنده: محسن خان
تاریخ: ۰:۲۴ ۱۳۹۲/۰۳/۱۸

نه. ToList یا ToDictionary اصطلاحاً یک نوع Projection هستند و پس از دریافت اطلاعات مطابق کوئری لینک شما اعمال خواهند شد (شکل دادن به اطلاعات دریافت شده از بانک اطلاعاتی؛ فرضاً 100 رکورد دریافت شده، حالا شما خواستید از این رکوردها برای استفاده، List درست کنید یا دیکشنری یا حالت‌های دیگر).

افزایش Performance یک سایت از موارد بسیار مهمی است که هر برنامه نویسی باید به آن توجه ویژه‌ای داشته باشد و در این زمینه لینک [Best Practices](#) می‌تواند بسیار کاربردی باشد. حال در این پست قصد داریم Style ها و Js های نوشته شده در سطح هر View را با Bundling and Minifying در Asp.Net MVC 4 بهینه نماییم. در ابتدا با استفاده از [Nuget](#) پکیج BundleMinifyInlineJsCss را به پروژه MVC خود مطابق شکل زیر اضافه می‌نماییم.



در مرحله بعدی کلاسی را با نام BundleMinifyingInlineCssJSAttribute ایجاد کرده و با ارث بردن از کلاس ActionFilterAttribute متد OnActionExecuting را override می‌نماییم. اکنون کلاس ما به شکل زیر است:

```
using System.Web;
using System.Web.Mvc;
using BundlingAndMinifyingInlineCssJs.ResponseFilters;
namespace UILayer.Filters
{
    public class BundleMinifyingInlineCssJSAttribute : ActionFilterAttribute
    {
        public override void OnActionExecuting(ActionExecutingContext filterContext)
        {
            filterContext.HttpContext.Response.Filter = new
            BundleAndMinifyResponseFilter(filterContext.HttpContext.Response.Filter);
        }
    }
}
```

و برای استفاده می‌توانیم بالای کنترلر خود کد زیر را اضافه نماییم.

```
[BundleMinifyingInlineCssJS]
public partial class HomeController : Controller
{
}
```

در ادامه پروژه را اجرا می‌کنیم. Style ها و Js های نوشته شده در سطح هر View به صورت زیر در می‌آیند.

```
1
2
3 <!DOCTYPE html>
4 <html>
5 <head>
6 </head>
7 <body>
8     <script>
9         var data1 = 'data1';
10        var data2 = 'data2'
11    </script>
12    <style type="text/css">
13        .style1
14        {
15            width: 160px;
16        }
17        .style2
18        {
19            width: 176px;
20        }
21    </style>
22    <h1>Hello</h1>
23    <script>
24        data1 = 'data1';
25        data2 = 'data2'
26    </script>
27 </body>
28 </html>
29
```

```
1
2
3 <!DOCTYPE html>
4 <html>
5 <head>
6 <style>.style1{width:160px}.style2{width:176px}</style></head>
7 <body>
8
9
10 <h1>Hello</h1>
11
12 <script>var data1="data1",data2="data2";data1="data1",data2="data2"</script></body>
13 </html>
14
```

نظرات خوانندگان

نویسنده: محسن خان
تاریخ: ۱۱:۲۶ ۱۳۹۲/۰۳/۲۴

با تشکر از این نکته جدید.

به نظر من اگر فشرده سازی Response فعال باشه اصلا نیازی به حذف فواصل خالی در HTML نهایی نیست. چون حداقل کاری رو که الگوریتم‌های فشرده سازی خوب انجام می‌دن، مدیریت فضاهای خالی است.

شاید بد نباشه به صورت یک کار تحقیقی بررسی بشه که اگر فشرده سازی رو فعال کردیم چند درصد روی حجم دریافتی تاثیر داره. اگر روش حذف فضاهای خالی رو بدون فشرده سازی اعمال کردیم، چند درصد فرقی هست.

نویسنده: م کریمی
تاریخ: ۷:۵۹ ۱۳۹۲/۰۳/۲۵

با سلام
دقیقا این همین کاریه که دارم آزمایش می‌کنم بعد از جمع آوری داده‌ها به اشتراک می‌زارم

نویسنده: ایلیا اکبری فرد
تاریخ: ۱۷:۳۲ ۱۳۹۲/۰۴/۲۷

با سلام. نحوه فشرده سازی response به چه صورت است؟ با تشکر.

نویسنده: وحید نصیری
تاریخ: ۱۸:۲ ۱۳۹۲/۰۴/۲۷

یک نمونه [در اینجا](#) ارسال شده

نویسنده: احمد پایان
تاریخ: ۱۲:۵۷ ۱۳۹۲/۰۴/۲۸

سلام ، مرسی از مطالب خوبتون
بحث Optimization در لینک زیر کاملتر بررسی شده، البته مطلب شما برایم تازگی داشت.

<http://go.microsoft.com/fwlink/?LinkId=254725>

نکاتی که میشه گفت

در وب فرم هم قابل استفاده است

برای single download کردن فایل‌های css و js دو روش وجود دارد:

1. تنظیم debug=false در بخش compilation در فایل Web.config

2. نوشتن کد زیر در کلاسی که باندل‌های خود را در bundleTable درج می‌کنید.

```
BundleTable.EnableOptimizations = true
```

برای نصب آن با NuGet Package، در کنسول عبارت زیر را وارد کنید

```
install-package Microsoft.AspNet.Web.Optimization
```


که با نصب آن، 4 کتابخانه به reference های پروژه اضافه می شود.
تفاوتی که در script ها ایجاد می کند می توان به حذف کردن description ها، تغییر در variable ها، و min کردن jz های شما اشاره کرد.
موفق باشید.

نویسنده: محسن خان
تاریخ: ۱۴:۱۰ ۱۳۹۲/۰۴/۲۸

[این موضوع](#) به بیان های مختلفی در سایت تابحال مطرح شده

[نحوه ارتقاء برنامه های موجود MVC3 به MVC4](#)

[قابلیت های کاربردی ASP.NET WebForms](#) -

[اسکرپت های خود را یکی کنید](#)

[فشرده سازی فایل های CSS و JavaScript بصورت خودکار توسط MS Ajax Minifier](#)

نویسنده: م کریمی
تاریخ: ۸:۵۸ ۱۳۹۲/۰۴/۲۹

با سلام
مباحثی همچون [Bundling and Minifying](#) قبلا توسط آقای نصیری توضیح کامل داده شده است . ما هم مباحث تکمیلی را اضافه می کنیم

در این سلسله مقالات قصد دارم چندین مطلب راجع به افزایش سرعت نرم افزارهای تحت وب مطرح نمایم. این مطالب هرچند بسیار مختصر می‌باشند ولی در کارایی و سرعت برنامه‌های شما در آینده تاثیر خواهند داشت.

1. کش کردن همیشه آخرین حربه می‌باشد

این مهم است که بخش‌های مختلف سایت شما در سطوح مختلف کش شوند (ASP.NET, Kernel, Server, Proxy Server, Browser) (...), ولی این موضوع باید همیشه آخرین حربه و نکته ای باشد که آن را در مورد سایت خود اعمال می‌کنید. یعنی همیشه مطمئن شوید ابتدا تمامی نکات مربوط به افزایش کارایی در برنامه خود را رعایت کرده اید، سپس اقدام به کش داده‌ها در سطوح مختلف نمایید. توجه کنید کش کردن داده‌ها و صفحات می‌تواند مشکلات را برای شما به عنوان یک برنامه نویس یا تست کننده برنامه پنهان کند و به شما اطمینان دهد که همه چیز خوب کار می‌کند در حالی که این چنین نیست! البته ذکر این نکته هم بی فایده نیست که کش کردن همه چیز بعضی مواقع دشمن برنامه شما محسوب می‌شود! هیچ وقت یادم نمی‌رود، در پورتال داخلی یک شرکت که در وقت استراحت به کارکنان اجازه مطالعه روزنامه‌های روز را می‌داد (به صورت آفلاین)، این نکته در بالای صفحه آورده شده بود: «لطفا برای به روز رساندن صفحات روزنامه‌ها از کلید Ctrl+F5 استفاده نمایید». این موضوع یعنی بحث کشینگ در برنامه آن پرتال در سطح فاجعه می‌باشد! حالا فرض کنید این مشکل در فرم ورود و یا مرور اطلاعات یک برنامه به وجود آید...

2. حذف View Engine های غیر ضروری

به عنوان یک برنامه نویس ASP.NET MVC، باید اطلاع داشته باشید که CLR به صورت خودکار View Engine های Razor و Web Forms را لود می‌کند. این موضوع به این دلیل است که اطلاعی از نحوه برنامه نویسی شما ندارد. اگر شما فقط از یکی از این دو View Engine استفاده می‌کنید، لطفا دیگری را غیر فعال کنید! فعال بودن هر دوی آنها یعنی اتلاف وقت گرانبهای CPU سرور شما برای رندر کردن تمامی صفحات شما توسط دو انجین! ابتدا view های شما با Web Forms Engine رندر شده سپس نتیجه به Razor Engine منتقل شده و مجدد توسط این انجین رندر می‌شود. این موضوع در سایت‌های با تعداد کاربر بالا یعنی فاجعه! برای حل این مشکل کافی است خطوط زیر را در فایل Global.asax و در رویداد بخش Application_Start وارد نمایید:

```
ViewEngines.Engines.Clear();
ViewEngines.Engines.Add(new RazorViewEngine());
```

این دو خط یعنی خداحافظ Web Forms Engine...

قبل از استفاده از این کد، اطمینان حاصل کنید کل برنامه شما توسط Razor تهیه شده است وگرنه بنده هیچ مسئولیتی در رابطه با فریادهای کارفرمای شما متقبل نمی‌شوم!

صد البته برای حذف Razor Engine و استفاده از Web Form Engine می‌توان از کد زیر در همان موقعیت فوق استفاده کرد:

```
ViewEngines.Engines.Clear();
ViewEngines.Engines.Add(new WebFormViewEngine());
```

البته همانطور که حتما دوستان مطلع هستند امکان گسترش Engine های فوق توسط ارث بری از کلاس BuildManagerViewEngine جهت ایجاد Engine های دیگر همیشه محیا است. در این صورت می‌توانید تنها انجین سفارشی مورد نظر خود را لود کرده و از لود دیگر انجین‌ها پرهیز کنید.

3. استفاده از فایل‌های PDB مایکروسافت برای دیباگ و یا پروفایل کردن DLL های دیگران

دیباگ یا پروفایل کردن برنامه‌ها، DLL ها، اسمبلی‌ها و منابعی از برنامه که شما آن را خود ننوشته اید (سورس آنها در دسترس شما نمی‌باشد) همیشه یکی از سخت‌ترین مراحل کار می‌باشد. جهت کمک به دیباگرها یا پروفایلرها، نیاز است فایل‌های PDB مرتبط با DLL ها را در اختیار آنها قرار دهید تا به بهترین نتیجه دسترسی پیدا کنید. این فایل‌ها محتوی نام توابع، شماره خطوط برنامه و metadata های دیگر برنامه اصلی قبل از optimize شدن توسط کامپایلر یا JIT می‌باشد. خوب حالا اگر نیاز شد این کار را در رابطه با DLL ها و کلاس‌های پایه Microsoft.NET انجام دهیم چه کار کنیم؟

خیلی ساده! خود Microsoft سروری جهت این موضوع تدارک دیده که فایل‌های PDB را جهت دیباگ کردن در اختیار تیم‌های برنامه نویسی قرار می‌دهد. کافی است از منوی Tools گزینه Options را انتخاب، سپس به بخش Debugging و به بخش Symbols بروید و گزینه Microsoft Symbol Servers as your source for Symbols را انتخاب کنید. برای اطمینان از اینکه هر مرتبه که برنامه را دیباگ می‌کنید مجبور به دانلود این فایل‌ها نشوید، فراموش نکنید پوشه‌ای را جهت کش این فایل‌ها ایجاد و آدرس آن را در بخش Cache symbols in this directory همین صفحه وارد نمایید.

این امکان در Visual Studio 2010, 2012 در دسترس می‌باشد.

نظرات خوانندگان

نویسنده:

محسن خان

تاریخ:

۱۷:۱۴ ۱۳۹۲/۰۵/۱۰

اثر View Engine های اضافی رو [با Glimpse](#) بهتر میشه دید.

قسمت اول

4. فشرده سازی HTTP را فعال کنید

اطمینان حاصل کنید که HTTP Compression در تمامی بخش‌های اصلی برنامه شما فعال است. حداقل کاری که می‌توانید در این رابطه بکنید این است که خروجی HTML که توسط برنامه شما تولید می‌شود را فشرده سازی کنید. جهت فعال سازی فشرده سازی در برنامه خود بهتر است در اولویت اول از مازول ویژه ای که جهت این کار در IIS در نظر گرفته شده استفاده کنید. این مازول تمامی کارها را به صورت خودکار برای شما انجام می‌دهد. اگر دسترسی به IIS جهت فعال سازی آن را ندارید، می‌توانید از مازول‌های ASP.NET که جهت این کار تهیه شده استفاده کنید. می‌توانید [کمی جستجو کنید](#) و یا خودتان یکی تهیه کنید!

5. تنظیم CacheControlMaxAge

مقدار [CacheControlMaxAge](#) را در فایل web.config را طوری تنظیم کنید تا هیچ کاربری هیچ فایل static را دیگر درخواست نکند. مثلاً می‌توانید این مقدار را بر روی چند ماه تنظیم کنید و البته فراموش نکنید این مقدار را در صفحات پویای خود بازنویسی (override) کنید تا مشکلی در رابطه با کش شدن فرم‌های اصلی برنامه (همانطور که در نکته اول بخش اول ذکر شد) پدید نیاید. البته کش کردن فایل‌های استاتیک برنامه بار مالی نیز برای شما و کاربران‌تان خواهد داشت. دیگر هزینه پهنای باند اضافی جهت دانلود این فایل‌ها در هر درخواست برای شما (در سمت سرور) و کاربران‌تان (در سمت کاربر) پرداخت نخواهد شد!

6. استفاده از OutputCache

اگر از MVC استفاده می‌کنید، فراموش نکنید که از [OutputCache](#) در کنترل‌های MVC استفاده نمایید. اگر سرور شما بتواند اطلاعات را از رم خود بازیابی کند بهتر از آن است که آن را مجدد از دیتابیس واکنشی نماید و عملیاتی نیز بر روی آن انجام دهد. البته مدیریت حافظه NET. به صورت خودکار کمبود حافظه را مدیریت کرده و از نشت حافظه جلوگیری خواهد کرد. برای توضیحات بیشتر در این رابطه می‌توانید از این [مقاله](#) کمک بگیرید.

7. بهره برداری از ORM Profiler

ORM Profiler ها تمامی فعالیت‌های ORM تحت نظر گرفته، دستورات T-SQL ارسالی به بانک اطلاعاتی را واکنشی کرده و برای شما نمایش می‌دهند. [تعدادی از آنها](#) نیز این دستورات را آنالیز کرده پیشنهاداتی در رابطه با بهبود کارایی به شما ارائه می‌دهند. برای مثال به جای اینکه شما 2000 رکورد را یکی یکی از بانک بازیابی کنید، می‌توانید آن را به صورت یک query به بانک ارسال کنید. این موضوع به سادگی توسط ORM Profiler ها قابل بررسی است. نمونه ای از این نرم افزارها را می‌توانید در [این سایت](#) یا [این سایت](#) پیدا کنید. البته در صورتی که نمی‌خواهید از نرم افزارهای جانبی استفاده کنید، می‌توانید از ابزارهای توکار بانک‌های اطلاعاتی مانند [SQL Profiler](#) نیز استفاده کنید ([راهنمایی](#)).

قسمت دوم**ORM Lazy Load.8**

در هنگام استفاده از ORM ها دقت کنید کجا از [Lazy Load](#) استفاده می‌کنید. Lazy Load باعث می‌شود وقتی شما اطلاعات مرتبط را از بانک اطلاعات واکشی می‌کنید، این واکشی اطلاعات در چند query از بانک انجام شود. در عوض عدم استفاده از [Lazy Load](#) باعث می‌شود تمامی اطلاعات مورد نیاز شما در یک query از بانک اطلاعاتی دریافت شود. این موضوع یعنی سر بار کمتر در شبکه، در بانک اطلاعاتی، در منابع حافظه و منابع پر ارزش cpu در سرورها. البته استفاده از include در حالت فعال بودن یا نبودن lazy هم داستان مجزایی دارد که اگر عمری باقی باشد راجع به آن مقاله ای خواهیم نوشت.

به این نمونه دقت کنید:

```
List<Customer> customers = context.Customers.ToList();
foreach (Customer cust in context.Customers){
    Console.WriteLine("Customer {0}, Account {1}", cust.Person.LastName.Trim() + ", " +
    cust.Person.FirstName, cust.AccountNumber);
}
```

همچنین کدی (در صورت فعال بودن Lazy Load در ORM) در صورتی که جدول Customers دارای 1000 رکورد باشد، باعث می‌شود برنامه 1001 دستور sql تولید و در بانک اجرا گردد.

برای اطلاع بیشتر می‌توانید به این [مقاله](#) مراجعه نمایید.

9. استفاده از MiniProfiler

سعی کنید از MiniProfiler در تمامی پروژه‌ها استفاده کنید. البته وقتی نرم افزار را در اختیار مصرف کننده قرار می‌دهید، آن را غیر فعال کنید. می‌توانید از متغیرهای compiler برای مجزا کردن build های متفاوت در برنامه خود استفاده کنید:

```
#if DEBUG then
// فعال سازی MiniProfiler
#endif
```

ایده دیگری هم وجود دارد. شما می‌توانید MiniProfiler را برای کاربر Admin یا کاربر Debugger فعال و برای بقیه غیر فعال کنید. در باب MiniProfiler مسائلی زیادی وجود دارد که چند نمونه از آن در همین سایت در [این مقاله](#) و [این مقاله](#) در دسترس است. البته می‌توانید از ابزارهای دیگری مانند [Glimpse](#) که در این زمینه وجود دارد نیز استفاده کنید. لب کلام این نکته استفاده از profiler برای نرم افزار خود می‌باشد.

10. Data Paging در بانک اطلاعاتی

هنگامیکه از کامپوننت‌های شرکت‌های دیگر (Third party) استفاده می‌کنید، اطمینان حاصل کنید که صفحه بندی اطلاعات در بانک اطلاعاتی انجام می‌شود. برای نمونه کاپوننت گرید شرکت Telerik چند نوع صفحه بندی را پشتیبانی می‌کند. صفحه بندی سمت کاربر (توسط JavaScript)، صفحه بندی سمت سرور توسط کامپوننت و صفحه بندی مجازی. صفحه بندی سمت کاربر یعنی تمامی اطلاعات از سرور به کاربر فرستاده شده و در سمت کاربر عمل صفحه بندی انجام می‌شود. این یعنی واکشی تمامی اطلاعات از بانک و در مورد نرم افزارهای پرکاربر با حجم اطلاعات زیاد یعنی فاجعه. صفحه بندی سمت سرور ASP.NET هم یعنی واکشی اطلاعات از سرور بانک به سرور برنامه و سپس صفحه بندی توسط برنامه. این موضوع هم ممکن است مشکلات زیادی را ایجاد نماید چون باید حداقل تمامی رکوردها از اولین رکورد تا آخرین رکورد صفحه جاری از بانک واکشی شود که این عمل علاوه بر ایجاد سر بار شبکه، سر بار IO در بانک اطلاعاتی و سر بار cpu در سرور ASP.NET ایجاد می‌کند. استفاده از صفحه بندی مجازی، شما را قادر می‌کند بتوانیم اطلاعات را در بانک صفحه بندی کرده و فقط صفحه مورد نظر خود را از بانک واکشی کنیم.

این حالت مجازی در اکثر componentها که توسط شرکت‌های مختلف ایجاد شده وجود دارد ولی ممکن است نام‌های متفاوتی داشته باشد. برای این موضوع باید به راهنمای component خریداری شده مراجعه کنید و یا به فروم‌ها و... مراجعه نمایید.

11. بررسی تعداد کوئری‌های صادر شده در یک صفحه و تعداد رکوردهای بازگشت داده شده توسط آن‌ها

این [به این معنا نیست](#) که برای هر query یک context مجزا ایجاد کنید، منظور این است که به بهانه اینکه اطلاعات مختلفی از جداول مختلف مورد نیاز است، query خود را آن قدر پیچیده یا گسترده ننویسیم که یا process آن در بانک زمان و سربار زیادی ایجاد کند و یا حجم اطلاعات بلا استفاده‌ای را از بانک به سرور برنامه لود نماید. به جای این موضوع می‌توانید در یک یا چند context دستورات مجزای واکنشی اطلاعات صادر کنید تا تنها اطلاعات مورد نیاز خود را واکنشی نمایید. البته این موضوع باعث نشود که تعداد queryها مثلاً به 1000 عدد برسد! یعنی باید فیمابین queryهای پیچیده و queryهای ساده ولی با تعداد یکی را که مناسب‌تر با پروژه است انتخاب کنید که این موضوع با تجربه و تست حاصل می‌شود.

نظرات خوانندگان

نویسنده: جواد

تاریخ: ۱۳۹۲/۰۵/۱۲ ۳:۱۵

"استفاده از صفحه بندی مجازی، شما را قادر می‌کند بتوانیم اطلاعات را در بانک صفحه بندی کرده و فقط صفحه مورد نظر خود را از بانک واکنشی کنیم. این حالت مجازی در اکثر componentها که توسط شرکت‌های مختلف ایجاد شده وجود دارد"

میشه این رو بیشتر توضیح بدید که منظورتون چیه. یا باید تمامی اطلاعات رو بفرستیم بعد صفحه بندی کنه یا اینکه به ازای هر صفحه یک کوئری به بانک بفرسته و اطلاعات رو نشون بده. حالا این صفحه بندی مجازی کجا کاربرد داره.

نویسنده: محسن خان

تاریخ: ۱۳۹۲/۰۵/۱۲ ۱۵:۱۷

هیچ کامپوننتی وجود خارجی نداره که قسمت مدیریت سمت بانک اطلاعاتی رو هم خودش به تنهایی انجام بده. همین کنترل‌های پیش فرض ASP.NET رو هم اگر ازشون درست استفاده کنیم، مشکلات کارایی ندارند. مثلاً: (نکته مهمش Skip.Take.ToList استفاده شده هست)

واکنشی اطلاعات به صورت chunk chunk (تکه تکه) و نمایش در ListView

نویسنده: مرتضی

تاریخ: ۱۳۹۲/۰۵/۱۲ ۲۱:۲۴

چرا وجود نداره!
از گرید Kendo استفاده کنید اگر paging رو فعال کنید خودش مدیریت میکنه

نویسنده: محسن خان

تاریخ: ۱۳۹۲/۰۵/۱۲ ۲۲:۲۷

بله. مدیریت می‌کنه، نه به تنهایی. اینجا هم باید اطلاعات Skip و Take رو بهش بدی تا صفحه بندی کم هزینه‌ای رو اعمال کنه. خود GridView در وب فرم‌ها هم paging داره. مشکلی اینه که در حالت پیش فرض کل اطلاعات رو از سرور واکنشی می‌کنه و بعد یک صفحه رو نمایش می‌ده. بحث اینه که گرید اگر قراره یک صفحه رو نمایش بده که 20 ردیف داره، بتونه فقط 20 رکورد رو واکنشی کنه و نه کل اطلاعات رو و این نیاز به کوئری‌های خاصی در سمت سرور داره. یک نمونه‌اش رو در واکنشی اطلاعات به صورت تکه تکه لینک دادم.

نویسنده: م منفرد

تاریخ: ۱۳۹۲/۰۵/۱۲ ۲۲:۴۶

1. اکثر کنترل‌های ASP.NET WebForm قابلیت bind به DataSet را دارد. اگر از آنها در این مدل استفاده نمایید کار صفحه بندی به عهده کنترل+DataSet می‌افتد.
 2. صفحه بندی مجازی یعنی شما به کنترل می‌گویید کل اطلاعات شما مثلاً 51 صفحه است، الان صفحه 4 را نمایش می‌دهی این هم اطلاعات صفحه 4. بعد وقتی کاربر بر روی گزینه صفحه بعد کلیک کرد، به کنترل می‌گویید کل اطلاعات 51 صفحه است، الان صفحه 5 را نمایش می‌دهی و این هم اطلاعات آن.
- برای مثال می‌توانی به این مثال در DataTables مراجعه کنید

نویسنده: مرتضی

تاریخ: ۱۳۹۲/۰۵/۱۳ ۰:۱۳

گفتم که خودش مدیریت می‌کنه یعنی اینکه اطلاعات Skip , Take رو نمی‌خواد بهش بدی و خودش اینکار رو انجام میده - من دارم ازش تو پروژم استفاده میکنم - کل اطلاعات رو واکنشی نمی‌کنه و همون 20 رکورد فرضا صفحه 3 رو واکنشی میکنه

نویسنده: محسن خان
تاریخ: ۱۳۹۲/۰۵/۱۳ ۰:۳۱

می‌تونم یک مثال با پروفایل SQL نهایی آن ارائه بدی. مطابق بررسی که کردم و حداقل دو تا لینکی که دادم در مورد این کتابخانه، موارد پردازش Skip و Take سمت سرور اون خودکار نیست.

نویسنده: مرتضی
تاریخ: ۱۳۹۲/۰۵/۱۳ ۴:۴۹

کد Razor - کد VB - خروجی SQL

```
@(Html.Kendo.Grid(Of Models.vProject).Name("ProjectsGrid") _
    .Columns(Sub(column)
        With column
            .Bound(Function(p) p.ProjectId).Hidden()
            .Bound(Function(p) p.Supervisor).Title("ناظر")
            .Bound(Function(p) p.MapNumber).Title("شماره نقشه")
            .Bound(Function(p) p.MapCode).Title("کد نقشه")
            .Bound(Function(p) p.NewStructureArea).Title("متراژ")
            .Bound(Function(p) p.NumberOfFloors).Title("تعداد طبقات")
            .Bound(Function(p) p.InsuranceName).Title("بیمه")
        End With
    End Sub).Pageable(Sub(p)
        p.Enabled(True)
        p.Info(True)
        p.PageSizes(True)
        p.Messages(Sub(m)
            m.Display("{0} - {1} رکورد {2} از")
            m.Empty("رکوردی برای نمایش وجود ندارد")
            m.Of("از")
            m.Page("صفحه")
            m.ItemsPerPage("رکورد در هر صفحه")
            m.Refresh("بروزرسانی")
        End Sub)
    End Sub).Selectable(Sub(s)
        s.Enabled(True).Mode(GridSelectionMode.Single).Type(GridSelectionType.Row)) _
        .DataSource(Sub(ds)
            ds.Ajax.ServerOperation(True).Read("GetProjects", "Home") _
                .Model(Sub(m) m.Id(Function(modelId)
                    modelId.ProjectId)))
        )
```

```
<HttpPost>
Function GetProjects(<DataSourceRequest> request As DataSourceRequest) As JsonResult
    Return Json(db.vProjects.ToDataSourceResult(request))
End Function
```

```
SELECT TOP (5) [Extent1].[ProjectId] AS [ProjectId],
               [Extent1].[Supervisor] AS [Supervisor],
               [Extent1].[MapCode] AS [MapCode],
               [Extent1].[MapNumber] AS [MapNumber],
               [Extent1].[EmployerName] AS [EmployerName],
               [Extent1].[InsuranceName] AS [InsuranceName],
               [Extent1].[NewStructureArea] AS [NewStructureArea],
               [Extent1].[NumberOfFloors] AS [NumberOfFloors]
FROM (SELECT [Extent1].[ProjectId] AS [ProjectId],
             [Extent1].[Supervisor] AS [Supervisor],
             [Extent1].[MapCode] AS [MapCode],
             [Extent1].[MapNumber] AS [MapNumber],
             [Extent1].[EmployerName] AS [EmployerName],
             [Extent1].[InsuranceName] AS [InsuranceName],
             [Extent1].[NewStructureArea] AS [NewStructureArea],
             [Extent1].[NumberOfFloors] AS [NumberOfFloors],
             row_number() OVER (ORDER BY [Extent1].[ProjectId] ASC) AS [row_number]
      FROM [vProject] AS [vProject]) AS [Extent1]
```

```
[vProject].[MapCode] AS [MapCode],
[vProject].[MapNumber] AS [MapNumber],
[vProject].[EmployerName] AS [EmployerName],
[vProject].[InsuranceName] AS [InsuranceName],
[vProject].[NewStructureArea] AS [NewStructureArea],
[vProject].[NumberOfFloors] AS [NumberOfFloors]
FROM [dbo].[vProject] AS [vProject]) AS [Extent1]) AS [Extent1]
WHERE [Extent1].[row_number] > 5
ORDER BY [Extent1].[ProjectId] ASC
```

نویسنده: محسن خان
تاریخ: ۱۳۹۲/۰۵/۱۳ ۸:۲۰

کد سمت سرور db.vProjects.ToDataSourceResult چطور تهیه شده؟ در موردش [اینجا بحث شده](#) . شما یک IQueryable باید در اختیارش قرار بدی (که از لحاظ لایه بندی کار مشکل داره) تا بر اساس اطلاعات شماره صفحه و غیره‌ای که از کلاینت می‌رسه خودش مباحث Take و Skip رو پیاده سازی کنه. در حقیقت این کتابخانه فقط [یک متد الحاقی](#) اضافه‌تر برای اینکار جهت مدیریت مباحث سمت سرور داره.

نویسنده: مرتضی
تاریخ: ۱۳۹۲/۰۵/۱۳ ۱۳:۴۵

درسته محسن خان- متد الحاقی ToDataSourceResult در خواست رو می‌گیره و....
بحث سر این بود که هیچ کامپوننتی وجود خارجی نداره که قسمت مدیریت سمت بانک اطلاعاتی رو هم خودش به تنهایی انجام بده

آره شیء DataSourceRequest شامل PageNumber , PageSize , میشه
ولی دیگه ما کاری بهش نداریم و خودش صفحه بندی رو انجام می‌ده حالا به طریقی
نمیشود گفت اگر ما از IQueryable استفاده کردیم حتما لایه بندی ما مشکل داره

نویسنده: محسن خان
تاریخ: ۱۳۹۲/۰۵/۱۳ ۱۴:۱۷

داره. [بهش می‌گن leaky abstraction](#) .

نویسنده: مرتضی
تاریخ: ۱۳۹۲/۰۵/۱۳ ۱۴:۲۴

درسته بهش می‌گن [leaky abstraction](#) اما نمی‌گن 100% ایراد

قسمت سوم

12. استفاده از validation سمت کاربر

برای جلوگیری از ارسال و دریافت‌های متناوب اطلاعات به سرور، از validation سمت کاربر استفاده نمایید. فرم‌های html 5 قابلیت‌های چک کردن نوع ورودی‌ها را به صورت خودکار دارد ولی اذتکای به آن پرهیز کنید چون ممکن است یا کاربران برنامه شما از مرورگری استفاده کنند که از html5 پشتیبانی نکند و یا پشتیبانی کاملی از آن نداشته باشند. برای حل این مشکل می‌توانید از کتابخانه‌هایی مانند JQuery و ابزارهایی مانند JQuery Validation استفاده کنید. البته [در MVC](#) استفاده وسیعی از JQuery Validation شده که می‌تواند مورد استفاده قرار گیرد.

فراموش نکنید می‌توانید از ابزارهایی مانند Regex برای چک کردن سختی کلمات عبور و... نیز در JavaScript بهره برداری نمایید. البته دقت کنید که حتما پیامی مرتبط با خطای به وقوع پیوسته در اختیار کاربر قرار دهید تا بتواند آن را بر طرف کند در غیر این صورت بنده مسئولیتی راجع به از دست دادن کاربران و یا عصبانیت کارفرما بر عهده نمی‌گیرم!

13. استفاده از validation سمت سرور

حتما به خود می‌گویید نویسنده دچار چندگانگی شخصیت شده است! ولی چنین نیست. این مطلب بیشتر از اینکه در رابطه با ایجاد سرعت بیشتر باشد مربوط به امنیت است. چون validation سمت کاربر به سادگی قابل دور زدن می‌باشد. اگر شما تنها validation را سمت کاربر انجام دهید و سمت سرور از آن چشم پوشی کنید، به سرعت تمام برنامه شما هک می‌شود. لطفا دقت کنید که امنیت را فدای هیچ چیز نکنید. این یک نکته کلیدی است. البته سوای اینکه این یک نکته امنیتی است، validation سمت سرور باعث می‌شود شما بخشی از درخواست‌ها را قبل از انجام process زیاد از گردونه خارج کنید و از ارسال اطلاعات اضافی به بانک و ایجاد سربار اضافی جلوگیری کنید.

14. چک کردن scriptهای مورد استفاده سمت کاربر

استفاده از master pageها بسیار سرعت کار را زیاد می‌کنند. بیشتر دوستان scriptهای سمت کاربر خود را در master page قرار می‌دهند تا در تمامی صفحات لود شوند. این موضوع از طرفی سرعت برنامه نویسی را زیاد می‌کند ولی از طرف دیگر به دلیل اینکه باعث می‌شود فایل‌های script در تمامی صفحات بارگذاری شوند، باعث هدر رفت منابع شبکه شما (و کاربران)، ایجاد سربار حافظه و cpu در سمت کاربر و در نتیجه سرعت پایین‌تر برنامه شما خواهد شد. سخت گیری در این موضوع می‌تواند این باشد که حتی شما function اضافی هم در سمت کاربر نداشته باشید.

برخی ناظران پروژه به این موضوعات دقت زیادی می‌کنند. در پروژه ای که به عنوان ناظر بودم مجری همین کار را انجام داده بود و به دلیل نیاز مبرم کارفرما به سرعت برنامه، این بخش از نظر اینجانب مردود اعلام شده و مجری مجبور به نوشتن دوباره کدهای آن گردید.

نظرات خوانندگان

نویسنده: علی یگانه مقدم
تاریخ: ۱۳۹۴/۰۵/۰۲ ۱:۴۹

یک نکته برای تکمیل کردن بحث بر سر مورد آخر: در MVC این مشکل توسط sectionها حل شده

[قسمت چهارم](#)**15. استفاده از using**

اگر از objectهایی استفاده می‌کنید که interface مربوط به [IDisposable](#) را پیاده سازی کرده اند، حتما از عبارت using استفاده کنید. استفاده از دستور using باعث می‌شود زمانی که دیگر نیازی به object شما نباشد، به صورت خودکار از حافظه حذف شود و در روال جمع آوری زباله (GC) قرار گیرد. این عمل باعث حداقل رسیدن احتمال نشت حافظه در نرم افزار شما می‌شود. برای مثال:

```
using System;
using System.Text;

class Program
{
    static void Main()
    {
        // Use using statement with class that implements Dispose.
        using (SystemResource resource = new SystemResource())
        {
            Console.WriteLine(1);
        }
        Console.WriteLine(2);
    }
}

class SystemResource : IDisposable
{
    public void Dispose()
    {
        // The implementation of this method not described here.
        // ... For now, just report the call.
        Console.WriteLine(0);
    }
}
```

برای اطلاعات بیشتر می‌توانید از [این مقاله](#) استفاده کنید.

16. اطلاعات ارسالی توسط شبکه را به حداقل برسانید

حجم اطلاعات ارسالی به شبکه را به حداقل برسانید. ارسال اطلاعات در شبکه به معنی گذر اطلاعات شما از 7 لایه مختلف شبکه در رایانه شما، گذر از media شبکه، گذر مجدد از 7 لایه شبکه در رایانه مقصد می‌باشد. به این معنی که هرچه اطلاعات بیشتری در شبکه ارسال کنید، سربار بیشتری متوجه سیستم شما خواهد بود. برای رفع این مشکل از فشرده سازهای css و javascript استفاده کنید. این فشرده سازها فواصل خالی، دستورات اضافی و... را از کد شما حذف و حجم آن را به حداقل می‌رسانند. کم کردن تعداد درخواست‌ها و در نتیجه کم کردن تعداد فایل‌های ارسالی از سرور به کاربر نیز حربه ای در این زمینه می‌باشد. برای مقایسه فشرده سازها به صورت آنلاین و استفاده از بهترین آنها (متناسب کد شما) می‌توانید از [این سایت](#) استفاده کنید. امکانات توکاری هم وجود دارد که در زمان اجرای برنامه css و javascript شما را فشرده و تلفیق می‌کند ولی با توجه به اینکه برای سرور در هر مرتبه فراخوانی سربار دارد (حتی در صورت کش کردن) اکیدا توصیه می‌شود از فشرده سازها قبل از اجرای برنامه (Pre-Compressed) به جای فشرده سازهای زمان اجرا (Run-time Compression) استفاده کنید.

نظرات خوانندگان

نویسنده: حسین حقیان
تاریخ: ۱۳۹۲/۰۵/۱۶ ۱۲:۳۸

با سلام و تشکر از مجموعه مطالب مرتبط که ارائه کردید
میشه برای این قسمت مثالی رو ذکر بفرمایید
اکیدا توصیه می‌شود از فشرده سازها قبل از اجرای برنامه (Pre-Compressed) به جای فشرده سازهای زمان اجرا (Run-time Compression) استفاده کنید.

با تشکر

نویسنده: م منفرد
تاریخ: ۱۳۹۲/۰۵/۱۶ ۱۹:۵۱

فشرده سازی قبل از اجرای برنامه (Pre-Compression) یعنی که شما قبل از اینکه برنامه خود را در محیط اصلی نصب و اجرا کنید، فایل‌های اسکریپت آن را فشرده کنید. یعنی کاربران فایل اسکریپت فشرده شده را درخواست و دانلود می‌کنند و عملیات اضافی در سمت سرور انجام نمی‌شود. به عنوان مثال شما از فایل JQuery.min.js به جای jquery.js استفاده کنید. یعنی استفاده از نسخه فشرده شده اسکریپت‌ها.

فشرده سازی زمان اجرا (Run-time Compression) یعنی فشرده سازی اسکریپت‌های مورد نیاز کاربر توسط خود برنامه وب (به صورت خودکار و یا توسط یک ماژول اضافی). این عمل باعث می‌شود که در هر بار درخواست هر کاربر برای یک فایل، برنامه آن را مجدد فشرده سازی کند (و یا از cache استفاده کند). این عمل به معنی استفاده بیشتر از منابع پر ارزش سرور شما می‌باشد. به عنوان مثال شما بخواهید در هر مرحله درخواست هر کاربر jquery.js را فشرده کنید!

از مقایسه دو حالت بالا مشخص است وقتی شما فقط یکبار اسکریپت‌های خود را فشرده می‌کنید بسیار از حالتی که در هر مرتبه از درخواست کاربران آن را فشرده کنید بهتر است و کمتر منابع سرور را هدر می‌دهد.

قسمت پنجم**17. پرهیز از استفاده نسخه debug**

وقتی به ASP.NET مراجعه می‌کنید، توجه فرمایید که از چه نوع build برای محصول نهایی استفاده می‌کنید. وقتی از نسخه debug برنامه استفاده می‌کنید، بهبود دهنده‌های سطح کامپایلر عمل نکرده و کد شما در حالت بهینه اجرا نخواهد شد (کد شما همانگونه که هست اجرا می‌شود!).

برای مثال هنگامی که از نسخه release استفاده می‌کنید، کامپایلر #c به صورت خودکار از StringBuilderها به جای تلفیق عادی رشته‌ها، از آرایه‌ها به جای لیست‌ها، از دستور switch/case به جای دستورات if/then/else، تلفیق شروط با یکدیگر و... استفاده کرده و کد شما را در حالت بهینه‌تری اجرا می‌کند. عدم استفاده از این نسخه شما را از این مزایا محروم می‌سازد و نرم افزار شما به کندی اجرا خواهد شد. البته ناگفته نماند این موضوع فقط باید برای محصول نهایی استفاده شود و جهت دیباگ کردن برنامه همچنان باید از نسخه debug استفاده نمایید.

توجه نمایید می‌توانید با استفاده از متغیرهای کامپایلر در کد خود بخشی از کد را مختص build خاصی از برنامه کنید. مثلاً اگر برنامه در حال debug کامپایل شد، MiniProfiler را فعال کن در غیر این صورت غیر فعال باشد.

```
#if DEBUG
// فعال کردن MiniProfiler
#endif
```

18. تنظیم دقیق لاگ‌های سیستم در محیط اجرا

وقتی محصول نهایی را آماده می‌کنید، فراموش نکنید که سطح لاگ گیری را در سطح مطلوبی قرار دهید تا بتوانید در صورت نیاز برنامه را اشکال زدایی کنید. البته زیاده روی در این مورد نیز می‌تواند مشکل‌زا باشد. اکثر برنامه نویسان هنگامی که محصول نهایی را برای مشتری آماده می‌کنند، لاگ را غیر فعال می‌کنند تا کاربر سرعت بیشتری را تجربه کند. این سیاست غلط شما را از امکانات بی نظیر لاگ کردن (مانند وقایع نگاری امنیتی، رفع سریع مشکلات و...) محروم می‌سازد. بنابر این حتماً سیستم لاگ خود را در زمان تولید محصول اصلی (و نصب بر روی سرور اصلی) در حالت متعادل تنظیم نمایید. کمی تست و تجربه شما را در این امر یاری می‌کند.

19. مشخص کردن اندازه عکس

مشخص کردن اندازه عکس در تک img به صورت css یا attribute باعث می‌شود که همان اولین بار که صفحه رندر می‌شود، اندازه مورد نیاز عکس به آن اختصاص یابد تا در صورت دانلود سریعاً جایگزین آن گردد. عدم مشخص کردن سایز عکس (طول و عرض) باعث رندر شدن مجدد تمامی المان‌های صفحه بعد از دانلود هر عکس از سرور می‌شود و منابع با ارزش cpu کاربر شما را به سادگی از بین می‌برد.

```

```

نظرات خوانندگان

نویسنده: مرتضی

تاریخ: ۱۶:۵۴ ۱۳۹۲/۰۷/۰۷

سلام. ممنون از مطلب خوبتون. بنده وقتی سایتم رو با ویژوال استادیو باز میکنم فقط مود دیباگ داره و مود release رو نداره. فقط برای برنامه‌های ویندوز فرم و wpf مود release داره. پس با این اوصاف بنده چطوری سایتم رو تویه مود release پابلیش کنم؟ فقط کد زیر رو تویه وب کانفیگ قرار میدم

```
<compilation debug="false" targetFramework="4.0">
```

آیا قطعه کد بالا همون کار release رو انجام میده. با تشکر

نویسنده: م منفرد

تاریخ: ۰:۱۰ ۱۳۹۲/۱۰/۱۴

در بخش publish از منوی debug می‌توانید نوع خروجی را مشخص کنید که برنامه با debug پابلیش شود یا release

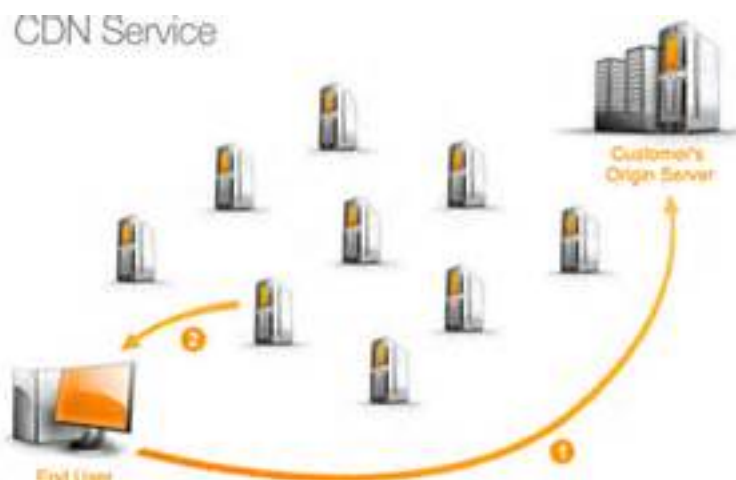
قسمت ششم

20. اسکریپت در پایین صفحه

لینک‌های مربوطه به javascriptهای خود را تا جای ممکن در پایین صفحه قرار دهید. وقتی parser مرورگر به فایل‌های javascript می‌رسد، تمامی فعالیت‌ها را متوقف کرده و سعی در دانلود و سپس اجرای آن دارد. برخلاف اینکه مرورگرها امکان دانلود چند فایل را به صورت همزمان از سرور دارند، هنگامی که به اسکریپت‌ها می‌رسند، تنها یک فایل را دانلود می‌کنند. یعنی اجرای برنامه و دانلودهای مرتبط با صفحه شما متوقف شده و پس از دانلود و اجرای اسکریپت اجرای آنها ادامه پیدا می‌کند. این مسئله وقتی نمود بیشتری پیدا می‌کند که شما فایل‌های اسکریپت با حجم و تعداد بالا در برنامه خود استفاده می‌کنید. برای فرار از این مشکل می‌توانید تگ‌های مربوط به اسکریپت را در آخر صفحات خود بگذارید. فقط دقت کنید اگر نیاز است که قبل از نمایش صفحه تغییری در DOM ایجاد کنید، باید حتما اسکریپت‌های مربوطه را بالای صفحه قرار دهید. روش دیگر دانلود فایل‌های اسکریپت به وسیله AJAX است که انشاء الله در آینده مقاله ای در این رابطه خواهم نوشت.

CDN.21

CDN یا Content Delivery Network سرورهای توزیع شده ای در سطح دنیا هستند که یک نسخه از برنامه شما برای اجرا بر روی آن قرار دارد. هنگامی که کاربر می‌خواهد به سایت شما دسترسی پیدا کند به صورت خودکار به نزدیکترین سرور منتقل می‌شود تا بتواند سرعت بیشتری را تجربه کند. علاوه بر این CDN باعث بالانس شدن بار ترافیک شبکه شما شده خط حملات D.O.S و D.D.O.S را به حداقل می‌رساند.



زمانیکه شما یک سیستم CDN را فعال میکنید تاثیر آن بصورت زیر خواهد بود:

- ۱- شبکه توزیع محتوا یا همان CDN تمامی سرورهای شبکه جهانی اینترنت را پوشش میدهد. بنابراین زمانیکه شما این سیستم را برای سایت خود فعال میکنید اطلاعات شما بر روی تمامی این سرورها کپی و ذخیره میشود و زمانیکه یک بازدیدکننده به سایت یا وبلاگ شما وارد میشود محتوای سایت شامل تصاویر و متون را از نزدیکترین سرور نزدیک به خود دریافت میکند و مستقیماً به هاست یا سرور شما متصل نمیشود. این کار موجب بهبودی چشمگیر در عملکرد سایت شما خواهد شد.
- ۲- تمام اطلاعات ثابت شما مانند تصاویر، کدهای CSS و javascript، mp3، pdf و فایل‌های ویدئویی شما را پشتیبانی میکند و تنها اطلاعاتی که قابل تغییر و بروزرسانی هستند مانند متون و کدهای HTML از سرور اصلی شما فراخوان میشوند. با این کار مصرف پهنای باند هاست شما کاهش یافته و هزینه ای که سالانه برای آن میپردازید کاهش چشمگیری خواهد داشت.
- ۳- تفاوت سرعت و عملکرد برای خودتان یا افرادی که در نزدیکی سرور اصلی شما هستند تفاوت زیادی نخواهد داشت ولی برای کسانی که از نقاط مختلف جهان به سایت شما وارد میشوند این افزایش سرعت ناشی از CDN کاملاً محسوس خواهد بود، با توجه به اینکه سایتهای ایرانی معمولاً سرور و هاست خود را از خارج و کشورهایی مانند آلمان و آمریکا تهیه میکنند و عموم بازدیدکنندگان

از داخل کشور هستند استفاده از CDN میتواند بسیار موثر باشد. برای تعیین تاثیر CDN بر سرعت سایت میتوانید عملکرد خود را با ابزارهایی مانند [Pingdom](#) و [GTmetrix](#) بعد و قبل از فعال سازی CDN بررسی و مقایسه کنید. ابزارها، تکنیک‌ها و روش‌های متفاوتی برای راه اندازی CDN وجود دارد که نیازمند مقاله ای مجزا می‌باشد.

در WPF، زیر ساخت‌های ComponentModel توسط کلاسی به نام [PropertyDescriptor](#)، منابع Binding موجود در قسمت‌های مختلف برنامه را در جدولی عمومی ذخیره و نگهداری می‌کند. هدف از آن، مطلع بودن از مواردی است که نیاز دارند توسط مکانیزم‌هایی مانند [INotifyPropertyChanged](#) و [DependencyProperty](#) ها، اطلاعات اشیاء متصل را به روز کنند. در این سیستم، کلیه اتصالاتی که Mode آن‌ها به OneTime تنظیم نشده است، به صورت اجباری دارای یک valueChangedHandlers متصل توسط سیستم PropertyDescriptor خواهند بود و در حافظه زنده نگه داشته می‌شوند؛ تا بتوان در صورت نیاز، توسط سیستم binding اطلاعات آن‌ها را به روز کرد. همین مساله سبب می‌شود تا اگر قرار نیست خاصیتی برای نمونه توسط مکانیزم INotifyPropertyChanged اطلاعات UI را به روز کند (یک خاصیت معمولی دات نت است) و همچنین حالت اتصال آن به OneTime نیز تنظیم نشده، سبب مصرف حافظه بیش از حد برنامه شود. اطلاعات بیشتر

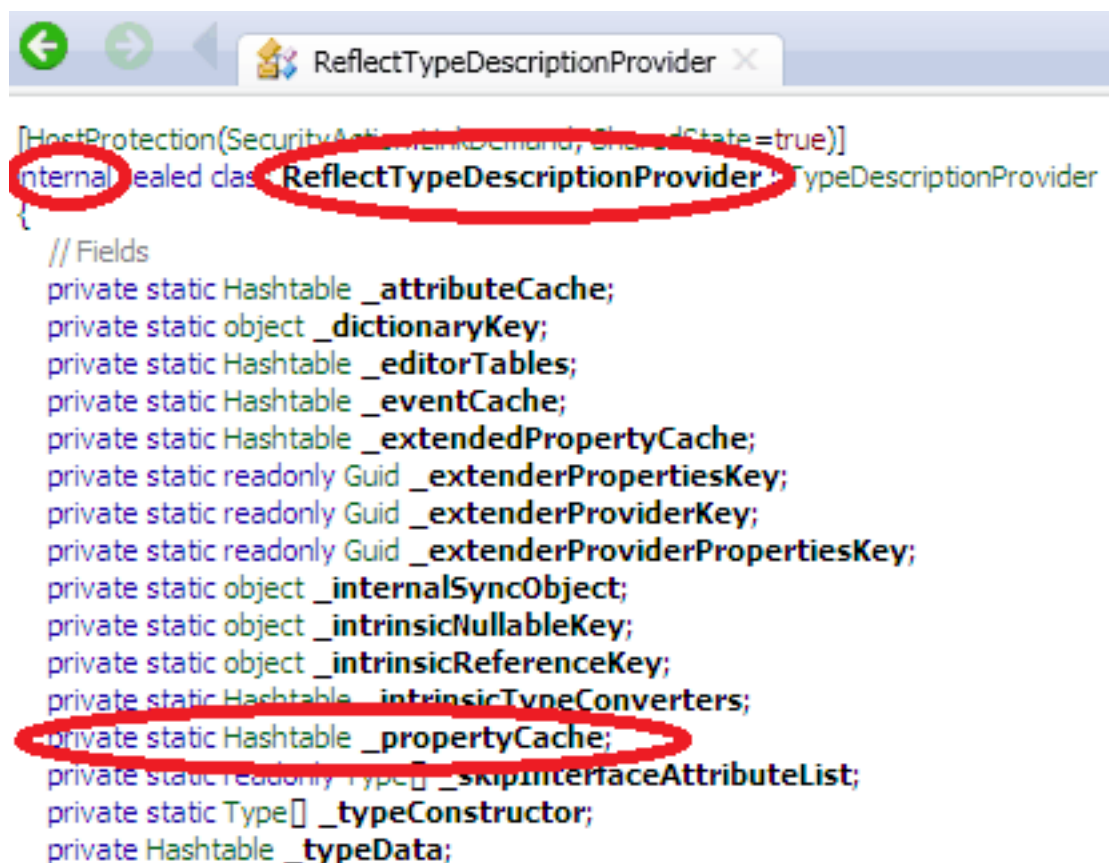
[A memory leak may occur when you use data binding in Windows Presentation Foundation](#)

راه حل آن هم ساده است. برای اینکه valueChangedHandler ایی به خاصیت ساده‌ای که قرار نیست بعدها UI را به روز کند، متصل نشود، حالت اتصال آن‌را باید به [OneTime](#) تنظیم کرد.

سؤال: در یک برنامه بزرگ که هم اکنون مشغول به کار است، چطور می‌توان این مسایل را ردیابی کرد؟

برای دستیابی به اطلاعات کش Binding در WPF، باید به Reflection متوسل شد. به این ترتیب در برنامه جاری، در کلاس PropertyDescriptor به دنبال یک کلاس خصوصی تو در توی دیگری به نام ReflectTypeDescriptionProvider خواهیم گشت (این اطلاعات از طریق مراجعه به سورس دات نت و یا حتی برنامه‌های ILSpy و Reflector قابل استخراج است) و سپس در این کلاس خصوصی داخلی، فیلد خصوصی propertyCache آن‌را که از نوع Hashtable است استخراج می‌کنیم:

```
var reflectTypeDescriptionProvider =  
typeof(PropertyDescriptor).Module.GetType("System.ComponentModel.ReflectTypeDescriptionProvider");  
var propertyCacheField = reflectTypeDescriptionProvider.GetField("_propertyCache",  
BindingFlags.Static | BindingFlags.NonPublic);
```



اکنون به لیست داخلی Binding نگهداری شونده توسط WPF دسترسی پیدا کرده‌ایم. در این لیست به دنبال مواردی خواهیم گشت که فیلد valueChangedHandlers به آن‌ها متصل شده است و در حال گوش فرا دادن به سیستم binding هستند (سورس کامل و طولانی این مبحث را در پروژه پیوست شده می‌توانید ملاحظه کنید).

یک مثال: تعریف یک کلاس ساده، اتصال آن و سپس بررسی اطلاعات درونی سیستم Binding

فرض کنید یک کلاس مدل ساده به نحو ذیل تعریف شده است:

```
namespace WpfOneTime.Models
{
    public class User
    {
        public string Name { set; get; }
    }
}
```

سپس این کلاس به صورت یک List، توسط ViewModel برنامه در اختیار View متناظر با آن قرار می‌گیرد:

```
using WpfOneTime.Models;
using System.Collections.Generic;

namespace WpfOneTime.ViewModels
{
    public class MainWindowViewModel
    {
        public IList<User> Users { set; get; }

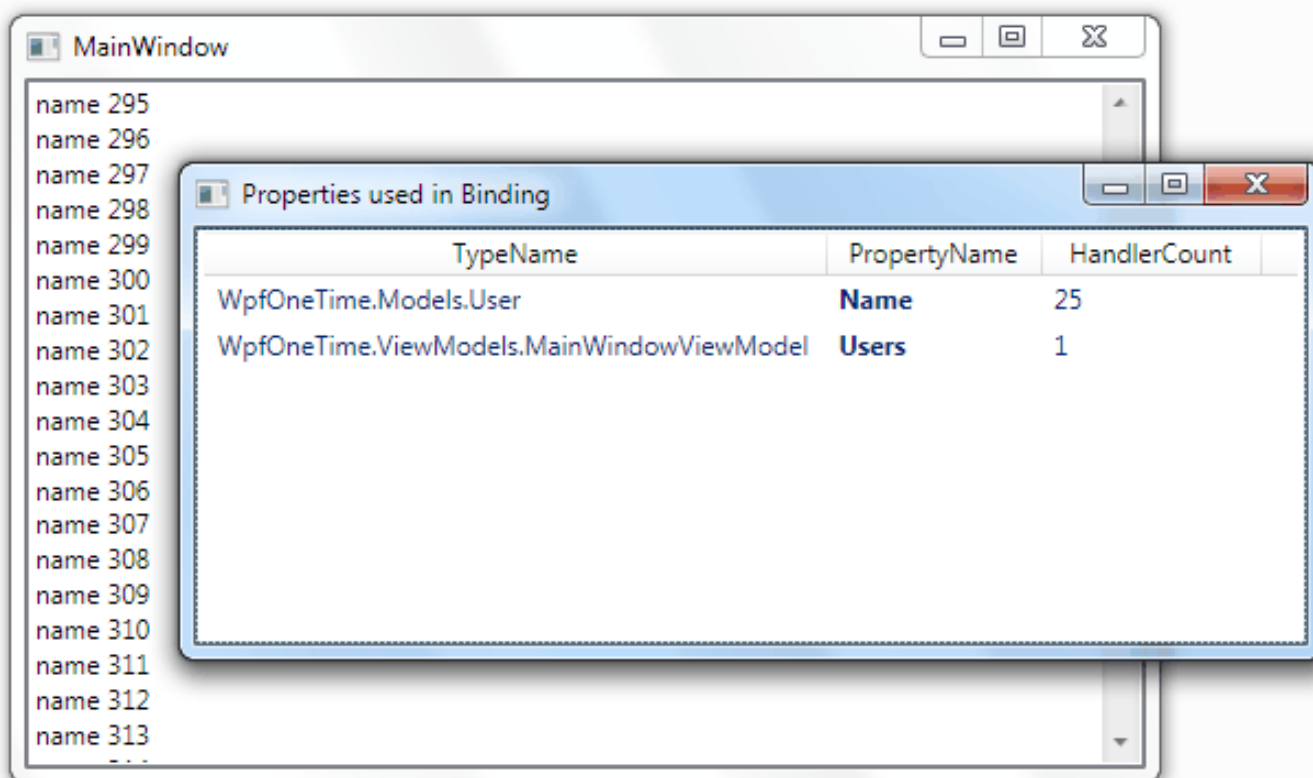
        public MainWindowViewModel()
        {
            Users = new List<User>();
        }
    }
}
```

```
        for (int i = 0; i < 1000; i++)
        {
            Users.Add(new User { Name = "name " + i });
        }
    }
}
```

تعاریف View برنامه نیز به نحو زیر است:

```
<Window x:Class="WpfOneTime.MainWindow"
        xmlns="http://schemas.microsoft.com/winfx/2006/xaml/presentation"
        xmlns:x="http://schemas.microsoft.com/winfx/2006/xaml"
        xmlns:ViewModels="clr-namespace:WpfOneTime.ViewModels"
        Title="MainWindow" Height="350" Width="525">
    <Window.Resources>
        <ViewModels:MainWindowViewModel x:Key="vmMainWindowViewModel" />
    </Window.Resources>
    <Grid DataContext="{Binding Source={StaticResource vmMainWindowViewModel}}">
        <ListBox ItemsSource="{Binding Users}">
            <ListBox.ItemTemplate>
                <DataTemplate>
                    <TextBlock Text="{Binding Name}" />
                </DataTemplate>
            </ListBox.ItemTemplate>
        </ListBox>
    </Grid>
</Window>
```

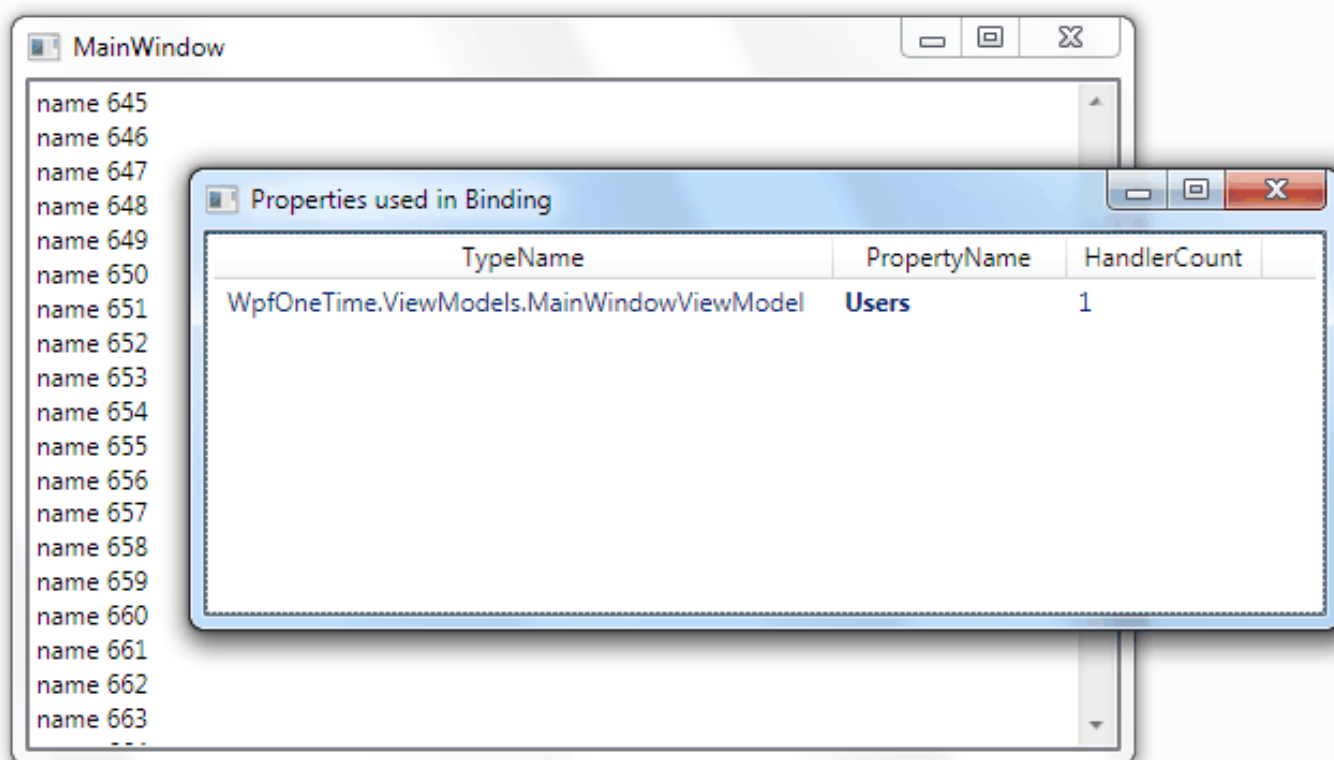
همه چیز در آن معمولی به نظر می‌رسد. ابتدا به ViewModel برنامه دسترسی یافته و DataContext را با آن مقدار دهی می‌کنیم. سپس اطلاعات این لیست را توسط یک ListBox نمایش خواهیم داد. خوب! اکنون اگر اطلاعات Hashtable داخلی سیستم Binding را در مورد View فوق بررسی کنیم به شکل زیر خواهیم رسید:



بله. تعداد زیادی خاصیت Name زنده و موجود در حافظه باقی هستند که تحت ردیابی سیستم Binding می‌باشند. در ادامه، نکته‌ی ابتدای بحث را جهت تعیین حالت Binding به [OneTime](#)، به View فوق اعمال می‌کنیم (یک سطر ذیل باید تغییر کند):

```
<TextBlock Text="{Binding Name, Mode=OneTime}" />
```

در این حالت اگر نگاهی به سیستم ردیابی WPF داشته باشیم، دیگر خبری از اشیاء زنده دارای خاصیت Name در حال ردیابی نیست:



به این ترتیب می‌توان در لیست‌های طولانی، به مصرف حافظه کمتری در برنامه WPF خود رسید. بدیهی است این نکته را تنها در مواردی می‌توان اعمال کرد که نیاز به به‌روز رسانی‌های ثانویه اطلاعات UI در کدهای برنامه وجود ندارند.

چطور از این نکته برای پروفایل یک برنامه موجود استفاده کنیم؟

کدهای برنامه را از انتهای بحث دریافت کنید. سپس دو فایل `ReflectPropertyDescriptorWindow.xaml` و `ReflectPropertyDescriptorWindow.xaml.cs` آن‌را به پروژه خود اضافه نمایید و در سازنده پنجره اصلی برنامه، کد ذیل را فراخوانی نمایید:

```
new ReflectPropertyDescriptorWindow().Show();
```

کمی با برنامه کار کرده و منتظر شوید تا لیست نهایی اطلاعات داخلی Binding ظاهر شود. سپس مواردی را که دارای `HandlerCount` بالا هستند، مدنظر قرار داده و بررسی نمایید که آیا واقعا این اشیاء نیاز به `valueChangedHandler` متصل دارند یا خیر؟ آیا قرار است بعدها UI را از طریق تغییر مقدار خاصیت آن‌ها به روز نمائیم یا خیر. اگر خیر، تنها کافی است نکته `Mode=OneTime` را به این Binding‌ها اعمال نمائیم.

دریافت کدهای کامل پروژه این مطلب

[WpfOneTime.zip](#)

نظرات خوانندگان

نویسنده: سیما

تاریخ: ۱۹:۲۱ ۱۳۹۳/۰۳/۲۳

سلام،

می‌خواستم بدونم به چه شکل میتوانم متوجه شوم کدام قسمت از برنامه من موجب افزایش مصرف رم شده است؟ برای مثال برنامه من بعد گذشت 1 دقیقه از اجرای آن مصرف رم معادل 5MB دارم ولی پس از گذشت 10 دقیقه به 1GB میرسد.

نویسنده: وحید نصیری

تاریخ: ۱۹:۱۷ ۱۳۹۳/۰۳/۲۳

از برنامه‌های Profiler باید استفاده کنید؛ مانند:

- [ابزارهای توکار VS.NET](#)

- [New Memory Usage Tool for WPF and Win32 Applications](#)

- [Windows Performance Toolkit](#)

- [dotMemory](#)

- [ANTS Memory Profiler](#)

کنترل‌های WPF در حالت پیش فرض و بدون اعمال قالب خاصی به آن‌ها عموماً خوب عمل می‌کنند. مشکل از جایی شروع می‌شود که قصد داشته باشیم حالت پیش فرض را اندکی تغییر دهیم و یا Visual tree این کنترل‌ها اندکی پیچیده شوند. برای نمونه مدل زیر را در نظر بگیرید:

```
using System;

namespace WpfLargeLists.Models
{
    public class User
    {
        public int Id { set; get; }
        public string FirstName { set; get; }
        public string LastName { set; get; }
        public string Address { set; get; }
        public DateTime DateOfBirth { set; get; }
    }
}
```

قصد داریم فقط 1000 رکورد ساده از این مدل را به یک ListView اعمال کنیم.

```
<ListView ItemsSource="{Binding UsersTab1}" Grid.Row="1" Margin="3">
    <ListView.View>
        <GridView>
            <GridViewColumn Header="Id" Width="50" DisplayMemberBinding="{Binding
Id, Mode=OneTime}" />
            <GridViewColumn Header="FirstName" Width="100"
DisplayMemberBinding="{Binding FirstName, Mode=OneTime}" />
            <GridViewColumn Header="LastName" Width="100"
DisplayMemberBinding="{Binding LastName, Mode=OneTime}" />
            <GridViewColumn Header="Address" Width="100"
DisplayMemberBinding="{Binding Address, Mode=OneTime}" />
            <GridViewColumn Header="DateOfBirth" Width="150"
DisplayMemberBinding="{Binding DateOfBirth, Mode=OneTime}" />
        </GridView>
    </ListView.View>
</ListView>
```

در اینجا UsersTab1، لیستی حاوی فقط 1000 رکورد از شیء User است. در حالت معمولی این لیست بدون مشکل بارگذاری می‌شود. اما با اعمال مثلاً قالب [MahApp.Metro](#)، بارگذاری همین لیست، حدود 5 ثانیه با CPU usage صد در صد طول می‌کشد. علت اینجا است که در این حالت WPF سعی می‌کند تا ابتدا در VisualTree، کل 1000 ردیف را کاملاً ایجاد کرده و سپس نمایش دهد.

راه حل توصیه شده برای بارگذاری تعداد بالایی رکورد در WPF : استفاده از UI Virtualization

UI Virtualization روشی است که در آن تنها المان‌هایی که توسط کاربر در حال مشاهده هستند، تولید و مدیریت خواهند شد. در این حالت اگر 1000 رکورد را به یک ListBox یا ListView ارسال کنید و کاربر بر اساس اندازه صفحه جاری خود تنها 10 رکورد را مشاهده می‌کند، WPF فقط 10 عنصر را در VisualTree مدیریت خواهد کرد. با اسکرول به سمت پایین، مواردی که دیگر نمایان نیستند dispose شده و مجموعه نمایان دیگری خلق خواهند شد. به این ترتیب می‌توان حجم بالایی از اطلاعات را در WPF با میزان مصرف پایین حافظه و همچنین مصرف CPU بسیار کم مدیریت کرد. این مجازی سازی در WPF به وسیله VirtualizingStackPanel در دسترس است.

برای اینکه WPF virtualization به درستی کار کند، نیاز است یک سری شرایط مقدماتی فراهم شوند:

- از کنترلی استفاده شود که از virtualization پشتیبانی می‌کند؛ مانند ListBox و ListView.
- ارتفاع کنترل لیستی باید دقیقاً مشخص باشد؛ یا درون یک ردیف از Grid ایی باشد که ارتفاع آن مشخص است. برای نمونه اگر ارتفاع ردیف Grid ایی که ListView را دربرگرفته است به * تنظیم شده، مشکلی نیست؛ اما اگر ارتفاع این ردیف به Auto تنظیم

شده، کنترل لیستی برای محاسبه vertical scroll bar خود دچار مشکل خواهد شد.

- کنترل مورد استفاده نباید در یک کنترل ScrollViewer محصور شود؛ در غیر اینصورت virtualization رخ نخواهد داد. علاوه بر آن در خود کنترل باید خاصیت ScrollViewer.HorizontalScrollBarVisibility نیز به Disabled تنظیم گردد.
- در کنترل در حال استفاده، ScrollViewer.CanContentScroll باید به true تنظیم شود.

مورد مشخص بودن ارتفاع بسیار مهم است. برای نمونه در برنامه‌ای پس از فعال سازی مجازی سازی، کنترل لیستی کلاً از کار افتاد و حرکت scroll bar آن سبب بروز CPU Usage بالایی می‌شد. این مشکل با تنظیم ارتفاع آن به شکل زیر برطرف شد:

```
Height="{Binding Path=RowDefinitions[1].ActualHeight, RelativeSource={RelativeSource AncestorType=Grid}}"
```

در این تنظیم، ارتفاع کنترل، به ارتفاع سطر دوم گرید دربرگیرنده ListView متصل شده است.

- پس از اعمال موارد یاد شده، باید VirtualizingStackPanel کنترل را فعال کرد. ابتدا دو خاصیت زیر باید مقدار دهی شوند:

```
VirtualizingStackPanel.IsVirtualizing="True"  
VirtualizingStackPanel.VirtualizationMode="Recycling"
```

سپس ItemsPanelTemplate کنترل باید به صورت یک VirtualizingStackPanel مقدار دهی شود. برای مثال اگر از ListBox استفاده می‌کنید، تنظیمات آن به نحو زیر است:

```
<ListBox.ItemsPanel>  
  <ItemsPanelTemplate>  
    <VirtualizingStackPanel IsVirtualizing="True" VirtualizationMode="Recycling" />  
  </ItemsPanelTemplate>  
</ListBox.ItemsPanel>
```

و اگر از ListView استفاده می‌شود، تنظیمات آن مشابه ListBox است:

```
<ListView.ItemsPanel>  
  <ItemsPanelTemplate>  
    <VirtualizingStackPanel  
      IsVirtualizing="True"  
      VirtualizationMode="Recycling" />  
  </ItemsPanelTemplate>  
</ListView.ItemsPanel>
```

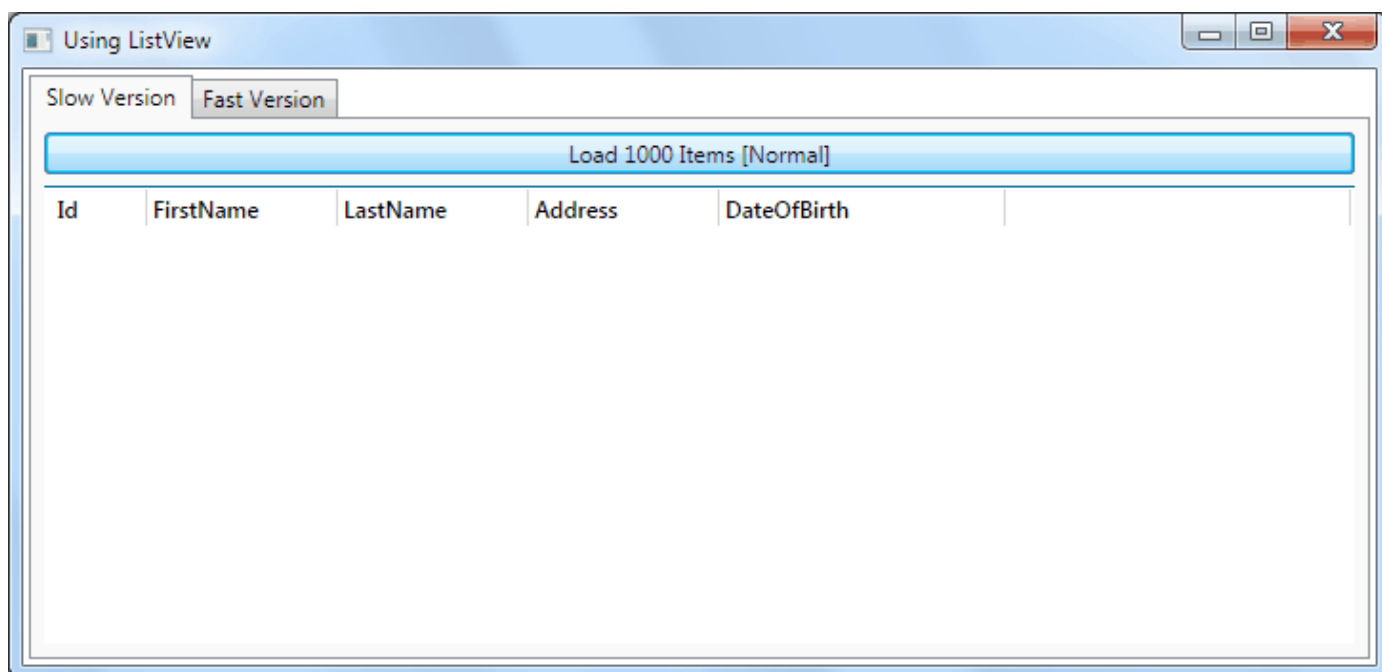
با این توضیحات ابتدای بحث به شکل زیر تغییر خواهد یافت تا مجازی سازی آن فعال گردد:

```
<ListView ItemsSource="{Binding UsersTab2}" Grid.Row="1" Margin="3"  
  ScrollViewer.HorizontalScrollBarVisibility="Disabled"  
  ScrollViewer.CanContentScroll="True"  
  VirtualizingStackPanel.IsVirtualizing="True"  
  VirtualizingStackPanel.VirtualizationMode="Recycling">  
  <ListView.ItemsPanel>  
    <ItemsPanelTemplate>  
      <VirtualizingStackPanel  
        IsVirtualizing="True"  
        VirtualizationMode="Recycling" />  
    </ItemsPanelTemplate>  
  </ListView.ItemsPanel>  
  <ListView.View>  
    <GridView>  
      <GridViewColumn Header="Id" Width="50" DisplayMemberBinding="{Binding  
Id, Mode=OneTime}" />  
      <GridViewColumn Header="FirstName" Width="100"  
DisplayMemberBinding="{Binding FirstName, Mode=OneTime}" />  
      <GridViewColumn Header="LastName" Width="100"  
DisplayMemberBinding="{Binding LastName, Mode=OneTime}" />  
      <GridViewColumn Header="Address" Width="100"  
DisplayMemberBinding="{Binding Address, Mode=OneTime}" />  
      <GridViewColumn Header="DateOfBirth" Width="150"
```

```
DisplayMemberBinding="{Binding DateOfBirth, Mode=OneTime}" />
    </GridView>
</ListView.View>
</ListView>
```

کدهای کامل مثال فوق را از اینجا می‌توانید دریافت کنید: [WpfLargeLists.zip](#)

در این مثال دو برگه را ملاحظه می‌کنید. برگه اول حالت normal ابتدای بحث است و برگه دوم پیاده سازی UI Virtualization را انجام داده است.



نظرات خوانندگان

نویسنده:

سیما

تاریخ:

۱۸:۵۳ ۱۳۹۳/۰۳/۲۳

با سلام،

میخواستم بدونم امکانش هست از WrapPanel هم بعنوان یک ItemsPanelTemplate با رعایت Virtualization استفاده کرد؟

نویسنده:

وحید نصیری

تاریخ:

۱۹:۲۲ ۱۳۹۳/۰۳/۲۳

Virtualizing WrapPanel : [^](#) و [^](#)

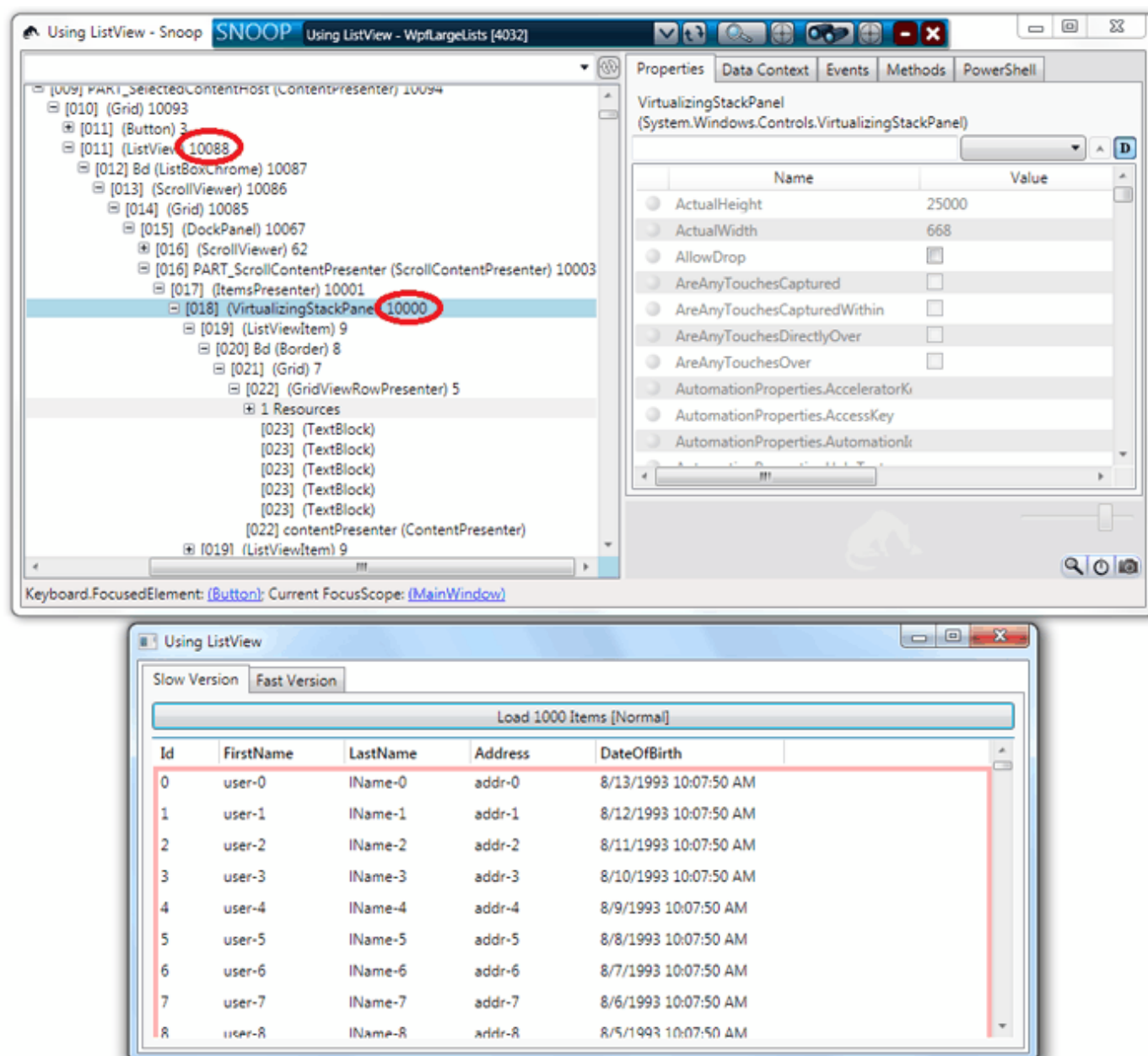
در مطلب « [بهبود کارایی کنترل‌های لیستی WPF در حین بارگذاری تعداد زیادی از رکوردها](#) » عنوان شد که در حالت فعال بودن UI Virtualization، فقط به تعداد ردیف‌های نمایان، اشیاء متناظری به یک کنترل لیستی اضافه می‌شوند و حالت برعکس آن زمانی است که ابتدا کلیه اشیاء بصری یک لیست تولید شده و سپس لیست نهایی نمایش داده می‌شود.

سؤال: چگونه می‌توان تعداد اشیاء اضافه شده به Visual tree یک کنترل لیستی را شمارش کرد؟

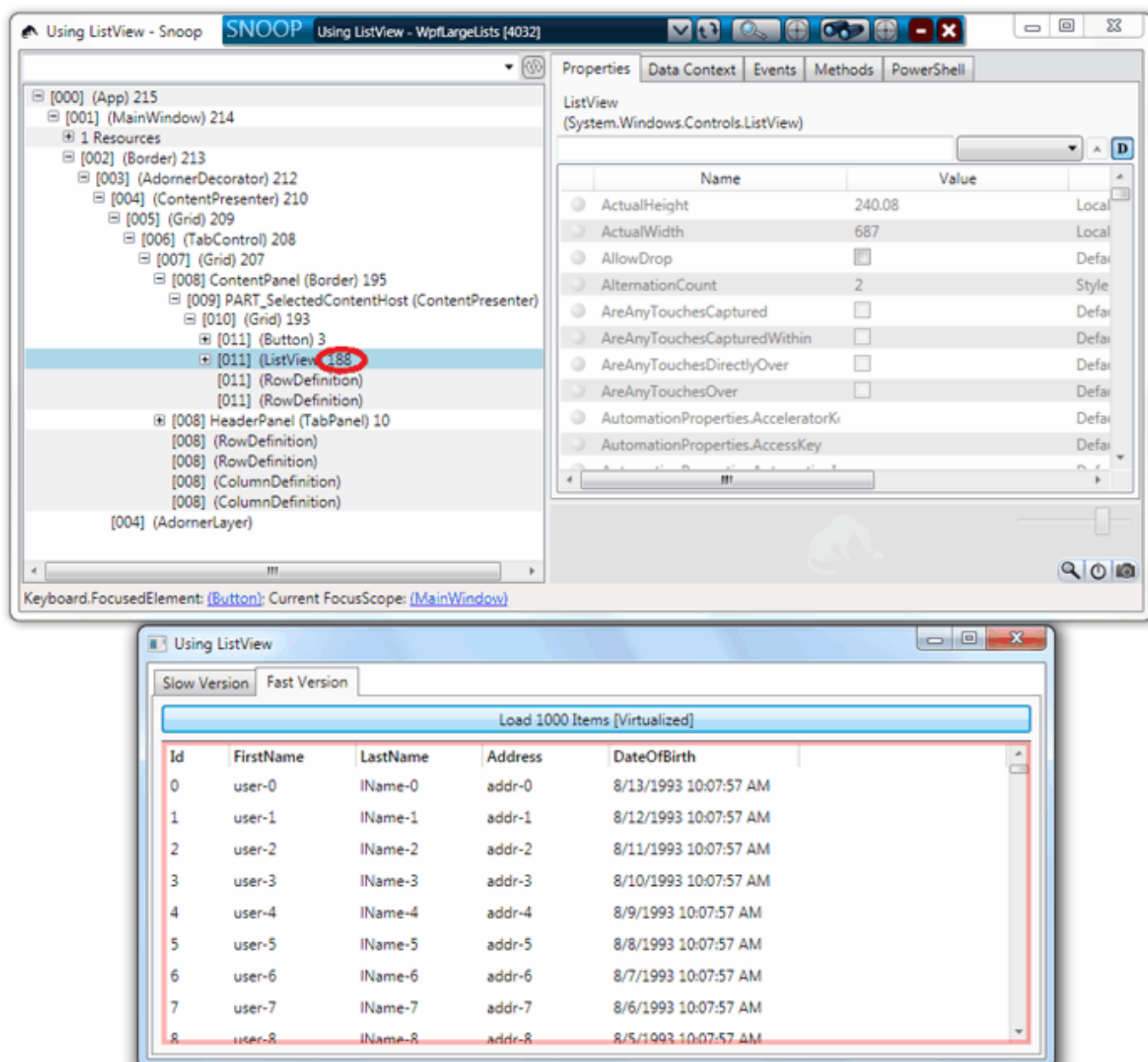
شبیه به افزونه FireBug فایرفاکس، برنامه‌ای به نام Snoop نیز جهت WPF تهیه شده است که با تزریق خود به درون پروسه برنامه، امکان بررسی ساختار Visual tree کل یک صفحه را فراهم می‌کند. برای دریافت آن به آدرس ذیل مراجعه نمایید:

<http://snoopwpf.codeplex.com>

پس از دریافت، ابتدا مثال انتهای بحث « [بهبود کارایی کنترل‌های لیستی WPF در حین بارگذاری تعداد زیادی از رکوردها](#) » را اجرا کرده و سپس برنامه Snoop را نیز جداگانه اجرا نمایید. اگر نام برنامه WPF مورد نظر، در لیست برنامه‌های تشخیص داده شده توسط Snoop ظاهر نشد، یکبار بر روی دکمه Refresh آن کلیک نمایید. پس از آن برنامه نمایش لیست‌ها را در Snoop انتخاب کرده و دکمه کنار آیکن minimize کردن Snoop را کشیده و بر روی پنجره برنامه رها کنید. شکل زیر ظاهر خواهد شد:



بله. همانطور که ملاحظه می‌کنید، در برگه Slow version به علت فعال نبودن مجازی سازی UI، تعداد اشیاء موجود در Visual tree کنترل لیستی، بالای 10 هزار مورد است. به همین جهت بارگذاری آن بسیار کند انجام می‌شود. اکنون همین عملیات کشیدن و رها کردن دکمه بررسی Snoop را بر روی برگه دوم برنامه انجام دهید:



در اینجا چون مجازی سازی UI فعال شده است، فقط به تعداد ردیف‌های نمایان به کاربر، اشیاء لازم جهت نمایش لیست، تولید و اضافه شده‌اند که در اینجا فقط 188 مورد است و در مقایسه با 10 هزار مورد برگه اول، بسیار کمتر می‌باشد و بدیهی است در این حالت مصرف حافظه برنامه بسیار کمتر بوده و همچنین سرعت نمایش لیست نیز بسیار بالا خواهد بود.

قسمت هفتم

22. استفاده از CSS Sprites

ایده اصلی این تکنیک به این صورت است که تمامی عکس‌های کوچک (در اینجا همه 100 عکس) در قالب یک تصویر بزرگ قرار خواهد گرفت و با استفاده از CSS مختصات هر عکس کوچک را در تصویر بزرگ پیدا کرده و نمایش می‌دهیم. یکی شدن 100 عکس کوچک به یک عکس بزرگ، تاثیر زیادی در پایین آمدن حجم عکس جدید خواهد داشت و مرورگر شما به جای درخواست 100 عکس از سرور، تنها یکی دانلود می‌کند و از این به بعد از کش مرورگر برای بازیابی آن استفاده می‌کند. این موضوع به معنی ترافیک کمتر شبکه و آزاد شدن منابع پر ارزش حافظه، cpu و پهنای باند در سمت سرور و کاربران. برای اطلاع بیشتر از این تکنیک می‌توانید [به این مقاله](#) مراجعه نمایید.

23. استفاده مطلوب از AJAX

شما می‌توانید برای لود کردن بخش‌های مخفی در صفحه خود از AJAX کمک بگیرید. به جای دانلود کردن تمامی بخش‌های صفحه در مرورگر کاربر، بخش‌هایی که در دید کاربر قرار ندارد را به صورت AJAX بارگیری کنید. نمونه ای از این تکنیک را در این [صفحه](#) مشاهده نموده و البته از کد آن استفاده نمایید.

24. حذف HTTP modules های اضافی

HTTP modules هایی را که در برنامه خود استفاده نمی‌کنید را حذف کنید. این کار یعنی سربار مدیریتی کمتر در مازول ASP.NET سرور شما. برای اجرای این مورد می‌توانید از کدی مشابه این کد در web.config خود استفاده کنید:

```
<httpModules>
  <remove name="OutputCache"/>
  <remove name="Session"/>
  <remove name="WindowsAuthentication"/>
  <remove name="FormsAuthentication"/>
  <remove name="PassportAuthentication"/>
  <remove name="RoleManager"/>
  <remove name="UrlAuthorization"/>
  <remove name="FileAuthorization"/>
  <remove name="AnonymousIdentification"/>
  <remove name="Profile"/>
  <remove name="ErrorHandlerModule"/>
  <remove name="ServiceModel"/>
</httpModules>
```

البته حواستان به این موضوع باشد مازول‌های مورد استفاده در برنامه خود را حذف نکنید که در این صورت ممکن است این آخرین پروژه شما با صاحب کارتان باشد!

چرا افسانه‌ای که می‌گوید PHP از ASP.NET سریعتر است اینقدر شایع است؟ در این مقاله به بیان حقایق می‌پردازیم که این افسانه را زیر سوال می‌برد؟

خیلی وقتها در بسیاری از نوشته‌ها و اظهارنظرها می‌بینیم ادعا می‌شود که PHP بسیار سریعتر از ASP.net است و اینکه ASP.net از لحاظ سرعت کند است. آزار دهنده‌ترین بخش این ادعاها، آن است که هر یک از آنها را که نگاه می‌کنی بصورت کاملاً غیر واقع بینانه به موضوع نگاه می‌کنند و فقط بدون دلیل این موضوع را ادعا می‌کنند. زیرا به این موضوع بصورتی کاملاً متعصبانه و بدون از واقعیتها نگاه می‌شود. به همین دلیل بصورت گسترده ای این افسانه در میان اهالی وب پذیرفته شده است.

حال بجای اینکه این موضوع را بارها و بارها در جاهای مختلف بیان کنیم، این مقاله را نوشته و در هر کجا که لازم باشد به آن ارجاع خواهیم داد. باید توجه کنید این حقیقت که زبان PHP یک زبان اصیل و قدرتمند است هیچ شکی در آن نیست اما اینکه خواهیم بصورت مغرضانه و به این دلیل که ما از این زبان استفاده می‌کنیم، آنرا از هر لحاظ برتر از سایر زبانها بدانیم (کمی که نه) بسیار اغراق آمیز است.

این مقاله برای این نیست که ما هریک از این زبانها را زیر سوال ببریم. بلکه برای آن است که این موضوع را با دلایل منطقی و حقیقی بررسی کنیم که آیا اینکه می‌گویند PHP از ASP.net سریعتر است واقعیت دارد یا نه؟

Compiled در مقابل Interpreted Languages:

قبل از هرچیز ذکر این نکته الزامی است که این دو زبان تفاوت‌های اساسی در base دارند. ASP.net یک زبان بهینه سازی و کامپایل شده است، به این معنی که کدهای نوشته شده در این زبان قبل از اینکه قابل اجرا شوند، به مجموعه ای از دستورالعمل‌های خاص ماشین تبدیل می‌شوند. از سوی دیگر PHP یک زبان تفسیر شده است، به این معنی که کدهای نوشته شده به همان شکل ذخیره شده و در زمان اجرا این کدها تفسیر می‌شوند. این موضوع بطور گسترده‌ای پذیرفته شده و ثابت شده است که برنامه‌های کامپایل شده به مراتب سریعتر از برنامه‌های تفسیر شده اجرا می‌شوند، به این دلیل که برنامه‌های تفسیر شده نیاز دارند تا در زمان اجرا به دستورالعمل‌های ماشین تبدیل شوند.

در اینجا به یک نقل قول از دانشنامه آزاد ویکی پدیا اشاره می‌کنم که میزان سریعتر بودن برنامه‌های کامپایل شده را نشان می‌دهد:

[A program translated by a compiler tends to be much faster than an interpreter executing the same program: even "a 10:1 ratio is not uncommon. The mixed solution's efficiency is typically somewhere in between](#)

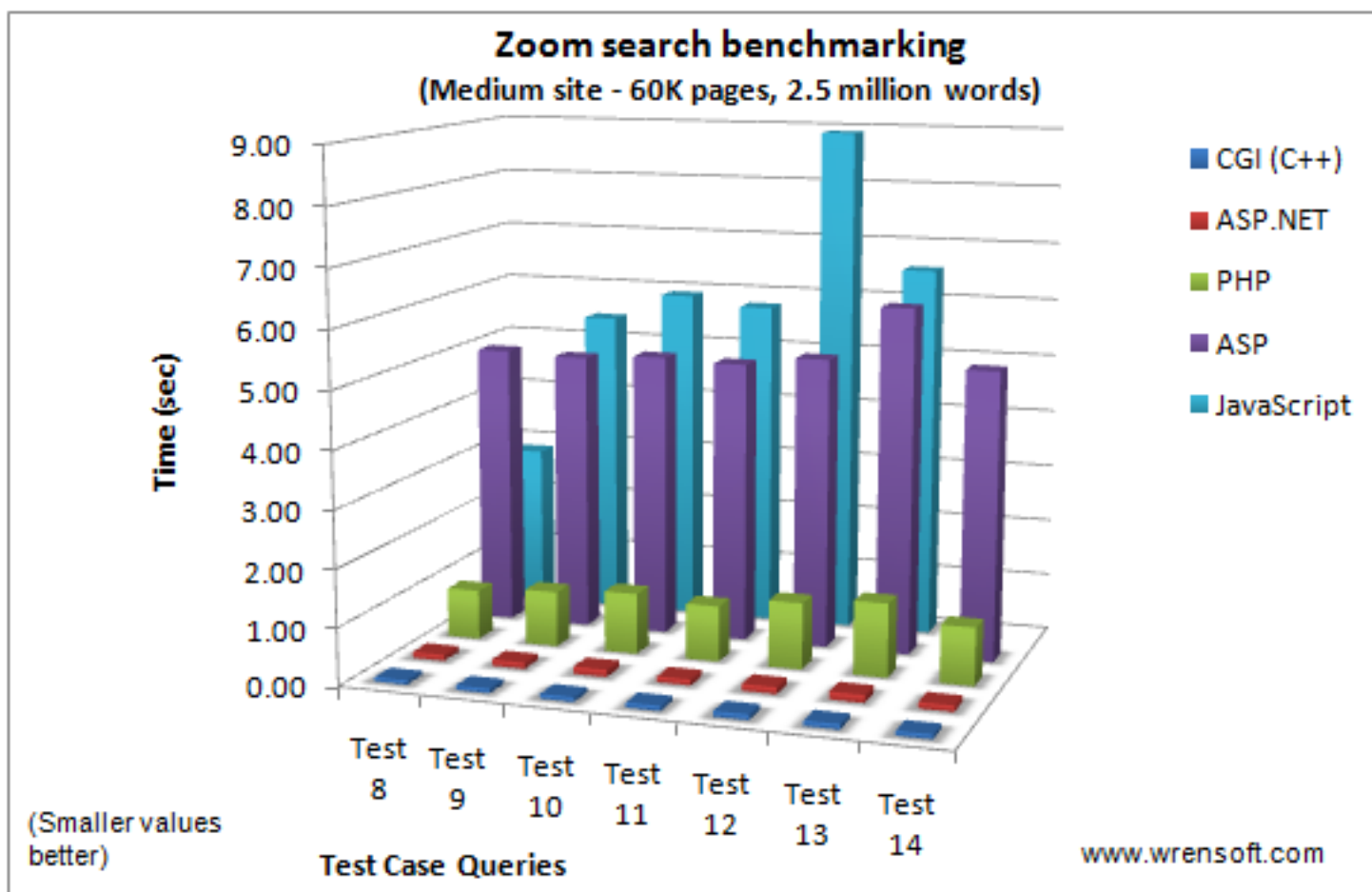
به این معنا که یک برنامه بصورت کامپایل شده بسیار سریعتر از همان برنامه بصورت تفسیر شده، اجرا می‌شود.

اعداد و ارقام:

حال که تئوری خود را مبنی بر دلیل سریعتر بودن ASP.net بیان کردیم بیایید با هم نگاهی به برخی آمارها بیاندازیم تا این تئوری را در عمل هم نشان داده باشیم.

آمارهای زیر توسط شرکت WrenSoft جهت مقایسه زمان اجرای یک کد مشابه در زبانهای مختلف تهیه شده است. اگر می‌خواهید توصیف عمیق‌تری از آمارها داشته باشید لطفاً لینک را دنبال کنید.

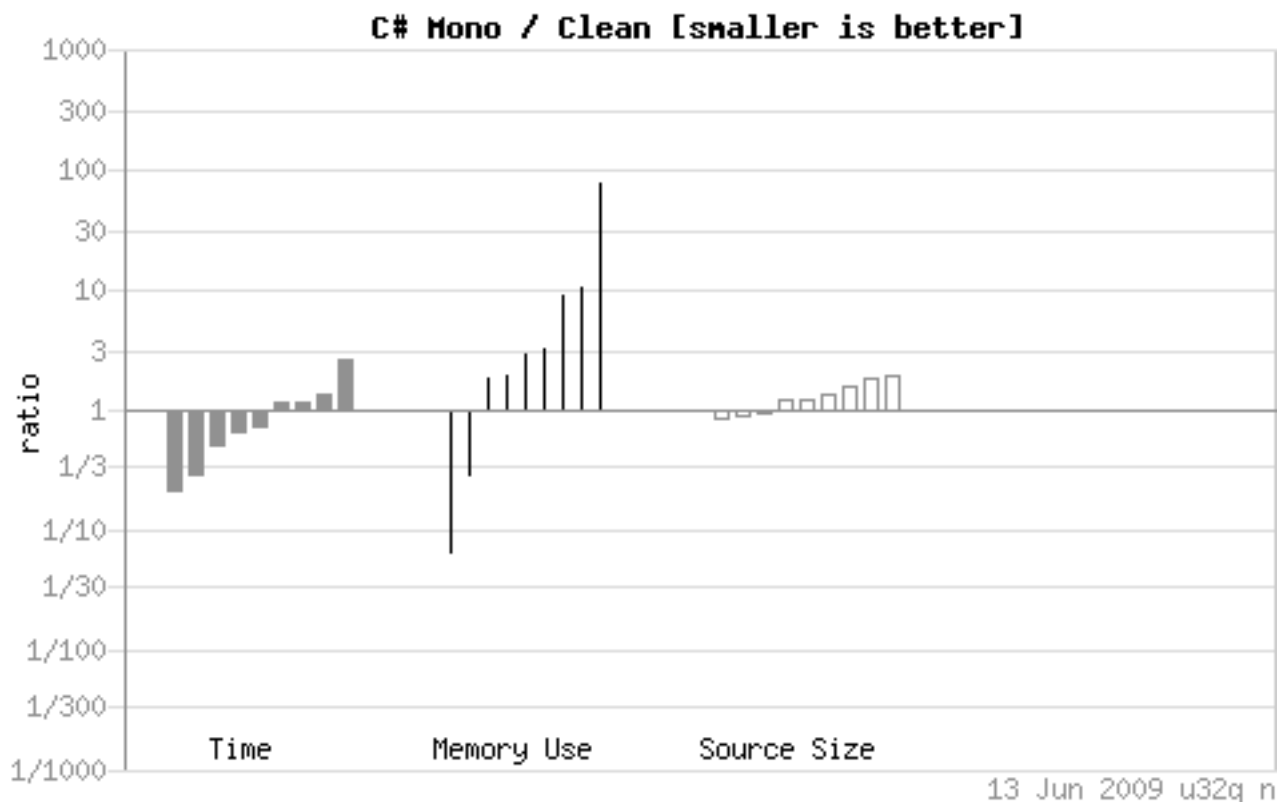
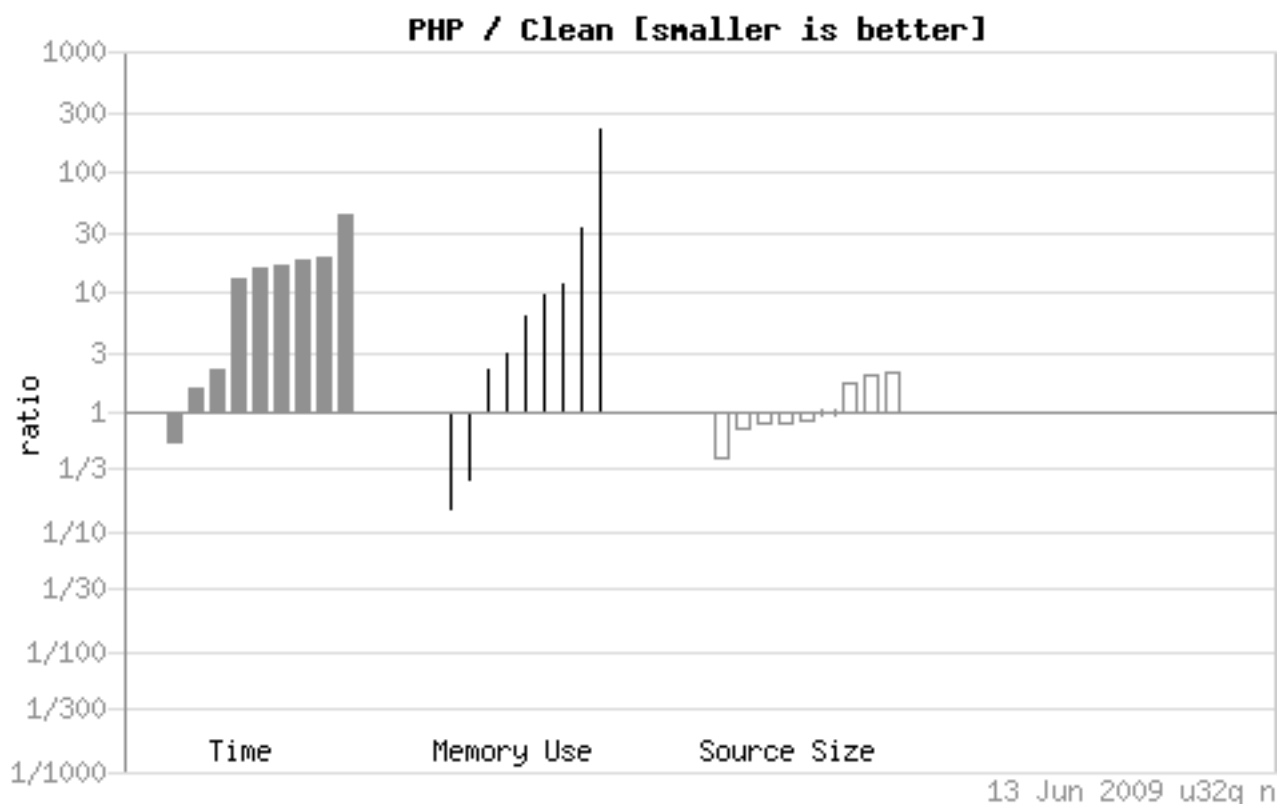
نمودار اول: زمان صرف شده برای تولید و نمایش نتایج برای جستجوی وب سایت‌های کوچک



PHP، 1.0097 ثانیه طول می‌کشد در حالی که ASP.net، 0.0810 ثانیه زمان نیاز دارد. می‌بینیم که PHP دوازده بار بیشتر از ASP.net زمان می‌برد.

در حال حاضر این آزمون با یک کد مشابه در زبانهای برنامه نویسی مختلف پیاده سازی و اجرا شد و نتیجه را مشاهده نمودید. حال این موضوع پیش می‌آید که این اجرای کدها بر روی سیستم عامل ویندوزی بوده است و این می‌تواند به نفع ASP.net باشد، پس همین آزمون را بر روی سیستم عامل لینوکسی مشاهده می‌کنیم.

آمارهای زیر از سایت معتبر shootout.alioth.debian.org گرفته شده است. این آمارها نحوه اجرای همان کد را بر روی سیستم عامل لینوکسی برای هردو زبان نشان می‌دهد:



همانطور که مشاهده می‌کنید در سیستم لینوکسی نیز همچنان ASP.NET سریعتر از PHP عمل می‌کند.

نتیجه گیری:

همین حالا جمله‌ی "asp.net vs php speed" را در google جستجو کنید. خواهید دید که در اکثر پست‌ها گفته شده که PHP از ASP.net سریعتر است اما دلیلی بر این ادعا نخواهید یافت و فقط در حد حرف است. مشکل این است که اکثر مردم وقتی چیزی را زیاد می‌بینند یا زیاد می‌شنوند بدون آنکه دلیل بخواهند آنرا می‌پذیرند و حتی بعضی اوقات از آن نیز دفاع می‌کنند که واقعا جای تاسف دارد.

توسعه وب بوسیله PHP کار خوبی است، بسیاری از اپلیکیشن‌ها و وبسایت‌های شگفت انگیز توسط این زبان نوشته شده اند. اگر احساس می‌کنید PHP یک زبان برتر است از آن استفاده کنید اما این دلیل نمی‌شود که اطلاعات غلط را به دیگران القاء کنید و بدون دلیل و مدرک این زبان را از هر لحاظ برتر بدانید حال آنکه در این مقاله دیدیم که براساس چیزی که ارائه شد، **ASP.net سرعت بیشتری نسبت به PHP دارد**.

اگر با من در این امر موافق نیستید می‌توانید با نظرهای مستدل خود ما را راهنمایی کنید.

نظرات خوانندگان

نویسنده: سیروس
تاریخ: ۱۳۹۲/۰۶/۲۶ ۱۳:۱

به دلیل وسعت استفاده بیشتر از php و نیز استفاده سایتها و شرکت‌های بزرگ از php خیلی‌ها فکر می‌کنند php بهتر و سریعتر از asp.net هست در حالیکه این وسعت استفاده بخاطر اوپن سورس و رایگان بودن php هست و چون وب سرور apache هم معمولاً رو لینوکس نصب میشه و خود لینوکس هم اپن سورس، تمام این دلایل دست به دست هم داده تا php بهتر به نظر بیاد. جدا از بحث سرعت اگر از لحاظ ساختاری بررسی کنیم php بیشتر یک زبان اسکریپتی است تا برنامه نویسی و ویژگی‌های زبان‌های خوب و شی گرا رو نداره.

نویسنده: مسلم
تاریخ: ۱۳۹۲/۰۶/۲۶ ۱۳:۴

نمی‌دونم چرا ولی توی عمل واسه استارت زدن دات نت خیلی دیر می‌جنبه. حتی بعد از کامپایل و پابلیش یه خورده تاخیر داره ولی یکم که باهاش کار کنی می‌فته رو دور و خوب میشه! ولی پی‌اچ‌پی همون اول سریعه. شاید بخاطر پیچیدگی فریم‌ورک هست. چرا که entity, linq هم در پروژه استفاده شده است.

نویسنده: محسن خان
تاریخ: ۱۳۹۲/۰۶/۲۶ ۱۳:۲۰

چندین علت داره:

- پروسه کامپایل کدهای دات نت یک مرحله‌ای نیست. کلاً در دات نت کدها به یک زبان میانی به نام IL ترجمه میشن. بعد این IL توسط JIT compiler تبدیل به کدهای ماشین میشه. در ASP.NET این مساله هم برای کدهای پشت صحنه برنامه و هم برای خود صفحات رخ می‌ده. بنابراین برای بار اول مشاهده، روند این پروسه الزامی هست. ولی برای دفعات بعدی مشاهده خیر. البته روش برای پیش کامپایل کردن کامل صفحات هم وجود داره و یا در IISهای جدید یک سری مبحث warmup توکار پیش بینی شده.

- همچنین IIS برای مدیریت منابع سرور، یک سایت رو مدام در حافظه نگه نمی‌داره. فقط زمانیکه اولین درخواست به سرور میرسه سایت رو بارگذاری می‌کنه و application pool اون رو استارت. این هم یک زمان اولیه اندکی رو ممکنه به خودش اختصاص بده. بعلاوه پس از مدتی، یک سایت بیکار رو از حافظه خارج می‌کنه. بهش می‌گن ریسایکل کردن. در ASP.NET 4.0 به بعد امکان تنظیم auto-start برای سایت‌ها هست.

نویسنده: محسن خان
تاریخ: ۱۳۹۲/۰۶/۲۶ ۱۲:۳۰

به نظر من برای بحث در مورد PHP مقایسه سرعت در رده آخر اهمیت هست. مسایل بهتری برای بحث وجود دارند. مثلاً:

- [بدترین زبانی که تابحال با آن کار کردید، کدوم بوده؟](#)

- [The PHP Singularity](#)

- [PHP: a fractal of bad design](#)

- [PHP Sadness](#)

نویسنده:

مهدی سعیدی فر

تاریخ:

۱۳:۵۰ ۱۳۹۲/۰۶/۲۶

هر کسی به من گفت php سریعتر هست و یا ASP.NET سریعتر هست؛ من هم در جواب گفتم شما درست می‌فرمایید و هیچ گاه با آن‌ها بحث نکردم. هنوز که هنوزه من نمی‌فهمم که واقعا دارید چی را با چی مقایسه کنید. ASP.NET و php کاملا دو مقوله‌ی متفاوت هستند. اگر قرار است مقایسه ای در سرعت عمومی انجام شود، معقول‌تر است که در سطح فریم ورک هایی با قدرت برابر انجام شود؛ برای مثال سرعت عمومی Zend بالاتر است یا ASP.NET MVC. اصلا بهتر است یه مقایسه با مستندات کافی برای شما مطرح کنم تا به کندی ASP.NET MVC پی ببرید: هدف از این برنامه نمایش عبارت Hello,World در مرورگر است. در php خام نوشتن کد زیر کفایت می‌کند:

```
echo 'Hello, World';
```

اما در ASP.NET MVC شما باید ابتدا یک کنترلر تعریف کرده سپس در یک Action عبارت Hello,World را بازگشت دهید. اگر این دو برنامه را اجرا کنید از سرعت فوق العاده‌ی php متحیر خواهید شد. البته بگذریم از اینکه در ASP.NET سربارهای به نام Routing و ... در ابتدای کار وجود دارد. نتیجه این هست که ASP.NET خیلی کند و حرفی برای گفتن ندارد در مقابل php. از این دست مقایسه‌ها من هم زیاد دیدم. واقعا خودشان هم نمی‌فهمند چی را با چی مقایسه می‌کنند. این نوع مقایسه‌ها بیشتر منو یاد کسی می‌اندازه که گوشی موبایل خریده بود چهار هسته ای و می‌گفت چقدر تکنولوژی پیشرفت کرده که از لپ‌تاپم دو هسته بیشتر دارد و سریعتره! من هم با خواندن این مقاله به جمله‌ی شما درست می‌فرمایید بسنده می‌کنم.

نویسنده:

محسن خان

تاریخ:

۱۲:۵۷ ۱۳۹۲/۰۶/۲۶

مثالت بی‌ربطه دوست عزیز. echo خام در PHP معادل هست با Response.Write خام در ASP.NET در حالیکه در یک فایل ashx اجرا می‌شود. احتمالا می‌دونی که این نوع فایل‌ها در حالت پیش فرض حتی مازول سشن هم براشون فعال نیست چه برسد به مسیریابی و در حداقل سربار کار می‌کنند.

نویسنده:

مهدی سعیدی فر

تاریخ:

۱۳:۳ ۱۳۹۲/۰۶/۲۶

خب منظور من دقیقا همین بود. یک مقایسه کاملا بی ربط. یک فریم ورک با کلی امکانات را با یک زبان خام مقایسه کردم! پی نوشت: من در آینده نظر قبلی را دادم. ساعت 13:50!

نویسنده:

محسن خان

تاریخ:

۱۳:۹ ۱۳۹۲/۰۶/۲۶

خوب، مثال‌های بحث جاری در ساده‌ترین حالت ممکن تهیه شدند. یعنی یک فریم ورک ASP.NET با کلی دم و دستگاه (ماژول سشن، ماژول مسیریابی، ماژول امنیت، ماژول اعتبارسنجی درخواست‌ها، ماژول فشرده سازی و ...) از یک سیستم ساده PHP سریعتر عمل می‌کنه. این چطور بی‌ربط به عنوان مقاله هست؟

نویسنده:

مهدی سعیدی فر

تاریخ:

۱۳:۲۲ ۱۳۹۲/۰۶/۲۶

آخه من از منبع این مقاله چیزی نفهمیدم.

کلا منبع داره در مورد یه چیز دیگه صحبت می‌کنه. <http://www.wrensoft.com/zoom/benchmarks.html>

نویسنده: احسان

تاریخ: ۱۴:۱۵ ۱۳۹۲/۰۶/۲۶

فکر کنم خیلی مشخصه که یک زبان کامپیل شده چقدر میتونه از یک واسطه سریعتر باشه کاش با جاوا مقایسه میکردی

نویسنده: رضا منصوری

تاریخ: ۱۸:۲۱ ۱۳۹۲/۰۶/۲۶

توسعه وب بوسیله PHP کار خوبی است، بسیاری از اپلیکیشن‌ها و وبسایت‌های شگفت انگیز توسط این زبان نوشته شده اند. اگر احساس می‌کنید PHP یک زبان برتر است از آن استفاده کنید اما این دلیل نمی‌شود که اطلاعات غلط را به دیگران القاء کنید و بدون دلیل و مدرک این زبان را از هر لحاظ برتر بدانید حال آنکه در این مقاله دیدیم که براساس چیزی که ارائه شد، **ASP.net سرعت بیشتری نسبت به PHP دارد**.

نویسنده: ناصر فرجی

تاریخ: ۲۲:۵۰ ۱۳۹۲/۰۶/۲۶

مطلبی مشابه که چند روز پیش خوندم. لحظاتی یاد بحث‌های فروم برنامه نویسی افتادم (:

<http://www.rezashirazi.com/post261.aspx>

نویسنده: آرایه

تاریخ: ۲۱:۳۷ ۱۳۹۲/۰۶/۲۷

به نظرم این بحث سود چندانی برای خوانندگان ندارد، benchmarkها بر مبنای کد مشابه هستند اما عملاً کد مشابه روی دو پلتفرم مختلف رو نمی‌شه مقایسه کرد. مطلبی در این خصوص نوشتم: [اینجا](#)

نویسنده: آرمان فرقانی

تاریخ: ۸:۹ ۱۳۹۲/۰۶/۲۸

فناوری-زبان PHP بسیار محترم است و بسیار قابل توصیه اما برای آن زمانی که در رقابت با ASP کلاسیک بود و پدیده ای به نام دات نت ظهور نکرده بود حال آنکه پدیده یاد شده در زمان حال به پختگی و پیشرفت چشمگیری دست یافته است. دنیای دات نت و مباحث مربوط به آن گسترده تر و پیچیده تر از آن است که برای توجیه استفاده از دات نت سرعت مقایسه شود. ده‌ها ویژگی منحصر به دات نت وجود دارد که برای انتخاب فناوری سمت سرور مجالی برای تفکر درباره سرعت باقی نمی‌گذارد و آنان که اهل تفکر باشند را جذب خود می‌کند و نه گمراهان (کسانی که یا تعصب دارند یا خسته تر از آن هستند که دنیای جدیدی را تجربه نمایند).

در مورد سرعت کافی است نکات ساده ای که چند دقیقه بیشتر زمان نمی‌برند رعایت شود تا وب سایت‌های دات نتی چندین برابر سریعتر عمل کنند.

فراموش نکنید دنیای دات نت تا حد بیشتری با اصول مهندسی نرم افزار گره خورده است و باب میل همگان نیست. عموماً سرویس دهنده بر حسب فناوری انتخاب می‌شود و نه برعکس! در مقالاتی که مقایسه انجام می‌دهند صحبت از رایگان بودن لینوکس و آپاچی و ... چندان ضرورتی ندارد.

بسیاری از آن‌ها که Open Source بودن PHP را با آب و تاب و به عنوان برتری مطرح می‌کردند هرگز در عمر خود به کدهای سورس آن نگاهی نکرده اند. البته اگر بدانند از کجا باید دانلود کنند. توصیه می‌کنم در مورد رویکرد و سیاست‌های چند سال اخیر مایکروسافت در رابطه با Open Source تحقیق بیشتری صورت گیرد. مثال هایی از سایت‌های موفق و یا تعداد سایت‌ها در یک فناوری هرگز دلیلی برای انتخاب فناوری نیست. ضمناً زمان ظهور این دو فناوری یکسان نبوده است که تعداد وب سایت‌ها معیار مقایسه باشد. فناوری-زبان PHP هنوز وجود دارد. هنوز قدرت خود را دارد. و هنوز هر کجا به هر دلیلی امکان استفاده از دات نت نبود، با کمال

افتخار و خوشحالی از این که چند سال عمری که برای آن گذاشته ام هدر نرفته است به آن باز می‌گردد و آن را مورد استفاده قرار می‌دهم.

نویسنده: نیما
تاریخ: ۱۳۹۲/۰۶/۳۰ ۷:۱۸

با سلام خدمت دوستان گرامی

نکته ای که باید به آن دقت داشت، فرق بین مجانی و سورس باز است. لطفا این موارد را از یکدیگر جدا نمایید.

نکته دوم هم اینست که چند نفر در کشور خودمون داریم که میتوانند سورس PHP، لینوکس و ... را سفارشی کنند؟

نویسنده: حامد وهابی املشی
تاریخ: ۱۳۹۲/۰۷/۰۱ ۱۴:۸

چرا در نمودار جاوا اسکریپت توی نمودار قرار گرفته ؟ اونها زبانهای سمت سرورند ، جاوا اسکریپت سمت کلاینت . البته ممکنه منظورش جاوا اسکریپت سمت سرور باشه ! در این صورت هم باز اشتباهه ، چون جاوا اسکریپت سمت سرور مربوط به تکنولوژی ASP کلاسیک بوده که توی نمودار اون رو هم آوردن . ASP کلاسیک هم وبی اسکریپت داشت و هم جاوا اسکریپت.

نویسنده: محسن خان
تاریخ: ۱۳۹۲/۰۷/۰۱ ۱۴:۱۷

منظورش [پیاده سازی خاص خودشون](#) بوده.

نویسنده: حامد وهابی املشی
تاریخ: ۱۳۹۲/۰۷/۰۱ ۱۵:۱۹

بلاخره پیاده سازی خودش ، سمت سرور بوده ؟

نویسنده: محسن خان
تاریخ: ۱۳۹۲/۰۷/۰۱ ۱۸:۸

بله. ضمناً [node.js](#) یک فناوری دیگر سمت سرور مبتنی بر جاوا اسکریپت است (و از این دست [دارن زیاد میشن](#))

نویسنده: علی یگانه مقدم
تاریخ: ۱۳۹۳/۰۹/۱۴ ۵:۱۰

البته این مورد هم هست که بسیاری از شرکت‌ها یا افراد که از php استفاده می‌کنن به خاطر عدم وابستگی‌شون به یک شرکت خاص چون مایکروسافت هم هست مثل گوگل که کلا استفاده از برنامه‌های شرکت مایکروسافت رو در شرکتش ممنوع اعلام کرده، کلا عدم وابستگی پی‌اچ‌پی و همچنین متن باز بودنش برای به روزآوری و تغییراتش توسط برنامه نویسان سراسر جهان بیشتر مورد توجه قرار گرفته.

البته این نکته هم هست که اکثر مردم با دیدن آدرس‌های بدون پسوند فکر می‌کنن که php هست مثلاً همین stack overflow رو خیلی‌ها فکر می‌کنن با php نوشتن.

البته بودن بیشتر سیستم‌های آماده از نوع php و همچنین ارزونتر بودن هاست‌های لینوکس به خصوص در ایران هم سبب این اتفاق بوده.

صفحات خروجی وب سایت زمانی که رندر شده و در مرورگر نشان داده می‌شود شامل فواصل اضافی است که تأثیری در نمایش سایت نداشته و صرفاً این کاراکترها فضای اضافی اشغال می‌کنند. با حذف این کاراکترهای اضافی می‌توان تا حد زیادی صفحه را کم حجم کرد. برای این کار در ASP.NET Webform کارهایی (^) انجام شده است.

روال کار به این صورت بوده که قبل از رندر شدن صفحه در سمت سرور خروجی نهایی بررسی شده و با استفاده از عبارات با قاعده الگوهای مورد نظر لیست شده و سپس حذف می‌شوند و در نهایت خروجی مورد نظر حاصل خواهد شد. برای راحتی کار و عدم نوشتن این روال در تمامی صفحات می‌تواند در مستر پیج این عمل را انجام داد. مثلاً:

```
private static readonly Regex RegexBetweenTags = new Regex(@">\s+<", RegexOptions.Compiled);
private static readonly Regex RegexLineBreaks = new Regex(@"\r\s+", RegexOptions.Compiled);

protected override void Render(HtmlTextWriter writer)
{
    using (var htmlwriter = new HtmlTextWriter(new System.IO.StringWriter()))
    {
        base.Render(htmlwriter);
        var html = htmlwriter.InnerWriter.ToString();

        html = RegexBetweenTags.Replace(html, "> <");
        html = RegexLineBreaks.Replace(html, string.Empty);
        html = html.Replace("/*<![CDATA["", ""].Replace("//"]>", "");
        html = html.Replace("// <![CDATA["", ""].Replace("// ]>", "");

        writer.Write(html.Trim());
    }
}
```

در هر صفحه رویدادی به نام Render وجود دارد که خروجی نهایی را می‌توان در آن تغییر داد. همانگونه که مشاهده می‌شود عملیات یافتن و حذف فضاهای خالی در این متد انجام می‌شود.

این عمل در ASP.NET Webform به آسانی انجام شده و باعث حذف فضاهای خالی در خروجی صفحه می‌شود.

برای انجام این عمل در ASP.NET MVC روال کار به این صورت نیست و نمی‌توان مانند ASP.NET Webform عمل کرد.

چون در MVC از ViewPage استفاده می‌شود و ما مستقیماً به خروجی آن دسترسی نداریم یک روش این است که می‌توانیم یک کلاس برای ViewPage تعریف کرده و رویداد Write آن را تحریف کرده و مانند مثال بالا فضای خالی را در خروجی حذف کرد. البته برای استفاده باید کلاس ایجاد شده را به عنوان فایل پایه جهت ایجاد صفحات در MVC فایل web.config معرفی کنیم. این روش در [اینجا](#) به وضوح شرح داده شده است.

اما هدف ما پیاده سازی با استفاده از اکشن فیلتر هاست. برای پیاده سازی ابتدا یک اکشن فیلتر به نام CompressAttribute تعریف می‌کنیم مانند زیر:

```
using System;
using System.IO;
using System.IO.Compression;
using System.Text;
using System.Text.RegularExpressions;
using System.Web;
using System.Web.Mvc;

namespace PWS.Common.ActionFilters
{
    public class CompressAttribute : ActionFilterAttribute
    {
        #region Methods (2)

        // Public Methods (1)

        /// <summary>
        /// Called by the ASP.NET MVC framework before the action method executes.
        /// </summary>
        /// <param name="filterContext">The filter context.</param>
```

```

public override void OnActionExecuting(ActionExecutingContext filterContext)
{
    var response = filterContext.HttpContext.Response;
    if (IsGZipSupported(filterContext.HttpContext.Request))
    {
        String acceptEncoding = filterContext.HttpContext.Request.Headers["Accept-Encoding"];
        if (acceptEncoding.Contains("gzip"))
        {
            response.Filter = new GZipStream(response.Filter, CompressionMode.Compress);
            response.AppendHeader("Content-Encoding", "gzip");
        }
        else
        {
            response.Filter = new DeflateStream(response.Filter, CompressionMode.Compress);
            response.AppendHeader("Content-Encoding", "deflate");
        }
    }
    // Allow proxy servers to cache encoded and unencoded versions separately
    response.AppendHeader("Vary", "Content-Encoding");
    // حذف فضاهای خالی

    response.Filter = new WhitespaceFilter(response.Filter);
}
// Private Methods (1)

/// <summary>
/// Determines whether [is G zip supported] [the specified request].
/// </summary>
/// <param name="request">The request.</param>
/// <returns></returns>
private Boolean IsGZipSupported(HttpRequestBase request)
{
    String acceptEncoding = request.Headers["Accept-Encoding"];

    if (acceptEncoding == null) return false;
    return !String.IsNullOrEmpty(acceptEncoding) && acceptEncoding.Contains("gzip") ||
acceptEncoding.Contains("deflate");
}

#endregion Methods
}

/// <summary>
/// Whitespace Filter
/// </summary>
public class WhitespaceFilter : Stream
{
#region Fields (3)

    private readonly Stream _filter;
    /// <summary>
    ///
    /// </summary>
    private static readonly Regex RegexAll = new Regex(@"\s+|\t\s+|\n\s+|\r\s+",
RegexOptions.Compiled);
    /// <summary>
    ///
    /// </summary>
    private static readonly Regex RegexTags = new Regex(@">\s+<", RegexOptions.Compiled);

#endregion Fields

#region Constructors (1)

    /// <summary>
    /// Initializes a new instance of the <see cref="WhitespaceFilter" /> class.
    /// </summary>
    /// <param name="filter">The filter.</param>
    public WhitespaceFilter(Stream filter)
    {
        _filter = filter;
    }

#endregion Constructors

#region Properties (5)

```

```

    /// 
    /// When overridden in a derived class, gets a value indicating whether the current stream
    supports reading.
    /// 
    /// true if the stream supports reading; otherwise, false.</returns>
    public override bool CanRead
    {
        get { return true; }
    }

    /// 
    /// When overridden in a derived class, gets a value indicating whether the current stream
    supports seeking.
    /// 
    /// true if the stream supports seeking; otherwise, false.</returns>
    public override bool CanSeek
    {
        get { return true; }
    }

    /// 
    /// When overridden in a derived class, gets a value indicating whether the current stream
    supports writing.
    /// 
    /// true if the stream supports writing; otherwise, false.</returns>
    public override bool CanWrite
    {
        get { return true; }
    }

    /// 
    /// When overridden in a derived class, gets the length in bytes of the stream.
    /// 
    /// A long value representing the length of the stream in bytes.</returns>
    public override long Length
    {
        get { return 0; }
    }

    /// 
    /// When overridden in a derived class, gets or sets the position within the current stream.
    /// 
    /// The current position within the stream.</returns>
    public override long Position { get; set; }

#endregion Properties

#region Methods (6)

// Public Methods (6)

    /// 
    /// Closes the current stream and releases any resources (such as sockets and file handles)
    associated with the current stream. Instead of calling this method, ensure that the stream is properly
    disposed.
    /// 
    public override void Close()
    {
        _filter.Close();
    }

    /// 
    /// When overridden in a derived class, clears all buffers for this stream and causes any
    buffered data to be written to the underlying device.
    /// 
    public override void Flush()
    {
        _filter.Flush();
    }

    /// 
    /// When overridden in a derived class, reads a sequence of bytes from the current stream and
    advances the position within the stream by the number of bytes read.
    /// 
    /// 

```

```

    /// <returns>
    /// The total number of bytes read into the buffer. This can be less than the number of bytes
    requested if that many bytes are not currently available, or zero (0) if the end of the stream has been
    reached.
    /// </returns>
    public override int Read(byte[] buffer, int offset, int count)
    {
        return _filter.Read(buffer, offset, count);
    }

    /// <summary>
    /// When overridden in a derived class, sets the position within the current stream.
    /// </summary>
    /// <param name="offset">A byte offset relative to the <paramref name="origin" />
    parameter.</param>
    /// <param name="origin">A value of type <see cref="T:System.IO.SeekOrigin" /> indicating the
    reference point used to obtain the new position.</param>
    /// <returns>
    /// The new position within the current stream.
    /// </returns>
    public override long Seek(long offset, SeekOrigin origin)
    {
        return _filter.Seek(offset, origin);
    }

    /// <summary>
    /// When overridden in a derived class, sets the length of the current stream.
    /// </summary>
    /// <param name="value">The desired length of the current stream in bytes.</param>
    public override void SetLength(long value)
    {
        _filter.SetLength(value);
    }

    /// <summary>
    /// When overridden in a derived class, writes a sequence of bytes to the current stream and
    advances the current position within this stream by the number of bytes written.
    /// </summary>
    /// <param name="buffer">An array of bytes. This method copies <paramref name="count" /> bytes
    from <paramref name="buffer" /> to the current stream.</param>
    /// <param name="offset">The zero-based byte offset in <paramref name="buffer" /> at which to
    begin copying bytes to the current stream.</param>
    /// <param name="count">The number of bytes to be written to the current stream.</param>
    public override void Write(byte[] buffer, int offset, int count)
    {
        string html = Encoding.Default.GetString(buffer);

        //remove whitespace
        html = RegexTags.Replace(html, "> <");
        html = RegexAll.Replace(html, " ");

        byte[] outdata = Encoding.Default.GetBytes(html);

        //write bytes to stream
        _filter.Write(outdata, 0, outdata.GetLength(0));
    }
}

#endregion Methods
}
}

```

در این کلاس فشرده سازی (gzip و deflate نیز اعمال شده است) در متد OnActionExecuting ابتدا در خط 24 بررسی می‌شود که آیا درخواست رسیده gzip را پشتیبانی می‌کند یا خیر. در صورت پشتیبانی خروجی صفحه را با استفاده از gzip یا deflate فشرده سازی می‌کند. تا اینجا کار ممکن است مورد نیاز ما نباشد. اصل کار ما (حذف کردن فضاهای خالی) در خط 42 اعمال شده است. در واقع برای حذف فضاهای خالی باید یک کلاس که از Stream ارث بری دارد تعریف شده و خروجی کلاس مورد نظر به فیلتر درخواست ما اعمال شود.

در کلاس WhitespaceFilter با تحریر متد Write الگوهای فضای خالی موجود در درخواست یافت شده و آنها را حذف می‌کنیم. در نهایت خروجی این کلاس که از نوع استریم است به ویژگی فیلتر صفحه اعمال می‌شود.

برای معرفی فیلتر تعریف شده می‌توان در فایل Global.asax در رویداد Application_Start به صورت زیر فیلتر مورد نظر را به فیلترهای MVC اعمال کرد.

```
GlobalFilters.Filters.Add(new CompressAttribute());
```

برای آشنایی بیشتر [فیلترها در ASP.NET MVC](#) را مطالعه نمایید.
پ.ن: جهت سهولت، در این کلاس ها، صفحات فشرده سازی و همزمان فضاهای خالی آنها حذف شده است.

نظرات خوانندگان

نویسنده: وحید نصیری
تاریخ: ۲۰:۵۹ ۱۳۹۲/۰۹/۲۲

با تشکر. من چندبار سعی کردم از روش حذف فواصل خالی استفاده کنم ولی هربار از خیرش گذشتم به این دلایل:

- در مرورگرهای قدیمی گاهی باعث کرش و بسته شدن آنی برنامه می‌شد.
- کدهای جاوا اسکریپت یا CSS اگر داخل صفحه قرار داشتند، مشکل پیدا می‌کردند.
- گاهی از همین فضاهای خالی برای اندکی ایجاد فاصله بین عناصر ممکن است استفاده شود. این‌ها با حذف فواصل خالی به هم می‌ریزند.
- بعضی مرورگرها علاقمند هستند که doctype ابتدای یک فایل HTML، حتما در یک سطر مجزا ذکر شود.
- زمانیکه شما codeایی در صفحه تعریف می‌کنید، برای پردازش صحیح تگ PRE توسط مرورگر، مهم است که سطر جدیدی وجود داشته باشد، یا فاصله بین عناصر حفظ شود. در غیراینصورت کد نمایش داده شده به هم می‌ریزد.
- الگوریتم‌های فشرده سازی اطلاعات مانند GZIP یا Deflate، حداقل کاری را که به خوبی انجام می‌دهند، فشرده سازی فواصل خالی است.

نویسنده: صابر فتح الهی
تاریخ: ۲۱:۵ ۱۳۹۲/۰۹/۲۲

بله کاملا حق با شماست و مشکل زمانی زیاد می‌شود که در صفحه کد sژ داشته باشیم و یکی از خطوط با استفاده از // کامنت کنیم.

با فشرده سازی دستورات بعدی کامنت خواهد شد و تا حدودی ممکن است صفحه از کار بیفتد.

که باید حتی الامکان از این نوع کامنت‌ها استفاده نشود.

در هر صورت از نظر شما متشکرم

نویسنده: میثم هوشمند
تاریخ: ۰:۳۱ ۱۳۹۲/۰۹/۲۳

با سلام
می‌شود این استثناها را در فشرده سازی لحاظ کرد؟

نویسنده: وحید نصیری
تاریخ: ۱:۱۲ ۱۳۹۲/۰۹/۲۳

البته فشرده سازی متفاوت است با حذف فواصل خالی بین تگ‌ها و سطرهای جدید. در حذف فواصل مثلا می‌شود تگ Pre را لحاظ نکرد:

```
var regex = new Regex(@"(?<=\s)\s+(?![^\<>]*</pre>)");
```

یکی از مسائل ریز و فنی در دنیای NET. استفاده یا عدم استفاده از NGEN است. در مقاله کوتاه زیر بهترین مکان های استفاده و عدم استفاده از آن را در چند بند خلاصه می کنم:

کجا از NGEN استفاده کنیم؟

برنامه هایی که مقدار زیادی کد مدیریت شده قبل از نمایش فرم برنامه دارند. مانند برنامه Blend که مقدار زیادی کد در شروع برنامه دارد. استفاده از ngen می تواند باعث افزایش کارایی و سرعت اجرای برنامه شود
فریم ورک ها، dll ها و کامپوننت های عمومی: کدهای تولید شده توسط JIT قابل اشتراک بین برنامه های مختلف نیستند ولی NGEN قابل اشتراک مابین برنامه های مختلف می باشد. بنابراین اگر کامپوننتی دارید که در بین برنامه های مختلف مشترک استفاده می شود، این کار می تواند سرعت شروع برنامه ها را بالا برده استفاده از منابع سیستم را کاهش دهد
برنامه هایی که در terminal serverها استفاده می شوند: توضیح فوق در مورد این برنامه ها نیز صادق است.

کجا از NGEN استفاده نکنیم؟

برنامه های کوچک: عملاً سرعت JIT آن قدر بالا است که NGEN کار را کندتر خواهد کرد!
برنامه های سروری که سرعت شروع آن مهم نیست: برنامه ها یا dll هایی که سرعت شروع آنها مهم نیست، اگر NGEN نشوند سرعت بیشتری برای شما به ارمغان خواهند داشت چون JIT در هنگام اجرا، کد را بهینه می کند ولی NGEN این کار را انجام نمی دهد.

چند نکته دیگر که باید در نظر داشته باشید این است که قرار نیست NGEN مثل یک جادوگر کد شما را جادو کند که سریع تر اجرا شود. تنها کد را از قبل به کد native مربوط به معماری cpu شما کامپایل خواهد کرد که شروع اجرای آن سریع تر شود. البته این جادوگر (: قربانی هم می خواهد. قربانی آن optimization های داخلی JIT است که در برنامه شما اعمال نخواهد شد. بنابراین در رابطه با استفاده از NGEN نهایت دقت را به خرج دهید.

نظرات خوانندگان

نویسنده: شهرز جعفری
تاریخ: ۱۶:۲۲ ۱۳۹۲/۱۱/۲۷

بد نیست به [این](#) مطلب (NGEN.EXE در دات نت) یک سری بزنید.

نویسنده: ناصر نیازی
تاریخ: ۱۹:۲۸ ۱۳۹۲/۱۱/۲۹

می خواستم بپرسم آیا می توان با این برنامه فایل اجرایی برنامه را Standalone کرد مثل فایل های دلفی ۷ که همه جا اجرا میشه ؟

نویسنده: م منفرد
تاریخ: ۰:۳۰ ۱۳۹۲/۱۱/۳۰

خیر. برای اجرای برنامه هنوز نیاز به فریمورک دات نت دارید.

مقدمه Profiler یک ابزار گرافیکی برای ردیابی و نظارت بر کارآئی SQL Server است. امکان ردیابی اطلاعاتی در خصوص رویدادهای مختلف و ثبت این داده‌ها در یک فایل (با پسوند trc) یا جدول برای تحلیل‌های آتی نیز وجود دارد. برای اجرای این ابزار مراحل زیر را انجام دهید:

Start > Programs> Microsoft SQL Server > Performance Tools> SQL Server Profiler

و یا در محیط Management Studio از منوی Tools گزینه SQL Server Profiler را انتخاب نمایید.

1- اصطلاحات

1-1- رویداد (Event): یک رویداد، کاری است که توسط موتور بانک اطلاعاتی (Database Engine) انجام می‌شود. برای مثال هر یک از موارد زیر یک رویداد هستند.

- متصل شدن کاربران (login connections) قطع شدن ارتباط یک login
- اجرای دستورات T-SQL، شروع و پایان اجرای یک رویه، شروع و پایان یک دستور در طول اجرای یک رویه، اجرای رویه‌های دور Remote Procedure Call
- باز شدن یک Cursor
- بررسی و کنترل مجوزهای امنیتی

1-2- کلاس رویداد (Event Class): برای بکارگیری رویدادها در Profiler، از یک Event Class استفاده می‌کنیم. یک Event Class رویدادی است که قابلیت ردیابی دارد. برای مثال بررسی ورود و اتصال کاربران با استفاده از کلاس Audit Login قابل پیاده سازی است. هر یک از موارد زیر یک Event Class هستند.

- SQL:BatchCompleted
- Audit Login
- Audit Logout
- Lock: Acquired
- Lock: Released

1-3- گروه رویداد (Event Category): یک گروه رویداد شامل رویدادهایی است که به صورت مفهومی دسته بندی شده اند. برای مثال، کلیه رویدادهای مربوط به قفل‌ها از جمله Lock: Acquired (بدست آوردن قفل) و Lock: Released (رها کردن قفل) در گروه رویداد Locks قرار دارند.

1-4- ستون داده ای (Data Column): یک ستون داده ای، خصوصیت و جزئیات یک رویداد را شامل می‌شود. برای مثال در یک Trace که رویدادهای Lock: Acquired را نظارت می‌کند، ستون Binary Data شامل شناسه (ID) یک صفحه و یا یک سطر قفل شده است و یا اینکه ستون Duration مدت زمان اجرای یک رویه را نمایش می‌دهد.

1-5- الگو (Template): یک الگو، مشخص کننده تنظیمات پیش گزیده برای یک Trace است، این تنظیمات شامل رویدادهایی است که نیاز دارید بر آنها نظارت داشته باشید. هنگامیکه یک Trace براساس یک الگو اجرا شود، رویدادهای مشخص شده، نظارت می‌شوند و نتیجه به صورت یک فایل یا جدول قابل مشاهده خواهد بود.

1-6 ردیاب (Trace): یک Trace داده‌ها را براساس رویدادهای انتخاب شده، جمع‌آوری می‌کند. امکان اجرای بلافاصله یک Trace برای جمع‌آوری اطلاعات با توجه به رویدادهای انتخاب شده و ذخیره کردن آن برای اجرای آتی وجود دارد.

1-7 فیلتر (Filter): هنگامی که یک Trace یا الگو ایجاد می‌شود، امکان تعریف شرایطی برای فیلتر کردن داده‌های جمع‌آوری شده نیز وجود دارد. این کار باعث کاهش حجم داده‌های گزارش شده می‌شود. برای مثال اطلاعات مربوط به یک کاربر خاص جمع‌آوری می‌شود و یا اینکه رشد یک بانک اطلاعاتی مشخص بررسی می‌شود.

2- انتخاب الگو (Profiler Trace Templates) از آنجائیکه اصولاً انتخاب Event‌های مناسب، کار سخت و تخصصی می‌باشد برای راحتی کار تعدادی Template‌های آماده وجود دارد، برای مثال TSQL_Duration تأکیدش روی مدت انجام کار است و یا SP_Counts در مواردی که بخواهیم رویه‌های ذخیره شده را بهینه کنیم استفاده می‌شود در جدول زیر به شرح هر یک پرداخته شده است:

الگو	هدف
Blank	ایجاد یک Trace کلی
SP_Counts	ثبت اجرای هر رویه ذخیره شده برای تشخیص اینکه هر رویه چند بار اجرا شده است
Standard	ثبت آمارهای کارائی برای هر رویه ذخیره شده و Query‌های عادی SQL که اجرا می‌شوند و عملیات ورود و خروج هر Login (پیش فرض)
TSQL	ثبت یک لیست از همه رویه‌های ذخیره شده و Query‌های عادی SQL که اجرا می‌شوند ولی آمارهای کارائی را شامل نمی‌شود
TSQL_Duration	ثبت مدت زمان اجرای هر رویه ذخیره شده و هر Query عادی SQL
TSQL_Grouped	ثبت تمام login‌ها و logout‌ها در طول اجرای رویه‌های ذخیره شده و هر Query عادی SQL، شامل اطلاعاتی برای شناسائی برنامه و کاربری که درخواست را اجرا می‌کند
TSQL_Locks	ثبت اطلاعات انسداد (blocking) و بن بست (deadlock) از قبیل blocked processes, deadlock chains, deadlock graphs, این الگو همچنین درخواست‌های تمام رویه‌های ذخیره شده و تمامی دستورات هر رویه و هر Query عادی SQL را دریافت می‌کند
TSQL_Replay	ثبت اجرای رویه‌های ذخیره شده و Query‌های SQL در یک SQL Instance و مهیا کردن امکان اجرای دوباره عملیات در سیستمی دیگر
TSQL_SPs	ثبت کارائی برای Query‌های SQL، رویه‌های ذخیره شده و تمامی دستورات درون یک رویه ذخیره شده و نیز عملیات ورود و خروج هر Login
Tuning	ثبت اطلاعات کارائی برای Query‌های عادی SQL و رویه‌های ذخیره شده و یا تمامی دستورات درون یک رویه ذخیره شده

3- انتخاب رویداد (SQL Trace Event Groups) رویدادها در 21 گروه رویداد دسته‌بندی می‌شوند که در جدول زیر لیست شده‌اند:

هدف	گروه رویداد
13 رویداد برای واسطه سرویس (Service Broker)	Broker
1 رویداد برای بارگذاری اسمبلی‌های CLR (Common Language Runtime)	CLR
7 رویداد برای ایجاد، دستیابی و در اختیار گرفتن Cursor	Cursors
6 رویداد برای رشد/کاهش (grow/shrink) فایل های Data/Log همچنین تغییرات حالت انعکاس (Mirroring)	Database
2 رویداد برای آگاه کردن وضعیت نابسامان درون یک SQL Instance	Deprecation
16 رویداد برای خطاها، هشدارها و پیغام‌های اطلاعاتی که ثبت شده است	Errors and Warnings
3 رویداد برای پیگیری یک شاخص متنی کامل	Full Text
9 رویداد برای بدست آوردن، رها کردن قفل و بن بست (Deadlock)	Locks
5 رویداد برای درخواست‌های توزیع شده و RPC (اجرای رویه‌های دور)	OleDb
3 رویداد برای وقتی که یک شی ایجاد، تغییر یا حذف می‌شود	Objects
14 رویداد برای ثبت نقشه درخواست‌ها (Query Plan) برای استفاده نقشه راهنما (Plan Guide) به منظور بهینه سازی کارائی درخواست‌ها، همچنین این گروه رویداد در خواست‌های متنی کامل (full text) را ثبت می‌کند	Performance
10 رویداد برای ایجاد Online Index	Progress Report
4 رویداد برای سرویس اطلاع رسان (Notification Service)	Query Notifications
2 رویداد برای وقتی که یک جدول یا شاخص، پوشش می‌شود	Scans
44 رویداد برای وقتی که مجوزی استفاده شود، جابجائی هویتی رخ دهد، تنظیمات امنیتی اشیائی تغییر کند، یک SQL Instance شروع و متوقف شود و یک Database جایگزین شود یا از آن پشتیبان گرفته شود	Security Audit
3 رویداد برای Mount Tape، تغییر کردن حافظه سرور و بستن یک فایل Trace	Server
3 رویداد برای وقتی که Connection‌ها موجود هستند و یک Trace فعال می‌شود، همچنین یک Trigger و یک تابع دسته بندی (classification functions) مربوط به مدیریت منابع (resource governor) رخ دهد	Sessions
12 رویداد برای اجرای یک رویه ذخیره شده و دستورات درون آن ، کامپایل مجدد و استفاده از حافظه نهانی (Cache)	Stored Procedures
13 رویداد برای شروع، ذخیره ، تأیید و لغو یک تراکنش	Transactions
9 رویداد برای اجرای Queryهای SQL و جستجوهای XQUERY (در داده‌های XML)	TSQL

گروه رویداد	هدف
User Configurable	10 رویداد که شما می‌توانید پیکربندی کنید

به طور معمول بیشتر از گروه رویدادهای Locks, Performance, Security Audit, Stored Procedures و TSQL استفاده می‌شود.

4- انتخاب ستون‌های داده ای (Data Columns) اگرچه می‌توان همه‌ی 64 ستون داده ای ممکن را برای ردیابی انتخاب کرد ولیکن داده‌های Trace شما زمانی مفید خواهند بود که اطلاعات ضروری را ثبت کرده باشید. برای مثال شماره ترتیب تراکنش‌ها را، برای یک رویداد RPC:Completed می‌توانید برگردانید، اما همه رویه‌های ذخیره شده مقادیر را تغییر نمی‌دهند بنابراین شماره ترتیب تراکنش‌ها فضای بیهوده ای را مصرف می‌کند. بعلاوه همه ستون‌های داده ای برای تمامی رویدادهای Trace معتبر نیستند. برای مثال CPU, Read, Write, Duration برای رویدادهای RPC:Starting و SQL:BatchStarting معتبر نیستند. SessionLoginName, مشخص می‌کنند چه کسی و از چه منشاء دستور را اجرا کرده است. ستون SessionLoginName معمولاً نام Login ای که از آن برای متصل شدن به SQL Instance استفاده شده است را نشان می‌دهد. در حالیکه ستون LoginName نام کاربری را که دستور را اجرا می‌کند نشان می‌دهد (EXECUTE AS). ستون ApplicationName خالی است مگر اینکه در ConnectionString برنامه کاربردی این خصوصیت (Property) مقداردهی شده باشد. ستون StartTime و EndTime زمان سرحدی برای هر رویداد را ثبت می‌کند این ستون‌ها بویژه در هنگامی که به عملیات Correlate نیاز دارید مفید هستند.

5- بررسی چند سناریو نمونه

• **یافتن درخواست هائی (Queries) که بدترین کارایی را دارا هستند.** برای ردیابی درخواست‌های ناکار، از رویداد RPC:Completed از دسته Stored Procedure و رویداد SQL:BatchCompleted از دسته TSQL استفاده می‌شود.

• **نظارت بر کارایی رویه ها** برای ردیابی کارائی رویه ها، از رویدادهای SP:Starting, SP:Completed, SP:StmtCompleted و SP:StmtStarting از کلاس Stored Procedure و رویدادهای SQL:BatchStarting, SQL:BatchCompleted از کلاس TSQL استفاده می‌شود.

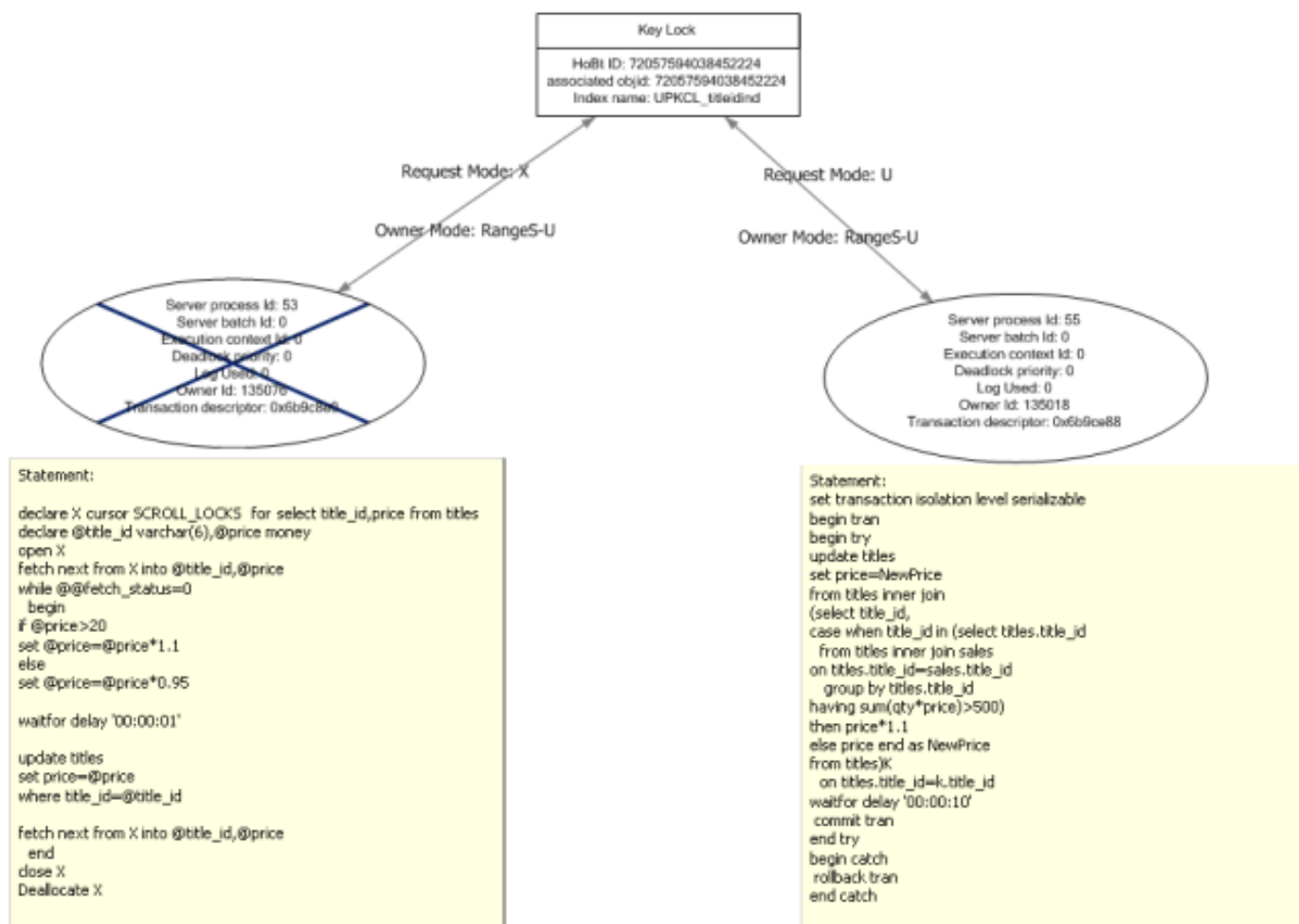
• **نظارت بر اجرای دستورات T-SQL توسط هر کاربر** برای ردیابی دستوراتی که توسط یک کاربر خاص اجرا می‌شود، نیاز به ایجاد یک Trace برای نظارت بر رویدادهای کلاس‌های Sessions, ExistingConnection و TSQL داریم همچنین لازم است نام کاربر در قسمت فیلتر و با استفاده از DBUserName مشخص شود.

• **اجرا دوباره ردیاب (Trace Replay)** این الگو معمولاً برای debugging استفاده می‌شود برای این منظور از الگوی Replay استفاده می‌شود. در ضمن امکان اجرای دوباره عملیات در سیستمی دیگر با استفاده از این الگو مهیا می‌شود.

• **ابزار Tuning Advisor (راهنمای تنظیم کارائی)** این ابزاری برای تحلیل کارائی یک یا چند بانک اطلاعاتی و تاثیر عملکرد آنها بر بار کاری (Workload) سرویس دهنده است. یک بار کاری مجموعه ای از دستورات T-SQL است که روی بانک اطلاعاتی اجرا می‌شود. بعد از تحلیل تاثیر بارکاری بر بانک اطلاعاتی، Tuning Advisor توصیه هائی برای اضافه کردن، حذف و یا تغییر طراحی فیزیکی ساختار بانک اطلاعاتی ارائه می‌دهد این تغییرات ساختاری شامل پیشنهاد برای تغییر ساختاری موارد Clustered Indexes, Nonclustered Indexes, Indexed View و Partitioning است. برای ایجاد بارکاری می‌توان از یک ردیاب تهیه شده در SQL Profiler استفاده کرد برای این منظور از الگوی Tuning استفاده می‌شود و یا رویدادهای RPC:Completed, SQL:BatchCompleted و SP:StmtCompleted را ردیابی نمائید.

• **ترکیب ابزارهای نظارتی (Correlating Performance and Monitoring Data)** یک Trace برای ثبت اطلاعاتی که در یک SQL Instance رخ می‌دهد، استفاده می‌شود. System Monitor برای ثبت شمارنده‌های کارائی (performance counters) استفاده می‌شود و همچنین از منابع سخت افزاری و اجزای دیگر که روی سرور اجرا می‌شوند، تصاویری فراهم می‌کند. توجه شود که در مورد Correlating یک فایل ردیاب (trace file) و یک Counter Log (ابزار Performance)، ستون داده ای StartTime و EndTime باید انتخاب شود، برای این کار از منوی File گزینه Import Performance Data انتخاب می‌شود.

- **جستجوی علت رخ دادن یک بن بست** برای ردیابی علت رخ دادن یک بن بست، از رویدادهای RPC:Starting، SQLBatchStarting از دسته Locks استفاده می‌شود. (در صورتی که نیاز به یک ارائه گرافیکی دارید از Deadlock graph استفاده نمائید، خروجی مطابق تصویر زیر می‌شود).



- 5-1- ایجاد یک Profiler Trace 1** را اجرا کنید از منوی File گزینه New Trace را انتخاب کنید و به SQL Instance مورد نظرتان متصل شوید.
- 2-** مطابق تصویر زیر برای Trace یک نام و الگو و تنظیمات ذخیره سازی فایل را مشخص کنید.

Trace Properties

General | **Events Selection**

Trace name: performance baseline

Trace provider name: 1936-603102405

Trace provider type: Microsoft SQL Server 2005 version: 9.0.4035

Use the template: Blank

☒ Save to file: C:\performance baseline.trc

Set maximum file size (MB): 5

☒ Enable file rollover

☐ Server processes trace data

☐ Save to table:

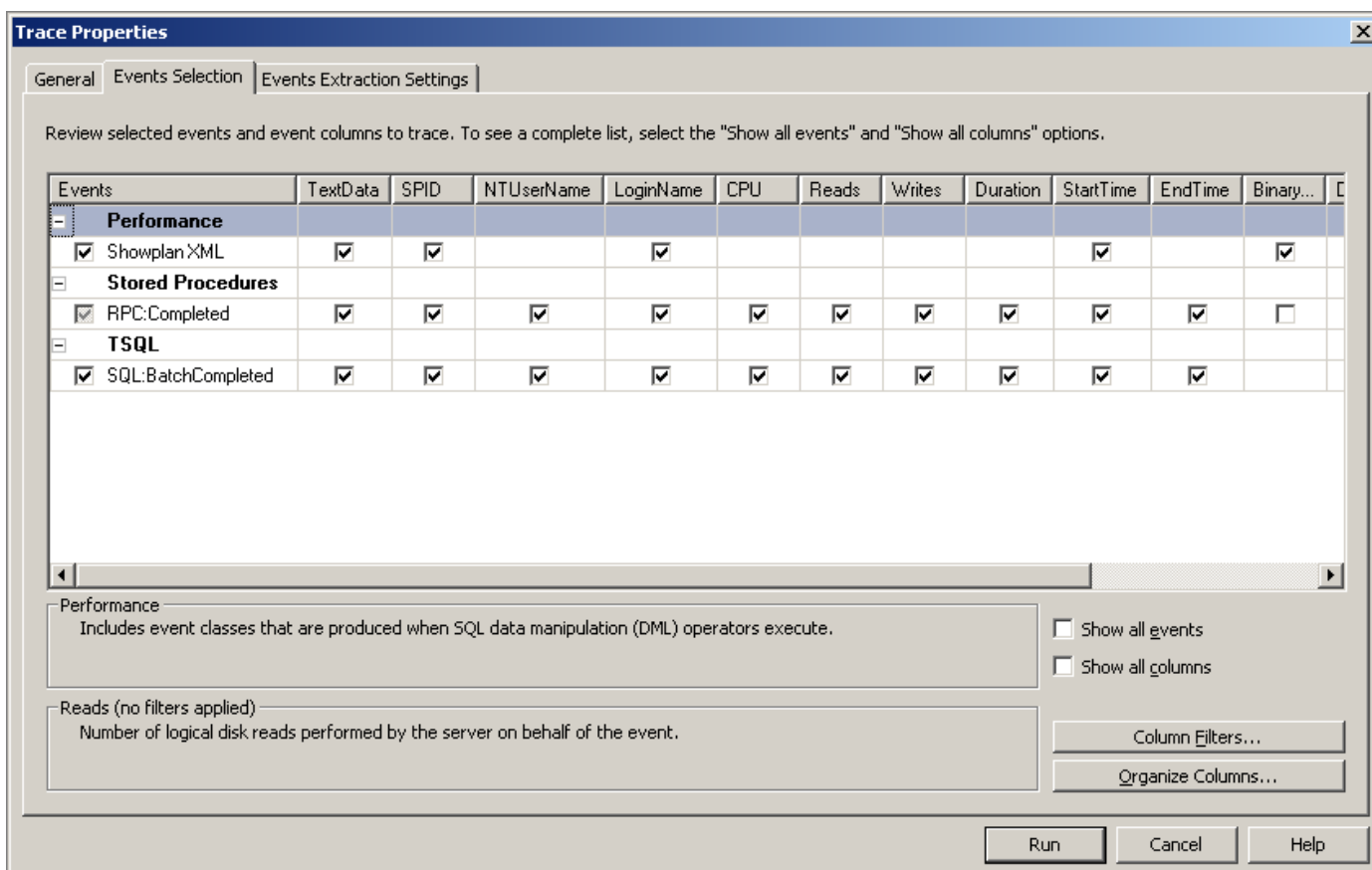
☐ Set maximum rows (in thousands): 1

☐ Enable trace stop time: 11/ 6/2010 1:20:21 PM

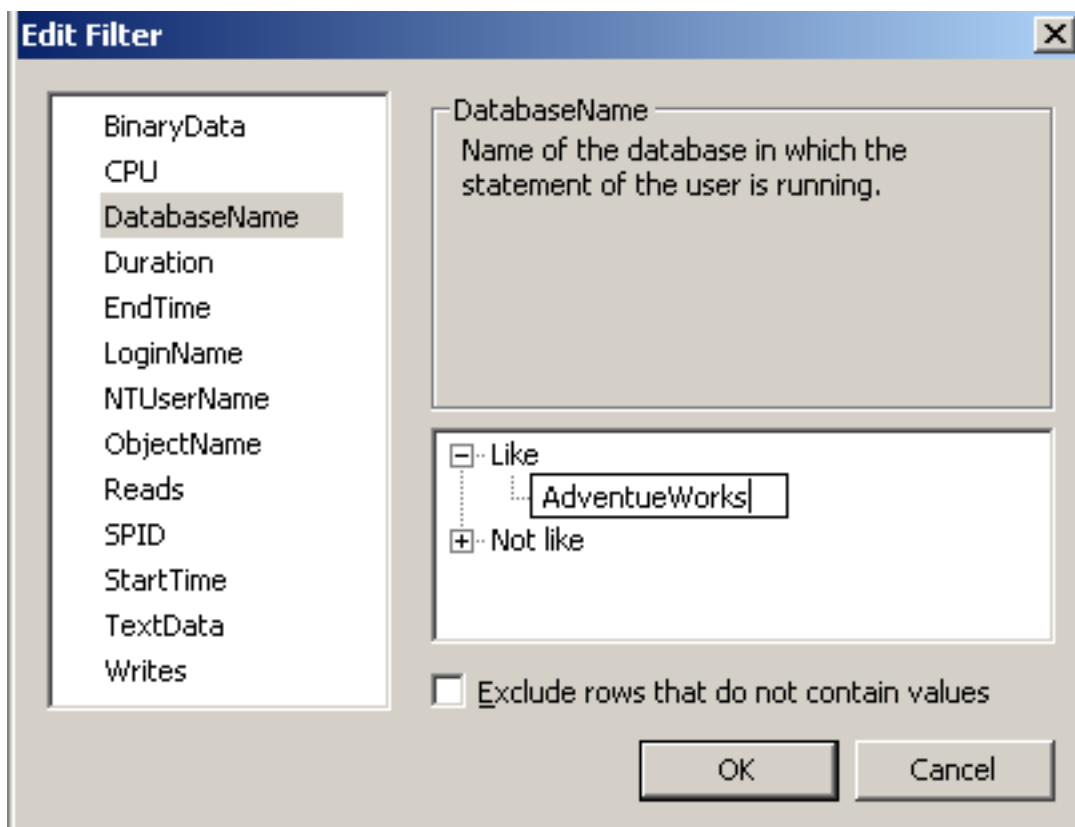
Run Cancel Help

3- بر روی قسمت Events Selection کلیک نمائید.

4- مطابق تصویر زیر رویدادها و کلاس رویدادها را انتخاب کنید، ستون‌های TextData, NTUserName, LoginName, CPU, Reads, Writes, Duration, SPID, StartTime, EndTime, BinaryData, DataBaseName, ServerName و ObjectName را انتخاب کنید.



5- روی Column Filters کلیک کنید و مطابق تصویر زیر برای DatabaseName فیلتری تنظیم کنید.



6- روی Run کلیک کنید. تعدادی Query و رویه ذخیره شده مرتبط با پایگاه داده AdventureWorks اجرا کنید .

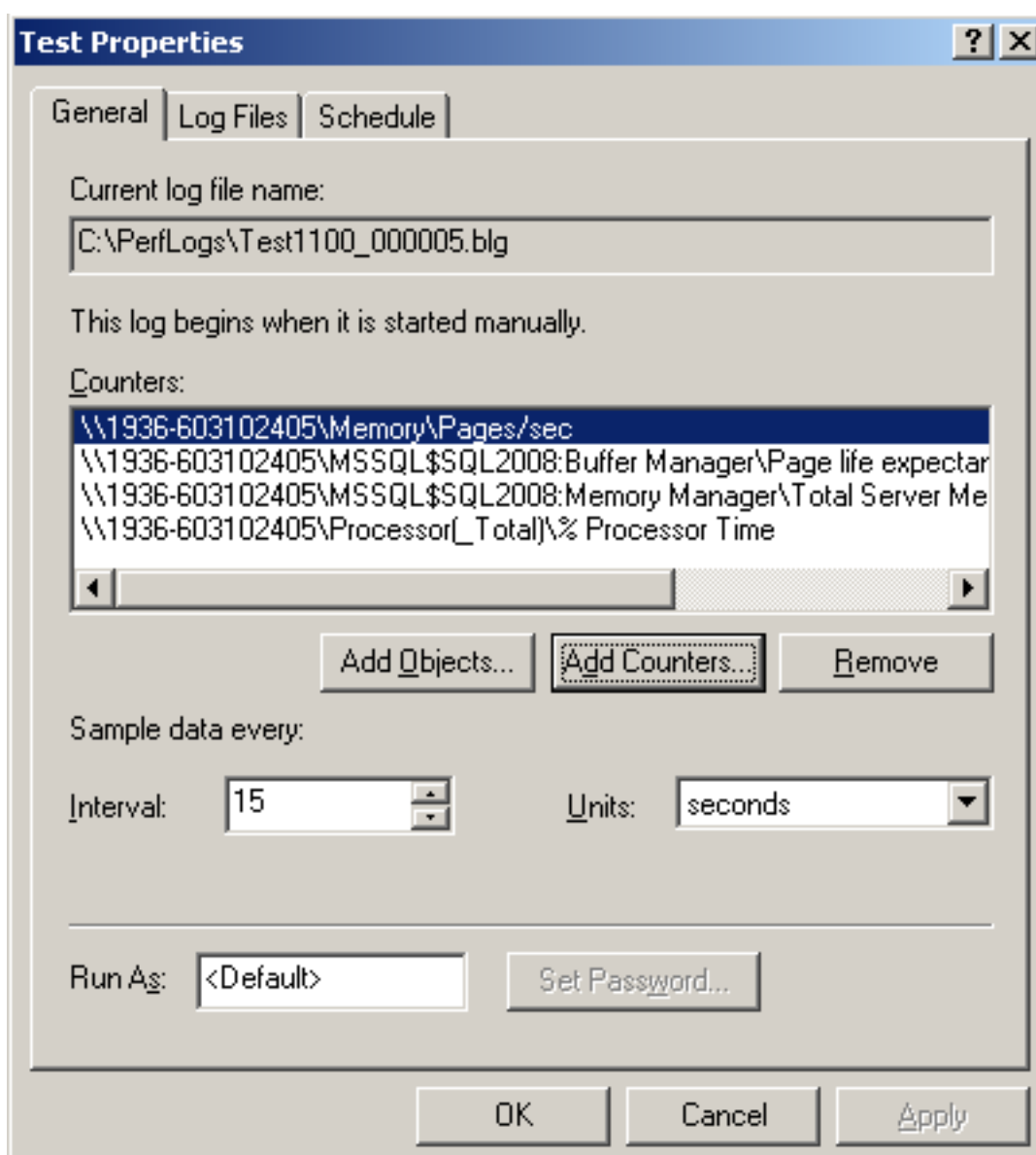
5-2- ایجاد یک Counter Log برای ایجاد یک Counter Log مراحل زیر را انجام دهید:

1- ابزار Performance را اجرا کنید (برای این کار عبارت PerfMon را در قسمت Run بنویسید).

2- در قسمت Counter Logs یک log ایجاد کنید.

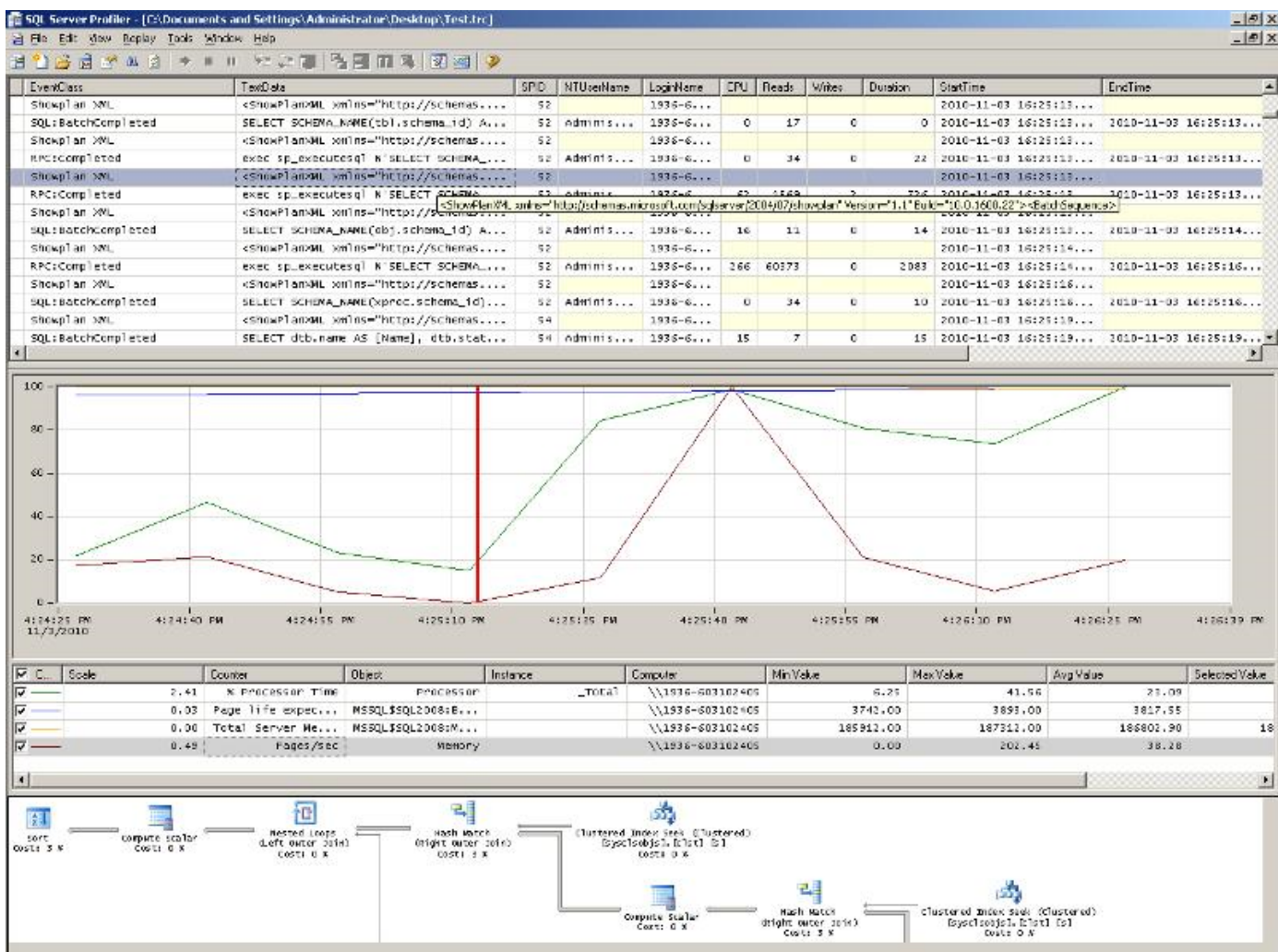
3- روی Add Counters کلیک کرده و مطابق تصویر موارد زیر را انتخاب کنید.

Performance Object	Select counters from list
Network Interface	Output Queue Length
Processor	% Processor Time
System	Processor Queue Length
SQLServer	Buffer Manager:Page life expectancy



4- روی Ok کلیک کنید تا Counter Log ذخیره شود سپس روی آن راست کلیک کرده و آنرا Start کنید.

5-3- ترکیب ابزارهای نظارتی (Profiler) 1- (Correlating SQL Trace and System Monitor Data) را اجرا کنید از منوی File گزینه Open و سپس Trace File را انتخاب کنید فایل trc را که در گام اول ایجاد کردید، باز نمایید.
2- از منوی File گزینه Import Performance Data را انتخاب کنید و فایل counter log را که در مرحله قبل ایجاد کردید انتخاب کنید.



نکته: اطلاعات فایل trc را می‌توان درون یک جدول وارد کرد، بدین ترتیب می‌توان آنالیز بیشتری داشت به عنوان مثال دستورات زیر این عمل را انجام می‌دهند.

```
SELECT * INTO dbo.BaselineTrace
FROM fn_trace_gettable(' c:\performance baseline.trc ', default);
```

با اجرای دستور زیر جدولی با نام BaselineTrace ایجاد و محتویات Trace مان (performance baseline.trc) در آن درج می‌گردد.

به صورت پیش فرض SQL Server از روش write-ahead log - WAL استفاده می‌کند. به این معنا که کلیه تغییرات، پیش از commit نهایی باید در لاگ فایل آن نوشته شوند. این مساله با تعداد بالای تراکنش‌ها تا حدودی بر روی سرعت سیستم می‌تواند تاثیرگذار باشد. برای بهبود این وضعیت، در SQL Server 2014 قابلیت به نام `delayed_durability` اضافه شده‌است که با فعال سازی آن، کلیه اعمال مرتبط با لاگ‌های تراکنش‌ها به صورت غیرهمزمان انجام می‌شوند. به این ترتیب تراکنش‌ها زودتر از معمول به پایان خواهد رسید؛ با این فرض که نوشته شدن تغییرات در لاگ فایل‌ها، در آینده‌ای محتمل انجام خواهند شد. این مساله به معنای فدا کردن D در ACID است ([Atomicity, Consistency, Isolation, Durability](#)). البته باید دقت داشت که رسیدن به ACID کامل هزینه‌بر است و شاید خیلی از اوقات تمام اجزای آن نیازی نباشند یا حتی بتوان با اندکی تخفیف آن‌ها را اعمال کرد؛ مانند D به تاخیر افتاده.

برای اینکار SQL Server از یک بافر 60 کیلوبایتی برای ذخیره سازی اطلاعات لاگ‌هایی که قرار است به صورت غیرهمزمان با تراکنش‌ها نوشته شوند، استفاده می‌کند. هر زمان که این 60KB پر شد، آن‌را flush کرده و ثبت خواهد نمود. به این ترتیب به دو مزیت خواهیم رسید:

- پردازش تراکنش‌ها بدون منتظر شدن جهت commit نهایی در دیسک سخت ادامه خواهند یافت. صبر کمتر به معنای امکان پردازش تراکنش‌های بیشتری در یک سیستم پر ترافیک است.
- با توجه به بافری که از آن صحبت شد، اینبار اعمال Write به صورت یک سری batch اعمال می‌شوند که کارایی و سرعت بیشتری نسبت به حالت تکی دارند.

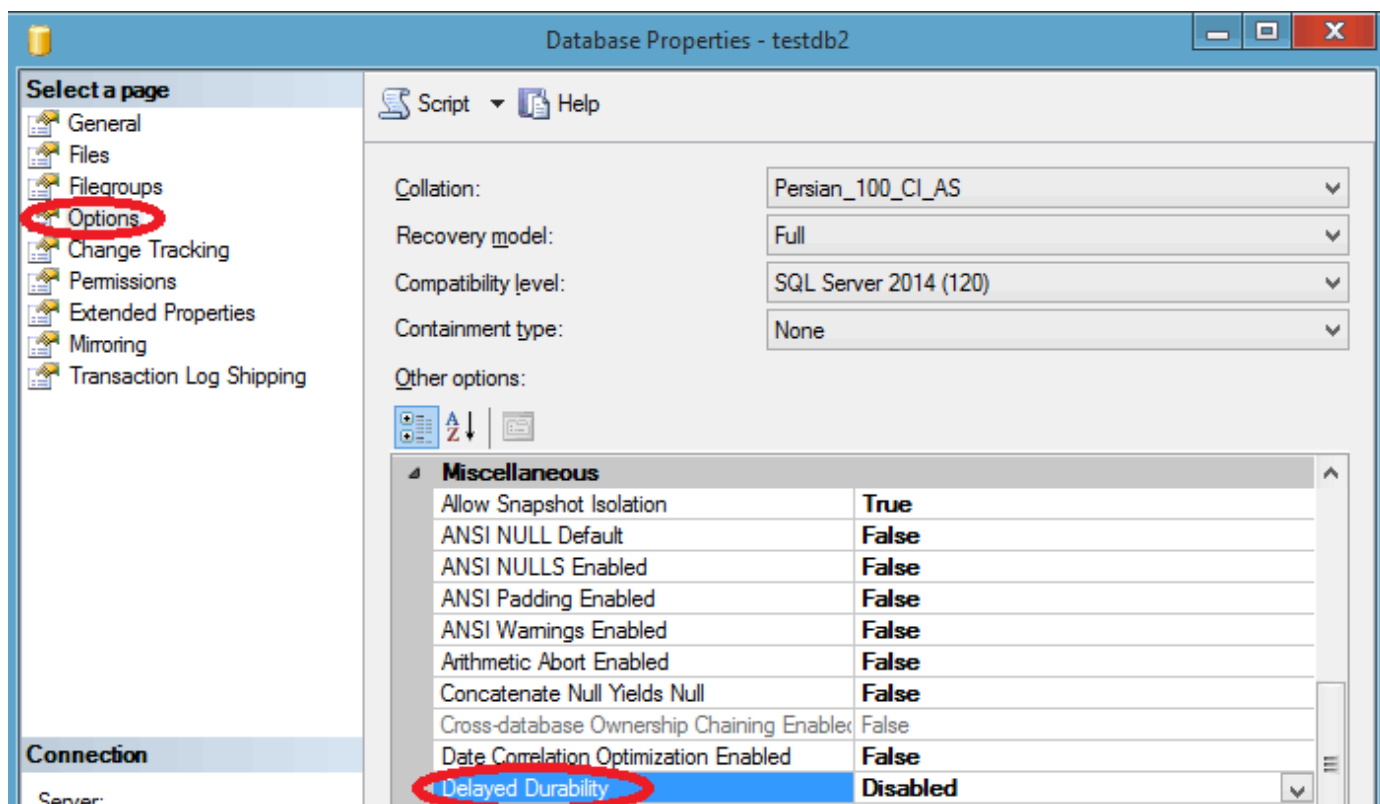
اندکی تاریخچه

ایده یک چنین عملی 28 سال قبل توسط [Hal Berenson](#) ارائه شده‌است! او را که آن‌را در سال 2006 تحت عنوان Asynchronous Commit پیاده سازی کرد و مایکروسافت در سال 2014 آن‌را ارائه داده‌است.

فعال سازی ماندگاری غیرهمزمان در SQL Server

فعال سازی این قابلیت در سطح بانک اطلاعاتی، در سطح یک تراکنش مشخص و یا در سطح رویه‌های ذخیره شده کامپایل شده مخصوص OLTP درون حافظه‌ای، میسر است. برای فعال سازی ماندگاری با تاخیر در سطح یک دیتابیس، خواهیم داشت:

```
ALTER DATABASE dbname SET DELAYED_DURABILITY = DISABLED | ALLOWED | FORCED;
```



در اینجا اگر ALLOWED را انتخاب کنید، به این معنا است که لاگ کلیه تراکنش‌های مرتبط با این بانک اطلاعاتی به صورت غیرهمزمان نوشته می‌شوند. حالت FORCED نیز دقیقاً به همین معنا است با این تفاوت که اگر حالت ALLOWED انتخاب شود، تراکنش‌های ماندگار (آن‌هایی که به صورت دستی DELAYED_DURABILITY را غیرفعال کرده‌اند)، سبب flush کلیه تراکنش‌هایی با ماندگاری به تاخیر افتاده خواهند شد و سپس اجرا می‌شوند. در حالت Forced تنظیم دسترسی DELAYED_DURABILITY = OFF در سطح تراکنش‌ها تأثیری نخواهد داشت؛ اما در حالت ALLOWED این مساله به صورت دستی در سطح یک تراکنش قابل لغو است. البته باید توجه داشت، صرفنظر از این تنظیمات، یک سری از تراکنش‌ها همیشه ماندگار هستند و بدون تاخیر؛ مانند تراکنش‌های سیستمی، تراکنش‌های بین دو یا چند بانک اطلاعاتی و کلیه تراکنش‌هایی که با Change Data Capture، FileTable، Change و Tracking سر و کار دارند.

در سطح تراکنش‌های می‌توان نوشت:

```
COMMIT TRANSACTION WITH (DELAYED_DURABILITY = ON);
```

و یا در رویه‌های ذخیره شده کامپایل شده مخصوص OLTP درون حافظه‌ای خواهیم داشت:

```
BEGIN ATOMIC WITH (DELAYED_DURABILITY = ON, ...)
```

سؤال: آیا فعال سازی DELAYED_DURABILITY بر روی مباحث locking و isolation levels تأثیر دارند؟

پاسخ: خیر. کلیه تنظیمات قفل گذاری‌ها همانند قبل و بر اساس isolation levels تعیین شده، رخ خواهند داد. تنها تفاوت در اینجا است که با فعال سازی DELAYED_DURABILITY، کار commit بدون صبر کردن برای پایان نوشته شدن اطلاعات در لاگ سیستم صورت می‌گیرد. به این ترتیب قفل‌های انجام شده زودتر آزاد خواهند شد.

سؤال: میزان از دست دادن اطلاعات احتمالی در این روش چقدر است؟

در صورتیکه سرور کرش کند یا ری‌استارت شود، حداکثر به اندازه‌ی 60KB اطلاعات را از دست خواهید داد (اندازه‌ی بافری که برای اینکار در نظر گرفته شده است). البته عنوان شده است که اگر ری‌استارت یا خاموشی سرور، از پیش تعیین شده باشد، ابتدا

کلیه لاگ‌های flush نشده، ذخیره شده و سپس ادامه‌ی کار صورت خواهد گرفت؛ ولی زیاد به آن اطمینان نکنید. اما همواره با فراخوانی sys.sp_flush_log، می‌توان به صورت دستی بافر لاگ‌های سیستم را flush کرد.

یک آزمایش

در ادامه قصد داریم یک جدول جدید را در بانک اطلاعاتی آزمایشی testdb2 ایجاد کنیم. سپس یکبار تنظیم DELAYED_DURABILITY = FORCED را انجام داده و 10 هزار رکورد را ثبت می‌کنیم و بار دیگر DELAYED_DURABILITY = DISABLED را تنظیم کرده و همین عملیات را تکرار خواهیم کرد:

```
CREATE TABLE tblData(
    ID INT IDENTITY(1, 1),
    Data1 VARCHAR(50),
    Data2 INT
);
CREATE CLUSTERED INDEX PK_tblData ON tblData(ID);
CREATE NONCLUSTERED INDEX IX_tblData_Data2 ON tblData(Data2);

-----

alter database testdb2 SET DELAYED_DURABILITY = FORCED;

-----

SET NOCOUNT ON
Print 'DELAYED_DURABILITY = FORCED'
DECLARE @counter AS INT = 0
DECLARE @start datetime = getdate()
WHILE (@counter < 10000)
BEGIN
    INSERT INTO tblData (Data1, Data2) VALUES('My Data', @counter)
    SET @counter += 1
END
Print DATEDIFF(ms,@start,getdate());
GO

-----

alter database testdb2 SET DELAYED_DURABILITY = DISABLED;
truncate table tblData;

-----

SET NOCOUNT ON
Print 'DELAYED_DURABILITY = DISABLED'
DECLARE @counter AS INT = 0
DECLARE @start datetime = getdate()
WHILE (@counter < 10000)
BEGIN
    INSERT INTO tblData (Data1, Data2) VALUES('My Data', @counter)
    SET @counter += 1
END
Print DATEDIFF(ms,@start,getdate());
GO

-----
```

با این خروجی:

```
DELAYED_DURABILITY = FORCED
666
DELAYED_DURABILITY = DISABLED
2883
```

در این آزمایش، سرعت insertها در حالت DELAYED_DURABILITY = FORCED حدود 4 برابر است نسبت به حالت معمولی.

برای مطالعه بیشتر

[Control Transaction Durability](#)

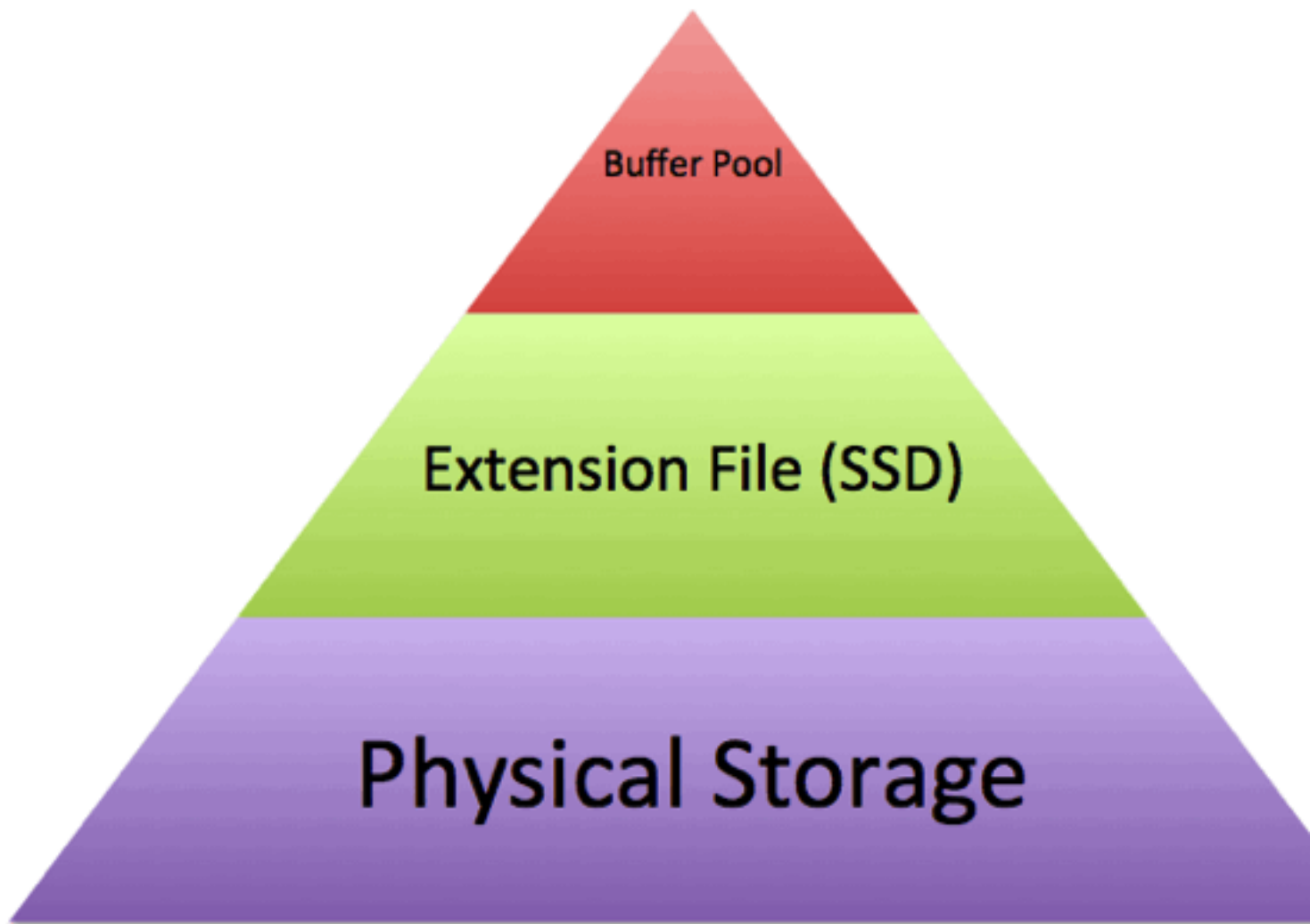
[SQL Server 2014 Delayed Durability/Lazy Commit](#)

[Delayed Durability in SQL Server 2014 - Part 1](#)

[Is In-Memory OLTP Always a silver bullet for achieving better transactional speed](#)

[Delayed Durability in SQL Server 2014](#)

Buffer Pool یکی از مصرف کنندگان اصلی حافظه در SQL Server است. برای مثال زمانیکه اطلاعاتی را از بانک اطلاعاتی دریافت می‌کنید، این داده‌ها در Buffer Pool کش می‌شوند. همچنین SQL Server اطلاعات کلیه Execution Plans را نیز در Plan Cache که جزئی از Buffer Pool است، برای استفاده‌ی مجدد نگهداری می‌کند. هر چقدر حافظه‌ی فیزیکی سرور شما بیشتر باشد، مقدار Buffer Pool نیز به همین میزان افزایش خواهد یافت که البته حداکثر آن‌را می‌توان در تنظیمات حافظه‌ی سرور محدود کرد (Max Server Memory setting).
در دنیای واقعی میزان حافظه‌ی فیزیکی سرورها محدود است. در SQL Server 2014 راه حلی برای این مشکل تحت عنوان Buffer Pool Extensions ارائه شده‌است که محل قرارگیری آن‌را در تصویر ذیل مشاهده می‌کنید:



Buffer Pool Extensions از یک فایل ساده که به آن Extension File نیز گفته می‌شود، تشکیل شده‌است و امکان ذخیره سازی آن بر روی هاردهای سریعی مانند SSD Drive میسر است. این فایل، ساختاری را همانند page file، در سیستم عامل ویندوز دارد. در این حالت بجای اضافه کردن RAM بیشتر به سرور، یک Extension File را می‌توان بکار گرفت. هر زمان که Buffer Pool اصلی تحت فشار قرار گیرد (به میزان حافظه‌ای بیش از حافظه‌ی فیزیکی سرور نیاز باشد)، از این افزونه‌ی فایلی استفاده خواهد

شد.

اطلاعات جزئیات Buffer Pool را توسط کوئری ذیل می‌توان بدست آورد:

```
Select * from sys.dm_os_buffer_descriptors
```

نحوه‌ی فعال سازی و تنظیم Buffer Pool Extensions

قبل از هر کاری بهتر است وضعیت افزونه‌ی Buffer pool را بررسی کرد:

```
select * from sys.dm_os_buffer_pool_extension_configuration
```

SQLQuery1.sql - (I...VahidPC\Vahid (60))* X

```
select * from sys.dm_os_buffer_pool_extension_configuration
```

100 %

Results Messages

	path	file_id	state	state_description	current_size_in_kb
1	NULL	-1	0	BUFFER POOL EXTENSION DISABLED	NULL

همانطور که ملاحظه می‌کنید، در حالت پیش فرض غیرفعال است. سپس یک فایل یک گیگابایتی را به عنوان افزونه‌ی Buffer pool ایجاد می‌کنیم.

```
ALTER SERVER CONFIGURATION
SET BUFFER POOL EXTENSION ON
(FILENAME = 'd:\BufferPoolExt.BPE', SIZE = 1GB);
```

توصیه شده‌است که این فایل را در یک درایور پر سرعت SSD قرار دهید؛ ولی محدودیتی از لحاظ محل قرارگیری ندارد (هر چند [به نظر فقط](#) در حالیکه از SSD Drive استفاده شود واقعا کار می‌کند). اینبار اگر کوئری اول را اجرا کنیم، چنین خروجی قابل مشاهده است:

SQLffQuery1.sql - ...VahidPC\Vahid (60))* X

```
select * from sys.dm_os_buffer_pool_extension_configuration
```

100 %

Results Messages

	path	file_id	state	state_description	current_size_in_kb
1	d:\BufferPoolExt.BPE	0	5	BUFFER POOL EXTENSION CLEAN PAGE CACHING ENABLED	1048576

این فایل به صورت خودکار در حین ری استارت یا خاموش شدن سرور، حذف شده و با راه اندازی مجدد آن، باز تولید خواهد شد.

تغییر اندازه‌ی افزونه‌ی Buffer pool

اگر سعی کنیم، یک گیگابایت را مثلا به 10 گیگابایت افزایش دهیم:

```
ALTER SERVER CONFIGURATION  
SET BUFFER POOL EXTENSION ON  
(FILENAME = 'd:\BufferPoolExt.BPE', SIZE = 10GB);
```

با خطای ذیل مواجه خواهیم شد:

```
Could not change the value of the 'BPoolExtensionPath' property
```

برای رفع این مشکل، ابتدا باید افزونه‌ی Buffer pool را غیرفعال کرد:

```
ALTER SERVER CONFIGURATION  
SET BUFFER POOL EXTENSION OFF
```

سپس می‌توان مجدداً اندازه و یا مسیر دیگری را مشخص کرد. بهتر است اندازه‌ی این فایل را حدود 16 برابر حداکثر میزان حافظه‌ی سرور (Max Server Memory) تعیین کنید. همچنین توصیه شده‌است که پس از غیرفعال کردن این افزونه، بهتر است یکبار instance جاری را ری استارت کنید.

چه زمانی بهتر است از افزونه‌ی Buffer pool استفاده شود؟

در محیط‌های read-heavy OLTP، استفاده از یک چنین افزونه‌ای می‌تواند میزان کارایی و پاسخگویی سیستم را به شدت افزایش دهد (تا [50 درصد](#)).

سؤال: آیا غیرفعال کردن افزونه‌ی Buffer pool سبب از دست رفتن اطلاعات می‌شود؟

خیر. BPE، تنها clean pages را در خود ذخیره می‌کند؛ یعنی تنها اطلاعاتی که Commit شده‌اند در آن حضور خواهند داشت و در این حالت حذف آن یا ری استارت کردن سرور، سبب از دست رفتن اطلاعات نخواهند شد.

برای مطالعه بیشتر

[Buffer Pool Extension](#)

[SQL Server 2014 Buffer Pool Extensions](#)

[Do you require a SSD to use the Buffer Pool Extension feature in SQL Server 2014](#)

[Buffer Pool Extensions in SQL Server 2014](#)

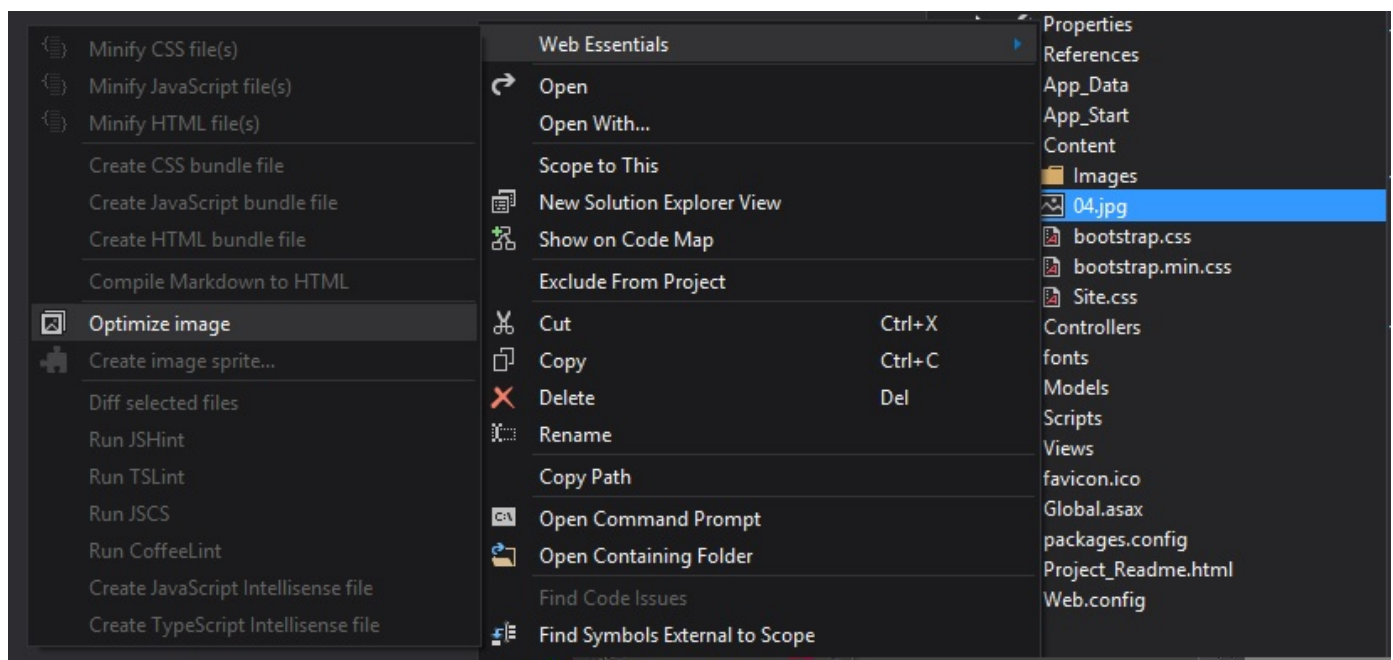
[SQL Server 2014 – Buffer Pool Extension](#)

در این مطلب نکات کار با تصاویر را توسط افزونه‌ی Web Essentials بررسی می‌کنیم. این افزونه قابلیت‌های زیر را در کار با تصاویر در اختیار شما قرار می‌دهد: **بهینه‌سازی تصاویر**

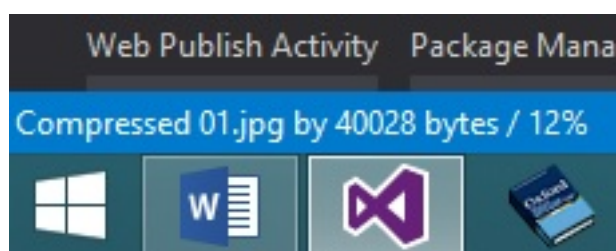
یکی از موارد مهمی که باید مورد توجه قرار بگیرد، استفاده از تصاویر کم حجم در وب‌سایت می‌باشد. روش‌های مختلفی جهت بهینه‌سازی تصاویر مورد استفاده در سایت وجود دارند، به طور مثال جهت بهینه‌سازی تصاویر PNG می‌توانید از ابزار PNGGauntlet استفاده کنید. همچنین [اینجا](#) نیز یک ابزار آنلاین موجود می‌باشد. افزونه‌ی Web Essentials این قابلیت را به آسانی در اختیار شما قرار می‌دهد؛ اینکار را می‌توانید توسط این افزونه به روش‌های زیر انجام دهید:

کلیک راست بر روی تصویر

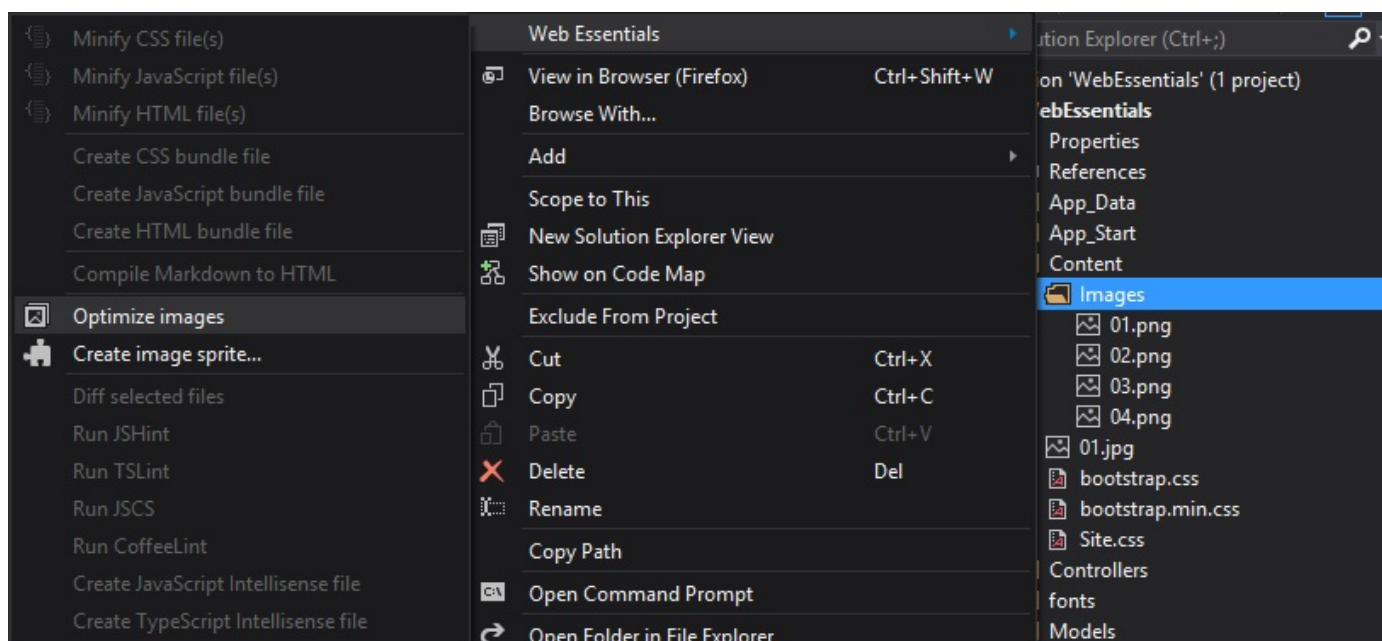
برای اینکار بر روی فایل‌ای که می‌خواهید optimize کنید، کلیک راست کرده و از منوی ظاهر شده گزینه Web Essentials و سپس Optimize Image را انتخاب کنید:



در قسمت status bar نیز می‌توانید نتیجه را مشاهده کنید:

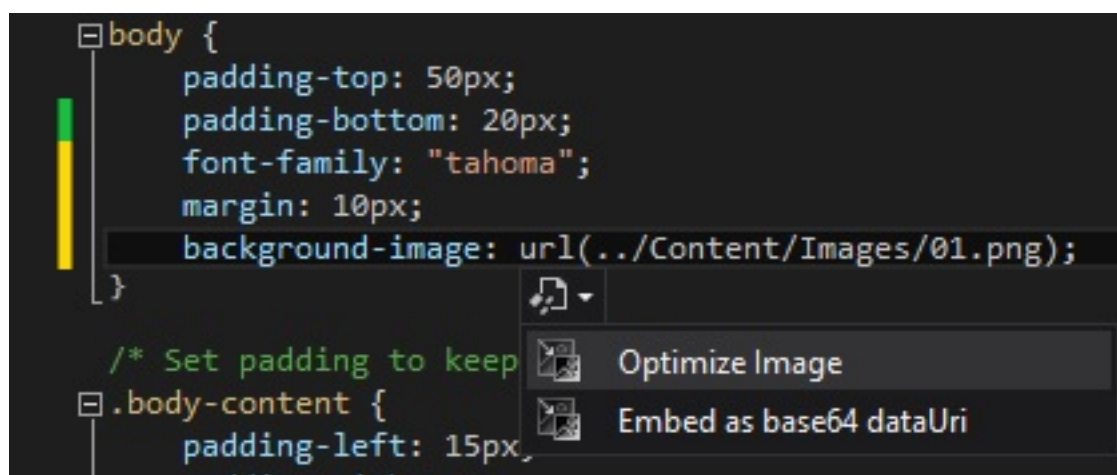


روال قبلی را می‌توانید برای چندین فایل انتخاب شده و یا یک پوشه تکرار کنید:



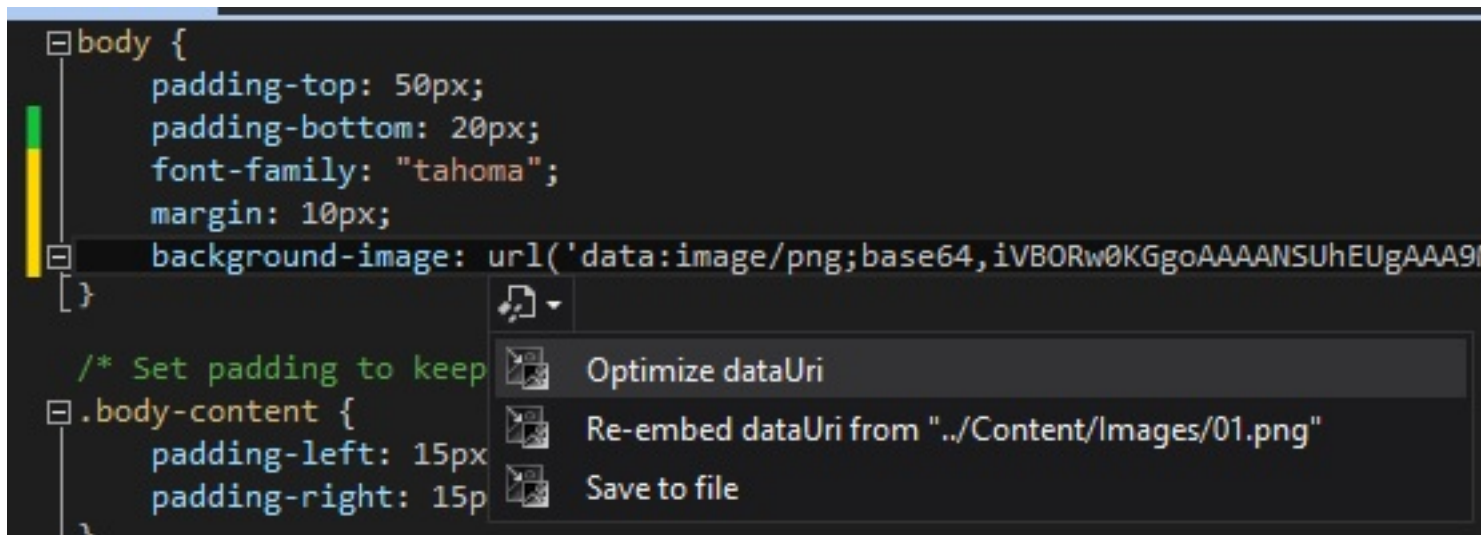
بهینه‌سازی تصاویر موجود در فایل‌های CSS

همچنین امکان بهینه‌سازی تصاویر داخل فایل‌های CSS نیز توسط این افزونه امکان پذیر است:



بهینه سازی تصاویر Base64 Encode

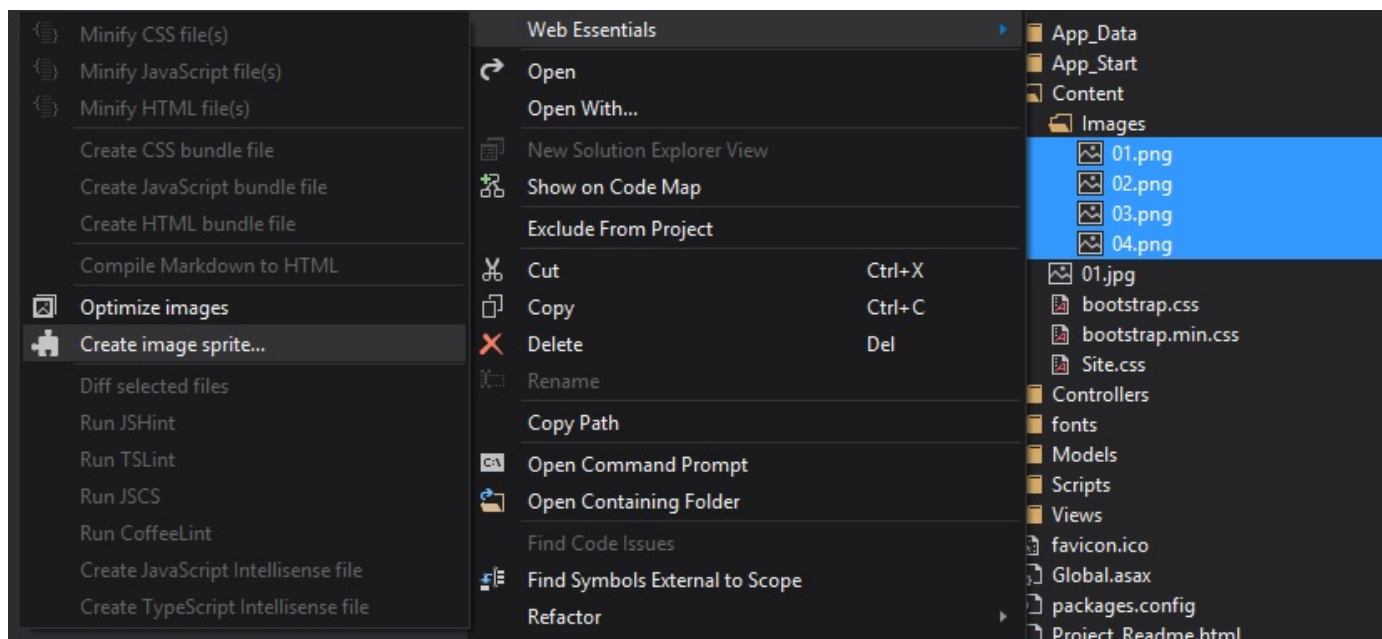
توسط این افزونه می‌توانیم تصاویر [Data Uri](#) را نیز بهینه سازی کنیم:



همانطور که در تصویر فوق مشاهده می‌کنید می‌توانیم تصاویری که به صورت Data Uri درون کد پیوست شده اند را با کلیک بر روی Save to file به صورت یک فایل ذخیره کنیم.

ایجاد تصاویر Sprite

یکی دیگر از قابلیت‌های افزونه Web Essentials امکان تهیه تصاویر به صورت [Sprite](#) می‌باشد. برای اینکار کافی است به این صورت عمل کنید:

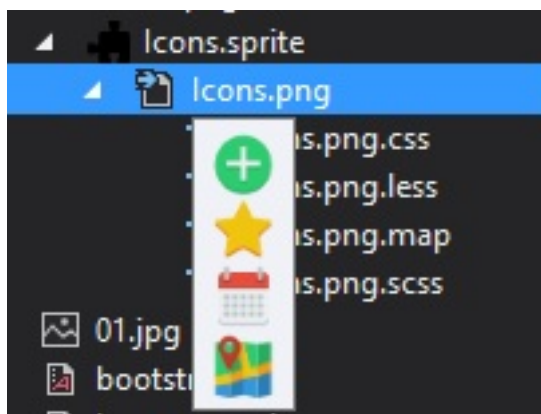


بعد از کلیک بر روی Create image sprite باید یک نام برای آن تعیین کنید و سپس بر روی کلید Save کلیک کنید. با اینکار یک فایل از نوع XML با پسوند sprite برای شما ساخته خواهد شد:

```
<?xml version="1.0" encoding="utf-8"?>
<sprite xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xsi:noNamespaceSchemaLocation="http://vswebessentials.com/schemas/v1/sprite.xsd">
  <settings>
    <!--Determines if the sprite image should be automatically optimized after creation/update.-->
    <optimize>true</optimize>
    <!--Determines the orientation of images to form this sprite. The value must be vertical or
```

```
horizontal.-->
<orientation>vertical</orientation>
<!--File extension of sprite image.-->
<outputType>png</outputType>
<!--Determin whether to generate/re-generate this sprite on building the solution.-->
<runOnBuild>>false</runOnBuild>
<!--Use full path to generate unique class or mixin name in CSS, LESS and SASS files. Consider
disabling this if you want class names to be filename only.-->
<fullPathForIdentifierName>>true</fullPathForIdentifierName>
<!--Use absolute path in the generated CSS-like files. By default, the URLs are relative to sprite
image file (and the location of CSS, LESS and SCSS).-->
<useAbsolutePath>>false</useAbsolutePath>
<!--Specifies a custom subfolder to save CSS files to. By default, compiled output will be placed
in the same folder and nested under the original file.-->
<outputDirectoryForCss />
<!--Specifies a custom subfolder to save LESS files to. By default, compiled output will be placed
in the same folder and nested under the original file.-->
<outputDirectoryForLess />
<!--Specifies a custom subfolder to save SCSS files to. By default, compiled output will be placed
in the same folder and nested under the original file.-->
<outputDirectoryForScss />
</settings>
<!--The order of the <file> elements determines the order of the images in the sprite.-->
<files>
<file>/Content/Images/01.png</file>
<file>/Content/Images/02.png</file>
<file>/Content/Images/03.png</file>
<file>/Content/Images/04.png</file>
</files>
</sprite>
```

یکی از زیر مجموعه‌های این فایل، تصویر نهایی می‌باشد، همچنین فایل‌های map، less، css و scss آن نیز تولید می‌شود:



به عنوان مثال فایل CSS تصویر فوق به صورت زیر می‌باشد:

```
/*
This is an example of how to use the image sprite in your own CSS files
*/
.Content-Images-01 {
/* You may have to set 'display: block' */
width: 32px;
height: 32px;
background: url('icons.png') 0 0;
}
.Content-Images-02 {
/* You may have to set 'display: block' */
width: 32px;
height: 32px;
background: url('icons.png') 0 -32px;
}
.Content-Images-03 {
/* You may have to set 'display: block' */
width: 32px;
height: 32px;
background: url('icons.png') 0 -64px;
```

```
}  
.Content-Images-04 {  
/* You may have to set 'display: block' */  
width: 32px;  
height: 32px;  
background: url('icons.png') 0 -96px;  
}
```

هر کدام از کلاس‌های فوق به یک تصویر در فایل مربوطه توسط image position اشاره می‌کند. شما می‌توانید با انتساب هر کدام از کلاس‌های فوق به یک المنت از آن تصویر استفاده نمائید:

```
<div class="Content-Images-01"></div>  
<div class="Content-Images-02"></div>  
<div class="Content-Images-03"></div>  
<div class="Content-Images-04"></div>
```

استفاده از تصاویر Data URIs

یکی دیگر از روش‌های کاهش درخواست‌های HTTP در یک سایت استفاده از [Data URIs](#) می‌باشد، توسط این روش می‌توانید فایل هایتان را درون HTML و یا CSS قرار دهید یا به اصطلاح embed کنید. به طور مثال جهت استفاده از یک تصویر می‌توانید به راحتی با آدرس دهی تصویر درون تگ img، تصویر را درون صفحه نمایش دهید:

```

```

همین کار را می‌توانیم توسط Data URIs انجام دهیم:

```

```

در کد فوق تصویر موردنظر را درون HTML به صورت embed شده قرار داده ایم، در این حالت دیگری نیازی به رفت و برگشت به سرور جهت نمایش تصویر نیست. **سینتکس Data URIs** به طور مثال تگ زیر را در نظر داشته باشید:

```

```

مقدار ویژگی src شامل موارد زیر است:

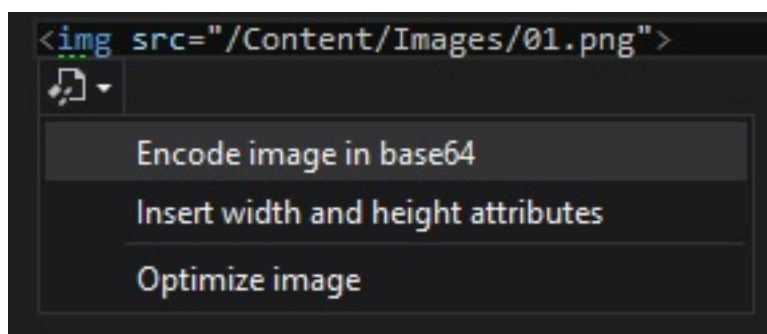
data: نام schema

image/png: نوع محتوا (content type)

base64: نوع encoding استفاده شده برای encode کردن اطلاعات

iVBOR...: اطلاعات encode شده.

توسط افزونه Web Essentials به راحتی می‌توانید تصویر موردنظرتان را به صورت Data URI تهیه کنید:



لطفا توجه فرمایید که جالب‌ترین قسمت این مقاله قابلیت استفاده از کلاس‌های دات نت در دل PowerShell می‌باشد. که در قسمت چهارم کدها مشاهده می‌فرمایید.

حذف تمام رکوردهای یک لیست شیرپوینت از طریق رابط کاربری SharePoint مسیر نمی‌باشد و لازم است برای آن چند خط کد نوشته شود که می‌توانید آن را با console و جالب‌تر از آن با PowerShell اجرا کنید. 1- ساده‌ترین روش حذف رکوردهای شیرپوینت را در روبرو مشاهده می‌فرمایید که به ازای حذف هر رکورد یک رفت و برگشت به پایگاه انجام می‌شود

```
SPList list = mWeb.GetList(strUrl);
if (list != null)
{
    for (int i = list.ItemCount - 1; i >= 0; i--)
    {
        list.Items[i].Delete();
    }
    list.Update();
}
```

2- با استفاده از [SPWeb.ProcessBatchData](#) در کد زیر می‌توانیم با سرعت بیشتر و هوشمندانه‌تری، حذف تمام رکوردها را در یک عمل انجام دهیم

```
public static void DeleteAllItems(string site, string list)
{
    using (SPSite spSite = new SPSite(site))
    {
        using (SPWeb spWeb = spSite.OpenWeb())
        {
            StringBuilder deletebuilder = BatchCommand(spWeb.Lists[list]);
            spSite.RootWeb.ProcessBatchData(deletebuilder.ToString());
        }
    }
}

private static StringBuilder BatchCommand(SPList spList)
{
    StringBuilder deletebuilder = new StringBuilder();
    deletebuilder.Append("<?xml version='1.0' encoding='UTF-8'><Batch>");
    string command = "<Method><SetList Scope='Request'>" + spList.ID +
        "</SetList><SetVar Name='ID'>{0}</SetVar><SetVar Name='Cmd'>Delete</SetVar></Method>";

    foreach (SPListItem item in spList.Items)
    {
        deletebuilder.Append(string.Format(command, item.ID.ToString()));
    }
    deletebuilder.Append("</Batch>");
    return deletebuilder;
}
```

3- در قسمت زیر همان روش batch قبلی را مشاهده می‌فرمایید که با تقسیم کردن batch ها به تکه‌های 1000 تایی کارایی آن را بالا برده ایم

```
// We prepare a String.Format with a String.Format, this is why we have a {{0}}
string command = String.Format("<Method><SetList Scope='Request'>{0}</SetList><SetVar Name='ID'>{0}</SetVar><SetVar Name='Cmd'>Delete</SetVar><SetVar Name='owsfileref'>{1}</SetVar></Method>", list.ID);
// We get everything but we limit the result to 100 rows
SPQuery q = new SPQuery();
q.RowLimit = 100;

// While there's something left
while (list.ItemCount > 0)
{
    // We get the results
    SPListItemCollection coll = list.GetItems(q);
```



```

StringBuilder sbDelete = new StringBuilder();
sbDelete.Append("<?xml version=\"1.0\" encoding=\"UTF-8\"?><Batch>");

Guid[] ids = new Guid[coll.Count];
for (int i=0;i<coll.Count;i++)
{
    SPlistItem item = coll[i];
    sbDelete.Append(string.Format(command, item.ID.ToString(), item.File.ServerRelativeUrl));
    ids[i] = item.UniqueId;
}
sbDelete.Append("</Batch>");

// We execute it
web.ProcessBatchData(sbDelete.ToString());

//We remove items from recyclebin
web.RecycleBin.Delete(ids);

list.Update();
}
}

```

4- در این قسمت به جالبترین و آموزندهترین قسمت این مطلب میپردازیم و آن import کردن namespaces ها و ساختن object های دات نت در دل PowerShell هست که می‌توانید به راحتی با مقایسه با کد قسمت قبلی که در console نوشته شده است، آنرا فرا بگیرید.

برای فهم script پاور شل زیر کافیت به چند نکته ساده زیر دقت کنید
ایجاد متغیرها به سادگی با شروع نوشتن نام متغیر با \$ و بدون تعریف نوع آنها انجام می‌شود
write-host حکم write را دارد و واضح است که نوشتن تنهای آن برای ایجاد یک line break می‌باشد.

کامنت کردن با #

عدم وجود semi colon برای اتمام فرامین

```

[System.Reflection.Assembly]::Load("Microsoft.SharePoint, Version=12.0.0.0, Culture=neutral,
PublicKeyToken=71e9bce111e9429c")
[System.Reflection.Assembly]::Load("Microsoft.SharePoint.Portal, Version=12.0.0.0, Culture=neutral,
PublicKeyToken=71e9bce111e9429c")
[System.Reflection.Assembly]::Load("Microsoft.SharePoint.Publishing, Version=12.0.0.0, Culture=neutral,
PublicKeyToken=71e9bce111e9429c")
[System.Reflection.Assembly]::Load("System.Web, Version=2.0.0.0, Culture=neutral,
PublicKeyToken=b03f5f7f11d50a3a")

write-host

# Enter your configuration here
$siteUrl = "http://mysharepointsite.example.com/"
$listName = "Name of my list"
$batchSize = 1000

write-host "Opening web at $siteUrl..."

$site = new-object Microsoft.SharePoint.SPSite($siteUrl)
$web = $site.OpenWeb()
write-host "Web is: $($web.Title)"

$list = $web.Lists[$listName];
write-host "List is: $($list.Title)"

while ($list.ItemCount -gt 0)
{
    write-host "Item count: $($list.ItemCount)"

    $batch = "<?xml version=`1.0`" encoding=`UTF-8`"?><Batch>"
    $i = 0

    foreach ($item in $list.Items)
    {
        $i++
        write-host "`rProcessing ID: $($item.ID) ($i of $batchSize)" -nonewline
    }
}

```

```
$batch += "<Method><SetList Scope=`Request`">${$list.ID}</SetList><SetVar  
Name=`ID`">${$item.ID}</SetVar><SetVar Name=`Cmd`">Delete</SetVar><SetVar  
Name=`owsfileref`">${$item.File.ServerRelativeUrl}</SetVar></Method>"  
  
    if ($i -ge $batchSize) { break }  
}  
  
$batch += "</Batch>"  
  
write-host  
  
write-host "Sending batch..."  
  
# We execute it  
$result = $web.ProcessBatchData($batch)  
  
write-host "Emptying Recycle Bin..."  
  
# We remove items from recyclebin  
$web.RecycleBin.DeleteAll()  
  
write-host  
  
$list.Update()  
}  
  
write-host "Done."
```

بر اساس [رفتار پیش فرض](#) در دیتابیس SQL Server، در زمان انجام دادن یک دستور که منجر به ایجاد تغییرات در اطلاعات موجود در جدول می‌شود (برای مثال دستور Update)، جدول مربوطه به صورت کامل Lock می‌شود، ولو آن دستور Update، فقط با یکی از رکوردهای آن جدول کار داشته باشد.

در سیستم‌های با تعداد تراکنش بالا و دارای تعداد زیاد کلاینت، این رفتار پیش فرض موجب ایجاد صفی از تراکنش‌های در حال انتظار بر روی جداولی می‌شود که ویرایش‌های زیادی بر روی آنها رخ می‌دهد. اگر چه که بنظر این مشکل [راه حل‌های زیادی دارد](#)، لکن آن راه حلی که همیشه موثر عمل می‌کند استفاده از SQL Server Table Hints است.

SQL Server Table Hints به تمامی آن دستوراتی گفته می‌شود که هنگام اجرای دستور اصلی (برای مثال Select و یا Update) رفتار پیش فرض SQL Server را بر اساس Hint ارائه شده تغییر می‌دهند.

لیست کامل این Hint ها را می‌توانید در [اینجا مشاهده کنید](#).

این Hint ای که در اینجا برای ما مفید است، آن است که به SQL Server بگوییم هنگام اجرای دستور Update، به جای Lock کردن کل جدول، فقط رکورد در حال ویرایش را Lock کند، و این باعث می‌شود تا باقی تراکنش‌ها، که ای بسا با سایر رکوردهای آن جدول کار داشته باشند متوقف نشوند، که البته این مسئله کمی به افزایش مصرف حافظه می‌انجامد، لکن مقدار افزایش بسیار ناچیز است.

این Hint که rowlock نام دارد در تراکنش‌های با Isolation Level تنظیم شده بر روی Snapshot باید با یک Table Hint دیگر با نام updlock ترکیب شود.

توضیحات مفصل‌تر این دو Hint در لینک مربوطه آمده است.

بنابر این، بجای دستور

```
update products
set Name = "Test"
Where Id = 1
```

داریم

```
update products with (nolock,updlock)
set Name = "Test"
where Id = 1
```

تا اینجا مشکل خاصی وجود ندارد، آنچه که از اینجا به بعد اهمیت دارد این است که در هنگام کار با Entity Framework، اساسا ما نویسنده دستورات Update نیستیم که به آنها Hint اضافه کنیم یا نه، بلکه دستورات SQL بوسیله Entity Framework ایجاد می‌شوند.

در Entity Framework، مکانیزمی تعبیه شده است با نام Db Command Interceptor که به شما اجازه می‌دهد دستورات SQL ساخته شده را [Log کنید](#) و یا قبل از اجرا [تغییر دهید](#)، که برای اضافه نمودن Table Hint ها ما از این روش استفاده می‌کنیم، برای انجام این کار داریم: (توضیحات در ادامه)

```
public class UpdateRowLockHintDbCommandInterceptor : IDbCommandInterceptor
{
    public void NonQueryExecuting(DbCommand command, DbCommandInterceptionContext<Int32> interceptionContext)
    {
        if (command.CommandType != CommandType.Text) return; // (1)
        if (!(command is SqlCommand)) return; // (2)
        SqlCommand sqlCommand = (SqlCommand)command;
        String commandText = sqlCommand.CommandText;
        String updateCommandRegularExpression = "(update) ";
```

```

        Boolean isUpdateCommand = Regex.IsMatch(commandText, updateCommandRegularExpression,
        RegexOptions.IgnoreCase | RegexOptions.Multiline); // You may use better regular expression pattern
        here.
        if (isUpdateCommand)
        {
            Boolean isSnapshotIsolationTransaction = sqlCommand.Transaction != null &&
            sqlCommand.Transaction.IsolationLevel == IsolationLevel.Snapshot;
            String tableHintToAdd = isSnapshotIsolationTransaction ? " with (rowlock , updlock) set
            " : " with (rowlock) set ";
            commandText = Regex.Replace(commandText, "^(set) ", (match) =>
            {
                return tableHintToAdd;
            }, RegexOptions.IgnoreCase | RegexOptions.Multiline);
            command.CommandText = commandText;
        }
    }
}

```

این کد در قسمت (1) ابتدا تشخیص می‌دهد که آیا این یک Command دارای Command Text است یا خیر، برای مثال اگر فراخوانی یک Stored Procedure است، ما با آن کاری نداریم.

در قسمت دوم تشخیص می‌دهیم که آیا با SQL Server در حال تعامل هستیم، یا برای مثال با Oracle و که ما برای Table Hintها فقط با SQL Server کار داریم.

سپس باید تشخیص دهیم که آیا این یک دستور update است یا خیر؟ برای این منظور از Regular Expressionها استفاده کرده ایم، که خیلی به بحث آموزش این پست مربوط نیست، به صورت کلی از Regular Expressionها برای یافتن و بررسی و جایگزینی عبارات با قاعده در هنگام کار با رشته‌ها استفاده می‌شود.

ممکن است Regular Expression ای که شما می‌نویسید بسیار بهتر از این نمونه باشد، که در این صورت خوشحال می‌شوم در قسمت نظرات آنرا قرار دهید.

در نهایت با بررسی Transaction Isolation Level مربوطه که Snapshot است یا خیر، به درج یک یا هر دو Table Hint مربوطه اقدام می‌نماییم.

مقدمه

OutputCaching باعث می‌شود خروجی یک اکشن متد در حافظه نگهداری شود. با اعمال این نوع کشینگ، ASP.NET در خواست‌های بعدی به این اکشن را تنها با بازگرداندن همان مقدار قبلی نگهداری شده در کش، پاسخ می‌دهد. در حقیقت با OutputCaching از تکرار چند باره کد درون یک اکشن در فراخوانی‌های مختلف جلوگیری کرده‌ایم. کش کردن باعث می‌شود که کارایی و سرعت سایت افزایش یابد؛ اما باید دقت کنیم که چه موقع و چرا از کش کردن استفاده می‌کنیم و چه موقع باید از این کار امتناع کرد.

فواید کش کردن

- انجام عملیات هزینه دار فقط یکبار صورت می‌گیرد. (هزینه از لحاظ فشار روی حافظه سرور و کاهش سرعت بالا آمدن سایت)

- بار روی سرور در زمان‌های پیک کاهش می‌یابد.

- سرعت بالا آمدن سایت بیشتر می‌شود.

چه زمانی باید کش کرد؟

- وقتی محتوای نمایشی برای همه کاربران یکسان است.

- وقتی محتوای نمایشی برای نمایش داده شدن، فشار زیادی روی سرور تحمیل می‌کند.

- وقتی محتوای نمایشی به شکل مکرر در طول روز باید نمایش داده شود.

- وقتی محتوای نمایشی به طور مکرر آپدیت نمی‌شود. (در مورد تعریف کیفیت "مکرر"، برنامه نویسی بهترین تصمیم گیرنده است)

طرح مساله

فرض کنید صفحه اول سایت شما دارای بخش‌های زیر است :

خلاصه اخبار بخش علمی، خلاصه اخبار بخش فرهنگی ، ده کامنت آخر، لیستی از کتگوری‌های موجود در سایت.

روش‌های مختلفی برای کوئری گرفتن وجود دارد، به عنوان مثال ما به کمک یک یا چند کوئری و توسط یک ViewModel جامع، می‌خواهیم اطلاعات را به سمت ویو ارسال کنیم. پس در اکشن متد Index ، حجم تقریباً کمی از اطلاعات را باید به کمک کوئری(کوئری‌های) تقریباً پیچیده ای دریافت کنیم و اینکار به ازای هر ریکوئست هزینه دارد و فشار به سرور وارد خواهد شد. از طرفی می‌دانیم صفحه اول ممکن است در طول یک یا چند روز تغییر نکند و همچنین شاید در طول یکساعت چند بار تغییر کند! به هر حال در جایی از سایت قرار داریم که کوئری (کوئری‌های) مورد نظر زیاد صدا زده میشوند ، در حقیقت صفحه اول احتمالاً بیشترین فشار ترافیکی را در بین صفحات ما دارد، البته این فقط یک احتمال است و ما دقیقاً از این موضوع اطلاع نداریم.

یکی از راه‌های انجام یک کش موفق و دانستن لزوم کش کردن، این است که دقیقاً بدانیم ترافیک سایت روی چه صفحه ای بیشتر است. در واقع باید بدانیم در کدام صفحه "هزینه‌ی اجرای عملیات موجود در کد" بیشترین است.

فشار ترافیکی (ریکوئست‌های زیاد) و آپدیت‌های روزانه‌ی دیتابیس را، در دو کفه ترازو قرار دهید؛ چه کار باید کرد؟ این تصمیمی است که شما باید بگیرید. نگرانی خود را در زمینه آپدیت‌های روزانه و ساعتی کمتر کنید؛ در ادامه راهی را معرفی میکنیم که آپدیت‌های هر از گاه شما، در پاسخ ریکوئست‌ها دیده شوند. کمی کفهی کش کردن را سنگین کنید.

به هر حال، فعال کردن قابلیت کش کردن برای یک اکشن، بسیار ساده است، کافیسیت ویژگی (attribute) آن را بالای اکشن بنویسید :

```
[OutputCache(Duration = "60", Location = OutputCacheLocation.Server)]
public ActionResult Index()
{
    // کوثری یا کوثری‌های لازم برای استفاده در صفحه اصلی و تبدیل آن به یک ویو مدل جامع//
}
```

```
[OutputCache(CacheProfile = "FirstPageIndex", Location=OutputCacheLocation.Server)]
public ActionResult Index()
{
    // کوثری یا کوثری‌های لازم برای استفاده در صفحه اصلی و تبدیل آن به یک ویو مدل جامع//
}
```

دو روش فوق برای کش کردن خروجی Index از لحاظ عملکرد یکسان است، به شرطی که در حالت دوم در وب کانفیگ و در بخش system.web آن، یک پروفایل ایجاد کنیم کنیم :

```
<caching>
  <outputCacheSettings>
    <outputCacheProfiles>
      <add name="FirstPageIndex" duration="60"/>
    </outputCacheProfiles>
  </outputCacheSettings>
</caching>
```

در حالت دوم ما یک پروفایل برای کشینگ ساخته ایم و در ویژگی بالای اکشن متد، آن پروفایل را صدا زده ایم. از لحاظ منطقی در حالت دوم، چون امکان استفاده مکرر از یک پروفایل در جاهای مختلف فراهم شده، روش بهتری است. محل ذخیره کش نیز در هر دو حالت سرور تعریف شده است.

برای تست عملیات کشینگ، کافیسیت یک BreakPoint درون Index قرار دهید و برنامه را اجرا کنید. پس از اجرا، برنامه روی Break Point می‌ایستد و اگر F5 را بزنیم، سایت بالا می‌آید. بار دیگر صفحه را رفرش کنیم، **اگر این "بار دیگر" در کمتر از 60 ثانیه پس از رفرش قبلی اتفاق افتاده باشد برنامه روی Break Point متوقف نخواهد شد**، چون خروجی اکشن، در کش بر روی سرور ذخیره شده است و این یعنی ما فشار کمتری به سرور تحمیل کرده ایم، صفحه با سرعت بالاتری در دسترس خواهد بود.

ما از تکرار اجرای کد جلوگیری کرده ایم و عدم اجرای کد بهترین نوع بهینه سازی برای یک سایت است. [اسکات الن، پلورال سایت]

چطور زمان مناسب برای کش کردن یک اکشن را انتخاب کنیم؟

- **کشینگ با زمان کوتاه** ؛ فرض کنید زمان کش را روی 1 ثانیه تنظیم کرده اید. این یعنی اگر ریکوئست هایی به یک اکشن ارسال شود و همه در طول یک ثانیه اتفاق بیفتد، آن اکشن فقط برای بار اول اجرا میشود، و در بارهای بعد(در طول یک ثانیه) فقط محتوای ذخیره شده در آن یک اجرا، بدون اجرای جدید، نمایش داده میشود. پس سرور شما فقط به یک ریکوئست در ثانیه در طول روز جواب خواهد داد و ریکوئست‌های تقریباً همزمان دیگر، در طول همان ثانیه، از نتایج آن ریکوئست (اگر موجود باشد) استفاده خواهند کرد

- **کشینگ با زمان طولانی** ؛ ما در حقیقت با اینکار منابع سرور را حفاظت میکنیم، چون عملیات هزینه دار(مثل کوثری‌های حجیم) تنها یکبار در طول زمان کشینگ اجرا خواهند شد. مثلاً اگر تنظیم زمان روی عدد 86400 تنظیم شود(یک روز کامل)، پس از اولین

ریکوئست به اکشن مورد نظر، تا 24 ساعت بعد، این اکشن اجرا نخواهد شد و فقط خروجی آن نمایش داده خواهد شد. آیا دلیلی دارد که یک کوئری هزینه دار را که قرار نیست خروجی اش در طول روز تغییر کند به ازای هر ریکوئست یک بار اجرا کنیم؟

اگر اطلاعات موجود در دیتابیس را تغییر دهیم چه کار کنیم که کشینگ رفرش شود؟

فرض کنید در همان مثال ابتدای این مقاله، شما یک پست به دیتابیس اضافه کرده اید، اما چون مثلاً duration مربوط به کشینگ را روی 86400 تعریف کرده اید تا 24 ساعت از زمان ریکوئست اولیه نگذرد، سایت آپدیت نخواهد شد و محتوا همان چیزهای قبلی باقی خواهند ماند. اما چاره چیست؟

کافیست در بخش ادمین، وقتی که یک پست ایجاد میکنید یا پستی را ویرایش میکند در اکشن‌های مرتبط با Create یا Edit یا Delete چنین کدی را پس از فرمان ذخیره تغییرات در دیتابیس، بنویسید:

```
Response.RemoveOutputCacheItem(Url.Action("index", "home"));
```

واضح است که ما داریم کشینگ مرتبط با یک اکشن متد مشخص را پاک میکنیم. با اینکار در اولین ریکوئست پس از تغییرات اعمال شده در دیتابیس، ASP.NET MVC چون میبند گشی برای این اکشن وجود ندارد، متد را اجرا میکند و کوئری‌های درونش را خواهد دید و اولین ریکوئست پیش از گش شدن را انجام خواهد داد. با اینکار کشینگ ریست شده است و پس از این ریکوئست و استخراج اطلاعات جدید، زمان کشینگ صفر شده و آغاز میشود.

میتوانید یک دکمه در بخش ادمین سایت طراحی کنید که هر موقع دلتان خواست کلیه کش‌ها را به روش فوق پاک کنید! تا اپلیکیشن منتظر ریکوئست‌های جدید بماند و کش‌ها دوباره ایجاد شوند.

جمع بندی

ویژگی OutputCach دارای پارامترهای زیادیست و در این مقاله فقط به توضیح عملکرد این اتریبیوت اکتفا شده است. بطور کلی این مبحث ظاهر ساده ای دارد، ولی نحوه استفاده از کشینگ کاملاً وابسته به هوش برنامه نویسی است و پیچیدگی‌های مرتبط با خود را دارد. در واقع خیلی مشکل است که بتوانید یک زمان مناسب برای کش کردن تعیین کنید. باید برنامه خود را در یک محیط شبیه سازی تحت بار قرار دهید و به کمک اندازه گیری و محاسبه به یک قضاوت درست از میزان زمان کش دست پیدا کنید. گاهی متوجه خواهید شد، از مقدار زیادی از حافظه سیستم برای کش کردن استفاده کرده اید و در حقیقت آنقدر ریکوئست ندارید که احتیاج به این هزینه کردن باشد.

یکی از روش‌های موثر برای دستیابی به زمان بهینه برای کش کردن استفاده از CacheProfile درون وب کانفیگ است. وقتی از کشینگ استفاده میکنید، در همان ابتدا مقدار زمانی مشخص برای آن در نظر نگرفته اید(در حقیقت مقدار زمان مشخصی نمیدانید) پس مجبور به آزمون و خطا و تست و اندازه گیری هستید تا بدانید چه مقدار زمانی را برای چه پروفایلی قرار دهید. مثلاً پروفایل هایی به شکل زیر تعریف کرده اید و نام آنها را به اکشن‌های مختلف نسبت داده اید. به راحتی میتوانید از طریق دستکاری وب کانفیگ مقادیر آن را تغییر دهید تا به حالت بهینه برسید، بدون آنکه کد خود را دستکاری کنید.

```
<caching>
  <outputCacheSettings>
    <outputCacheProfiles>
      <add name="Long" duration="86400"/>
      <add name="Average" duration="43600"/>
      <add name="Short" duration="600"/>
    </outputCacheProfiles>
  </outputCacheSettings>
</caching>
```

برای مطالعه جزئیات بیشتر در مورد OutputCaching مقالات زیر منابع مناسبی هستند.

[اینجا] و [اینجا]

نظرات خوانندگان

نویسنده: ابوالفضل رجب پور
تاریخ: ۹:۵۳ ۱۳۹۳/۰۴/۲۹

سلام

یک ابهام در یک مثال واقعی مثلا سایت خبری.

اگر بخواهیم خروجی اکشن اخبار رو کش کنیم، و در عین حال تعداد بازدید از هر خبر رو هم ثبت کنیم، چطور باید این کار رو انجام داد؟

نویسنده: مرتضی دلیل
تاریخ: ۱۰:۲۴ ۱۳۹۳/۰۴/۲۹

قاعدتا اگر اکشن مربوط به نمایش هر خبر مستقل از اکشن نمایش "آخرین اخبار" باشد، با کش کردن اکشن "آخرین اخبار" مشکلی برای اکشن نمایش دهنده هر خبر بوجود نخواهد آمد و میتوان در این اکشن، متدها یا عملیات مورد نظر را بدون نگرانی اعمال کرد. (اگر منظور از "ثبت"، ذخیره‌ی اطلاعات باشد)

نویسنده: وحید نصیری
تاریخ: ۱۰:۲۶ ۱۳۹۳/۰۴/۲۹

- با استفاده از jQuery که یک بحث سمت کاربر است، زمانیکه صفحه نمایش داده شد، یک درخواست Ajax ایی به اکشن متدی خاص، جهت به روز رسانی تعداد بار مشاهده ارسال کنید. به این روش client side tracking هم می‌گویند (کل اساس کار Google analytics به همین نحو است).

- روش دوم استفاده از [Donut Caching](#) است. در یک چنین حالتی، کد زیر مجاز است:

```
[LogThis]
[DonutOutputCache(Duration=5, Order=100)]
public ActionResult Index()
```

[اطلاعات بیشتر](#)

نویسنده: ایلیا اکبری فرد
تاریخ: ۱۶:۵۲ ۱۳۹۳/۰۸/۱۲

با سلام.

متدی به روش زیر در کنترلر خود ایجاد کرده ام:

```
[OutputCache(Duration = (7 * 24 * 60 * 60), VaryByParam = "none")]
[AllowAnonymous]
public virtual ActionResult Notification()
{
    ....
}
```

و در قسمت ادمین سیستم که در یک area جداگانه قرار دارد در اکشن متد خود اینگونه نوشتم:

```
Response.RemoveOutputCacheItem(Url.Action("Notification", "Article"));
Response.RemoveOutputCacheItem(Url.Action("Notification", "Article", new { area = "" }));
```

هیچکدام از دو روش بالا برایم جواب نمی‌دهد و کش خالی نمی‌شود. علت چیست؟

نویسنده: محسن خان
تاریخ: ۱۸:۴۱ ۱۳۹۳/۰۸/۱۲

آیا از `Html.RenderAction` برای نمایش آن استفاده کردید؟ اگر بله، متد یاد شده تاثیری روی کش آن نداره، چون نحوه‌ی کش شدن `child action`ها متفاوت.

نویسنده: ایلیا اکبری فرد
تاریخ: ۱۳۹۳/۰۸/۱۲ ۱۹:۱۵

بله. راه حل مشکل چیست؟

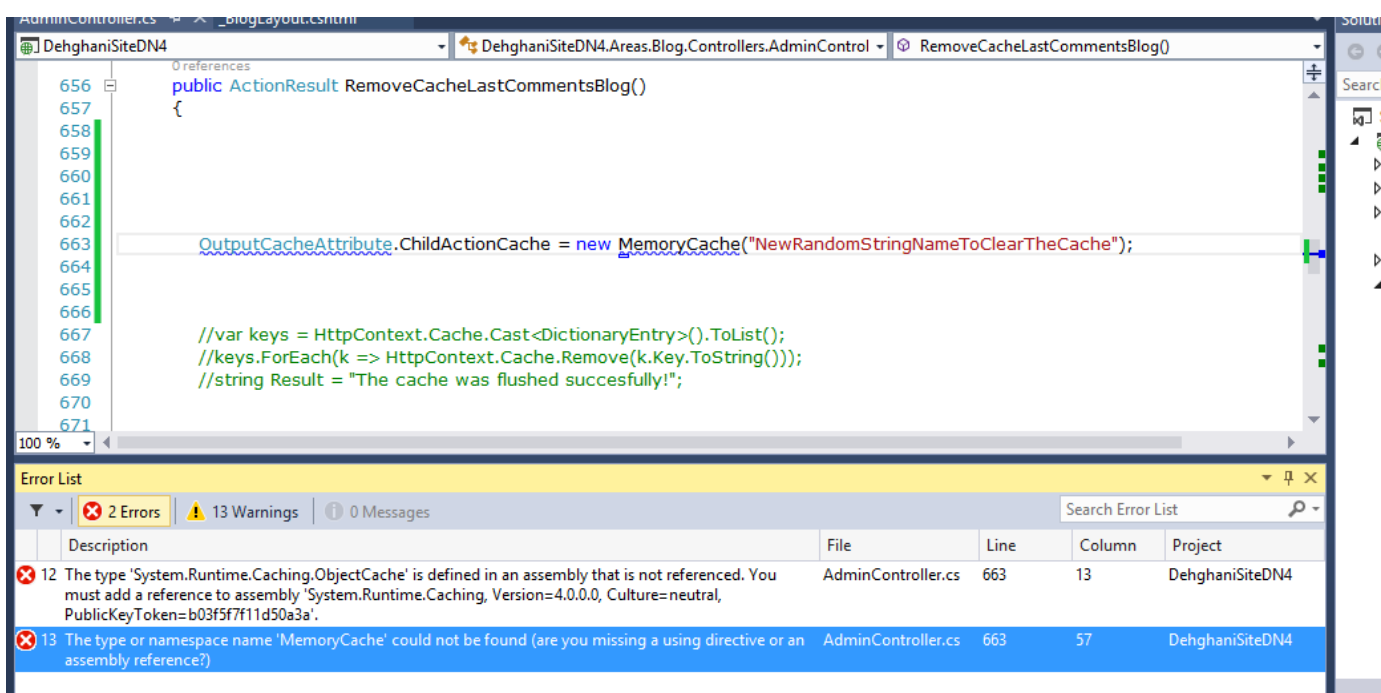
نویسنده: وحید نصیری
تاریخ: ۱۳۹۳/۰۸/۱۳ ۱۵:۱۴

به این صورت؛ البته این روش کش تمام `child action`ها را با هم پاک می‌کند:

```
OutputCacheAttribute.ChildActionCache = new MemoryCache("NewRandomStringNameToClearTheCache");
```

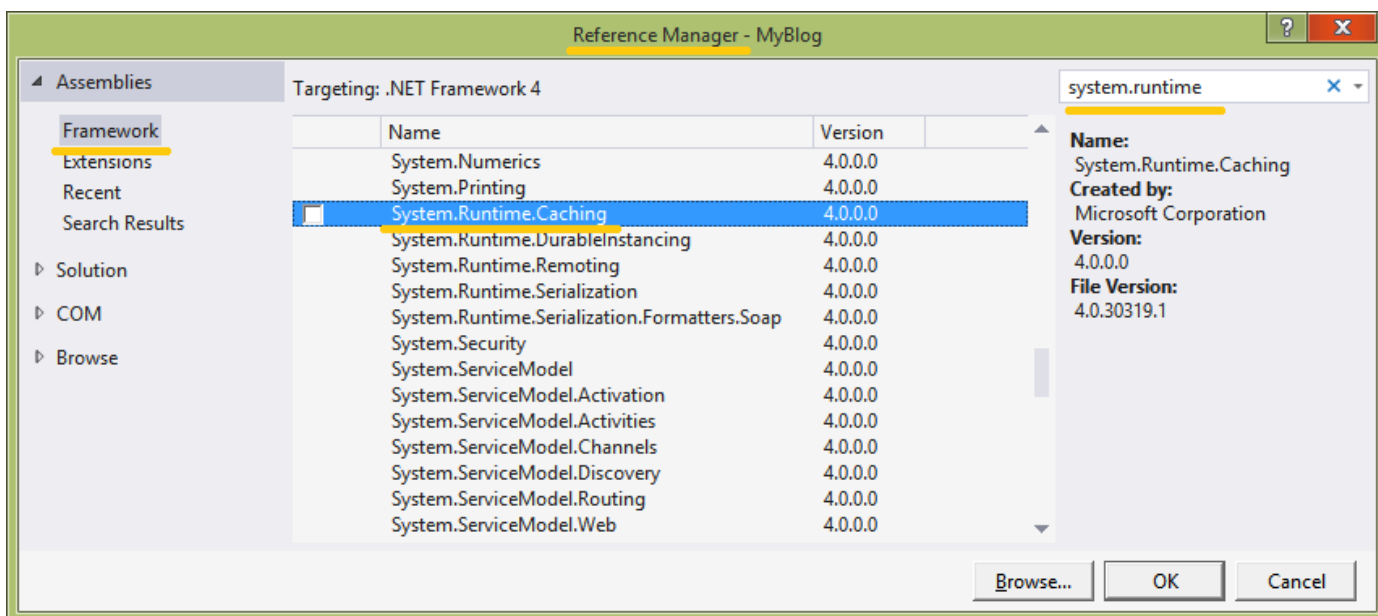
نویسنده: امیر عظیمی
تاریخ: ۱۳۹۴/۰۸/۰۴ ۱۷:۸

با سلام؛ من به `Layout` دارم که توش آخرین نظرات رو بصورت یک اکشن که یک `PartialView` را صدا میرنه البته از طریق فرمان `Html.RenderAction` این کار انجام میشه. طبق فرموده شما که همیشه `child action`ها رو بصورت فرمان عمومی کششون رو ریست کرد. من هم از همین فرمان شما در نظر بالا استفاده کردم ولی یک ارور میده که در عکس پیوست شده است.



نویسنده: وحید نصیری
تاریخ: ۱۳۹۴/۰۸/۰۴ ۱۹:۴۱

ارجاعی را به اسمبلی استاندارد `System.Runtime.Caching` اضافه کنید.



نویسنده: امیر عظیمی
تاریخ: ۱۳۹۴/۰۸/۰۴ ۲۱:۲۳

متشکرم بابت پاسخ شما

اما من چنین متدی رو نوشتم. یک Action دارم که یک View رو برمی گردونه و یک Action که یک PartialView را صدا میزنه؛ البته توسط Html.renderAction. حال من متد زیر را برای ریست کردن کش نوشتم. اما بازهم ریست نکرد

```
public ActionResult RemoveCacheLastCommentsBlog()
{
    OutputCacheAttribute.ChildActionCache =
        new MemoryCache("NewRandomStringNameToClearTheCache");
    HttpResponseMessage.RemoveOutputCacheItem(Url.Action("Index", "Blog"));
}
```

نویسنده: وحید نصیری
تاریخ: ۱۳۹۴/۰۸/۰۵ ۱۱:۲۲

- سمت سرور این کش حذف شده است (از حافظه ی IIS). برای اجبار به حذف کش سمت کلاینت از نکات مطلب « [غیرفعال کردن کش مرورگر در MVC](#) » استفاده کنید.
- همچنین مطلب « [بازنویسی سطح دوم کش برای Entity framework 6](#) » شاید برای کار شما مناسب تر باشد.

Multicore JIT یکی از قابلیت‌های کلیدی در دات نت 4.5 می‌باشد که در واقع راه حلی برای بهبود سرعت اجرای برنامه‌های دات نت است. قبل از معرفی این قابلیت ابتدا اجازه دهید نحوه کامپایل یک برنامه دات نت را بررسی کنیم.

انواع compilation

در حالت کلی دو نوع فرآیند کامپایل داریم:

Explicit

در این حالت دستورات قبل از اجرای برنامه به زبان ماشین تبدیل می‌شوند. به این نوع کامپایلرها AOT یا Ahead Of Time گفته می‌شود. این نوع از کامپایلرها برای اطمینان از اینکه CPU بتواند قبل از انجام تعاملی تمام خطوط کد را تشخیص دهد، طراحی شده اند.

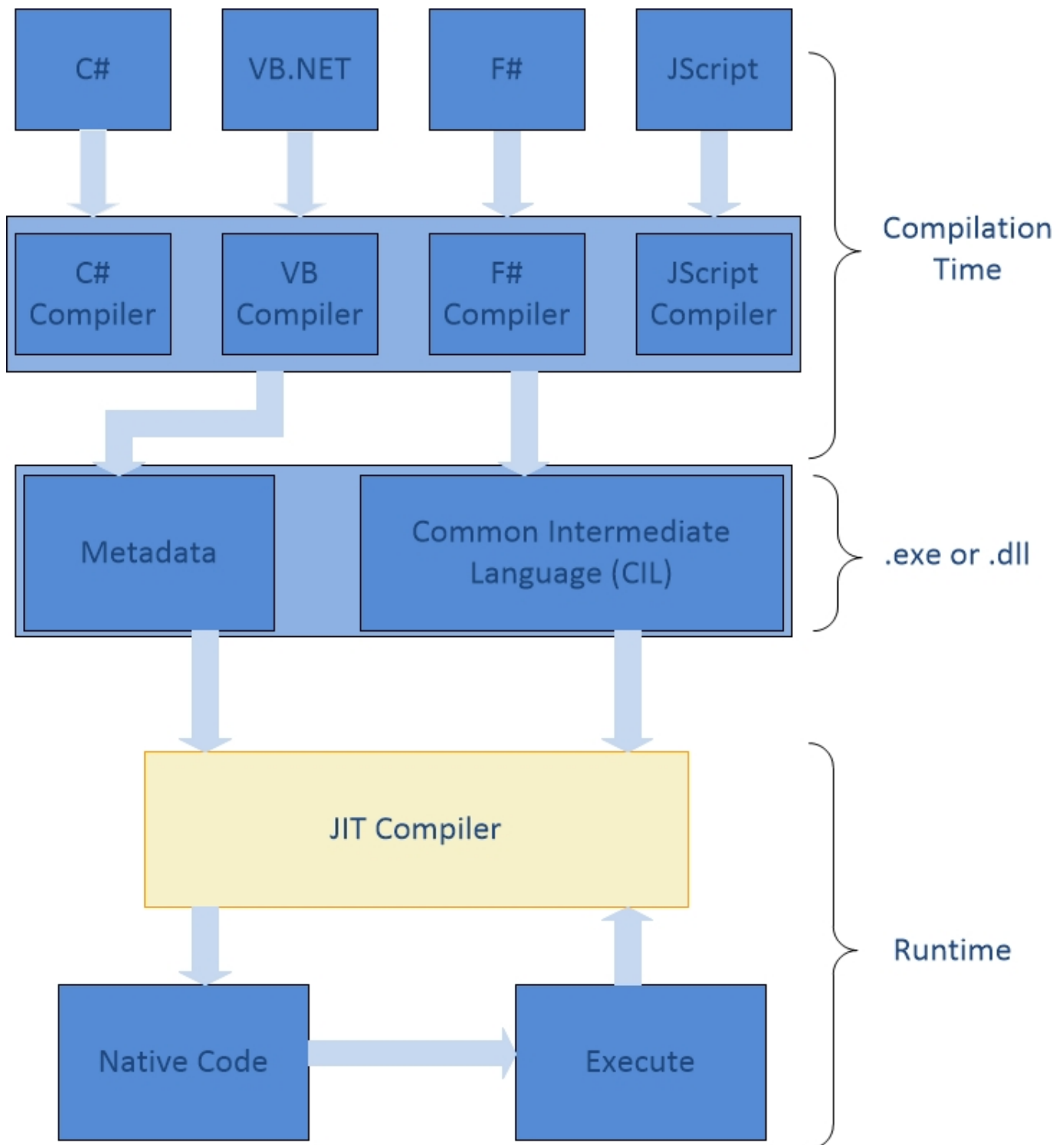
Implicit

این نوع compilation به صورت دو مرحله ای صورت می‌گیرد. در اولین قدم سورس کد توسط یک کامپایلر به یک زبان سطح میانی (IL) تبدیل می‌شود. در مرحله بعدی کد IL به دستورات زبان ماشین تبدیل می‌شوند. در دات نت فریم ورک به این کامپایلر JIT یا Just-In-Time گفته می‌شود.

در حالت دوم قابلیت جابجایی برنامه به آسانی امکان پذیر است، زیرا اولین قدم از فرآیند به اصطلاح platform agnostic می‌باشد، یعنی قابلیت اجرا بر روی گستره وسیعی از پلت فرم‌ها را دارد.

کامپایلر JIT

JIT بخشی از Common Language Runtime یا CLR می‌باشد. CLR در واقع وظیفه مدیریت اجرای تمام برنامه‌های دات نت را برعهده دارد.

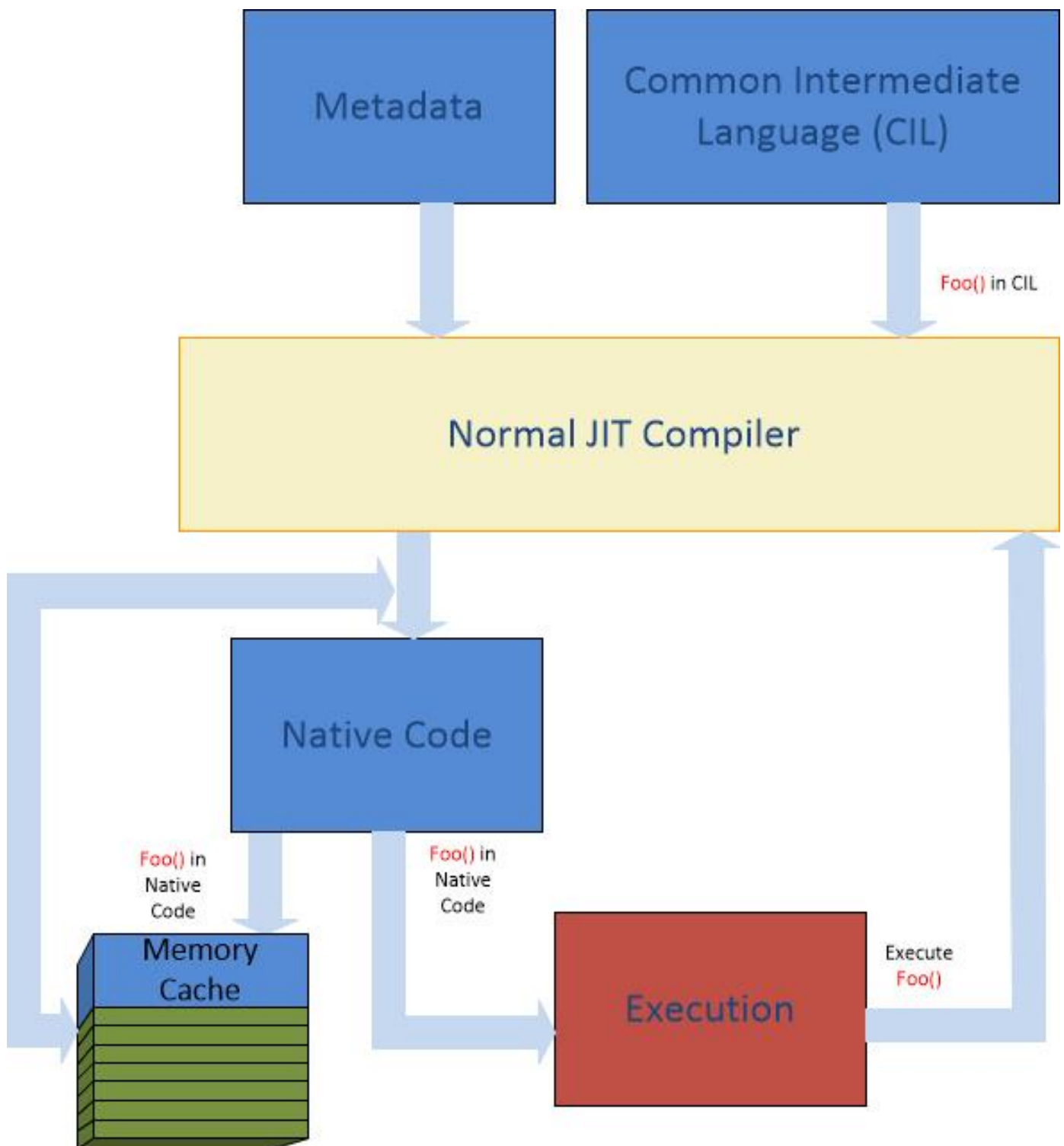


همانطور که در تصویر فوق مشاهده می‌کنید، سورس کد توسط کامپایلر دات نت به exe و یا dll کامپایل می‌شود. کامپایلر JIT تنها متدهایی را که در زمان اجرا (runtime) فراخوانی می‌شوند را کامپایل می‌کند. در دات نت فریم ورک سه نوع JIT Compilation داریم:

Normal JIT Compilation

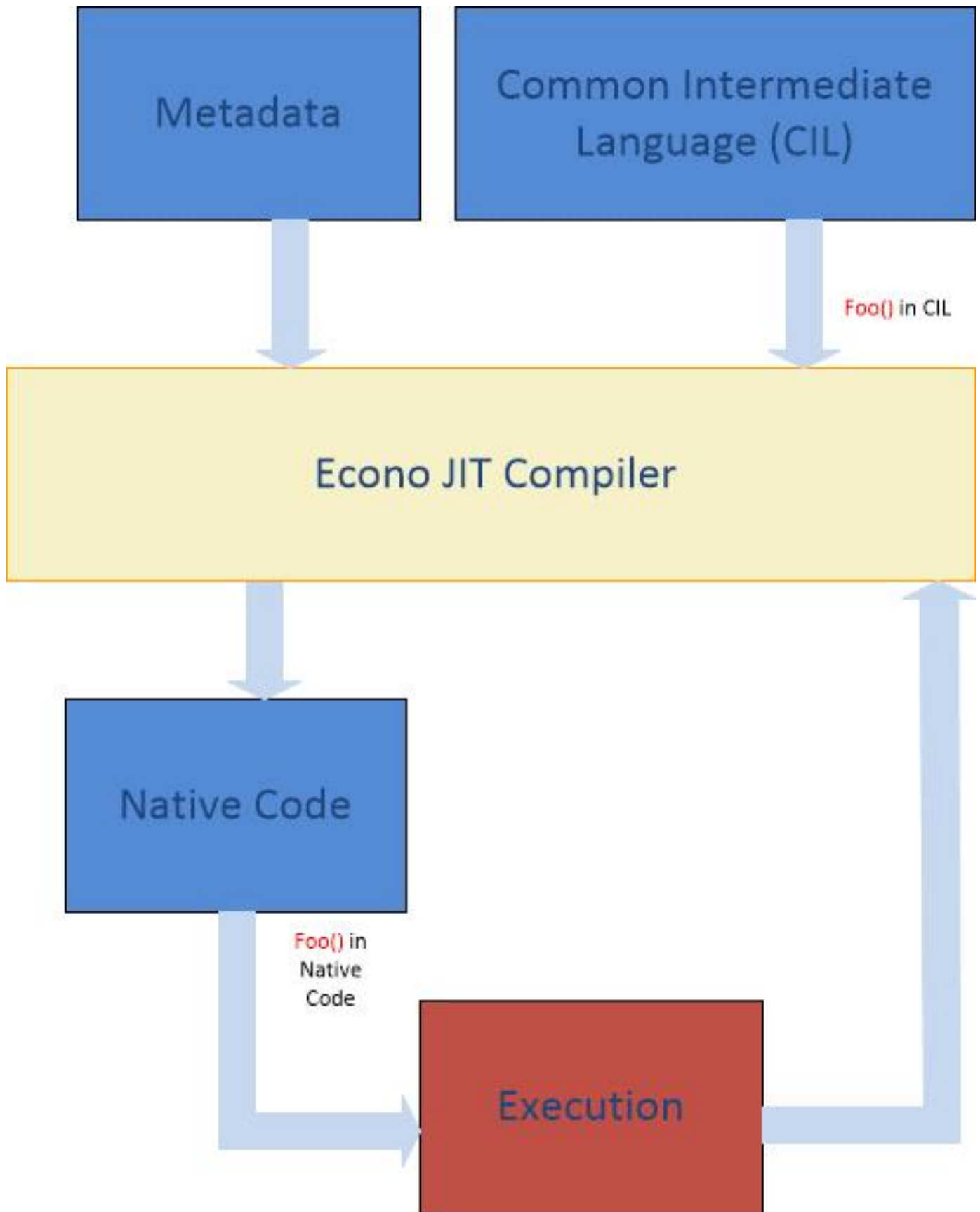
در این نوع کامپایل، متدها در زمان فراخوانی در زمان اجرا کامپایل می‌شوند. بعد از اجرا، متد داخل حافظه ذخیره می‌شود. به متدهای ذخیره شده در حافظه jitted گفته می‌شود. دیگر نیازی به کامپایل متد jit شده نیست. در فراخوانی بعدی، متد مستقیماً

از حافظه کش در دسترس خواهد بود.



Econo JIT Compilation

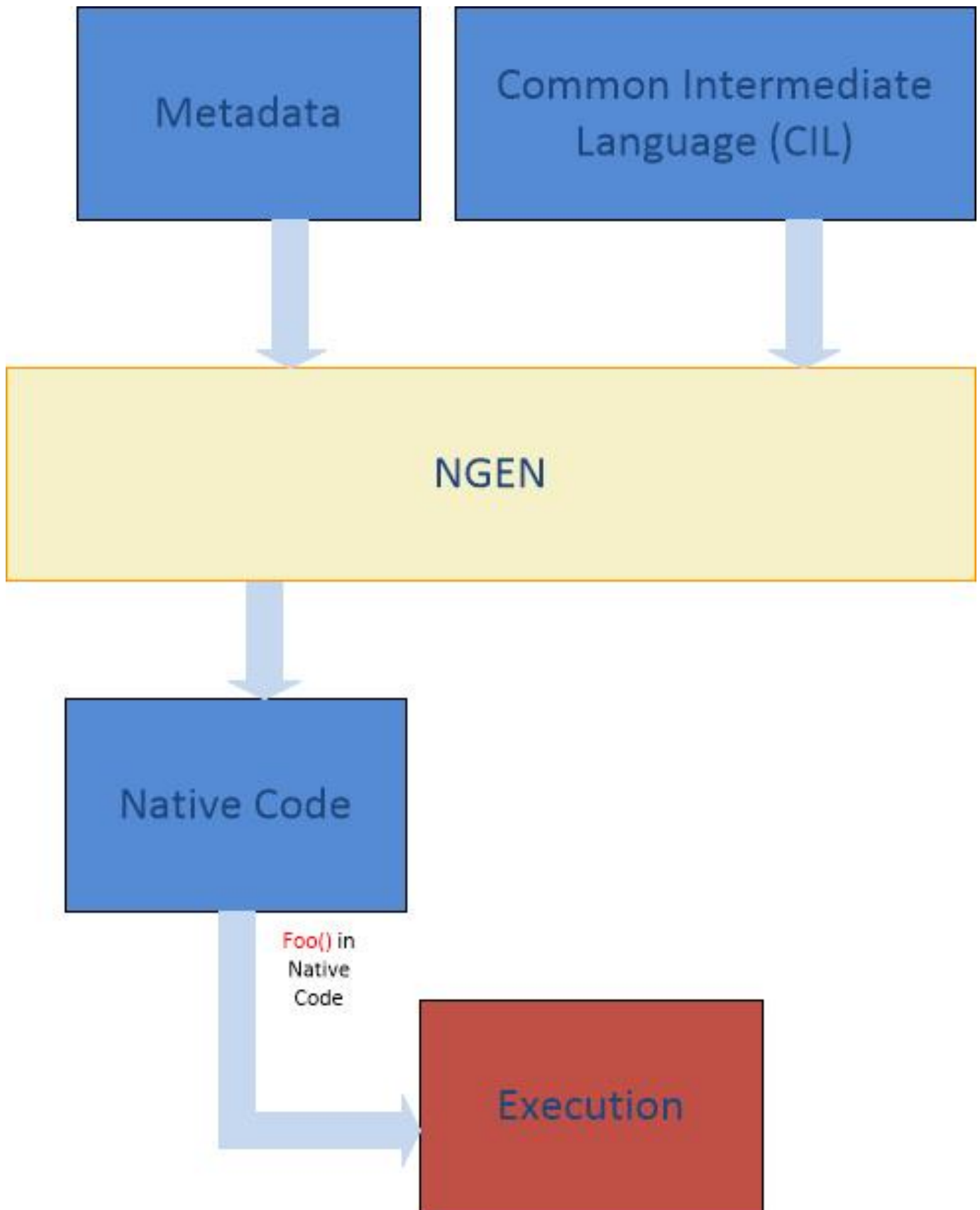
این نوع کامپایل شبیه به حالت Normal JIT است با این تفاوت که متدها بلافاصله بعد از اجرا از حافظه حذف می‌شوند.



Pre-JIT Compilation

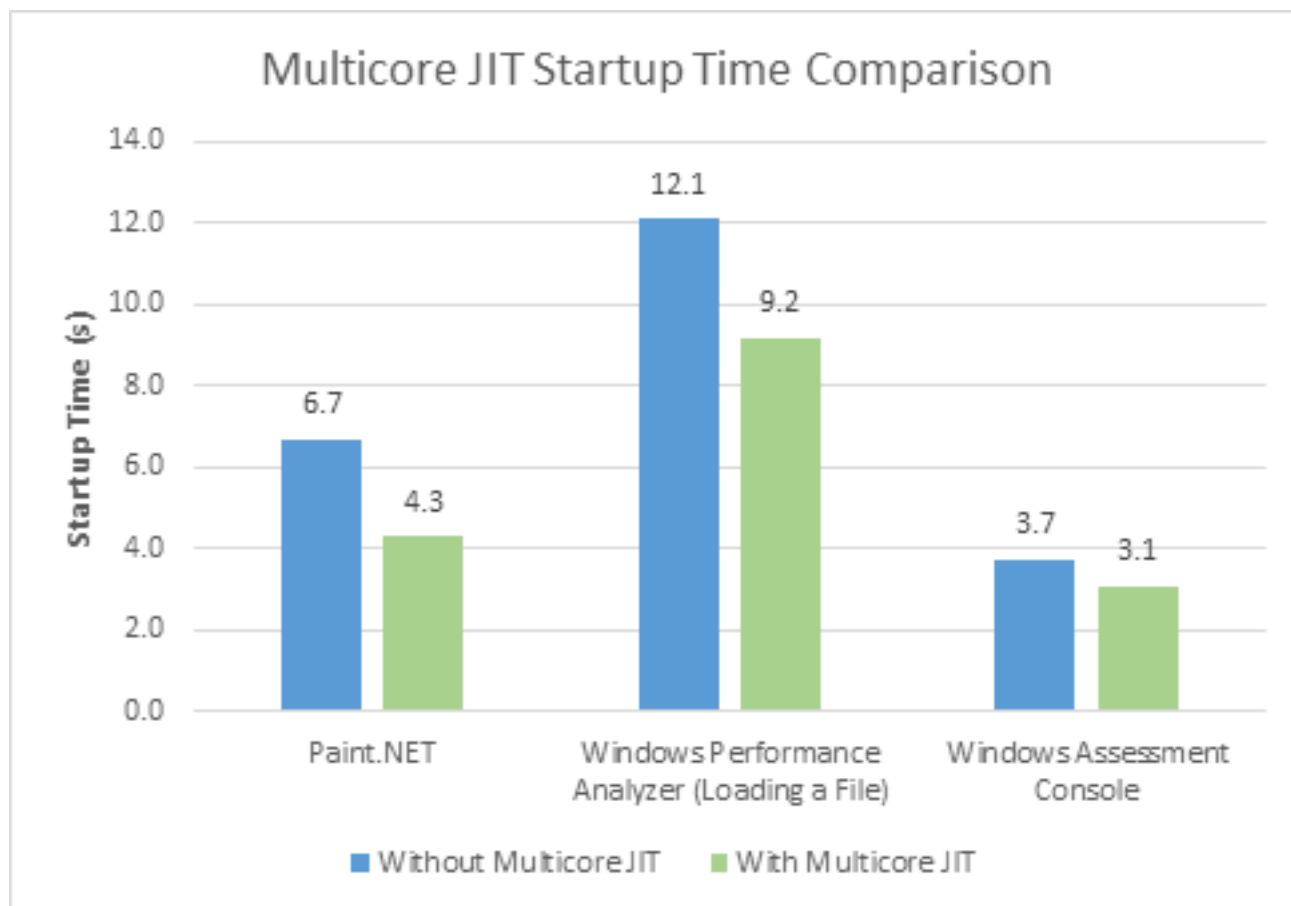
یکی دیگر از حالت‌های کامپایل برنامه‌های دات نتی Pre-JIT Compilation می باشد. در این حالت به جای متدهای مورد استفاده،

کل اسمبلی کامپایل می‌شود. در دات نت می‌توان اینکار را توسط [Ngen.exe](#) یا (Native Image Generator) انجام داد. تمام دستورالعمل‌های CIL قبل از اجرا به کد محلی (Native Code) کامپایل می‌شوند. در این حالت runtime می‌تواند از native images به جای کامپایلر JIT استفاده کند. این نوع کامپایل عملیات تولید کد را در زمان اجرای برنامه به زمان Installation منتقل می‌کند، در اینصورت برنامه نیاز به یک Installer برای اینکار دارد.



همانطور که عنوان شد Ngen.exe برای در دسترس بودن نیاز به Installer برای برنامه دارد. توسط Multicore JIT متدها بر روی دو هسته به صورت موازی کامپایل می‌شوند، در اینصورت می‌توانید تا 50 درصد از JIT Time صرفه جویی کنید.

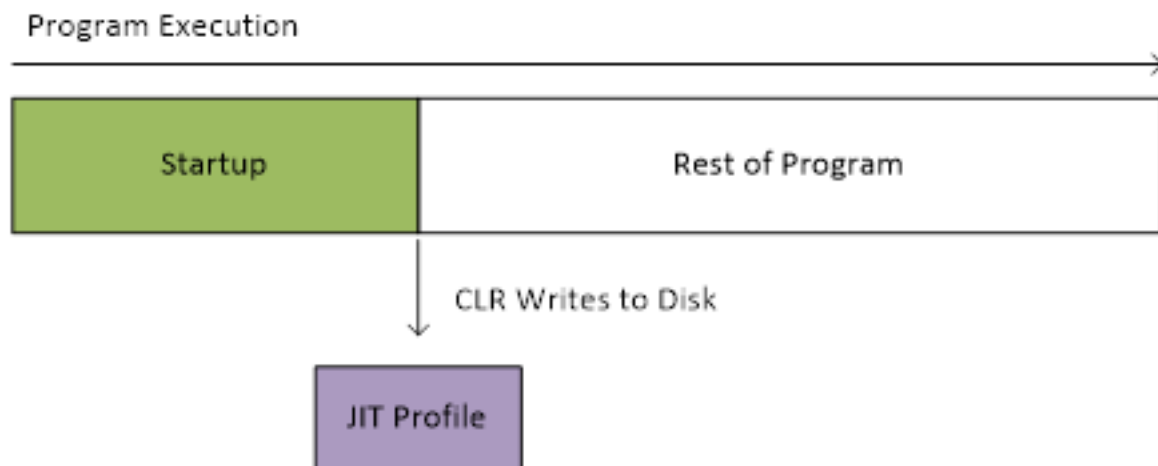
Multicore JIT همچنین می‌تواند باعث بهبود سرعت در برنامه‌های WPF شود. در نمودار زیر می‌توانید حالت‌های استفاده و عدم استفاده از Multicore JIT را در سه برنامه WPF نوشته شده مشاهده کنید.



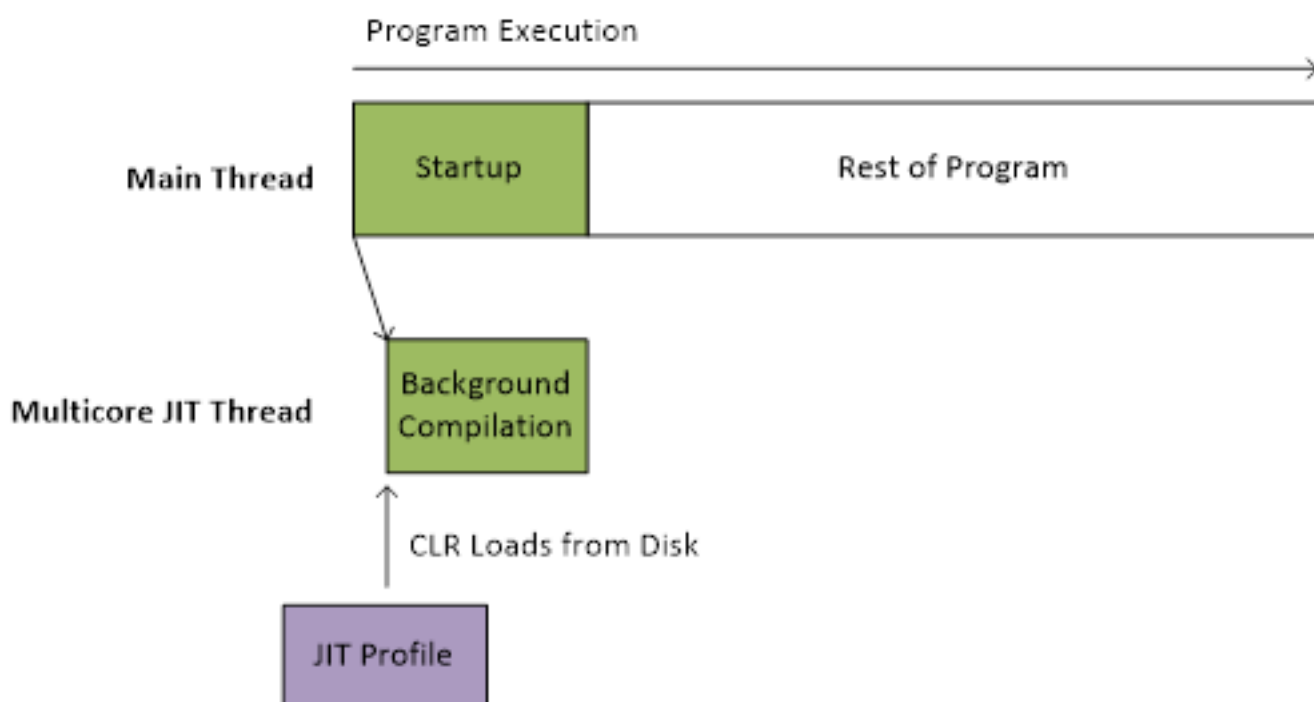
Multicore JIT در عمل

Multicore JIT از دو مد عملیاتی استفاده می‌کند: مد ثبت (Recording mode)، مد بازپخش (Playback mode)

در حالت ثبت کامپایلر JIT هر متدی که نیاز به کامپایل داشته باشد را رکورد می‌کند. بعد از اینکه CLR تعیین کند که اجرای برنامه به اتمام رسیده است، تمام متدهایی که اجرا شده اند را به صورت یک پروفایل بر روی دیسک ذخیره می‌کند.



هنگامیکه Multicore JIT فعال می‌شود، با اولین اجرای برنامه، حالت ثبت مورد استفاده قرار می‌گیرد. در اجراهای بعدی، از حالت بازپخش استفاده می‌شود. حالت بازپخش پروفایل را از طریق دیسک بارگیری کرده، و قبل از اینکه این اطلاعات توسط ترد اصلی مورد استفاده قرار گیرد، از آنها برای تفسیر (کامپایل) متدها در پیش‌زمینه استفاده می‌کند.



در نتیجه، ترد اصلی به کامپایل دیگری نیاز ندارد، در این حالت سرعت اجرای برنامه بیشتر می‌شود. حالت‌های ثبت و بازپخش تنها برای کامپیوترهایی با چندین هسته فعال می‌باشند.

استفاده از Multicore JIT

در برنامه‌های ASP.NET 4.5 و Silverlight 5 به صورت پیش فرض این ویژگی فعال می‌باشد. از آنجائیکه این برنامه‌ها hosted application هستند؛ در نتیجه فضای مناسبی برای ذخیره سازی پروفایل در این نوع برنامه‌ها موجود می‌باشد. اما برای برنامه‌های

Desktop این ویژگی باید فعال شود. برای اینکار کافی است دو خط زیر را به نقطه شروع برنامه تان اضافه کنید:

```
public App()
{
    ProfileOptimization.SetProfileRoot(@"C:\MyAppFolder");
    ProfileOptimization.StartProfile("Startup.Profile");
}
```

توسط متد [SetProfileRoot](#) می‌توانیم مسیر ذخیره سازی پروفایل JIT را مشخص کنیم. در خط بعدی نیز توسط متد StartProfile نام پروفایل را برای فعال سازی Multicore JIT تعیین می‌کنیم. در این حالت در اولین اجرای برنامه پروفایلی وجود ندارد، Multicore JIT در حالت ثبت عمل می‌کند و پروفایل را در مسیر تعیین شده ایجاد می‌کند. در دومین بار اجرای برنامه CRL پروفایل را از اجرای قبلی برنامه بارگذاری می‌کند؛ در این حالت Multicore JIT به صورت بازپخش عمل می‌کند.

همانطور که عنوان شد در برنامه‌های ASP.NET 4.5 و Silverlight 5 قابلیت Multicore JIT به صورت پیش فرض فعال می‌باشد. برای غیر فعال سازی آن می‌توانید با تغییر فلگ profileGuidedOptimizations به None اینکار را انجام دهید:

```
<?xml version="1.0" encoding="utf-8" ?>
<configuration>
  <!-- ... -->
  <system.web>
    <compilation profileGuidedOptimizations="None" />
  <!-- ... -->
  </system.web>
</configuration>
```

در این مقاله سعی داریم تا سرعت یافت و جستجوی View های متناظر با هر اکشن را در View Engine، با پیاده سازی قابلیت Caching نتیجه یافت آدرس فیزیکی view ها در درخواست های متوالی، افزایش دهیم تا عملاً بازده سیستم را تا حدودی بهبود ببخشیم.

طی مطالعاتی که بنده بر روی سورس MVC داشتم، به صورت پیش فرض، در زمانیکه پروژه در حالت Release اجرا می شود، نتیجه حاصل از یافت آدرس فیزیکی ویوهای متناظر با اکشن متدها در Application cache ذخیره می شود (HttpContext.Cache). این امر سبب اجتناب از عمل یافت چند باره بر روی آدرس فیزیکی ویوها در درخواست های متوالی ارسال شده برای رندر یک ویو خواهد شد.

نکته ای که وجود دارد این هست که علاوه بر مفید بودن این امر و بهبود سرعت در درخواست های متوالی برای اکشن متدها، این عمل با توجه به مشاهدات بنده از سورس MVC علاوه بر مفید بودن، تا حدودی هزینه بر هم هست و هزینه ای که متوجه سیستم می شود شامل مسائل مدیریت توکار حافظه کش توسط MVC است که مسائلی مانند سیاست های مدیریت زمان انقضای مداخل موجود در حافظه ی کش اختصاص داده شده به Lookup Caching و مدیریت مسائل thread-safe و ... را شامل می شود.

همانطور که می دانید، معمولاً تعداد ویوها اینقدر زیاد نیست که Caching نتایج یافت مسیر فیزیکی view ها، حجم زیادی از حافظه Ram را اشغال کند پس با این وجود به نظر می رسد که اشغال کردن این میزان اندک از حافظه در مقابل بهبود سرعت، قابل چشم پوشی است و سیاست های توکار نامبرده فقط عملاً تأثیر منفی در روند Lookup Caching پیشفرض MVC خواهند گذاشت. برای جلوگیری از تأثیرات منفی سیاست های نامبرده و عملاً بهبود سرعت Caching نتایج Lookup آدرس فیزیکی ویوها میتوانیم یک لایه Caching سطح بالاتر به View Engine اضافه کنیم.

خوشبختانه تمامی View Engine های MVC شامل Web Forms و Razor از کلاس VirtualPathProviderViewEngine مشتق شده اند که نکته مثبت که توسعه Caching اختصاصی نامبرده را برای ما مقدور می کند. در اینجا خاصیت (Property) قابل تنظیم ViewLocationCache از نوع IViewLocationCache هست.

بنابراین ما یک کلاس جدید ایجاد کرده و از اینترفیس IViewLocationCache مشتق میکنیم تا به صورت دلخواه بتوانیم اعضای این اینترفیس را پیاده سازی کنیم.

خوب؛ بنابر این اوصاف، من کلاس یاد شده را به شکل زیر پیاده سازی کردم:

```
public class CustomViewCache : IViewLocationCache
{
    private readonly static string s_key = "_customLookupCach" + Guid.NewGuid().ToString();
    private readonly IViewLocationCache _cache;

    public CustomViewCache(IViewLocationCache cache)
    {
        _cache = cache;
    }

    private static IDictionary<string, string> GetRequestCache(HttpContextBase httpContext)
    {
        var d = httpContext.Cache[s_key] as IDictionary<string, string>;
        if (d == null)
        {
            d = new Dictionary<string, string>();
            httpContext.Cache.Insert(s_key, d, null, Cache.NoAbsoluteExpiration, new TimeSpan(0,
15, 0));
        }
        return d;
    }
}
```

```
public string GetViewLocation(HttpContextBase httpContext, string key)
{
    var d = GetRequestCache(httpContext);
    string location;
    if (!d.TryGetValue(key, out location))
    {
        location = _cache.GetViewLocation(httpContext, key);
        d[key] = location;
    }
    return location;
}

public void InsertViewLocation(HttpContextBase httpContext, string key, string virtualPath)
{
    _cache.InsertViewLocation(httpContext, key, virtualPath);
}
}
```

و به صورت زیر می‌توانید از آن استفاده کنید:

```
protected void Application_Start() {
    ViewEngines.Engines.Clear();
    var ve = new RazorViewEngine();
    ve.ViewLocationCache = new CustomViewCache(ve.ViewLocationCache);
    ViewEngines.Engines.Add(ve);
    ...
}
```

نکته: فقط به یاد داشته باشید که اگر View جدیدی اضافه کردید یا یک View را حذف کردید، برای جلوگیری از بروز مشکل، حتماً و حتماً اگر پروژه در مراحل توسعه بر روی IIS قرار دارد app domain را ری‌استارت کنید تا حافظه کش مربوط به یافت‌ها پاک شود (و به روز رسانی) تا عدم وجود آدرس فیزیکی View جدید در کش، شما را دچار مشکل نکند.

نظرات خوانندگان

نویسنده: محسن خان
تاریخ: ۱۳۹۳/۰۵/۰۲ ۹:۵۶

ضمن تشکر از ایده‌ای که مطرح کردید. طول عمر HttpContext.Items فقط [محدوده به یک درخواست](#) و پس از پایان درخواست از بین می‌رود. مثلاً یکی از کاربردهای ذخیره اطلاعات Unit of work در طول یک درخواست هست و بعد از بین رفتن خودکار آن. بنابراین در این مثال cache.GetViewLocation اصلی بعد از یک درخواست مجدداً فراخوانی میشه، چون GetRequestCache نه فقط طول عمر کوتاهی داره، بلکه اساساً کاری به key متد GetViewLocation نداره. کار s_key تعریف شده [عموماً تعریف lock هست](#) نه استفاده ازش به عنوان کلید دیکشنری. بنابراین اگر خود MVC از HttpContext.Cache استفاده کرده، کار درستی بوده، چون به ازای هر درخواست نیازی نیست مجدداً محاسبه بشه.

نویسنده: سید مهران موسوی
تاریخ: ۱۳۹۳/۰۵/۰۲ ۱۲:۲۱

ممنون از توجهتون، بله من اشتباهات HttpContext.Items رو به کار برده بودم. کد موجود در مقاله اصلاح شد

نویسنده: حامد سبزیان
تاریخ: ۱۳۹۳/۰۵/۰۲ ۱۸:۴

بهبودی حاصل نشده. در [DefaultViewLocationCache](#) خود MVC مسیرها از HttpContext.Cache خوانده می‌شود، در کد شما هم از همان استفاده از HttpContext.Items در کد شما ممکن است اندکی بهینه بودن را افزایش دهد، به شرط استفاده بیش از یک بار از یک (چند) View در طول یک درخواست.

[Optimizing ASP.NET MVC view lookup performance](#)

همان طور که در انتهای مقاله اشاره شده است، استفاده از یک ConcurrentDictionary می‌تواند کارایی خوبی داشته باشد اما خوب استاتیک است و به حذف و اضافه شدن فیزیکی Viewها حساس نیست.

در خیلی مواقع ملاحظه میشود که برای نمایش تعدادی از رکوردهای یک جدول در پایگاه داده، کل مقادیر موجود در آن توسط یک دستور select به دست می آید و صفحه بندی خروجی، به کنترل های موجود سپرده میشود. اگر پایگاه داده ما دارای تعداد زیادی رکورد باشد، آن موقع است که دچار مشکل می شویم. فرض کنید به طور همزمان ۵ نفر (که تعداد زیادی نیستند) از برنامه ما که شامل ۱۰۰۰۰۰ سطر داده میباشد استفاده کنند و در هر صفحه، ۱۰ رکورد نمایش داده شود و صفحه بندی ما از نوع معقولی نباشد. در این صورت به جای اینکه با ۱۰×۵ رکورد داده را بارگزاری کنیم، ۱۰۰۰۰۰×۵ رکورد یعنی ۵۰۰۰۰۰ رکورد را برای به دست آوردن ۵۰ رکورد بارگزاری میکنیم. در زیر روشی شرح داده میشود که توسط آن، این سربار اضافه از روی برنامه و سرورهای مربوطه حذف شود. به stored procedure و توضیحات مربوط به آن توجه فرمایید :

```
CREATE PROCEDURE sp_PagedItems
(
    @Page int,
    @RecsPerPage int
)
AS

-- We don't want to return the # of rows inserted
-- into our temporary table, so turn NOCOUNT ON
SET NOCOUNT ON

--Create a temporary table
CREATE TABLE #TempItems
(
    ID int IDENTITY,
    Name varchar(50),
    Price currency
)

-- Insert the rows from tblItems into the temp. table
INSERT INTO #TempItems (Name, Price)
SELECT Name, Price FROM tblItem ORDER BY Price

-- Find out the first and last record we want
DECLARE @FirstRec int, @LastRec int
SELECT @FirstRec = (@Page - 1) * @RecsPerPage
SELECT @LastRec = (@Page * @RecsPerPage + 1)

-- Now, return the set of paged records, plus, an indication of we
-- have more records or not!
SELECT *,
MoreRecords =
(
    SELECT COUNT(*)
    FROM #TempItems TI
    WHERE TI.ID >= @LastRec
)
FROM #TempItems
WHERE ID > @FirstRec AND ID < @LastRec

-- Turn NOCOUNT back OFF
SET NOCOUNT OFF
```

در این کد دو پارامتر از نوع integer تعریف میکنیم. اول پارامتر @Page که مربوط به شماره صفحه ای می باشد که قصد دارید آن را بارگزاری نمایید. دومین پارامتر با نام @RecsPerPage تعداد رکوردهایی است که هر بار میخواهید بارگزاری شوند. مثلاً اگر میخواهید هر بار ۱۵ عدد از رکوردها را نمایش دهید، این مقدار را باید برابر ۱۵ قرار دهیم. در مرحله بعد یک جدول موقت با نام #TempItems ساخته شده است که به طور موقت مقادیری را در حافظه نگه میدارد. نکته کلیدی که جلوتر از آن استفاده شده، ستون با نام ID است که از نوع auto-increment بوده و روی جدول موقت تعریف شده است. این ستون شناسه هر سطر را در خود نگه میدارد که به صورت اتوماتیک بالا میرود و جزء لاینفکی از این نوع paging میباشد. پس از آن جدول موقت را توسط

رکوردهای جدول واقعی با نام tblItem توسط دستور select پر میکنیم.

در مرحله بعد شماره اولین و آخرین سطر مورد نظر را بر اساس پارامترهای ورودی محاسبه کرده و در متغیرهای @FirstRec و @LastRec می‌ریزیم.

برای استفاده از این کد فقط کافیست که پارامترهای ورودی را مقداردهی نمایید. مثلاً اگر میخواهید در یک کنترل Grid از آن استفاده کنید باید ابتدا یک کوئری داشته باشید که تعداد کل سطرها را به شما بدهد و بر اساس این مقدار تعداد صفحات مورد نظر را به دست آورید. پس از آن با کلیک روی هر کدام از شماره صفحات آن را به عنوان مقدار به پارامتر مورد نظر بفرستید و از آن لذت ببرید.

نظرات خوانندگان

نویسنده:

محسن خان

تاریخ:

۱۹:۵۹ ۱۳۹۳/۰۷/۲۲

ضمن تشکر از شما. یک اصلاح کوچک: جدول موقتی ایجاد شده در پایان کار رویه ذخیره شده باید drop بشه.

نویسنده:

محمد حسین عزتی

تاریخ:

۲۰:۱۸ ۱۳۹۳/۰۷/۲۲

از دقت شما به این نکته ظریف ممنونم

این موضوع در راستای آموزش عنوان مطلبش بود اما نکته شما جهت بالا بردن کیفیت و بهینه کردن کد مورد استفاده قرار میگیرد و عدم drop مشکلی در رسیدن به هدف مورد نظر ایجاد نمی کند
متشکرم

نویسنده:

امید

تاریخ:

۲۱:۲۰ ۱۳۹۳/۰۷/۲۲

در [sql 2012](#) به بعد جهت صفحه بندی دستورات offset و fetch اضافه شده که از لحاظ Performance بهینه تر از باقی روش های می باشد . [مقایسه صفحه بندی های مختلف](#)

نویسنده:

محمد حسین عزتی

تاریخ:

۲۱:۵۴ ۱۳۹۳/۰۷/۲۲

ممنونم بخاطر لینک مفیدی که در قسمت نظر ارسال نمودید

تنها نقطه ضعف این مقاله همینطور که خود شما هم متذکر شده اید این است که برای ورژن های بانک اطلاعاتی بعد از 2012 قابل استفاده است. هنوز بسیاری از نرم افزارها و سازمان های ما هنوز با ورژن های قدیمی تر کار می کنند.

متشکرم

نویسنده:

حمیدرضا کاظم نادی

تاریخ:

۱۱:۲۹ ۱۳۹۳/۰۷/۲۳

ممنون از مطلب خوبتون

به نظرم اگه جوری Sp را مینوشتید که یک ورودی متنی Query یا یک جدول موقت می گرفت و عمل Paging را روی اون انجام میداد مطلبتون بسیار کامل تر بود. مثلا ورودی Sp به این صورت بود که ('select * from Tb1_1',1,10) بازهم ممنون

نویسنده:

محمد حسین عزتی

تاریخ:

۱۲:۵۴ ۱۳۹۳/۰۷/۲۳

سلام

ممنونم از نظرتون

دوتا پارامتر داره از ورودی دریافت می کنه و هدف نحوه انجام صفحه بندی بوده

متشکرم

نویسنده:

رحمت اله رضایی

تاریخ:

۱۳:۳۶ ۱۳۹۳/۰۷/۲۳

روشی بسیار قدیمی است و این روزها آنچنان کاربرد ندارد.

برای صفحه بندی :

- در SQL Server 2008 از [ROW_NUMBER](#) استفاده می کنند.

- در SQL Server 2012 به بعد از [OFFSET FETCH](#) استفاده می کنند .

نویسنده:

محسن خان

تاریخ:

۱۳۹۳/۰۷/۲۳ ۱۴:۱۰

تاریخچه ای از روش های مختلف صفحه بندی اطلاعات در SQL Server در این مقاله بحث شده به همراه بررسی کارایی آن ها:

[Comparing performance for different SQL Server paging methods](#)

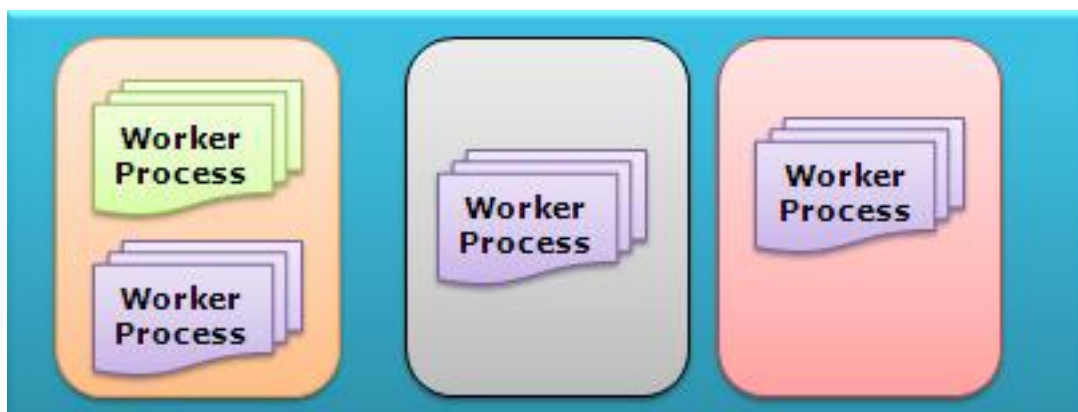
در قسمت قبلی گفتیم که IIS از تعدادی کامپوننت تشکیل شده است و به یکی از آن‌ها به نام Http.sys پرداختیم. در این قسمت قصد داریم به WWW Services بپردازیم. اجازه بدهید قبل از هر چیزی به دو مفهوم اصلی در IIS بپردازیم:

1. Worker Process

2. Application Pool

پرونده‌های کارگر w3wp.exe وظیفه‌ی اجرای برنامه‌های asp.net را در IIS، به عهده دارند. این پرونده‌ها مسئولیت پردازش تمامی درخواست و پاسخ‌ها از/به کلاینت را دارند. هر کاری که باید در asp.net انجام بشود، توسط این‌ها صورت می‌گیرد. به بیان ساده‌تر این پرونده‌ها قلب برنامه‌های ASP.Net بر روی IIS هستند.

Application Pool: این پول‌ها در واقع ظرفی یا در برگیرنده‌ای برای پرونده‌های کارگر به حساب می‌آیند. این پول‌ها پرونده‌های کارگر را از هم جدا و دسته‌بندی می‌کنند تا قابلیت اعتماد، امنیت و در دسترس بودن بدهند. موقعی که یک پرونده یا حتی یک پول دچار مشکل می‌شود، این اطمینان داده می‌شود که تاثیری بر دیگر پول‌ها یا پرونده‌های کارگر، ندارد. یعنی موقعی که یک web application دچار مشکل شود، هیچ تاثیری بر اجرای web application های دیگر ندارد. به یک application pool با چند پرونده کارگر web garden می‌گویند.



World Wide Web Publishing Services

یکی از قدیمی‌ترین امکانات موجود در IIS هست که از نسخه 7 به بعد، کار خود را با یک سرورس جدید به اسم Windows Process Activation Service یا به اختصار WAS که به صورت local system بر روی پرونده Svchost.exe با یک کد باینری یکسان اجرا می‌شود، شریک شده است. ممکن است در بعضی جاها WWW Service به صورت W3SVC هم نوشته شود.

اصلاً این WWW Service چه کاری انجام می‌دهد و به چه دردی می‌خورد؟

این سرویس در سه بخش مهم IIS 6 به فعالیت می‌پردازد:

HTTP administration and configuration

Performance monitoring

Process management

HTTP Administration and Configuration

سرویس WWW وظیفه خواندن اطلاعات پیکربندی IIS از متابیس را بر عهده دارد و از این اطلاعات خوانده شده برای پیکربندی و به روز کردن Http.sys استفاده می‌کند. به غیر از این کار، وظیفه آغاز و توقف و نظارت یا مانیتورینگ و همچنین مدیریت کامل پروسه‌های کارگر در زمینه http request را هم عهده دار است.

Performance Monitoring

سرویس WWW بر کارایی وب سایت‌ها و کش IIS نظارت می‌کند و البته یک شمارنده کارایی performance counter هم ایجاد می‌کند. کار شمارنده کارایی این است که اطلاعات یک سرویس یا سیستم عامل یا یک برنامه کاربردی را جمع آوری می‌کند تا به ما بگوید که این بخش‌ها به چه میزانی بهینه کار خود را انجام می‌دهند و به ما کمک می‌کنند که سیستم را به بهترین کارایی برسانیم. سیستم عامل، شبکه و درایورها، داده‌های شمارشی را تهیه و در قالب یک سیستم نظارتی گرافیکی به کارشناس سیستم یا شبکه نشان می‌دهند. برنامه نویسی‌ها هم از این طریق می‌توانند برنامه‌های خود را بنویسند که در [اینجا](#) لیستی از شمارنده‌ها در دانت نت را می‌توانید ببینید و بیشتر آن‌ها از طریق فضای نام system.diagnostics در دسترس هستند.

Process Management

سرویس www مدیریت application pool و پروسه‌های کارگر را هم به عهده دارد. این مدیریت شامل شروع و توقف و بازیابی پروسه‌های کارگر می‌شود. به علاوه اینکه این سرویس کار نظارت بر صحت انجام عملیات پروسه‌های کارگر را هم جز وظایف خود می‌داند. وقتی که چندین بار کار پروسه‌های کارگر در یک دوره زمانی که در فایل پیکربندی مشخص شده با مشکل مواجه شود، از شروع یک پروسه کارگر دیگر جلوگیری می‌کند.

در نسخه‌های جدیدتر IIS چکاری بر عهده WWW Service است؟

در IIS7 به بعد، دیگر مدیریت پروسه‌های کارگر را به عهده ندارد؛ ولی به جای آن سمتی جدید را به اسم listener adapter دریافت کرده است که یک listener adapter برای http listener یعنی Http.sys است. اصلی‌ترین وظیفه فعلی را که انجام می‌دهد پیکربندی Http.sys می‌باشد. موقعی که اطلاعات پیکربندی به روز می‌شوند باید این تغییرات بر روی Http.sys اعمال شوند. دومین وظیفه آن این است موقعی که درخواست جدیدی وارد صف درخواست‌ها می‌شود این مورد را به اطلاع WAS برساند. WAS در قسمت سوم این مقاله توضیح داده خواهد شد.

چندی قبل مطلبی را در مورد پیاده سازی سطح دوم کش در EF در این سایت [مطالعه کردید](#) . اساس آن مقاله‌ای بود که نحوه‌ی کش کردن اطلاعات حاصل از LINQ to Objects را بیان کرده بود ([^](#)). این مقاله پایه‌ی بسیاری از سیستم‌های کش مشابه نیز شده‌است ([^](#) و [^](#) و ...).

مشکل مهم این روش عدم سازگاری کامل آن با EF است. برای مثال در آن تفاوتی بین Include(x=>x.Tags) و Include(x=>x.Users) وجود ندارد. به همین جهت در این نوع موارد، قادر به تولید کلید منحصر بفردی جهت کش کردن اطلاعات یک کوئری مشخص نیست. در اینجا یک کوئری LINQ، به معادل رشته‌ای آن تبدیل می‌شود و سپس Hash آن محاسبه می‌گردد. این هش، کلید ذخیره سازی اطلاعات حاصل از کوئری، در سیستم کش خواهد بود. زمانیکه دو کوئری Include دار متفاوت EF، هش‌های یکسانی را تولید کنند، عملاً این سیستم کش، کارایی خودش را از دست می‌دهد. برای رفع این مشکل پروژه‌ی دیگری به نام [EF cache](#) ارائه شده‌است. این پروژه بسیار عالی طراحی شده و می‌تواند جهت ایده دادن به تیم EF نیز بکار رود. اما در آن فرض بر این است که شما می‌خواهید کل سیستم را در یک کش قرار دهید. وارد مکانیزم DBCommand و SqlDataReader می‌شود و در آن‌جا کار کش کردن تمام کوئری‌ها را انجام می‌دهد؛ مگر آنکه به آن اعلام کنید از کوئری‌های خاصی صرف‌نظر کند. با توجه به این مشکلات، روش بهتری برای تولید هش یک کوئری LINQ to Entities بر اساس کوئری واقعی SQL تولید شده توسط EF، پیش از ارسال آن به بانک اطلاعاتی به صورت زیر وجود دارد:

```
private static ObjectQuery TryGetObjectQuery<T>(IQueryable<T> source)
{
    var dbQuery = source as DbQuery<T>;
    if (dbQuery != null)
    {
        const BindingFlags privateFieldFlags =
            BindingFlags.NonPublic | BindingFlags.Instance | BindingFlags.Public;
        var internalQuery =
            source.GetType().GetProperty("InternalQuery", privateFieldFlags)
                .GetValue(source);
        return
            (ObjectQuery)internalQuery.GetType().GetProperty("ObjectQuery", privateFieldFlags)
                .GetValue(internalQuery);
    }
    return null;
}
```

این متد یک کوئری LINQ مخصوص EF را دریافت می‌کند و با کمک Reflection، اطلاعات درونی آن که شامل ObjectQuery اصلی است را استخراج می‌کند. سپس فراخوانی متد objectQuery.ToTraceString بر روی حاصل آن، سبب تولید SQL معادل کوئری LINQ اصلی می‌گردد. همچنین objectQuery امکان دسترسی به پارامترهای تنظیم شده‌ی کوئری را نیز میسر می‌کند. به این ترتیب می‌توان به معادل رشته‌ای منطقی‌تری از یک کوئری LINQ رسید که قابلیت تشخیص JOIN ها و متد Include نیز به صورت خودکار در آن لحاظ شده‌است.

این اطلاعات، پایه‌ی تهیه‌ی کتابخانه‌ی جدیدی به نام [EFSecondLevelCache](#) گردید. برای نصب آن کافی است دستور ذیل را در کنسول پاورشل نیوگت صادر کنید:

```
PM> Install-Package EFSecondLevelCache
```

سپس برای کش کردن کوئری معمولی مانند:

```
var products = context.Products.Include(x => x.Tags).FirstOrDefault();
```

می‌توان از متد جدید Cacheable آن به نحو ذیل استفاده کرد (این روش بسیار تمیزتر است از روش [مقاله‌ی قبلی](#) و امکان استفاده‌ی از انواع و اقسام متدهای EF را به صورت متداولی میسر می‌کند):

```
var products = context.Products.Include(x => x.Tags).Cacheable().FirstOrDefault(); // Async methods are supported too.
```

پس از آن نیاز است کدهای کلاس Context خود را نیز به نحو ذیل ویرایش کنید:

```
namespace EFSecondLevelCache.TestDataLayer.DataLayer
{
    public class SampleContext : DbContext
    {
        // public DbSet<Product> Products { get; set; }

        public SampleContext()
            : base("connectionString1")
        {
        }

        public override int SaveChanges()
        {
            return SaveAllChanges(invalidateCacheDependencies: true);
        }

        public int SaveAllChanges(bool invalidateCacheDependencies = true)
        {
            var changedEntityNames = getChangedEntityNames();
            var result = base.SaveChanges();
            if (invalidateCacheDependencies)
            {
                new EFCacheServiceProvider().InvalidateCacheDependencies(changedEntityNames);
            }
            return result;
        }

        private string[] getChangedEntityNames()
        {
            return this.ChangeTracker.Entries()
                .Where(x => x.State == EntityState.Added ||
                           x.State == EntityState.Modified ||
                           x.State == EntityState.Deleted)
                .Select(x => ObjectContext.GetObjectType(x.Entity.GetType()).FullName)
                .Distinct()
                .ToArray();
        }
    }
}
```

متد InvalidateCacheDependencies سبب می‌شود تا اگر تغییری در بانک اطلاعاتی رخداد، به صورت خودکار کش‌های کوئری‌های مرتبط غیر معتبر شوند و برنامه اطلاعات قدیمی را از کش نخواند.

کدهای کامل این پروژه را از مخزن کد ذیل می‌توانید دریافت کنید:

[EFSecondLevelCache](#)

پ.ن.

این کتابخانه هم اکنون در سایت جاری در حال استفاده است.

نظرات خوانندگان

نویسنده: اس حیدری
تاریخ: ۹:۹ ۱۳۹۳/۱۱/۰۷

برای داشتن دو یا چند Context و یا تغییر کانکشن Context می‌توان از این Cash استفاده کرد؟

چرا که کلید بر اساس معادل اسکول عبارت Linq ایجاد می‌شود

نویسنده: ایمان دارابی
تاریخ: ۹:۴۹ ۱۳۹۳/۱۱/۰۷

[این هم](#) کتابخانه خوبی هست. البته expire شدن کش را با استفاده از تگ هندل می‌کنه. خوبیش اینه بچ دلیت و آپدیت و امکانات دیگه هم داره. می‌شه از تگ به صورت اتوماتیک با روش شما ایجاد کرد و از کش همین کتابخانه استفاده کرد.

نویسنده: وحید نصیری
تاریخ: ۹:۵۵ ۱۳۹۳/۱۱/۰۷

رشته اتصالی هم در حین تولید کلید [در نظر گرفته شده‌است](#). همچنین در صورت نیاز یک عبارت دلخواه را که به آن در اینجا saltKey گفته می‌شود، می‌توانید به رشته‌ی نهایی که از آن کلید تولید می‌شود، اضافه کنید. برای اینکار پارامتر [EFCachePolicy](#) را مقدار دهی کنید.

نویسنده: وحید نصیری
تاریخ: ۱۰:۵ ۱۳۹۳/۱۱/۰۷

در انتهای سطر دوم مطلب، به این کتابخانه اشاره شده‌است. این مشکلات را دارد:

- چون از روش LINQ to Objects برای تهیه معادل رشته‌ای کوئری درخواستی استفاده می‌کند (دقیقا این روش: [^](#)) قادر نیست Include ها و جوین‌های EF را پردازش کند و در این حالت برای تمام جوین‌ها یک هش مساوی را در سیستم خواهید داشت.
- چون قادر نیست cache dependencies را از کوئری به صورت خودکار استخراج کند، شما نیاز خواهید داشت تا پارامتر تگ‌های آن‌را به صورت دستی به ازای هر کوئری تنظیم کنید. این کار [به صورت خودکار](#) در پروژه‌ی جاری انجام می‌شود. cache dependencies به این معنا است که کوئری جاری به چه موجودیت‌هایی در سیستم وابستگی دارد. از آن برای به روز رسانی کش استفاده می‌شود. برای مثال اگر یک کوئری به سه موجودیت وابستگی دارد، با تغییر یکی از آن‌ها، باید کش غیرمعتبر شده و در درخواست بعدی مجددا ساخته شود.

نویسنده: محمد عیدی مراد
تاریخ: ۱۰:۲۲ ۱۳۹۳/۱۱/۰۷

ظاهرا در حالت Lazy Loading زمانی که آبجکتی از کش لود میشه، پراپرتی‌های Navigation استثنای زیر را صادر میکنن:
TheObjectContext instance has been disposed and can no longer be used for operations that require a connection

تیکه کدی که این ارور رو بر میگرددونه:

```
var userInRoles = user.UserInRoles.Union(user.UsersSurrogate.Where(a => a.SurrogateFromDate != null && a.SurrogateToDate != null && a.SurrogateFromDate <= DateTime.Now && a.SurrogateToDate >= DateTime.Now).SelectMany(a => a.UserInRoles));
result = userInRoles.Any(a => a.Role.FormRoles.Any(b => b.IsActive && (b.Select && b.Form.SelectPath != null && b.Form.SelectPath.ToLower().Split(',').Contains(roleName))));
```

نویسنده: وحید نصیری
تاریخ: ۱۳۹۳/۱۱/۰۷ ۱۰:۴۴

Lazy loading با کش سازگاری ندارد؛ چون اتصال اشیاء موجود در کش از context قطع شده‌اند. در بار اول فراخوانی یک کوئری که قرار است کش شود، از context و دیتابیس استفاده می‌شود. اما در بارهای بعد دیگر به context و دیتابیس مراجعه نخواهد شد. تمام اطلاعات از کش سیستم بارگذاری می‌شوند و حتی یک کوئری اضافی نیز به بانک اطلاعاتی ارسال نخواهد شد. به همین جهت در این موارد باید از متد Include برای eager loading اشیایی که نیاز دارید استفاده کنید.

نویسنده: اس حیدری
تاریخ: ۱۳۹۳/۱۱/۰۷ ۱۱:۲۹

همچنین اتوماتیک بودن Cash به ازای کلیه Queryها هم می‌تواند یک آپشن در نظر گرفته شود و در مواردی که دسترسی به کوئری‌های داخلی نیست مفید واقع شود.

مثلا اگر برای اعتبار سنجی کاربر از Identity استفاده شود عملا نمی‌توان به کوئری‌های داخلی Identity دسترسی پیدا کرد و نیاز است که آن کوئری‌ها Cash شود، چرا که بسیار پرکاربرد می‌باشند.

نویسنده: وحید نصیری
تاریخ: ۱۳۹۳/۱۱/۰۷ ۱۱:۳۷

- کش سطح دوم نباید برای کش کردن اطلاعات خصوصی استفاده شود؛ یا کلا اطلاعاتی که نیاز به سطح دسترسی دارند. هدف آن کش کردن اطلاعات عمومی و پر مصرف است. اطلاعات خاص یک کاربر نباید کش شوند.
- در تمام سیستم‌ها، برای مواردی که به کوئری‌های آن دسترسی ندارید تا متد Cacheable را به آن‌ها اضافه کنید، نتیجه‌ی کوئری‌ها را باید خودتان از طریق [روش‌های متداول](#) کش کنید (مانند کلاس CacheManager مطلب یاد شده).

نویسنده: امین کاشانی
تاریخ: ۱۳۹۳/۱۲/۰۸ ۱:۲۵

من در کد زیر expiretime را 60 ثانیه گذاشتم. ولی در هربار فراخوانی در بازه زمانی 60 ثانیه از کش نمی‌خواند و دیتا از دیتابیس برمی‌گرداند. ایراد کار کجاست؟ ولی بدون نوشتن پارامتر زمان در متد cacheable کش درست عمل می‌کند.

```
public async Task<IList<Bestankaran>> GetBestankaran()
{
    EFCachePolicy expirationTime = new EFCachePolicy { AbsoluteExpiration = new
    DateTime().AddSeconds(60) };
    var result =
        Task.Run(() =>
            _bestankaran.Cacheable(expirationTime).ToListAsync());
    return await result;
}
```

نویسنده: وحید نصیری
تاریخ: ۱۳۹۳/۱۲/۰۸ ۱:۳۲

- زمانیکه از متد ToListAsync استفاده می‌کنید، نیازی به استفاده از Task.Run نیست. [اطلاعات بیشتر](#)
- بجای new DateTime باید از [DateTime.Now](#) استفاده کنید.

نویسنده: غلامرضا ربال
تاریخ: ۱۳۹۴/۰۵/۰۹ ۱۶:۴۷

با تشکر.

آیا این کتابخانه با کتابخانه EntityFramework.Extended سازگاری دارد؟
چون قصد دارم از این دو کنار هم استفاده کنم. یه کاری شبیه به کار زیر

```
public IList<string> GetUserPermissions(int[] roleIds, int userId)
{
    var permissionsOfRoles = (from p in _permissions
                              from r in p.ApplicationRoles
                              where roleIds.Contains(r.Id)
                              select p.Name).Cacheable().Future();

    var permissionsOfUser = (from p in _permissions
                              from r in p.AssignedUsers
                              where userId == r.Id
                              select p.Name).Cacheable().Future().ToList();
    return permissionsOfUser.Union(permissionsOfRoles).ToList();
}
```

ولی با خطای

The source query must be of type ObjectQuery or DbQuery.
Parameter name: source

[ArgumentException: The source query must be of type ObjectQuery or DbQuery.
Parameter name: source]
EntityFramework.Extensions.FutureExtensions.Future(IQueryable`1 source) +249

مواجه شدم. که مشخص است برای اعمال متد Future باید مبدا از نوع IQueryable باشد. آیا اعمال متد AsQueryable در روند کار کتابخانه EFSecondLevelCache مشکلی ایجاد نخواهد شد؟

نویسنده: وحید نصیری
تاریخ: ۱۷:۲ ۱۳۹۴/۰۵/۰۹

ترکیب Cacheable().Future() غیر ضروری است. چون این کوئری‌های Cacheable برای بار دوم به بعد، از کش و از حافظه خوانده می‌شوند و کاری به اطلاعات Context ندارند. بار اول اتصال به دیتابیس آن هم فقط یکبار انجام می‌شود و سر بار آنچنانی ندارد.

بعد از معرفی نسخه‌ی 2 از Asp.Net Web Api و پشتیبانی رسمی آن از OData بسیاری از توسعه دهندگان سیستم نفس راحتی کشیدند؛ زیرا از آن پس می‌توانستند علاوه بر امکانات جالب و مهمی که تحت پروتکل OData میسر بود، از سایر امکانات تعبیه شده در نسخه‌ی دوم web Api نیز استفاده نمایند. یکی از این قابلیت‌ها، مبحث مهم [Batching Processing](#) است که در طی این پست با آن آشنا خواهیم شد.

منظور از Batch Request این است که درخواست دهنده بتواند چندین درخواست (Multiple Http Request) را به صورت یک Pack جامع، در قالب فقط یک درخواست (Single Http Request) ارسال نماید و به همین روال تمام پاسخ‌های معادل درخواست ارسال شده را به صورت یک Pack دیگر دریافت کرده و آن را پردازش نماید. نوع درخواست نیز مهم نیست یعنی می‌توان در قالب یک Pack چندین درخواست از نوع Post و Get یا حتی Put و ... نیز داشته باشید. بدیهی است که پیاده سازی این قابلیت در جای مناسب و در پروژه‌هایی با تعداد کاربران زیاد می‌تواند باعث بهبود چشمگیر کارایی پروژه شود.

برای شروع همانند سایر مطالب می‌توانید از این [پست](#) جهت راه اندازی هاست سرویس‌های Web Api استفاده نمایید. برای فعال سازی قابلیت batching Request نیاز به یک MessageHandler داریم تا بتوانند درخواست‌هایی از این نوع را پردازش نمایند. خوشبختانه به صورت پیش فرض این Handler پیاده سازی شده‌است و ما فقط باید آن را با استفاده از متد MapHttpBatchRoute به بخش مسیر یابی (Route Handler) پروژه معرفی نماییم.

```
public class Startup
{
    public void Configuration(IAppBuilder appBuilder)
    {
        var config = new HttpConfiguration();

        config.Routes.MapHttpBatchRoute(
            routeName: "Batch",
            routeTemplate: "api/$batch",
            batchHandler: new DefaultHttpBatchHandler(GlobalConfiguration.DefaultServer));

        config.MapHttpAttributeRoutes();

        config.Routes.MapHttpRoute(
            name: "Default",
            routeTemplate: "{controller}/{action}/{name}",
            defaults: new { name = RouteParameter.Optional }
        );

        config.Formatters.Clear();
        config.Formatters.Add(new JsonMediaTypeFormatter());
        config.Formatters.JsonFormatter.SerializerSettings.Formatting =
Newtonsoft.Json.Formatting.Indented;
        config.Formatters.JsonFormatter.SerializerSettings.ContractResolver = new
CamelCasePropertyNamesContractResolver();

        config.EnsureInitialized();
        appBuilder.UseWebApi(config);
    }
}
```

مهم‌ترین نکته‌ی آن استفاده از DefaultHttpBatchHandler و معرفی آن به بخش batchHandler مسیریابی است. کلاس DefaultHttpBatchHandler برای وهله سازی نیاز به آبجکت سروری که سرویس‌های WebApi در آن هاست شده‌اند دارد که با دستور GlobalConfiguration.DefaultServer به آن دسترسی خواهید داشت. در صورتی که HttpServer خاص خود را دارید به صورت زیر عمل نمایید:

```
var config = new HttpConfiguration();
HttpServer server = new HttpServer(config);
```

تنظیمات بخش سرور به اتمام رسید. حال نیاز داریم بخش کلاینت را طوری طراحی نماییم که بتواند درخواست را به صورت دسته‌ای ارسال نماید. در زیر یک مثال قرار داده شده است:

```
using System.Net.Http;
using System.Net.Http.Formatting;

public class Program
{
    private static void Main(string[] args)
    {
        string baseAddress = "http://localhost:8080";
        var client = new HttpClient();
        var batchRequest = new HttpRequestMessage(HttpMethod.Post, baseAddress + "/api/$batch")
        {
            Content = new MultipartContent("mixed")
            {
                new HttpResponseMessage(new HttpRequestMessage(HttpMethod.Post, baseAddress +
"/api/Book/Add")
                {
                    Content = new ObjectContent<string>("myBook", new JsonMediaTypeFormatter())
                }),
                new HttpResponseMessage(new HttpRequestMessage(HttpMethod.Get, baseAddress +
"/api/Book/GetAll"))
            };
        };

        var batchResponse = client.SendAsync(batchRequest).Result;

        MultipartStreamProvider streamProvider =
batchResponse.Content.ReadAsMultipartAsync().Result;
        foreach (var content in streamProvider.Contents)
        {
            var response = content.ReadAsHttpResponseMessageAsync().Result;
        }
    }
}
```

همان طور که می‌دانیم برای ارسال درخواست به سرویس Web Api باید یک نمونه از کلاس `HttpRequestMessage` و هله سازی شود سازنده‌ی آن به نوع `HttpMethod` اکشن نظیر (`POST` یا `GET`) و آدرس سرویس مورد نظر نیاز دارد. نکته‌ی مهم آن این است که خاصیت `Content` این درخواست باید از نوع `MultipartContent` و `subType` آن نیز باید `mixed` باشد. در بدنه‌ی آن نیز می‌توان تمام درخواست‌ها را به ترتیب و با استفاده از و هله سازی از کلاس `HttpMessageContent` تعریف کرد. برای دریافت پاسخ این گونه درخواست‌ها نیز از متد الحاقی `ReadAsMultipartAsync` استفاده می‌شود که امکان پیمایش بر بدنه‌ی پیام دریافتی را می‌دهد.

مدیریت ترتیب درخواست ها

شاید این سوال به ذهن شما نیز خطور کرده باشد که ترتیب پردازش این گونه پیام‌ها چگونه خواهد بود؟ به صورت پیش فرض ترتیب اجرای درخواست‌ها حائز اهمیت است. یعنی تا زمانیکه پردازش درخواست اول به اتمام نرسد، کنترل اجرای برنامه، به درخواست بعدی نخواهد رسید که این مورد بیشتر زمانی رخ می‌دهد که قصد دریافت اطلاعاتی را داشته باشید که قبل از آن باید عمل `Persist` در پایگاه داده اتفاق بیافتد. اما در حالاتی غیر از این می‌توانید این گزینه را غیر فعال کرده تا تمام درخواست‌ها به صورت موازی پردازش شوند که به طور قطع کارایی آن نسبت به حالت قبلی بهینه‌تر است. برای غیر فعال کردن گزینه‌ی ترتیب اجرای درخواست‌ها، به صورت زیر عمل نمایید:

```
config.Routes.MapHttpBatchRoute(
    routeName: "WebApiBatch",
    routeTemplate: "api/$batch",
    batchHandler: new DefaultHttpBatchHandler(GlobalConfiguration.DefaultServer)
    {
        ExecutionOrder = BatchExecutionOrder.NonSequential
    });
```

تفاوت آن فقط در مقدار دهی خاصیت `ExecutionOrder` به صورت `NonSequential` است.

رشته، مجموعه‌ای از کاراکترهاست که پشت سرهم، در مکانی از حافظه قرار گرفته‌اند. هر کاراکتر حاوی یک شماره سریال در جدول [یونیکد](#) هست. به طور پیش فرض دات نت برای هر کاراکتر (نوع داده char) شانزده بیت در نظر گرفته است که برای 65536 کاراکتر کافی است.

برای نگهداری از رشته‌ها و انجام عملیات بر روی آنها در دات نت از نوع system.string استفاده می‌کنیم:

```
string greeting = "Hello, C#";
```

که در این حالت مجموعه‌ای از کاراکترها را ایجاد خواهد کرد:

H	e	l	l	o	,		C	#
---	---	---	---	---	---	--	---	---

اتفاقاتی که در داخل کلاس string رخ می‌دهد بسیار ساده است و ما را از تعریف char[] بی‌نیاز می‌کند تا مجبور نشویم خانه‌های آرایه را به ترتیب پر کنیم. از معایب استفاده از آرایه char میتوان موارد زیر را برشمرد: خانه‌های آن یک ضرب پر نمیشوند بلکه به ترتیب، خانه به خانه پر می‌شوند. قبل از انتساب متن باید از طول متن مطمئن شویم تا بتوانیم تعداد خانه‌ها را بر اساس آن ایجاد کنیم. همه عملیات آرایه‌ها از پر کردن ابتدای کار گرفته تا هر عملی، نیاز است به صورت دستی صورت بگیرد و تعداد خطوط کد برای هر کاری هم بالا می‌رود.

البته استفاده از string هم راه حل نهایی برای کار با متون نیست. در انتهای این مطلب مورد دیگری را نیز بررسی خواهیم کرد. از ویژگی دیگر رشته‌ها این است که آن‌ها شباهت زیادی به آرایه‌ای از کاراکترها دارند؛ ولی اصلاً شبیه آن‌ها نیستند و نمی‌توانید به صورت یک آرایه آن‌ها را مقداردهی کنید. البته کلاس string امکاناتی را با استفاده از indexer [] مهیا کرده است که می‌توانید بر اساس اندیس‌ها به کاراکترها به صورت جداگانه دسترسی داشته باشید ولی نمی‌توانید آن‌ها را مقدار دهی کنید. این اندیس‌ها از 0 تا طول آن length-1 ادامه دارند.

```
string str = "abcde";
char ch = str[1]; // ch == 'b'
str[1] = 'a'; // Compilation error!
ch = str[50]; // IndexOutOfRangeException
```

همانطور که میدانیم برای مقداردهی رشته‌ها از علامت‌های نقل قول "" استفاده میکنیم که باعث میشود اگر بخواهیم علامت " را در رشته‌ها داشته باشیم نتوانیم. برای حل این مشکل از علامت \ استفاده میکنیم که البته باعث استفاده از بعضی کاراکترهای خاص دیگر هم می‌شود:

```
string a="Hello \"C#\"";
string b="Hello \r\n C#"; // مساوی با اینتر
string c="C:\\a.jpg"; // چاپ خود علامت \ -مسیردهی
```

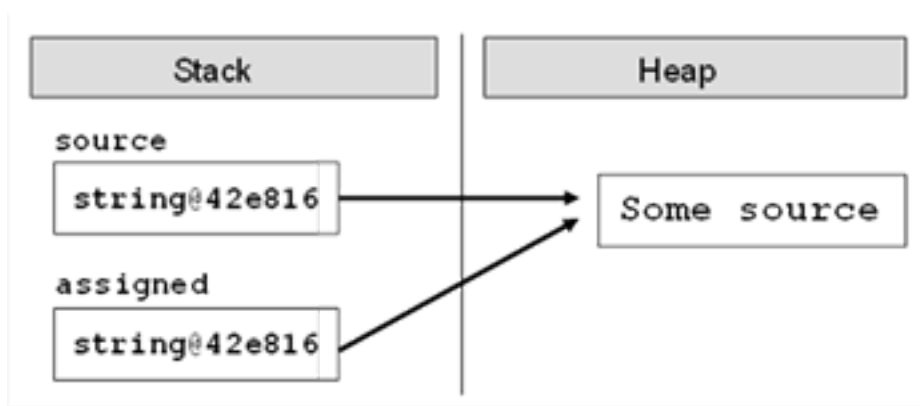
البته اگر از علامت @ در قبل از رشته استفاده شود علامت \ بی اثر خواهد شد.

```
string c=@"C:\a.jpg"; // == "C:\a.jpg"
```

مقداردهی رشته‌ها و پایدار (تغییر ناپذیر) بودن آنها Immutable

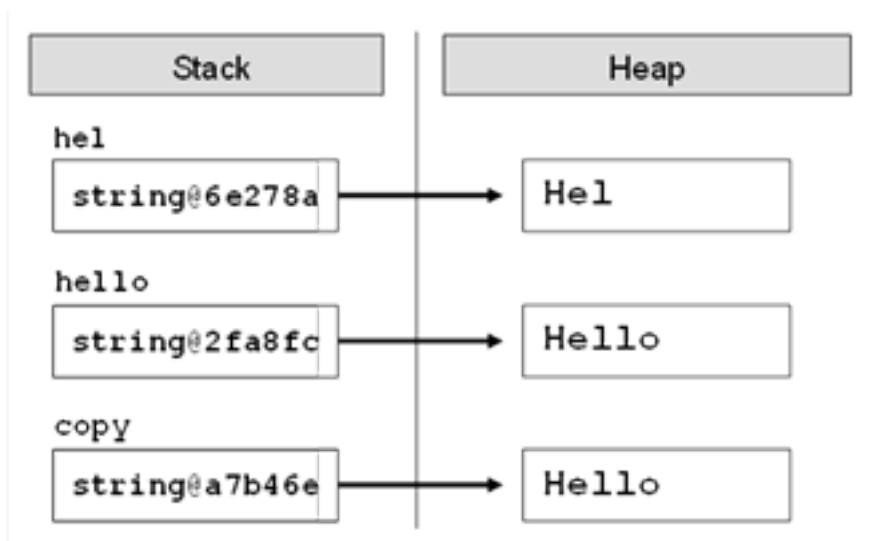
رشته‌ها ساختاری پایدار هستند؛ به این معنی که به صورت reference مقداردهی می‌شوند. موقعی که شما مقداری را به یک رشته انتساب می‌دهید، مقدار متغیر در String pool یا [لینک](#) در Heap ذخیره می‌شوند و اگر همین متغیر را به یک متغیر دیگر انتساب دهیم، متغیر جدید مقدار آن را دیگر در حافظه پویا (داینامیک) Heap به عنوان مقدار جدید ذخیره نخواهد کرد؛ بلکه تنها یک pointer خواهد بود که به آدرس حافظه متغیر اولی اشاره می‌کند. به مثال زیر دقت کنید. متغیر source مقدار some source را ذخیره می‌کند و بعد همین متغیر، به متغیر assigned انتساب داده می‌شود؛ ولی مقداری جابجا نمی‌شود. بلکه متغیر assign به آدرسی در حافظه اشاره می‌کند که متغیر source اشاره می‌کند. هرگاه که در یکی از متغیرها، تغییری رخ دهد، همان متغیری که تغییر کرده است، به آدرس جدید با محتوای تغییر داده شده اشاره می‌کند.

```
string source = "Some source";
string assigned = source;
```

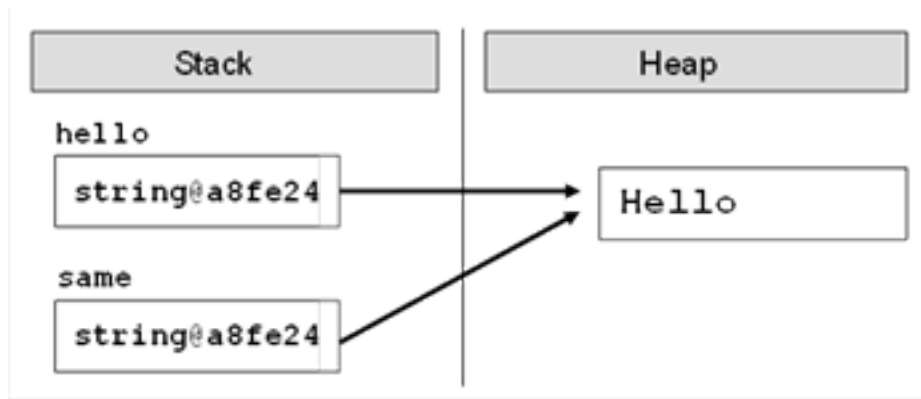


این ویژگی نوع reference فقط برای ساختارهای Immutable به معنی پایدار رخ می‌دهد و نه برای ساختارهای ناپایدار (تغییر پذیر) mutable؛ به این خاطر که آن‌ها مقادیرشان را مستقیماً تغییر می‌دهند و اشاره‌ای در حافظه صورت نمی‌گیرد.

```
string hel = "Hel";
string hello = "Hello";
string copy = hel + "lo";
```



```
string hello = "Hello";
string same = "Hello";
```



برای اطلاعات بیشتر در این زمینه این [لینک](#) را مطالعه نمایید.

مقایسه رشته‌ها

برای مقایسه دو رشته می‌توان از علامت == یا از متد Equals استفاده نماییم. در این حالت به خاطر اینکه کد حروف کوچک و بزرگ متفاوت است، مقایسه حروف هم متفاوت خواهد بود. برای اینکه حروف کوچک و بزرگ تاثیری بر مقایسه ما نگذارند و C# با برابر بدانند باید از متد Equals به شکل زیر استفاده کنیم:

```
Console.WriteLine(word1.Equals(word2,
    StringComparison.CurrentCultureIgnoreCase));
```

برای اینکه بزرگی و کوچکی اعداد را مشخص کنیم از علامت‌های < و > استفاده می‌کنیم ولی برای رشته‌ها از متد CompareTo بهره می‌بریم که چینش قرارگیری آن‌ها را بر اساس حروف الفبا مقایسه می‌کند و سه عدد، می‌تواند خروجی آن باشند. اگر 0 باشد یعنی برابر هستند، اگر 1- باشد رشته اولی قبل از رشته دومی است و اگر 1 باشد رشته دومی قبل از رشته اولی است.

```
string score = "sCore";
string scary = "scary";

Console.WriteLine(score.CompareTo(scary));
Console.WriteLine(scary.CompareTo(score));
Console.WriteLine(scary.CompareTo(scary));

// Console output:
// 1
// -1
// 0
```

اینبار هم برای اینکه حروف کوچک و بزرگ، دخالتی در کار نداشته باشند، می‌توانید از داده شمارشی StringComparison در متد ایستای string.Compare(s1,s2,StringComparison) استفاده نمایید؛ یا از نوع داده‌ای boolean برای تعیین نوع مقایسه استفاده کنید.

```
string alpha = "alpha";
string score1 = "sCorE";
string score2 = "score";

Console.WriteLine(string.Compare(alpha, score1, false));
Console.WriteLine(string.Compare(score1, score2, false));
Console.WriteLine(string.Compare(score1, score2, true));
```

```
Console.WriteLine(string.Compare(score1, score2,
    StringComparison.CurrentCultureIgnoreCase));
// Console output:
// -1
// 1
// 0
// 0
```

نکته : برای مقایسه برابری دو رشته از متد Equals یا == استفاده کنید و فقط برای تعیین کوچک یا بزرگ بودن از compare استفاده نمایید. دلیل آن هم این است که برای مقایسه از فرهنگ culture فعلی سیستم استفاده میشود و نظم جدول یونیکد را رعایت نمی کنند و ممکن است بعضی رشته های نابرابر با یکدیگر برابر باشند. برای مثال در زبان آلمانی دو رشته "SS" و "ß" با یکدیگر برابر هستند.

عبارات با قاعده Regular Expression

این عبارات الگوهایی هستند که قرار است عبارات مشابه الگویی را در رشته ها پیدا کنند. برای مثال الگوی [A-Z0-9]+ مشخص می کند که رشته مورد نظر نباید خالی باشد و حداقل با یکی از حروف بزرگ یا اعداد پر شده باشد. این الگوها میتوانند برای واکنشی داده ها یا قالب های خاص در رشته ها به کار بروند. برای مثال شماره تماس ها، [پست الکترونیکی](#) و ... در [اینجا](#) میتواند نحوه ی الگوسازی را بیاموزید. کد زیر بر اساس یک الگو، شماره تماس های مورد نظر را یافته و البته با فیلتر گذاری آن ها را نمایش می دهد:

```
string doc = "Smith's number: 0898880022\nFranky can be " +
    "found at 0888445566.\nSteven's mobile number: 0887654321";
string replacedDoc = Regex.Replace(
    doc, "(08)[0-9]{8}", "$1*****");
Console.WriteLine(replacedDoc);
// Console output:
// Smith's number: 08*****
// Franky can be found at 08*****.
// Steven' mobile number: 08*****
```

سه شماره تماس در رشته ی بالا با الگوی ما همخوانی دارند که بعد با استفاده از متد replace در شی Regex عبارات دلخواه خودمان را جایگزین شماره تماس ها خواهیم کرد. الگوی بالا شماره تماس هایی را میابد که با 08 آغاز شده اند و بعد از آن 8 عدد دیگر از 0 تا 9 قرار گرفته اند. بعد از اینکه متن مطابق الگو یافت شد، ما آن را با الگوی \$1***** جایگزین می کنیم که علامت \$ یک placeholder برای یک گروه است. هر عبارت () در عبارات با قاعده یک گروه حساب میشود و اولین پرانتز \$1 و دومین پرانتز یا گروه میشود \$2 که در عبارت بالا (08) میشود \$1 و به جای مابقی الگو، 8 علامت ستاره نمایش داده میشود.

اتصال رشته ها در Loop

برای اتصال رشته ها ما از علامت + یا متد ایستای string.concat استفاده می کنیم ولی استفاده ی از آن در داخل یک حلقه باعث کاهش کارایی برنامه خواهد شد. برای همین بیاید ببینم در حین انتقال رشته ها در حافظه چه اتفاقی رخ میدهد. ما در اینجا دو رشته str1 و str2 داریم که عبارات "super" و "star" را نگه داری می کنند و در واقع دو متغیر هستند که به حافظه ی پویای Heap اشاره می کنند. اگر این دو را با هم جمع کنیم و نتیجه را در متغیر result قرار دهیم، سه متغیر میشوند که هر کدام به حافظه ای جداگانه در heap اشاره می کنند. در واقع برای این اتصال، قسمت جدیدی از حافظه تخصیص داده شده و مقدار جدید در آن نشسته است. در این حالت یک متغیر جدید ساخته شد که به آدرس آن اشاره می کند. کل این فرآیند یک فرآیند کاملاً زمانبر است که با تکرار این عمل موجب از دست دادن کارایی برنامه می شود؛ به خصوص اگر در یک حلقه این کار صورت بگیرد. سیستم دات نت همانطور که میدانید شامل [GC](#) یا سیستم خودکار پاکسازی حافظه است که برنامه نویس را از dispose کردن بسیاری از اشیاء بی نیاز می کند. موقعی که متغیری به قسمتی از حافظه اشاره می کند که دیگر بلا استفاده است، سیستم GC به صورت خودکار آنها را پاکسازی می کند که این عمل زمان بر هم خودش موجب کاهش کارایی می شود. همچنین انتقال رشته ها از یک مکان حافظه به مکانی دیگر، باز خودش یک فرآیند زمانبر است؛ به خصوص اگر رشته مورد نظر طولانی هم باشد. **مثال عملی:** در تکه کد زیر قصد داریم اعداد 1 تا 20000 را در یک رشته الحاق کنیم:

```
DateTime dt = DateTime.Now;
string s = "";
for (int index = 1; index <= 20000; index++)
{
    s += index.ToString();
}
Console.WriteLine(s);
```

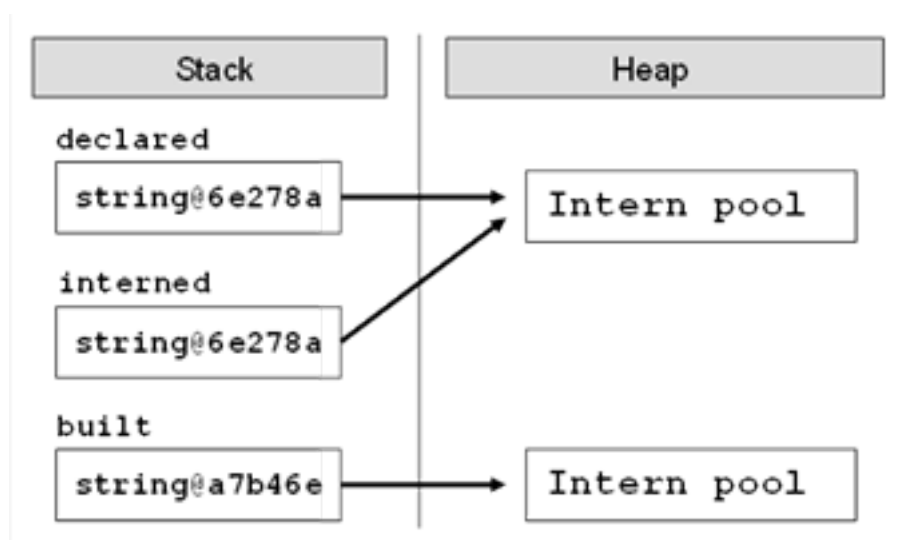
```
Console.WriteLine(dt);
Console.WriteLine(DateTime.Now);
Console.ReadKey();
```

کد بالا تا زمان نمایش کامل، بسته به قدرت سیستم ممکن است یکی دو ثانیه طول بکشد. حالا عدد را به 200000 تغییر دهید (یک صفر اضافه تر). برنامه را اجرا کنید و مجدداً تست بزنید. در این حالت چند دقیقه ای بسته به قدرت سیستم زمان خواهد برد؛ مثلاً دو دقیقه یا سه دقیقه یا کمتر و بیشتر. عملیاتی که در حافظه صورت میگیرد این چند گام را طی میکند: قسمتی از حافظه به طور موقت برای این دور جدید حلقه، گرفته میشود که به آن بافر میگوییم. رشته قبلی به بافر انتقال میابد که بسته به مقدار آن زمان بر و کند است؛ 5 کیلو یا 5 مگابایت یا 50 مگابایت و ... شماره تولید شده جدید به بافر چسبانده میشود. بافر به یک رشته تبدیل میشود و جایی برای خود در حافظه Heap میگیرد. حافظه رشته قدیمی و بافر دیگر بلا استفاده شده اند و توسط GC پاکسازی میشوند که ممکن است عملیاتی زمان بر باشد.

String Builder

این کلاس ناپایدار و تغییر پذیر است. به کد و شکل زیر دقت کنید:

```
string declared = "Intern pool";
string built = new StringBuilder("Intern pool").ToString();
```



این کلاس دیگر مشکل الحاق رشته ها یا دیگر عملیات پردازشی را ندارد. بیایید مثال قبل را برای این کلاس هم بررسی نماییم:

```
StringBuilder sb = new StringBuilder();
sb.Append("Numbers: ");

DateTime dt = DateTime.Now;
for (int index = 1; index <= 200000; index++)
{
    sb.Append(index);
}
Console.WriteLine(sb.ToString());
Console.WriteLine(dt);
Console.WriteLine(DateTime.Now);
Console.ReadKey();
```

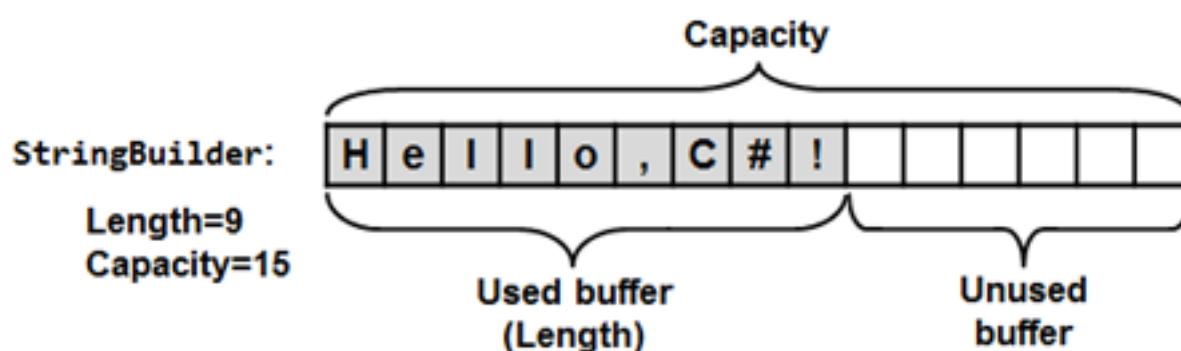

اکنون همین عملیات چند دقیقه‌ای قبل، در زمانی کمتر، مثلاً دو ثانیه انجام میشود. حال این سوال پیش می‌آید مگر کلاس *stringbuilder* چه میکند که زمان پردازش آن قدر کوتاه است؟

همانطور که گفتیم این کلاس *mutable* یا تغییر پذیر است و برای انجام عملیات‌های ویرایشی نیازی به ایجاد شیء جدید در حافظه ندارد؛ در نتیجه باعث کاهش انتقال غیرضروری داده‌ها برای عملیات پایه‌ای چون الحاق رشته‌ها میگردد.

stringbuilder شامل یک بافر با ظرفیتی مشخص است (به طور پیش فرض 16 کاراکتر). این کلاس آرایه‌هایی از کاراکترها را پیاده سازی میکند که برای عملیات و پردازش‌هایش از یک رابط کاربرپسند برای برنامه نویسان استفاده می‌کند. اگر تعداد کاراکترها کمتر از 16 باشد مثلاً 5، فقط 5 خانه آرایه استفاده میشود و مابقی خانه‌ها خالی میماند و با اضافه شدن یک کاراکتر جدید، دیگر شیء جدیدی در حافظه درست نمی‌شود؛ بلکه در خانه ششم قرار می‌گیرد و اگر تعداد کاراکترهایی که اضافه می‌شوند باعث شود از 16 کاراکتر رد شود، مقدار خانه‌ها دو برابر میشوند؛ هر چند این عملیات دو برابر شدن *resizing* عملیاتی کند است ولی این اتفاق به ندرت رخ می‌دهد.

کد زیر یک آرایه 15 کاراکتری ایجاد می‌کند و عبارت *Hello C#* را در آن قرار می‌دهد.

```
StringBuilder sb = new StringBuilder(15);
sb.Append("Hello, C#!");
```



در شکل بالا خانه‌هایی خالی مانده است *Unused* و جا برای کاراکترهای جدید به اندازه خانه‌های *unused* هست و اگر بیشتر شود همانطور که گفتیم تعداد خانه‌ها 2 برابر می‌شوند که در اینجا میشود 30.

استفاده از متد ایستای *string.Format*

از این متد برای نوشتن یک متن به صورت قالب و سپس جایگزینی مقادیر استفاده می‌شود:

```
DateTime date = DateTime.Now;
string name = "David Scott";
string task = "Introduction to C# book";
string location = "his office";

string formattedText = String.Format(
    "Today is {0:MM/dd/yyyy} and {1} is working on {2} in {3}.",
    date, name, task, location);
Console.WriteLine(formattedText);
```

در کد بالا ابتدا ساختار قرار گرفتن تاریخ را بر اساس الگو بین *{}* مشخص می‌کنیم و متغیر *date* در آن قرار می‌گیرد و سپس برای *{1}*, *{2}*, *{3}* به ترتیب قرار گیری آن‌ها متغیرهای *name*, *last*, *location* قرار می‌گیرند. از *ToString()* هم می‌توان برای فرمت بندی خروجی استفاده کرد؛ مثل همین عبارت *MM/dd/yyyy* در خروجی نوع داده تاریخ و زمان.

نظرات خوانندگان

نویسنده: شهرز جعفری
تاریخ: ۱۸:۱۰ ۱۳۹۳/۱۱/۲۹

یک سوال منظور از Gac اینجا چیه؟

نویسنده: علی یگانه مقدم
تاریخ: ۱۸:۴۸ ۱۳۹۳/۱۱/۲۹

ممنون که گوشزد کردید؛ عذر میخوام. مطلب ویرایش شد. منظور GC بود. بنده اشتباهها نوشتم GAC.

هدف از توابع خطی (Inline)

استفاده از توابع، مقداری بر زمان اجرای برنامه می‌افزاید؛ هرچند که این زمان بسیار کم و در حد میلی ثانیه است، اما باری را بر روی برنامه قرار می‌دهد و علت این تاخیر زمانی این است که در فراخوانی و اعلان توابع، کامپایلر یک کپی از تابع مورد نظر را در حافظه قرار می‌دهد و در فراخوانی تابع، به آدرس مذکور مراجعه می‌کند و در عین حال آدرس موقعیت توقف دستورات در تابع main را نیز ذخیره می‌کند تا پس از پایان تابع، به آدرس قبل برگردد و ادامه‌ی دستورات را اجرا کند. در نتیجه این آدرس‌دهی‌ها و نقل و انتقالات بین آنها بار زمانی را در برنامه ایجاد می‌کند که در صورت زیاد بودن توابع در برنامه و تعداد فراخوانی‌های لازم، زمان قابل توجهی خواهد شد.

یکی از تکنیک‌های بهینه که برای کاهش زمان اجرای برنامه توسط کامپایلرها استفاده می‌شود استفاده از توابع خطی (inline) است. این امکان در زبان C با عنوان توابع ماکرو (Macro function) و در C++ با عنوان توابع خطی (inline function) وجود دارد. در واقع توابع خطی به کامپایلر پیشنهاد می‌دهند، زمانی که سربار فراخوانی تابع بیشتر از سربار بدنه خود متد باشد، برای کاهش هزینه و زمان اجرای برنامه از تابع به صورت خطی استفاده کند و یک کپی از بنده‌ی تابع را در قسمتی که تابع ما فراخوانی شده است، قرار دهد که مورد آدرس‌دهی از میان خواهد رفت!

نمونه ای از پیاده سازی این تکنیک در زبان C++ :

```
inline type name(parameters)
{
    ...
}
```

بررسی متدهای خطی در سی شارپ به مثال زیر توجه کنید:

قسمت‌های getter و setter مربوط به پراپرتی‌ها سربار اضافی بر کلاس Vector می‌افزایند. این موضوع شاید آنچنان مسئله‌ی مهمی نباشد. ولی فرض کنید این پراپرتی‌ها به شکل زیر داخل حلقه‌ای طولانی قرار گیرند. اگر با استفاده از یک پروفایلر زمان اجرای برنامه را زیر نظر بگیرید، خواهید دید که بیش از 90 درصد آن صرف فراخوانی‌های متدهای بخش‌های get , set پراپرتی‌ها است. برای این منظور باید مطمئن شویم که فراخوانی این متدها، به صورت خطی صورت می‌گیرد!

```
public class Vector
{
    public double X { get; set; }
    public double Y { get; set; }
    public double Z { get; set; }

    // ...
}
```

برای این منظور آزمایشی را انجام می‌دهیم. فرض کنید کلاسی را به شکل زیر داشته باشیم:

```
public class MyClass
{
    public int A { get; set; }
    public int C;
}
```

و برای استفاده از آن به شکل زیر عمل کنیم:

```
static void Main()
{
    MyClass target = new MyClass();
    int a = target.A;
    Console.WriteLine("A = {0}", a);
    int c = target.C;
```

```
Console.WriteLine("C = {0}", c);
}
```

بعد از دیباگ برنامه و مشاهده‌ی کدهای ماشین مربوط به آن خواهیم دید که متد مربوط به getter پراپرتی A به صورت خطی فراخوانی نشده است:

```
int a = target.A;
0000003e mov     ecx,edi
00000040 cmp     dword ptr [ecx],ecx
00000042 call    dword ptr ds:[05FA29A8h]
00000048 mov     esi,eax
0000004a mov     dword ptr [esp+4],esi
        int c = target.C;
00000098 mov     edi,dword ptr [edi+4]
MyClass.get_A() looks like this:
00000000 push    esi
00000001 mov     esi,ecx
00000003 cmp     dword ptr ds:[03B701DCh],0
0000000a je      00000011
0000000c call    76BA6BA7
00000011 mov     eax,dword ptr [esi+0Ch]
00000014 pop     esi
00000015 ret
```

چه اتفاقی افتاده است؟

کامپایلر سی شارپ در زمان کامپایل، کدهای برنامه را به کدهای IL تبدیل می‌کند و JIT کامپایلر، این کدهای IL را گرفته و کد ساده‌ی ماشین را تولید می‌کند. لذا به دلیل اینکه JIT با معماری پردازنده آشنایی کافی دارد، مسئولیت تصمیم‌گیری اینکه کدام متد به صورت خطی فراخوانی شود برعهده‌ی آن است. در واقع این JIT است که تشخیص می‌دهد که آیا فراخوانی متد به صورت خطی مناسب است یا نه و به صورت یک معاوضه کار بین خط لوله دستورالعمل‌ها و کش است. اگر شما برنامه‌ی خود را با (F5) و همگام با دیباگ اجرا کنید، تمام بهینه‌سازی‌های JIT که Inline Method هم یکی از آنهاست، از کار خواهند افتاد. برای مشاهده‌ی کد بهینه شده باید با بدون دیباگ (CTRL+F5) برنامه خود را اجرا کنید که در آن صورت مشاهده خواهید کرد، متد getter مربوط به پراپرتی A به صورت خطی استفاده شده است.

```
int a = target.A;
00000024 mov     ebx,dword ptr [edi+0Ch]
```

JIT محدودیت‌هایی برای فراخوانی به صورت خطی متدها دارد :

متدهایی که حجم کد IL آنها بیشتر از 32 بایت است.
متدهای بازگشتی.

متدهایی که با اتریبیوتMethodImpl علامتگذاری شدند و MethodImplOptions.NoInlining اعمال شده بر آن
متدهای virtual

متدهایی که دارای کد مدیریت خطا هستند

Methods that take a large value type as a parameter

Methods with complicated flowgraphs

برای اینکه در سی شارپ به کامپایلر اعلام کنیم تا متد مورد نظر به صورت خطی مورد استفاده قرار گیرد، در دات نت 4.5 توسط اتریبیوتMethodImpl و اعمال MethodImplOptions.AggressiveInlining که یک نوع شمارشی است می‌توان این کار را انجام داد. مثال:

```
using System;
using System.Diagnostics;
using System.Runtime.CompilerServices;
class Program
{
    const int _max = 10000000;
    static void Main()
```

```

{
    // ... Compile the methods.
    Method1();
    Method2();
    int sum = 0;
    var s1 = Stopwatch.StartNew();
    for (int i = 0; i < _max; i++)
    {
        sum += Method1();
    }
    s1.Stop();
    var s2 = Stopwatch.StartNew();
    for (int i = 0; i < _max; i++)
    {
        sum += Method2();
    }
    s2.Stop();
    Console.WriteLine(((double)(s1.Elapsed.TotalMilliseconds * 1000000) /
_max).ToString("0.00 ns"));
    Console.WriteLine(((double)(s2.Elapsed.TotalMilliseconds * 1000000) /
_max).ToString("0.00 ns"));
    Console.Read();
}
static int Method1()
{
    // ... No inlining suggestion.
    return "one".Length + "two".Length + "three".Length +
        "four".Length + "five".Length + "six".Length +
        "seven".Length + "eight".Length + "nine".Length +
        "ten".Length;
}
[MethodImpl(MethodImplOptions.AggressiveInlining)]
static int Method2()
{
    // ... Aggressive inlining.
    return "one".Length + "two".Length + "three".Length +
        "four".Length + "five".Length + "six".Length +
        "seven".Length + "eight".Length + "nine".Length +
        "ten".Length;
}
}
Output
7.34 ns      No options
0.32 ns      MethodImplOptions.AggressiveInlining

```

در واقع با اعمال این اتریبیوت، محدودیت شماره یک مبنی بر محدودیت حجم کد IL مربوط به متد، در نظر گرفته نخواهد شد.

مطالعه بیشتر: <http://dotnet.dzone.com/news/aggressive-inlining-clr-45-jit>

<http://www.ademiller.com/blogs/tech/2008/08/c-inline-methods-and-optimization>

<http://www.dotnetperls.com/aggressiveinlining>

<http://blogs.msdn.com/b/ericgu/archive/2004/01/29/64644.aspx>

https://msdn.microsoft.com/en-us/library/ms973858.aspx#highperfmanagedapps_topic10

جهت « [بهبود کارایی کنترل‌های لیستی WPF در حین بارگذاری تعداد زیادی از رکوردها](#) » توصیه شده‌است که مجازی سازی UI فعال گردد. به این ترتیب بجای تولید یکباره‌ی برای مثال 1000 ردیف، تنها 10 ردیفی که نمایان هستند تولید می‌شوند. بنابراین مصرف حافظه و سرعت برنامه به نحو قابل ملاحظه‌ای افزایش خواهد یافت. اما ... این مجازی سازی، اسکرول مطلوبی ندارد و بریده بریده به نظر می‌رسد.

خاصیت‌های جدید VirtualizingPanel در دات نت 4.5

تمام کنترل‌های مشتق شده‌ی از ListBox مانند ListView و امثال آن، در WPF 4.5 امکان تنظیم یک چنین خواصی را یافته‌اند:

```
<ListBox ItemsSource="{Binding Persons}"
    VirtualizingPanel.IsVirtualizing="True"
    VirtualizingPanel.ScrollUnit="Pixel"
    VirtualizingPanel.IsVirtualizingWhenGrouping="True"
    VirtualizingPanel.CacheLength="100"
    VirtualizingPanel.CacheLengthUnit="Pixel"/>
```

در WPF 4.5 پس از نمایش تعداد ردیف‌های قابل ملاحظه‌ی توسط کاربر، سیستم کش خودکاری با حق تقدم پایین فعال شده و آیتم‌هایی را که هنوز نمایان نیستند، ایجاد و کش می‌کند. به این ترتیب کاربر با اسکرول به سمت پایین یا بالا، کندی رندر یا بریده بریده به نظر رسیدن اسکرول را حس نخواهد کرد.

در اینجا می‌توان دو خاصیت CacheLength و CacheLengthUnit را مقدار دهی کرد. مقدار پیش فرض CacheLength مساوی 1.1 است. CacheLengthUnit می‌تواند یکی از مقادیر Item، Pixel یا Page را بپذیرد. هر Page در اینجا بر اساس اندازه‌ی viewport یا قسمت نمایان لیست، تعریف می‌شود. مقدار پیش فرض آن نیز Item است. همچنین در اینجا می‌توان ScrollUnit را نیز تنظیم کرد که مقادیر Item یا Pixel را می‌پذیرد. حالت پیش فرض آن Item است؛ به این معنا که اگر ردیفی کاملاً در viewport جای نگرفت، نمایش داده نمی‌شود. این مورد را با تنظیم مقدار ScrollUnit به Pixel می‌توان بهبود بخشید.

IsVirtualizingWhenGrouping سبب فعال سازی مجازی سازی حتی در حالت گروه بندی اطلاعات می‌گردد. به صورت پیش فرض اگر گروه بندی فعال شود، دیگر مجازی سازی رخ نخواهد داد.

فعال سازی قابلیت‌های دات نت 4.5 در برنامه‌های WPF 4

اگر برنامه‌ی WPF 4 خود را فعلاً قصد ندارید به دات نت 4.5 ارتقاء دهید، با توجه به اینکه اگر کاربر دات نت 4.5 را نصب کرده باشد، برنامه‌ی شما به صورت خودکار همانند یک برنامه‌ی WPF 4.5 رفتار می‌کند (دات نت 4.5 جایگزین دات نت 4 می‌شود)، می‌توان با اندکی Reflection این قابلیت‌ها را در صورت وجود، فعال کرد:

```
using System;
using System.Reflection;
using System.Windows;
using System.Windows.Controls;

namespace DNTProfiler.Common.Behaviors
{
    /// <summary>
    /// Smooth scrolling VirtualizingStackPanels, without sacrificing virtualization.
    /// </summary>
    public static class PixelBasedScrollingBehavior
    {
        private static readonly MethodInfo _setScrollUnit = typeof(VirtualizingPanel)
            .GetMethod("SetScrollUnit", BindingFlags.Public | BindingFlags.Static);

        private static readonly MethodInfo _setCacheLengthUnit = typeof(VirtualizingPanel)
            .GetMethod("SetCacheLengthUnit", BindingFlags.Public | BindingFlags.Static);
    }
}
```

```

private static readonly MethodInfo _setCacheLength = typeof(VirtualizingPanel)
    .GetMethod("SetCacheLength", BindingFlags.Public | BindingFlags.Static);

public static bool GetIsEnabled(DependencyObject obj)
{
    return (bool)obj.GetValue(IsEnabledProperty);
}

public static void SetIsEnabled(DependencyObject obj, bool value)
{
    obj.SetValue(IsEnabledProperty, value);
}

public static readonly DependencyProperty IsEnabledProperty =
    DependencyProperty.RegisterAttached("IsEnabled", typeof(bool),
    typeof(PixelBasedScrollingBehavior), new UIPropertyMetadata(false, handleIsEnabledChanged));

private static void handleIsEnabledChanged(DependencyObject obj,
    DependencyPropertyChangedEventArgs e)
{
    var listView = obj as ListView;
    if (listView == null)
    {
        throw new InvalidOperationException("This behavior can only be attached to a
        ListView.");
    }

    if (_setScrollUnit != null)
    {
        // It's .NET 4.5
        _setScrollUnit.Invoke(listView, new object[] { listView, /*Pixel*/ 0 });
    }

    if (_setCacheLengthUnit != null)
    {
        // It's .NET 4.5
        _setCacheLengthUnit.Invoke(listView, new object[] { listView, /*Pixel*/ 0 });
    }

    if (_setCacheLength != null)
    {
        // It's .NET 4.5
        var type = Type.GetType("System.Windows.Controls.VirtualizationCacheLength,
        PresentationFramework, Version=4.0.0.0, Culture=neutral, PublicKeyToken=31bf3856ad364e35");
        if (type == null) return;
        var instance = Activator.CreateInstance(type, 100.0);

        _setCacheLength.Invoke(listView, new[] { listView, instance });
    }
}
}
}

```

در اینجا با تعریف یک Attached property جدید، خواصی مانند CacheLength و ScrollUnit، CacheLengthUnit توسط Reflection یافت خواهند شد. اگر مقدار آن‌ها نال نباشند، یعنی برنامه‌ی دات نت 4 در سیستمی که بر روی آن دات نت 4.5 نصب است، در حال اجرا می‌باشد. بنابراین در این حالت می‌توان اسکرول مبتنی بر Pixel را فعال کرد تا برنامه اسکرول روان‌تری را پیدا کند.

برای استفاده از آن خواهیم داشت:

```

<ListView ItemsSource="{Binding}"
    behaviors:PixelBasedScrollingBehavior.IsEnabled="True">

```

فرض کنید یک چنین کلاسی طراحی شده‌است:

```
public class NestedClass
{
    private int _field2;
    public NestedClass()
    {
        _field2 = 12;
    }
}

public class MyClass
{
    private int _field1;
    private NestedClass _nestedClass;

    public MyClass()
    {
        _field1 = 1;
        _nestedClass = new NestedClass();
    }

    private string GetData()
    {
        return "Test";
    }
}
```

می‌خواهیم از طریق Reflection مقادیر فیلدها و متدهای مخفی آن‌را بخوانیم.
حالت متداول دسترسی به فیلد خصوصی آن از طریق Reflection، یک چنین شکلی را دارد:

```
var myClass = new MyClass();

var field1Obj = myClass.GetType().GetField("_field1", BindingFlags.NonPublic | BindingFlags.Instance);
if (field1Obj != null)
{
    Console.WriteLine(Convert.ToInt32(field1Obj.GetValue(myClass)));
}
```

و یا دسترسی به مقدار خروجی متد خصوصی آن، به نحو زیر است:

```
var getDataMethod = myClass.GetType().GetMethod("GetData", BindingFlags.NonPublic | BindingFlags.Instance);
if (getDataMethod != null)
{
    Console.WriteLine(getDataMethod.Invoke(myClass, null));
}
```

در اینجا دسترسی به مقدار فیلد مخفی NestedClass، شامل مراحل زیر است:

```
var nestedClassObj = myClass.GetType().GetField("_nestedClass", BindingFlags.NonPublic | BindingFlags.Instance);
if (nestedClassObj != null)
{
    var nestedClassFieldValue = nestedClassObj.GetValue(myClass);
    var field2Obj = nestedClassFieldValue.GetType().GetField("_field2", BindingFlags.NonPublic | BindingFlags.Instance);
    if (field2Obj != null)
    {
        Console.WriteLine(Convert.ToInt32(field2Obj.GetValue(nestedClassFieldValue)));
    }
}
```


البته این مقدار کد فقط برای دسترسی به دو سطح تو در تو بود.

چقدر خوب بود اگر می‌شد بجای این همه کد، نوشت:

```
myClass._field1
myClass._nestedClass._field2
myClass.GetData()
```

نه؟!

برای این مشکل راه حلی معرفی شده‌است به نام Dynamic Proxy که در ادامه به معرفی آن خواهیم پرداخت.

معرفی Dynamic Proxy

Dynamic Proxy یکی از [مفاهیم AOP](#) است. به این معنا که توسط آن یک محصور کننده نامرئی، اطراف یک شیء تشکیل خواهد شد. از این غشای نامرئی عموماً جهت مباحث ردیابی اطلاعات، مانند پروکسی‌های Entity framework، همانجایی که تشخیص می‌دهد کدام خاصیت به روز شده‌است یا خیر، استفاده می‌شود و یا این غشای نامرئی کمک می‌کند که در حین دسترسی به خاصیت یا متدی، بتوان منطق خاصی را در این بین تزریق کرد. برای مثال فرآیند تکراری logging سیستم را به این غشای نامرئی منتقل کرد و به این ترتیب می‌توان به کدهای تمیزتری رسید. یکی دیگر از کاربردهای این محصور کننده یا غشای نامرئی، ساده سازی مباحث Reflection است که نمونه‌ای از آن در پروژه‌ی [EntityFramework.Extended](#) بکار رفته‌است. در اینجا، کار با محصور سازی نمونه‌ای از کلاس مورد نظر با Dynamic Proxy شروع می‌شود. سپس کل عملیات Reflection فوق در همین چند سطر ذیل به نحوی کاملاً عادی و طبیعی قابل انجام است:

```
// Accessing a private field
dynamic myClassProxy = new DynamicProxy(myClass);
dynamic field1 = myClassProxy._field1;
Console.WriteLine((int)field1);

// Accessing a nested private field
dynamic field2 = myClassProxy._nestedClass._field2;
Console.WriteLine((int)field2);

// Accessing a private method
dynamic data = myClassProxy.GetData();
Console.WriteLine((string)data);
```

خروجی Dynamic Proxy از نوع dynamic دات نت 4 است. پس از آن می‌توان در اینجا هر نوع خاصیت یا متد دلخواهی را به شکل dynamic تعریف کرد و سپس به مقادیر آن‌ها دسترسی داشت.

بنابراین با استفاده از Dynamic Proxy فوق می‌توان به دو مهم دست یافت:

1) ساده سازی و زیبا سازی کدهای کار با Reflection

2) استفاده‌ی ضمنی از مباحث [Fast Reflection](#). در کتابخانه‌ی Dynamic Proxy معرفی شده، دسترسی به خواص و متدها، توسط [کدهای IL](#) بهینه سازی شده‌است و در دفعات آتی کار با آن‌ها، دیگر شاهد سربار بالای Reflection نخواهیم بود.

کدهای کامل این مثال را از اینجا می‌توانید دریافت کنید:

[DynamicProxyTests.zip](#)

به احتمال زیاد برنامه نویسانی که از AngularJS در پروژه‌های خود استفاده می‌کنند، در برخی موارد کند شدن در Rendering و Binding صفحات را تجربه کرده اند. این مقاله مطالب خیلی ساده و راحتی در خصوص استفاده درست و بهینه از Binding می‌باشد.

قدم اول و مهم بحث on time binding هست:

در برخی موارد ما اطلاعاتی که فقط یکبار Bind می‌شوند و در طول اجرا هیچ تغییری نمی‌کنند را درست Bind نمی‌کنیم. برای مثال فرض کنید می‌خواهیم عنوان صفحه را در یک تگ h1 نمایش دهیم. به صورت معمول همه‌ی ما از روش زیر استفاده می‌کنیم.

```
<h1>{{title}}</h1>
```

اما این روش درست نیست! چرا؟

AngularJS برای Rendering طرف View از Watcherها استفاده می‌کند که در هر لحظه Bindingها را رصد می‌کنند و در صورت تغییر قسمت مورد نظر از View، دوباره Render می‌شود. بحث اصلی خود Watcherها هستند که حتی Bindingهایی که هیچ وقت مقادیر آنها تغییر نمی‌کنند نیز رصد می‌شود و این باعث کندی عمل Watching در AngularJS می‌شود. در AngularJS نسخه‌ی 1.3 به بعد امکانی فراهم شده‌است که شما بتوانید Bindingهایی را که یک بار بیشتر تغییر نمی‌کنند، مشخص کنید. به کد زیر دقت کنید:

```
<h1>{{::title}}</h1>
```

بله دقیقاً به همین راحتی! شما با اضافه کردن :: در ابتدای هر Binding مشخص می‌کنید که این قسمت از View فقط یک بار Render شود. این عمل باعث می‌شود AngularJS Watcher کار کمتری انجام دهد. به زودی مقاله‌های بیشتری در خصوص Performance در AngularJS خواهیم نوشت. امیدوارم لذت برده باشید.

نظرات خوانندگان

نویسنده: مسعود پاکدل
تاریخ: ۱۳۹۴/۰۳/۲۳ ۱۳:۳۹

در نسخه‌های قدیمی‌تر نیز می‌توانید از [Angular bindonce](#) استفاده کنید.

[در مقاله‌ی قبل](#) روش درست استفاده کردن از Binding را برای بهبود Performance، توضیح دادم. در این مقاله می‌خواهم در مورد ng-if و فرق آن با ng-show صحبت کنم و اینکه کدامیک Performance بهتری را برای AngularJS فراهم می‌کنند. **سول اول، کار ng-show چیست؟**

ng-show یکی از پر کاربردترین Directive‌های AngularJS است که وظیفه‌ی Show و Hide قسمتی از View را به عهده دارد. به کد زیر توجه کنید:

```
<div ng-show="has">
  <div ng-repeat="item in items">
    <span>{{item.title}}</span>
  </div>
</div>
```

در این کد ما از ng-show استفاده کرده‌ایم. اگر has مقدار true داشته باشد div نمایش داده می‌شود و اگر false باشد، div نمایش داده نمی‌شود. در داخل div، لیستی وجود دارد که توسط ng-repeat تکرار شده و یک لیست ساده را درست می‌کند. **سول دوم، کار ng-if چیست؟**

کد بالا را دوباره تکرار می‌کنیم. ولی با این تفاوت که اینبار بجای ng-show از ng-if استفاده خواهیم کرد:

```
<div ng-if="has">
  <div ng-repeat="item in items">
    <span>{{item.title}}</span>
  </div>
</div>
```

خوب؟ آیا عملکرد این دو کد با هم تفاوت دارد؟ جواب سؤال در ظاهر خیر هست. یعنی مانند کد بالایی، اگر has مقدار true داشته باشد، div نمایش داده می‌شود؛ در غیر اینصورت، خیر.

سوال سوم، پس اگر عملکرد یکسانی دارند، تفاوت آن‌ها در چیست؟

تفاوت این دو Directive در Performance هست. اجازه دهید بیشتر توضیح دهم. ng-show اگر مقدار false دریافت کند، tag مورد نظر را نمایش نمی‌دهد؛ ولی تگ‌های داخلی آن توسط AngularJS پردازش می‌شوند. یعنی چه ng-show مقدار true بگیرد و یا false، لیست داخل tag، توسط AngularJS پردازش و Render می‌شود. ولی در ظاهر ما در View چیزی را نمی‌بینیم. ng-if بر حسب مقادیری که دریافت می‌کند، می‌تواند به بالا رفتن Performance AngularJS کمک کند. فرض کنید ng-if مقدار false گرفته است؛ یعنی has مقدار false دارد. علاوه بر اینکه tag div نمایش داده نمی‌شود، بلکه داخل tag نیز پردازش نمی‌شود. یعنی لیستی که ما در کد نوشته‌ایم، به هیچ عنوان توسط AngularJS پردازش نخواهد شد که باعث می‌شود Watcher، کار کمتری انجام دهد. پس در نتیجه بهبودی را در کارآیی Rendering و Binding خواهیم داشت.

خیلی خوشحالم که تا این مرحله، [این مقاله‌ها](#) را دنبال می‌کنید. [در مقالات قبل](#) مسائل ساده و مهمی در بحث Performance مطرح شد. در این مقاله می‌خواهم قدم سوم در بهبود Performance را توضیح دهم که رعایت کردن این مسائل می‌تواند کمک زیادی در بهبود عملکرد برنامه‌های مبتنی بر AngularJS داشته باشد.

scope؟

همه‌ی برنامه نویسان و توسعه دهندگان، یکی از اولین مفاهیمی را که در AngularJS یاد می‌گیرند، scope هست. اما scope چیست؟ به صورت خیلی ساده می‌توان گفت scope مشخص کننده‌ی حوزه متغیرها و توابعی هست که قرار است در View تاثیر داشته باشند. کد زیر را مشاهده کنید:

```
<div>{{name}} نام و نام خانوادگی</div>
<div>{{avg()}} معدل</div>
```

و کد سمت controller

```
scope.nums=[19,20,17,16,15,18,19];
scope.name='بهنام محمدی';
scope.avg= function(){
    return scope.nums.reduce(function(previousValue, currentValue) {
        return previousValue + currentValue;
    })/scope.nums.length;
}
```

و اما خروجی نهایی

نام و نام خانوادگی: بهنام محمدی
معدل: 17.71

خوب چون ما در قسمت controller به صورت scope.name و scope.avg عمل کرده‌ایم، می‌توانیم در View به صورت name و avg از این متغیرها استفاده کنیم. در نتیجه اگر ما در controller، به جای scope.name بنویسیم name و یا به جای scope.avg بنویسیم avg به مشکل بر می‌خوریم؛ چون قسمت View ما متغیرهای داخل scope را در View دخیل می‌کند و متغیرهای داخل scope توسط Watcherها رصد می‌شود.

خوب سؤال، همه چیز که عالی هست پس مشکل در کجاست؟

مشکل در متغیر scope.nums هست! به کد زیر توجه کنید:

```
var nums=[19,20,17,16,15,18,19];
scope.name='بهنام محمدی';
scope.avg= function(){
    return nums.reduce(function(previousValue, currentValue) {
        return previousValue + currentValue;
    })/nums.length;
}
```

فکر کنم متوجه تفاوت این کد با کد بالایی شده‌اید. اما کدام کد درست است؟ یا بهتر بگوییم کدام کد بر روی Performance تاثیر مناسبی دارد؟ کد پایینی Performance بالایی دارد. دلیل این موضوع این است وقتی ما از nums در View هیچ استفاده‌ای نمی‌کنیم، بهتر است به صورت var nums تعریف شود. در کد بالایی که این متغیر به صورت scope.nums تعریف شده بود، با اینکه در View استفاده نشده بود، ولی توسط AngularJS Watcher در هر لحظه رصد می‌شود و این کار باعث کندی و کاهش عملکرد AngularJS خواهد شد. بنابراین در کل متغیرهایی را که در View استفاده نمی‌کنید، به صورت var test استفاده نمایید تا AngularJS Watcher این متغیرها را رصد نکند.

امیدوارم از این مقاله لذت برده باشید. منتظر مقاله بعدی من باشید.

امیدوارم [از مقالات قبلی](#) لذت برده باشید. در این مقاله می‌خواهم در مورد watch\$ صحبت کنم.

سوال اول: watch\$ چیست و چه کاربردی دارد؟

watch\$ همان عملکرد Watching در AngularJS را انجام می‌دهد؛ ولی کاربردهای جالبی دارد. به کد زیر دقت کنید.

```
var errorChat=false;
$scope.$watch(function () {
    return errorChat;
}, function (newValue, oldValue) {
    if(newValue ===true){
        alert('قسمت محاوره سامانه با مشکل روبرو شده است لطفا با مدیریت تماس بگیرید')
    }
});
```

فرض کنید سناریوی پروژه به این صورت است که ما قسمت‌های مختلفی را در صفحه داریم و یکی از این قسمت‌ها، چت روم می‌باشد. می‌خواهیم در صورتیکه خطای اتصال و یا هر نوع خطایی در این قسمت بود، با پیغامی به کاربر گزارش داده شود. ما از متغیر errorChat برای این کار استفاده کرده‌ایم. با فرض اینکه در توابع دیگر در صورتیکه خطایی وجود داشته باشد، مقدار این متغیر را true می‌کند و ما بر حسب رصد این متغیر پیغام خطا را نمایش می‌دهیم.

سوال دوم: این کد به نحوه احسن کار می‌کند؛ پس مشکل کجاست؟

مشکل اینجاست بعد از اینکه متغیر errorChat مقدار true گرفت و ما هشدار را نمایش دادیم، باز این متغیر رصد می‌شود که لزومی به این کار نیست (فرض ما بر این است که بعد از بروز خطا دیگر قسمت چت روم کار نخواهد کرد) و متوقف کردن این متغیر باعث می‌شود Performance در نهایت بهتر شود. خوب برای اینکه رصد این متغیر را متوقف کنیم از کد زیر استفاده می‌کنیم. به کد زیر توجه کنید:

```
var errorChat=false;
var stop=$scope.$watch(function () {
    return errorChat;
}, function (newValue, oldValue) {
    if(newValue ===true){
        stop();
        alert('قسمت محاوره سامانه با مشکل روبرو شده است لطفا با مدیریت تماس بگیرید')
    }
});
```

خوب؛ فکر کنم تفاوت این دو کد با هم روشن است. ما یک متغیر stop به کد اضافه کردیم. این متغیر با تابع متوقف کننده watch\$ مربوط به errorChat پر می‌شود و در نهایت با اجرای این تابع، عمل watching متغیر errorChat متوقف می‌شود. اگر با setInterval یا setTimeout در Javascript کار کرده باشید، شباهت این موارد را متوجه خواهید شد.

در این مقاله موضوعی را مطرح خواهیم کرد که شاید برای خیلی‌ها این نوع کد نویسی خوشایند نباشد. حتی برای خود من هم خوشایند نیست! ولی نهایتاً در بهبود Performance تاثیر خیلی زیادی دارد. به کد زیر دقت کنید.

```
<div ng-repeat="item in items">
  <div ng-if="setting.header">{{item.header}}</div>
  <div>{{item.title}}</div>
  <div ng-if="setting.footer">{{item.footer}}</div>
</div>
```

توضیح کد: فرض کنید سناریوی پروژه ما به این صورت هست که ما یک لیست داریم، شامل 3 فیلد که header، title و footer را در تنظیمات می‌توانیم مشخص کنیم که header و footer در شرایطی نمایش داده شود و در شرایطی نمایش داده نشود و یا حالت‌های دیگر.

خوب مشکل چیست و راهکار چیست؟

فرض کنید لیست ما شامل 100 رکورد هست و در تنظیمات مشخص کرده‌ایم که header نمایش داده شود و footer نمایش داده نشود. اما اتفاقی بدی که می‌فتد این است که وقتی لیست در View ساخته می‌شود، 100 بار ng-if مربوط به header و footer چک می‌شود؛ در جمع 200 بار می‌شود. چه این مقادیر true باشند چه false فرقی نمی‌کند و 200 بار بررسی می‌شود. راهکار این مشکل به این صورت است که ما باید از ng-if داخل ng-repeat استفاده نکنیم. اما برای پیاده سازی تنظیمات باید ng-ifها را قبل از ng-repeat بررسی کنیم. پس مسلماً ng-repeatها باید قالب پیش بینی کرده ما را نسبت به ng-ifها درست کند. نتیجه‌ی کار به صورت کد زیر است که شاید برای شما هم خوشایند نباشد:

```
<div ng-if="setting.header && setting.footer">
  <div ng-repeat="item in items">
    <div>{{item.header}}</div>
    <div>{{item.title}}</div>
    <div>{{item.footer}}</div>
  </div>
</div>
<div ng-if="setting.header && setting.footer==false">
  <div ng-repeat="item in items">
    <div>{{item.header}}</div>
    <div>{{item.title}}</div>
  </div>
</div>
<div ng-if="setting.header==false && setting.footer">
  <div ng-repeat="item in items">
    <div>{{item.title}}</div>
    <div>{{item.footer}}</div>
  </div>
</div>
<div ng-if="setting.header==false && setting.footer==false">
  <div ng-repeat="item in items">
    <div>{{item.title}}</div>
  </div>
</div>
```

درست است من هم با شما موافق هستم که خوشایند نیست. در این کد ما همه‌ی حالت‌ها را پیش بینی و قالب مناسب هر شرط را درست کرده‌ایم. حجم کد چند برابر شده، ولی از لحاظ Performance در ساخت لیست در View در حد 98% بهبود پیدا کرده‌است. همان مثال قبلی را در نظر بگیرید. ng-if مربوط به header و footer در این کد فقط 4 بار بررسی می‌شود. چه 100 رکورد باشد، چه 1000 تا، چه 10 تا رکورد.

در مورد ng-repeatها هم نگران نباشید فقط یک بار اجرا میشوند. اگر کارکرد ng-if را در [مقاله‌ی قبلی من](#)، خوانده باشید، متوجه‌ی این موضوع می‌شوید که elementهای داخلی و directionهای AngularJS داخلی ng-if زمانی پردازش می‌شوند که شرط true باشد. از این روش زمانی استفاده کنید که تعداد داده‌ها و حالت‌های زیادی دارید و Performance اهمیت بیشتری دارد. امیدوارم مقاله‌ی مفیدی باشد.

برنامه‌های قدیمی، الزاما خیلی قدیمی هم نیستند؛ برنامه‌هایی هستند پر از کوئری‌های ذیل:

```
SELECT * FROM table1 WHERE OrderDate = '12 Mar 2004'

SET @SQL = 'SELECT * FROM table2 WHERE OrderDate = ' + @Var + ''
EXEC (@SQL)
```

ویژگی مهم این نوع کوئری‌ها که با جمع زدن رشته‌ها و یا مقدار دهی مستقیم فیلدها تشکیل شده‌اند، «غیر پارامتری» بودن آن‌ها است.

این نوع مشکلات با بکار گیری ORM‌ها به نحو قابل توجهی کاهش یافته‌است؛ زیرا این نوع واسط‌ها در اغلب موارد، در آخر کار کوئری‌هایی پارامتری را تولید می‌کنند.

مشکل کوئری‌های غیر پارامتری چیست؟

استفاده‌ی وسیع از کوئری‌های غیرپارامتری با SQL Server، مشکلی را پدید می‌آورد به نام «Cache bloat» یا «کش پُف کرده» و این «پُف» به این معنا است که کش کوئری‌های اجرا شده‌ی بر روی SQL Server بیش از اندازه با Query plan‌های مختلف حاصل از بررسی نحوه‌ی اجرای بهینه‌ی آن‌ها پر شده‌است. هر کوئری که به SQL Server می‌رسد، جهت اجرای بهینه، ابتدا پردازش می‌شود و دستور العملی خاص آن، تهیه و سپس در حافظه کش می‌شود. وجود این کش به این خاطر است که SQL Server هر بار به ازای هر کوئری رسیده، این عملیات پردازشی را تکرار نکند. مشکل از زمانی شروع می‌شود که SQL Server کوئری‌هایی را که از نظر یک برنامه نویس مانند هم هستند را به علت عدم استفاده‌ی از پارامترها، یکسان تشخیص نداده و برای هر کدام یک Plan جداگانه را محاسبه و کش می‌کند. این مساله با حجم بالای کوئری‌های رسیده دو مشکل را ایجاد می‌کند:

الف) مصرف حافظه‌ی بالای SQL Server که گاهی اوقات این حافظه‌ی اختصاص داده شده‌ی به کش کوئری‌ها به بالای یک گیگابایت نیز می‌رسد.

ب) CPU Usage بالای سیستم

سیستم قدیمی است؛ امکان تغییر کدها را نداریم.

بدیهی است بهترین راه حلی که در اینجا وجود دارد، پارامتری ارسال کردن کوئری‌ها به SQL Server است تا به ازای هر تغییری در مقادیر آن‌ها، این کوئری‌ها باز هم یکسان به نظر برسند و SQL Server سعی در محاسبه‌ی مجدد Plan آن‌ها نکند. اما ... اگر این امکان را ندارید، خود SQL Server یک چنین قابلیت‌هایی را به صورت توکار تدارک دیده‌است که باید فعال شوند.

فعال سازی پارامتری کردن خودکار کوئری‌ها در SQL Server

اگر نمی‌توانید کدهای یک سیستم قدیمی را تغییر دهید، SQL Server می‌تواند به صورت خودکار این کار را برای شما انجام دهد. در این حالت فقط کافی است یکی از دو دستور ذیل را اجرا کنید:

```
--Forced
ALTER DATABASE dbName SET PARAMETERIZATION FORCED

--Simple
ALTER DATABASE dbName SET PARAMETERIZATION SIMPLE
```

حالت simple بیشتر جهت پارامتری کردن خودکار کوئری‌های select بکار می‌رود. اگر می‌خواهید تمام کوئری‌های select, insert, update و delete را نیز پارامتری کنید، باید از حالت forced استفاده نمایید.

فعال سازی بهبود کارایی SQL Server با کوئری‌های Ad-Hoc زیاد

به کوئری‌های غیرپارامتری، کوئری‌های Ad-Hoc نیز گفته می‌شود. اگر سیستم فعلی شما، تعداد زیادی کوئری Ad-Hoc تولید می‌کند، می‌توان فشار کاری SQL Server را برای این مورد خاص، تنظیم و بهینه سازی کرد. فعال سازی گزینه‌ی ویژه‌ی «Optimize for Ad hoc Workloads» سبب می‌شود تا SQL Server پس از مدتی به صورت خودکار کش Plan کوئری‌هایی را که به ندرت استفاده می‌شوند، حذف کند. همین مساله سبب آزاد شدن حافظه و بهبود کارایی کلی سیستم می‌گردد. همچنین باید در نظر داشت که کش Plan کوئری‌ها نامحدود نیست و سقفی دارد. به همین جهت آزاد شدن آن، کش کردن کوئری‌هایی را که بیشتر استفاده می‌شوند، ساده‌تر می‌کند. برای اعمال آن به یک بانک اطلاعاتی خاص، نیاز است دستورات ذیل را اجرا کرد:

```
use dbName;
-- Optimizing for Ad hoc Workloads
exec sp_configure 'show advanced options',1;
RECONFIGURE;
go
exec sp_configure 'optimize for ad hoc workloads',1;
RECONFIGURE;
Go
```

برای مطالعه‌ی بیشتر

[Fixing Cache Bloat Problems With Guide Plans and Forced Parameterization](#)

[Optimizing ad-hoc workloads](#)

[Optimizing for Ad hoc Workloads](#)

ورود سیستم‌های ORM مانند EF تحولی عظیم در در مباحث کار و تغییرات بر روی داده‌ها یا Data Manipulation بود. به طور خلاصه اصلی‌ترین هدف یک ORM، ایجاد فرامین شیء گرا به جای فرامین رابطه‌ای است؛ ولی در این بین نکات دیگری هم مد نظر گرفته شده است که یکی از آن‌ها پشتیبانی از چندین دیتابیس هست تا توسعه گران از یک سیستم واحد جهت اتصال به همه‌ی دیتابیسی‌ها استفاده کنند و نیازی به دانش اضافه و سیستم جداگانه‌ای برای هر دیتابیس نباشد؛ مانند ADO که در دات نت به چندین دسته تقسیم شده بود و هم اینکه در صورتی که تمایلی به تغییر دیتابیس در آینده داشتید، کدها برای توسعه باز باشند و نیازی به تغییر سیستم نباشد.

ولی اگر کمی بیشتر به دنیای واقعی وارد شویم گاهی اوقات نیاز است که تنها بر روی یک دیتابیس فعالیت کنیم و یک دیتابیس نیاز است تا حد ممکن بهینه طراحی شود تا کارآیی بانک در حال حاضر و به خصوص در آینده تا حدی تضمین شود. من همیشه در مورد EF یک نظری داشتم و آن اینست که با اینکه یک ORM، یک هدف مهم را در نظر دارد و آن اینست که تا حد ممکن استانداردهایی را که بین تمامی دیتابیسی‌ها مشترک است، رعایت کند، ولی باز قابل قبول است اگر بگوییم که کاربران EF انتظار داشته باشند تا اطلاعات بیشتری در مورد sql server در آن نهفته باشد. از یک سو هر دو محصول میکروسافت هستند و از سوی دیگر مطمئناً توسعه گران محصولات دات نت بیش از هر چیزی به sql server نگاه ویژه‌تری دارند. پس میکروسافت در کنار حفظ آن ویژگی‌های مشترک، باید به حفظ استانداردهای جدایی برای sql server هم باشد.

تعدادی از برنامه نویسان در هنگام ایجاد Domain Model کم لطفی‌های زیادی را می‌کنند که یکی از آن‌ها عدم کنترل نوع داده‌های خود است. مثلاً برای رشته‌ها هیچ محدودیتی را در نظر نمی‌گیرند. شاید در سمت کلاینت اینکار را انجام می‌دهند؛ ولی نکته‌ی مهم در طرف دیتابیس است که چگونه تعریف می‌شود. در این حالت nvarchar(MAX) در نظر گرفته میشود که به معنی اشاره به منطقه دوگیگابایتی از اطلاعات است. در نکات بعدی، قصد داریم این مرحله را یک گام به جلوتر پیش ببریم و آن هم ایجاد نوع داده‌های بهینه‌تر در Sql Server است.

نکته مهم: بدیهی است که تغییرات زیر، ORM شما را تنها به sql server مقید می‌کند که بعدها در صورت تغییر دیتابیس نیاز به حذف موارد زیر را خواهید داشت؛ در غیر اینصورت به مشکل عدم ایجاد دیتابیس برخورد خواهید کرد.

اولین مورد مهم بحث تاریخ و زمان است؛ وقتی ما یک نوع داده را تنها در DateTime در نظر بگیریم، در Sql Server هم همین نوع داده وجود دارد و انتخاب میشود. ولی اگر شما واقعاً نیازی به این نوع داده نداشته باشید چطور؟ در حال حاضر من بر روی یک برنامه‌ی کارخانه کار میکنم که بخش کارمندان و گارگران آن سه داده زمانی زیر را شامل می‌شود:

```
public DateTime BirthDate { get; set; }
public DateTime HireDate { get; set; }
public DateTime? LeaveDate { get; set; }
```

حال به جدول زیر نگاه کنید که هر نوع داده چه مقدار فضا را به خود اختصاص می‌دهد:

4 بایت	SmallDateTime
8 بایت	DateTime
6 تا 8 بایت	DateTime2
8 تا 10 بایت	DateTimeOffset
3 بایت	Date

4 بایت	SmallDateTime
3 تا 5 بایت	Time

از این جدول چه می‌فهمید؟ با یک نگاه می‌توان فهمید که ساختار بالای من باید 24 بایت گرفته باشد؛ برای ساختاری که هم تاریخ و هم زمان (ساعت) را پشتیبانی می‌کند. ولی با نگاه دقیق‌تر به نام پراپرتی‌ها این نکته روشن می‌شود که ما یک گپ (Gap فضای بیهوده) داریم چون زمان تولد، استخدام و ترک سازمان اصلاً نیازی به ساعت ندارند و همان تاریخ کافی است. یعنی نوع Date با حجم کلی 9 بایت؛ که در نتیجه 15 بایت صرفه جویی در یک رکورد صورت خواهد گرفت.

پس کد بالا را به شکل زیر تغییر می‌دهم:

```
[Column(TypeName = "date")]
public DateTime BirthDate { get; set; }

[Column(TypeName = "date")]
public DateTime HireDate { get; set; }

[Column(TypeName = "date")]
public DateTime? LeaveDate { get; set; }
```

خصوصیت [Column](#) از نسخه 4.5 دات نت فریم ورک اضافه شده و در فضای نام `System.ComponentModel.DataAnnotations.Schema` قرار گرفته است.

نوع‌هایی که در بالا با سایز متغیر هستند، به نسبت دقتی که برای آن تعیین می‌کنید، سایز می‌گیرند. مثل `time(0)` که 3 بایت از حافظه را می‌گیرد. در صورتی که `time` معرفی کنید، به جای اینکه از شیء `DateTime` استفاده کنید، از شیء `Timespan` استفاده کنید، تا در پشت صحنه از نوع داده `time` استفاده کند. در این حالت حداکثر حافظه یعنی 5 بایت را برخواهد داشت و بهترین حالت ممکن این هست که نیاز خود را بسنجید و خودتان دقت آن را مشخص کنید. دو شکل زیر نحوه‌ی تعریف نوع زمان را مشخص می‌کنند. یکی حالت پیش فرض و دیگری انتخاب دقت:

```
public class Testtypes
{
    public TimeSpan CloseTime { get; set; }
    public TimeSpan CloseTime2 { get; set; }
}

public class TestConfig : EntityTypeConfiguration<Testtypes>
{
    public TestConfig()
    {
        this.Property(x => x.CloseTime2).HasPrecision(3);
    }
}
```

در تکه کد بالا همه از نوع `time` تعریف می‌شوند ولی در خصوصیت شماره یک نهایت استفاده از نوع تایم یعنی `time(7)` مشخص می‌شود. ولی در خصوصیت بعدی چون در کانفیگ دقت آن را مشخص کرده‌ایم از نوع `time(3)` تعریف می‌شود.

مورد دوم در مورد داده‌های اعشاری است:

بسیاری از برنامه نویسان تا آنجا که دیده‌ام از نوع `float` و `single` و `double` برای اعداد اعشاری استفاده می‌کنند ولی باید دید که در آن سمت دیتابیس، برای این نوع داده‌ها چه اتفاقی می‌افتد. نوع `float` در دات نت، با نوع `single` برابری می‌کند؛ هر دو یک نام جدا دارند، ولی در واقع یکی هستند. عموماً برنامه نویسان به طور کلی بیشتر از همان `single` استفاده می‌کنند و برای انتساب برای این دو نوع هم حتماً باید حرف `f` را بعد از عدد نوشت:

```
float flExample=23.2f;
```

باید توجه کنید که اگر مثلاً `float` انتخاب کردید، تصور نکنید که همان `float` در دیتابیس خواهد بود. این دو متفاوت هستند تبدیلات به شکل زیر رخ می‌دهد:

```
//real
```

```
public float FloatData { get; set; }
//real
public Single SingleData { get; set; }
//float
public double DoubleData { get; set; }
```

همه نوع‌های بالا اعداد اعشاری هستند که به صورت تقریبی و به صورت نماد اعشاری ذخیره می‌گردند و برای به دست آوردن مقدار ذخیره شده، هیچ تضمینی نیست همان عددی که وارد شده است بازگردانده شود. اگر تا به حال در برنامه هایتان به چنین مشکلی برخوردید دلیلش اعداد اعشاری کوچک بوده است. ولی با بزرگتر شدن عدد، این تفاوت به خوبی دیده می‌شود. حالا اگر بخواهیم اعداد اعشاری را به طور دقیق ذخیره کنیم، مجبور به استفاده از نوع decimal هستیم. در دات نت آنچنان محدودیتی بر سر استفاده‌ی از آن نداریم. ولی در سمت سرور داده‌ها بهتر هست برای آن تدابیری اندیشیده شود. هر عدد دسیمال از دقت و مقیاس تشکیل می‌شود. دقت آن تعداد ارقامی است که در عدد وجود دارد و مقیاس آن تعداد ارقام اعشاری است. به عنوان مثال عدد زیر دقتش 7 و مقیاسش 3 است:

```
4235.254
```

در صورتی که عدد اعشاری را به دسیمال نسبت دهیم باید حرف m را بعد از عدد وارد کنیم:

```
decimal d1=4545.112m;
```

برای اعداد صحیح نیازی نیست. برای تعیین نوع دسیمال از fluent api استفاده می‌کنیم:

```
modelBuilder.Entity<Class>().Property(object => object.property).HasPrecision(7, 3);
```

کد زیر برای خصوصیت شماره یک، دقت 18 و مقیاس 2 را در نظر می‌گیرد و دومین خصوصیت طبق آنچه که برایش تعریف کرده ایم دقت 7 و مقیاس 3 است:

```
public class Testtypes
{
    public Decimal Decimal1 { get; set; }
    public Decimal Decimal2 { get; set; }
}

public class TestConfig : EntityTypeConfiguration<Testtypes>
{
    public TestConfig()
    {
        this.Property(x => x.Decimal2).HasPrecision(7, 3);
    }
}
```

مورد سوم مبحث رشته هاست :

کدهای زیر را مطالعه فرمایید:

```
[StringLength(25)]
public string FirstName { get; set; }

[StringLength(30)]
[Column(TypeName = "varchar")]
public string EnProductTitle { get; set; }

public string ArticleContent { get; set; }
[Column(TypeName = "varchar(max)")]
public string ArticleContentEn { get; set; }
```

اولین رشته بالا (نام) را به محدوده‌ای از کاراکترها محدود کرده‌ایم. به طور پیش فرض تمامی رشته‌ها به صورت nvarchar در نظر گرفته می‌شوند. بدین ترتیب در رشته نام کوچک (nvarchar(25) در نظر گرفته خواهد شد. حال اگر بخواهیم فقط حروف انگلیسی

پشتیبانی شوند، مثلاً نام فنی کالا را بخواهید وارد کنید، بهتر هست که نوع آن به طرز صحیحی تعریف شود که در کد بالا با استفاده از خصوصیت Column نوع varchar را معرفی می‌کنم. بدین ترتیب تعریف نهایی نوع به شکل `varchar(30)` خواهد بود. استفاده از `fluentApi` ها هم در این رابطه به شکل زیر است:

```
this.Property(e => e.EnProductTitle).HasColumnType("VARCHAR").HasMaxLength(30);
```

برای مواردی که محدوده‌ای تعریف نشود `nvarchar(MAX)` در نظر گرفته میشود مانند پراپرتی `ArticleContent` بالا. ولی اگر قصد دارید فقط حروف اسکی پشتیبانی گردند، مثلاً متن انگلیسی مقاله را نیز نگه می‌دارید بهتر هست که نوع آن به‌هیته‌ترین حالت در نظر گرفته شود که برای پراپرتی `ArticleContentEn` نوع `varchar(MAX)` تعریف کرده‌ایم. همانطور که گفتیم پیش فرض رشته‌ها `nvarchar` است، در صورتی که دوست دارید این پیش فرض را تغییر دهید روش زیر را دنبال کنید:

```
modelBuilder.Properties<string>().Configure(c => c.HasColumnType("varchar"));
```

//===== یا

```
modelBuilder.Properties<string>().Configure(c => c.IsUnicode(false));
```

جهت تکمیل بحث نیز هر کدام از متغیرهای عددی در سی شارپ معادل نوع‌های زیر در `Sql Server` هستند:

```
//tinyInt
public byte Age { get; set; }

//smallInt
public Int16 OldInt { get; set; }

//int
public int Int32 { get; set; }

//Bigint
public Int64 HighNumbers { get; set; }
```

نظرات خوانندگان

نویسنده: مرتضی ریسی
تاریخ: ۲۰:۴۴ ۱۳۹۴/۰۵/۰۴

سلام. ممنون.
میشه بفرمائید برای مقادیر مالی به ریال و تومان بهترین نوع داده ای چیست؟
من اینچنین استفاده میکنم:

```
public virtual decimal CostPrice { set; get; }
```

و در کانفیگ:

```
this.Property(x => x.CostPrice)  
    .HasColumnType("money")  
    .IsRequired();
```

همین نوع و همین اندازه تنظیم کافیه؟ آیا تنظیم بیشتری نیاز دارد؟

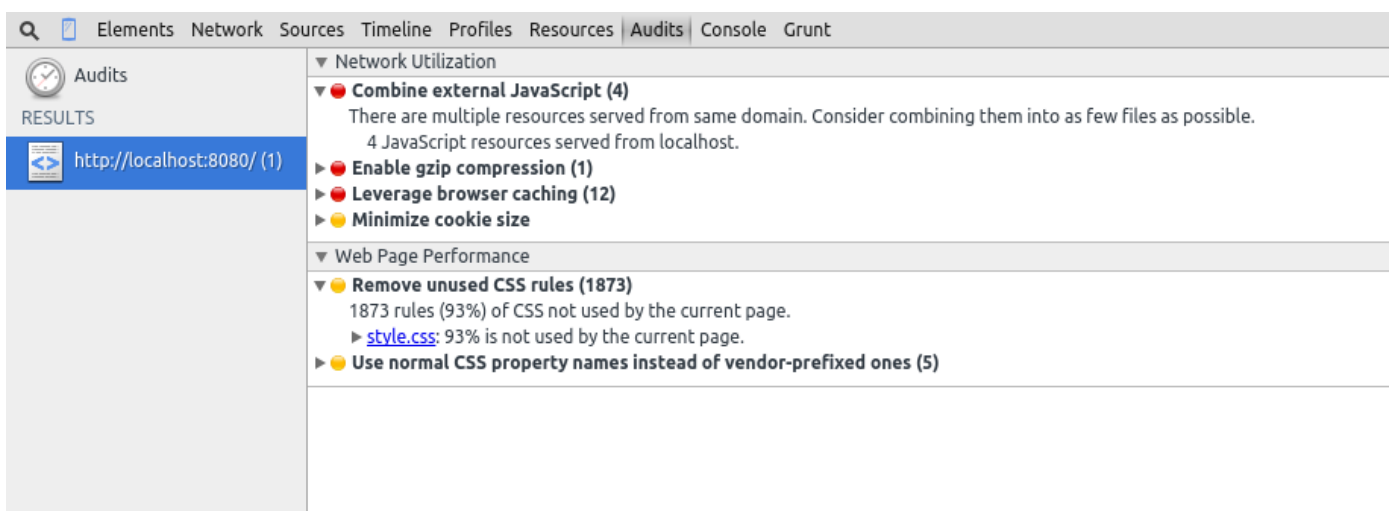
نویسنده: علی یگانه مقدم
تاریخ: ۲۲:۲ ۱۳۹۴/۰۵/۰۴

توصیه می‌کنم برای واحد پولی ایران از این جنس استفاده نکنید. از همان واحدهای عددی دقیق استفاده کنید.
اگر واحد مالی ما مانند کشورهای خارجی به صورت اعشار بیان میشد بله بهینه بود ولی در حال حاضر خیر.
من خودم تا به الان از Int استفاده کردم و می‌توان برای واحدهای بزرگتر BigInt را مورد استفاده قرار داد. هر چند int تا میلیارد را به خوبی پشتیبانی می‌کند

در [مقالات قبلی](#) به طور کامل با گالپ آشنا شدیم و گفتیم که می‌تواند ما را در بهینه سازی ورک فلویمان کمک کند. در این قسمت یاد خواهیم گرفت که چگونه تجربه‌ی کاربری بهتری را از سرعت بارگذاری سایتمان ایجاد کنیم.

افزایش کارآیی Performance وب با گالپ

برای اینکه بفهمیم چه کارهایی می‌تواند سایت یا اپلیکیشن ما را کاراتر کند، از Developer tools با زدن Ctrl+Shift+I درون گوگل کروم، کار خود را شروع می‌کنیم. به برگه‌ی Audits می‌رویم و دکمه‌ی Run را با تنظیمات پیش فرض می‌زنیم. نتایج آن بعد از اندکی صبر، برای من به صورت شکل زیر است:



ما قصد داریم بدانیم گالپ چه ابزاری را برای راه حل‌های داده شده توسط مرورگر دارد؟

۱- کنار هم قرار دادن و فشرده کردن فایل‌های جاوا اسکریپت

پلاگین‌های گالپ برای اینکار، [gulp-concat](#) و [gulp-uglify](#) هستند. آنها را در مسیر ریشه‌ی پروژه نصب می‌کنیم.

```
npm install --save-dev gulp-uglify gulp-concat
```

بعد از نصب، فایل gulpfile.js را به صورت زیر ویرایش و تسک js را به آن اضافه می‌کنیم.

```
// first load all required js files
// concat them in to script.min.js
// and minify it.
gulp.task('js', function() {
  return gulp.src([
    config.bowerDir + '/jquery/dist/jquery.min.js', // این فایل وابستگی فایل‌های زیر است
    config.bowerDir + '/materialize/dist/js/materialize.min.js',
    './resources/js/app.js'
  ])
  .pipe(concat('script.min.js'))
  .pipe(uglify())
  .pipe(size())
  .pipe(gulp.dest('./public/js'));
});
```

خروجی:

```
mmdsharifi@mmdsharifi ~/Work/Lenus $ gulp js
[13:27:20] Using gulpfile ~/Work/Lenus/gulpfile.js
[13:27:20] Starting 'js'...
[13:27:23] all files 206.05 kB
[13:27:23] Finished 'js' after 3.18 s
```

فشرده کردن جاوا اسکریپت، حجم فایل‌ها را ۳۰ تا ۹۰ درصد [کاهش](#) می‌دهد.

۲- حذف سکتورهای بدون استفاده css

همانگونه که در عکس اول آمده، ۹۳ درصد سکتورها در این صفحه بلا استفاده هستند! و این یعنی کاهش فوق العاده زیاد حجم فایل فشرده شده css. به طور معمول، توسعه دهندگان ۸۵ درصد حجم فایل css خود را می‌توانند با این کار [کاهش](#) دهند (البته بیشتر این اتفاق هنگام استفاده از فریک ورک‌هایی مانند bootstrap, ... می‌افتد) برای این کار از پلاگین [gulp-uncss](#) استفاده می‌کنیم. نصب:

```
npm install gulp-uncss --save-dev
```

سپس تسک مربوطه را می‌نویسیم:

```
gulp.task('css', function() {
  return sass(config.sassPath + '/style.scss', {
    style: 'compressed',
    loadPath: [
      './resources/sass',
      config.bowerDir + '/materialize/sass'
    ]
  })
  .on('error', util.log)
  .pipe(size())
  .pipe(uncss({
    html: ['./index.html', './posts.html']
  }))
  .pipe(gulp.dest('./public/css'))
  .pipe(size())
  .pipe(connect.reload());
});
```

نتیجه:

```
mmdsharifi@mmdsharifi ~/Work/Lenus $ gulp css
[14:24:58] Using gulpfile ~/Work/Lenus/gulpfile.js
[14:24:58] Starting 'css'...
[14:25:01] all files 136.68 kB
[14:25:03] all files 17.34 kB
[14:25:03] Finished 'css' after 4.46 s
```

نتیجه فوق العاده است! ۸۷ درصد [کاهش](#) حجم css! اما ممکن است بعضی از استایل‌های شما توسط javascript به صفحه تزریق شوند. در این صورت نباید سکتورهای لازم را حذف کرد و آنها را داخل آرایه‌ی ignore قرار می‌دهیم. برای این منظور، تسک بالا را به صورت زیر به روز رسانی می‌کنیم.


```

gulp.task('css', function() {
  return sass(config.sassPath + '/style.scss', {
    style: 'compressed',
    loadPath: [
      './resources/sass',
      config.bowerDir + '/materialize/sass'
    ]
  })
  .on('error', util.log)
  .pipe(size())
  .pipe(uncss({
    html: ['./index.html', './posts.html'],
    timeout: 2000, // wait for load js files
    ignore: [
      ".waves-ripple ",
      ".drag-target",
      "#sidenav-overlay",
      ".waves-effect",
      ".waves-effect .waves-ripple",
      ".waves-effect.waves-pinck .waves-ripple",
      ".waves-block.waves-light"
    ]
  }))
  .pipe(minifyCss())
  .pipe(size())
  .pipe(gulp.dest('./public/css'))
  .pipe(connect.reload());
});

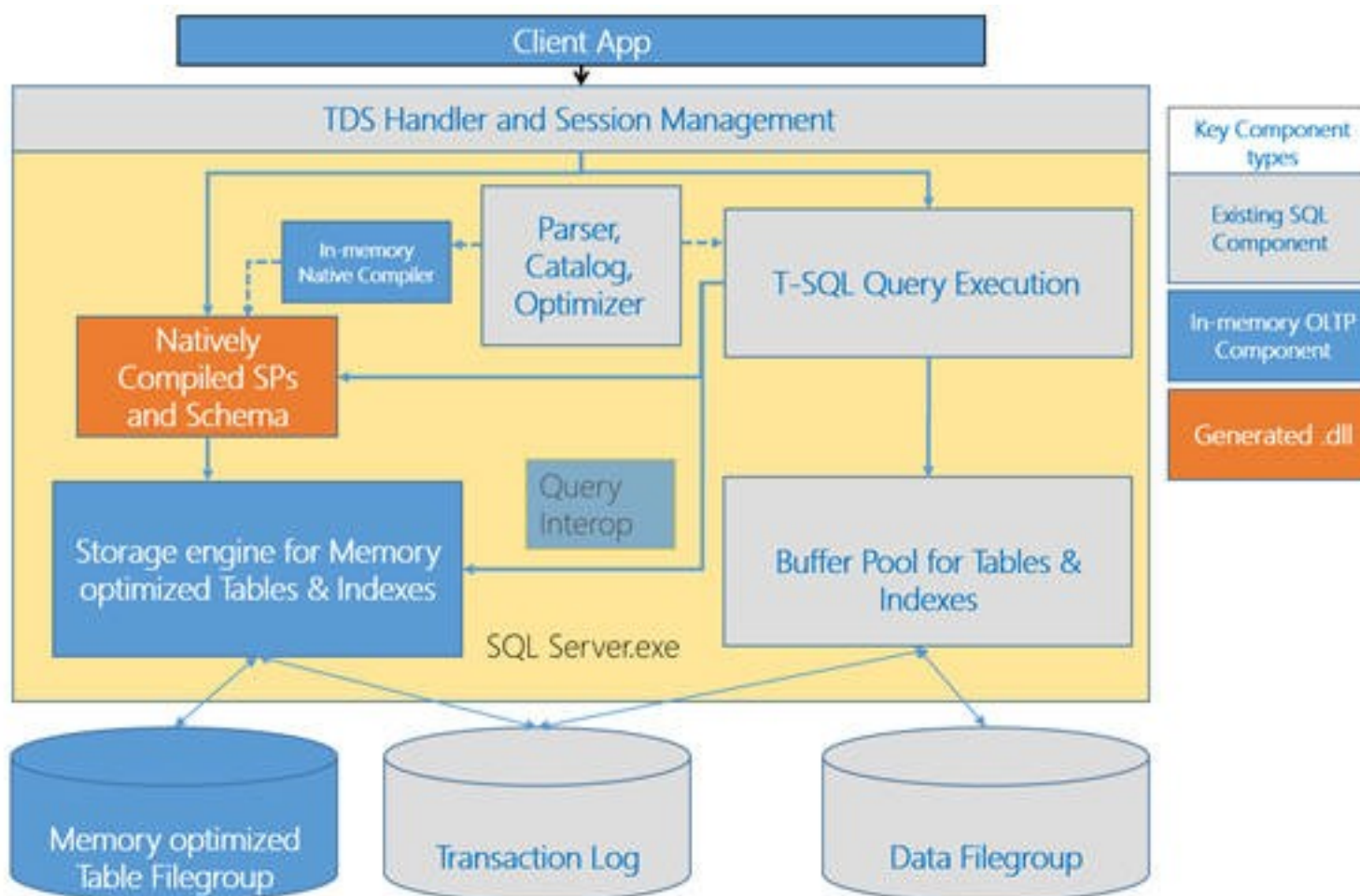
```

بعد از اینکار حجم فایل css من کمی افزایش پیدا کرد ولی بعد از فشرده کردن، نهایتاً به حدود ۱۴KB رسید و این یعنی ۸۷ درصد کاهش حجم فایل css؛ تنها بعد از حذف سکلتورهای اضافی و فشرده کردن آنها. می‌توانید [پلاگین‌های بیشتری](#) را در اینجا ببینید و استفاده کنید.

[Gist](#)

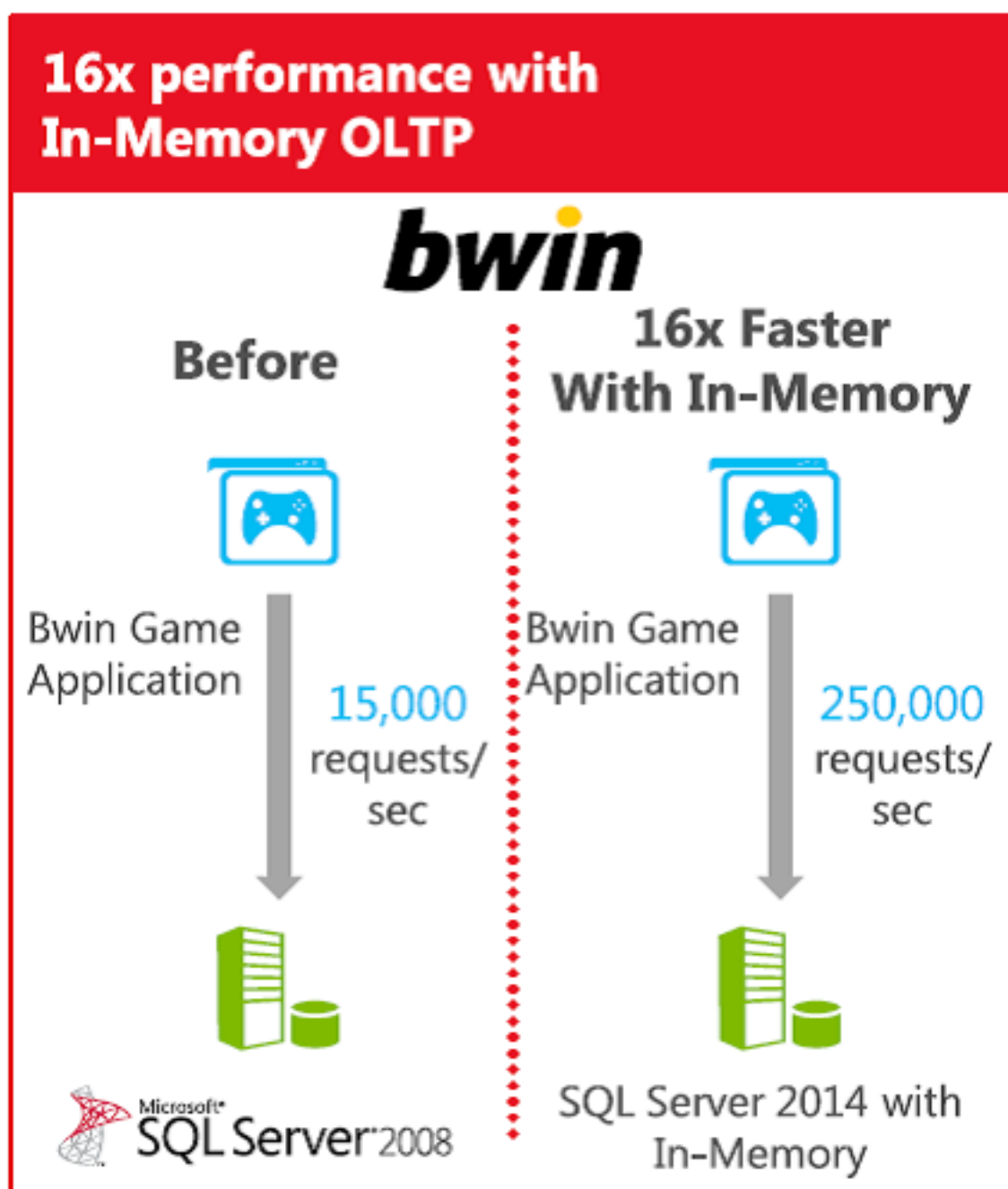
از سال 1970 تا به حال سیستم‌های مدیریت پایگاه داده عملیاتی - ODBMS - مختلفی ایجاد شده‌اند. بعضی از آنها به مرور زمان از بین رفته‌اند و برخی قدرتمندتر شده‌اند. در دهه‌های اخیر بین سیستم‌های مدیریت پایگاه داده عملیاتی، محصولات شرکت‌های اوراکل، مایکروسافت، IBM و SAP از بقیه موفق‌تر بوده‌اند. اما مسلماً در این بین بهترین سیستم مدیریت پایگاه داده، محصول شرکت اوراکل بوده است و سخن گزافی نیست که بگوییم محصول شرکت اوراکل در دهه‌های اخیر در بین محصولات دیگر شرکت‌ها پادشاهی می‌کرده است.

تا حدود 4 سال پیش بین کیفیت oracle db و sql server اختلاف فاحشی وجود داشت. چه از نظر سرعت و چه از نظر دیگر امکانات، اوراکل کاملاً برتر از رقیب خود بود. در نسخه‌ی sql server 2012، امکانات قابل توجهی به محصول شرکت مایکروسافت افزوده شد. از مهمترین این امکانات می‌توان به ویژگی AlwaysOn و ColumnStore Indexها اشاره کرد. امکانات این نسخه باعث شد که اختلاف بین oracle db و sql server تا حدی کاهش یابد. مایکروسافت سرانجام در نسخه‌ی sql server 2014 خود تغییرات اساسی بوجود آورد. مهمترین این تغییرات ایجاد موتور درونی In-Memory OLTP می‌باشد که برای تراکنش‌های درون حافظه بهینه شده است. با استفاده از امکانات این نسخه می‌توان بدون نیاز به دوباره نویسی محصولات، سرعت اجرای کوئری‌های آنها را به طور متوسط ده برابر کرد. در شکل ذیل ساختار جدید sql server مشاهده می‌شود.

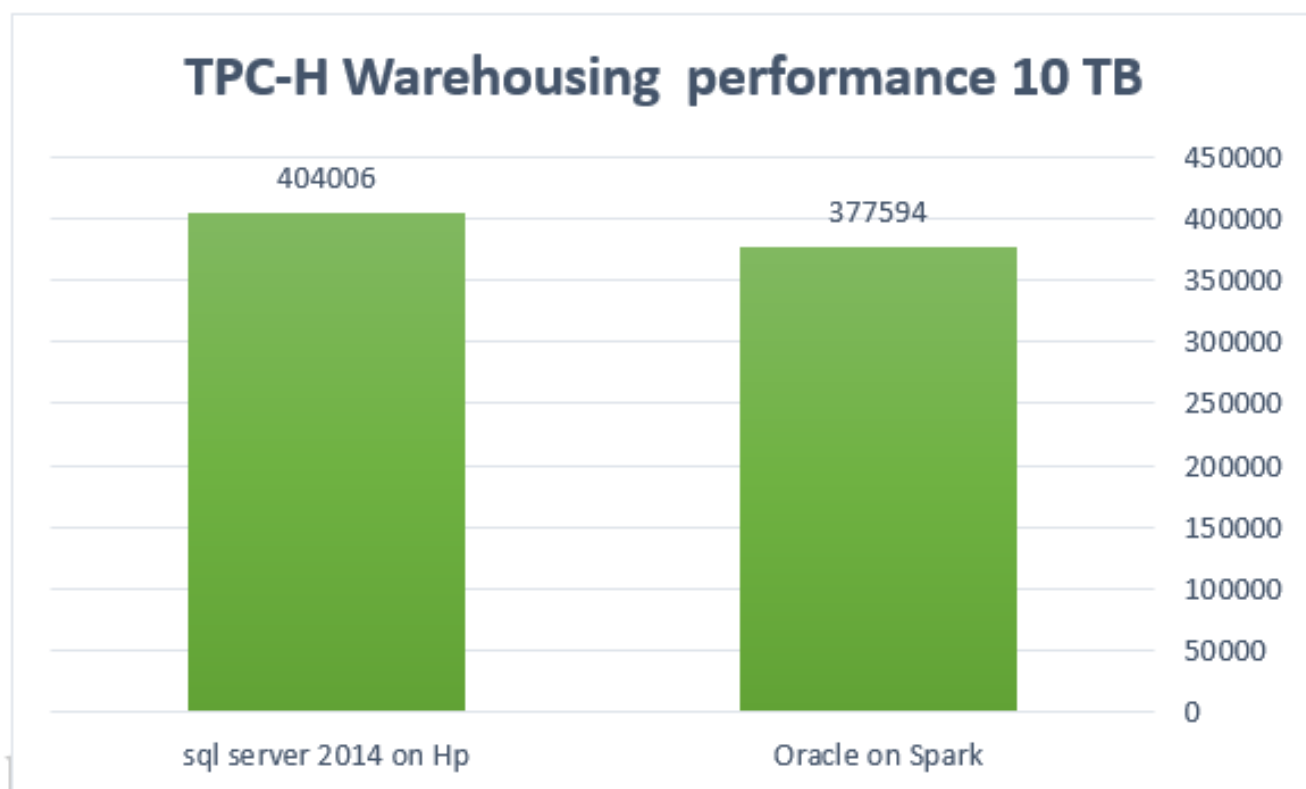





شرکت بوین که یک شرکت مشهور ارائه خدمات آنلاین و پیش بینی بازی‌های ورزشی است و در هر لحظه، کاربران آنلاین بسیاری در وب سایت شرکت، کوئری اجرا می‌کنند، از قابلیت‌های جدید اس کیو ال سرور 2014 استفاده کرده است و با استفاده از این




قابلیت‌ها توانسته سرعت اجرای پرس و جوی مشتریانش را از 15 هزار پرس و جو در ثانیه به 250 هزار پرس و جو در ثانیه برساند. در نتیجه کارایی سرور این شرکت 16 برابر شده است.



در تحقیقی دیگر، یک محقق، با استفاده از قابلیت‌های جدید اس کیو ال سرور 2014 توانسته است دو رکورد جدید را از اجرای کوئری‌های انبار داده ای برای حجم‌های 3 ترابایت و 10 ترابایت و نوع پارتیشن بندی نشده به ثبت برساند و رکوردهای قبلی را که متعلق به اوراکل بوده، بشکند. این محقق توانسته 404005 کوئری نسبتاً سنگین انبار داده‌ای را در پایگاه داده‌ای با 10 ترابایت اطلاعات، در یک ساعت اجرا کند و رکورد قبلی را که متعلق به اوراکل و برابر 377594 کوئری با همین شرایط بوده، بشکند. همچنین هزینه‌ی اجرای کوئری‌های سرور اس کیو ال مذکور برابر 2.04 دلار در هر ساعت اجرای کوئری بوده است. به این معنی است که کمتر از نصف هزینه‌ی مشابه در رکورد ثبت شده‌ی اوراکل که برابر 4.65 دلار در ساعت اجرای کوئری بوده است، هزینه داشته است.



3,000 GB Results									
Rank	Company	System	QphH	Price/QphH	Watts/KQphH	System Availability	Database	Operating System	Date Submitted
1		DL580 G8	461,837	2.04 USD	NR	04/16/14	Microsoft SQL Server 2014 Enterprise Edition	Windows Server 2012 R2 Std Edition	04/15/14
2		SPARC T5-4 Server	409,721	3.94 USD	NR	09/24/13	Oracle Database 11g R2 Enterprise Edition w/Partitioning	Oracle Solaris 11.1	06/07/13
3		Cisco UCS C420 M3 Server	230,119	1.29 USD	NR	12/30/13	Sybase IQ 16.0 SP02	Red Hat Enterprise Linux 6.4	10/31/13

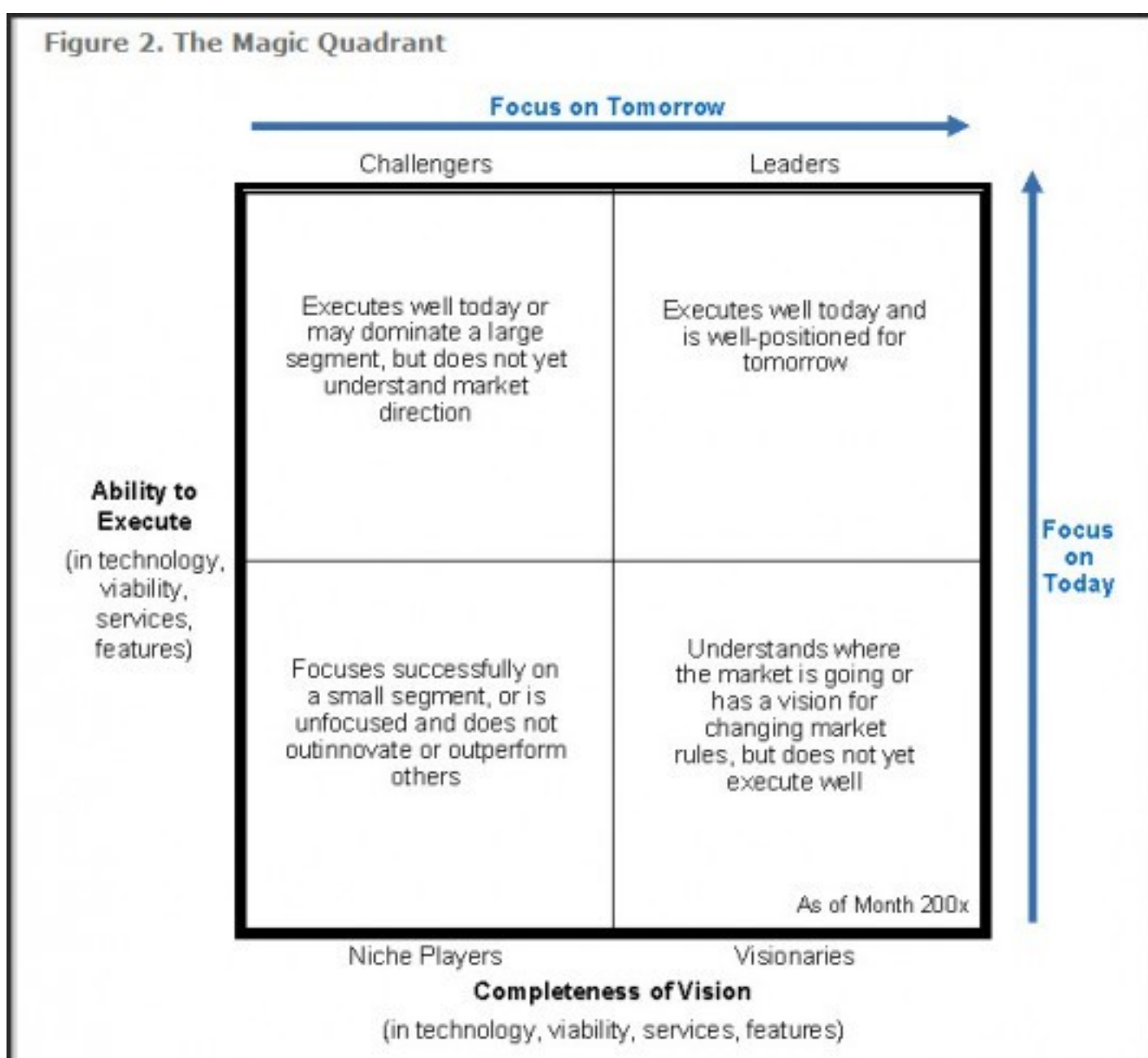
10,000 GB Results									
Rank	Company	System	QphH	Price/QphH	Watts/KQphH	System Availability	Database	Operating System	Date Submitted
1		DL580 G8	404,005	2.34 USD	NR	04/16/14	Microsoft SQL Server 2014 Enterprise Edition	Windows Server 2012 R2 Std Edition	04/15/14
2		SPARC T5-4 Server	377,594	4.65 USD	NR	11/26/13	Oracle Database 11g R2 Enterprise Edition w/Partitioning	Oracle Solaris 11.1	11/25/13
3		HP ProLiant DL980 G7	158,108	6.49 USD	NR	04/15/13	Microsoft SQL Server 2012 Enterprise Edition	Microsoft Windows Server 2012 Standard Edition	04/15/13

http://www.tpc.org/tpch/results/tpch_perf_results.asp?resulttype=noncluster&version=2%¤cyID=0

در واقع اگر بخواهیم سیستم‌های مدیریت پایگاه داده عملیاتی را رتبه بندی کنیم، به جز سرعت، باید عوامل مختلفی را در نظر بگیریم که چنین کاری نیاز به همکاری گروهی بزرگ دارد. خوشبختانه چنین گروه‌هایی وجود دارند و آن قدر معتبر هستند که اکثر شرکت‌های بزرگ به آمارهای آنها استناد می‌کنند. در فناوری‌های مربوط به آی تی، برای رسیدن به معتبرترین نتایج باید به گزارش‌های ارائه شده‌ی شرکت گارتنر رجوع کنیم. گارتنر، شرکت پژوهشی و مشاوره‌ی آمریکایی است، که در زمینه‌ی ارائه

خدمات برون‌سپاری، تحقیق و پژوهش و مشاوره فناوری اطلاعات فعالیت می‌نماید. این شرکت در سال 1979 راه‌اندازی شد و در سال 2014 بیش از 6500 نفر کارمند داشته که در 85 کشور بوده‌اند. در این بین حدود 1500 نفر از آنها در بخش تحقیق و توسعه فعالیت داشته‌اند. همچنین در این سال، درآمد شرکت گارتنر که عمدتاً از طریق مشاوره دادن به شرکت‌های مختلف بوده، بیش از 2 میلیارد دلار در سال 2014 بوده است.

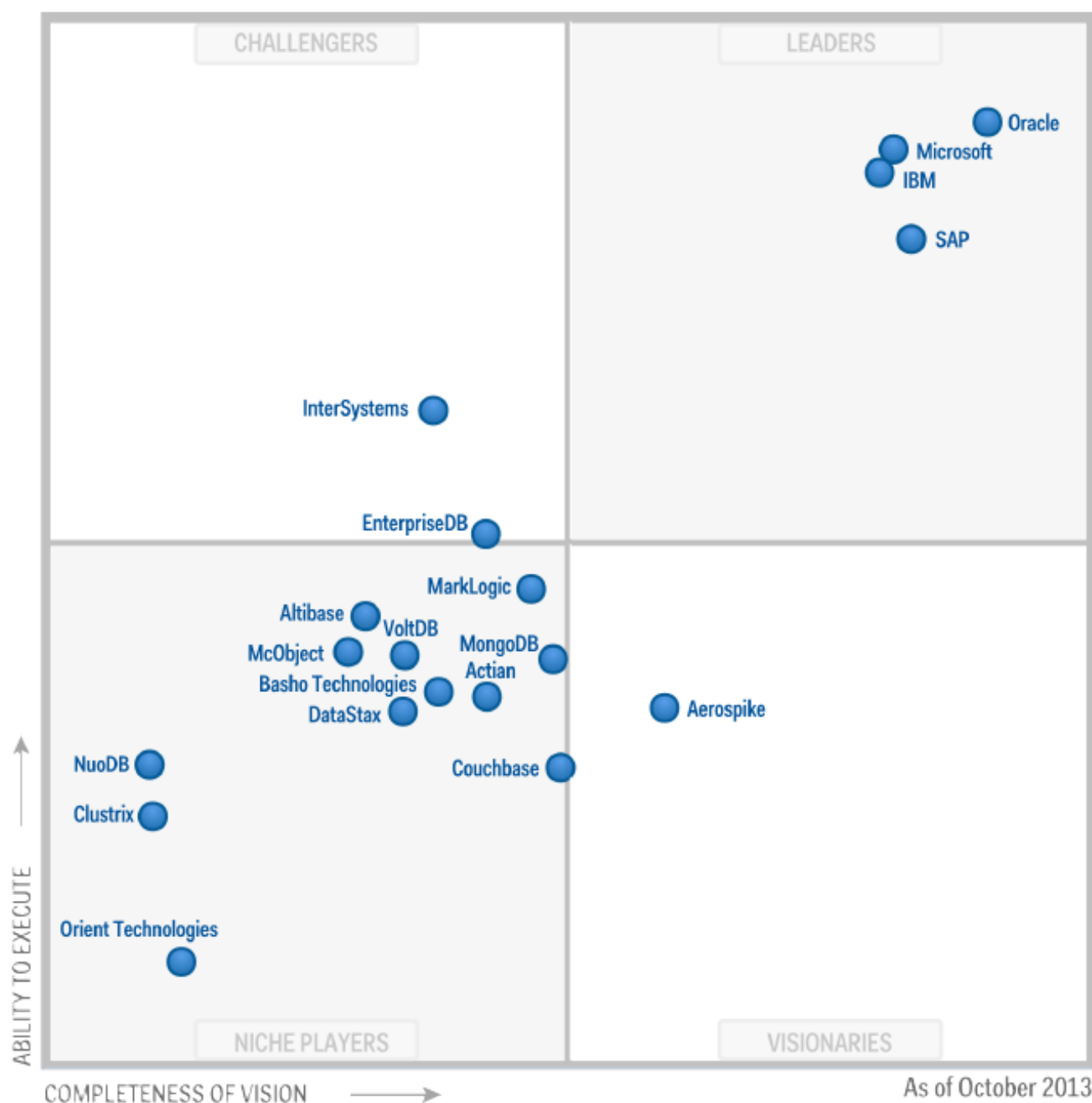
شرکت گارتنر معمولاً خلاصه‌ی نتیجه‌ی بررسی‌های خود را در نمودارهایی خاص به نام مربع جادویی گارتنر ارائه می‌کند. در این نمودار، قابلیت‌های اجرایی که بیانگر کیفیت فعلی محصول هستند، در محور عمودی نمایش داده می‌شوند و از پایین به بالا زیاد می‌شوند. یعنی هر چه محصولی بالاتر باشد، در حال حاضر کیفیت بهتری دارد. محور افقی نمودار بیانگر بصیرت و آینده‌نگری محصول می‌باشد و از چپ به راست زیاد می‌شود. به این ترتیب رهبران یک حوزه‌ی خاص، در ربع بالا و سمت راست مربع جای می‌گیرند.



حال که با نحوه‌ی تفسیر مربع جادویی گارتنر آشنا شدیم، به بررسی نمودارهای مربوط به سیستم‌های مدیریت پایگاه داده عملیاتی در سه سال اخیر می‌پردازیم.

در شکل ذیل می‌بینیم که در سال 2013 و پس از ارائه‌ی نسخه‌ی sql server 2012 توسط مایکروسافت، اوراکل همچنان پیشتاز است و شرکت‌های مایکروسافت، آی بی ام و SAP پس از آن قرار گرفته‌اند. البته در این سال شرکت مایکروسافت فاصله‌ی زیاد قبلی خود را با اوراکل، کم کرده است.

Figure 1. Magic Quadrant for Operational Database Management Systems



Source: Gartner (October 2013)

در سال 2014، شرکت مایکروسافت از نظر آینده نگری و بصیرت، از اوراکل پیشی گرفته ولی هنوز در قابلیت‌های اجرایی عقب‌تر از اوراکل قرار دارد.



اما چند روز پیش در تاریخ 12 اکتبر 2015، شرکت گارتنر گزارشی ارائه کرد که خیلی از فعالان آی تی را شگفت زده کرد. این گزارش در حال حاضر در وب سایت شرکت گارتنر قابل دسترسی است؛ ولی معمولاً گارتنر پس از مدتی آن را از حالت رایگان به پولی تغییر می‌دهد.

[لینک موقت گزارش](#)

در گزارش سال 2015 و پس از ارائه نسخه‌ی 2014 sql server و کاربردی شدن و تست قابلیت‌های آن در عمل توسط شرکت‌های مختلف، بالاخره طلسم چند ده ساله‌ی اوراکل شکسته شده و اگرچه اوراکل نسبت به سال قبل رشد داشته است، ولی sql server مایکروسافت توانسته، هم در قابلیت اجرای فعلی و هم در بصیرت و آینده نگری بالاتر از محصول شرکت اوراکل بایستد. بنابراین عملاً دوران پادشاهی مطلق اوراکل در حوزه‌ی پایگاه‌های داده‌ی عملیاتی به سر رسیده است.

Figure 1. Magic Quadrant for Operational Database Management Systems



در انتها لازم می‌بینم به نکاتی مهم اشاره کنم:

- شرکت اوراکل بر خلاف تصور خیلی از افراد، همانند شرکت‌های مایکروسافت، آی بی ام و ... محصولات گسترده و مختلفی دارد و این بررسی و نتایج تنها در حوزه‌ی سیستم‌های مدیریت پایگاه داده عملیاتی بود.
- بالاتر بودن sql server مایکروسافت از اوراکل در سال 2015 به این معنا نیست که اوراکل نمی‌تواند به جایگاه قبلی خود برگردد؛ بلکه شاید در سال‌های آینده این رتبه بندی باز هم تغییر کند. در واقع این گزارش به این معنا است که فاصله‌ی زیاد قدیم بین sql server و oracle db از بین رفته و در حال حاضر این دو به رقیب سر سختی برای یکدیگر تبدیل شده‌اند.
- وجود رقابت نزدیک بین شرکت‌های بزرگ باعث می‌شود که این شرکت‌ها حداکثر تلاش خود را برای بهتر کردن محصولات خود انجام بدهند و برندگان اصلی این وضعیت، استفاده کنندگان از این محصولات هستند.
- بنده به عنوان نگارنده‌ی این پست شخصا با هر دو محصول oracle db و sql server کار می‌کنم و تلاش کردم که این پست بی طرفانه باشد؛ پس لطفا متعصبانه قضاوت نکنید.

نظرات خوانندگان

نویسنده: علی یگانه مقدم
تاریخ: ۲۰۹ ۱۳۹۴/۰۷/۲۹

اوراکل هم سیستمی مشابه in memory oltp را راه اندازی کرده است که در مقایسه با sql server سه برابر سریعتر بوده است. امروزه اکثر شرکت‌ها چه کوچک و چه بزرگ به سمت sql server حرکت میکنند. این حرکت به دلایل زیر صورت میگیرد برتری sql server بر اوراکل از لحاظ کارایی ارزان‌تر بودن sql server: اوراکل در حال حاضر قیمتی حدود 250 هزار دلار دارد در صورتی که بهترین و گرانترین نسخه sql server قیمتی به مراتب پایین‌تر دارد

نویسنده: دل محسن
تاریخ: ۱۰:۱۶ ۱۳۹۴/۰۷/۲۹

سلام
طبق [این](#) مطلب من فکر نکنم 100 هزار دلار! قیمت sql server باشه.

نویسنده: علی یگانه مقدم
تاریخ: ۱۱:۲ ۱۳۹۴/۰۷/۲۹

بله مثل اینکه اشتباه خوندم
تو [سرچ گوگل](#) سمت راست قیمت‌های مختلف زده شده

نویسنده: زمرد 2020
تاریخ: ۲۱:۱۷ ۱۳۹۴/۰۷/۳۰

سلام و خیلی ممنون از نظرتون.
ولی با توجه به مستند بودن آمار و ارقامی که من دادم، امکانش هست که منبع این سه برابر بودن سرعت رو بفرمایید؟

نویسنده: علی یگانه مقدم
تاریخ: ۲:۲ ۱۳۹۴/۰۸/۰۱

یکی از دوستان که در زمینه sql server کار میکنه این رو به من گفته بود
فکر کنم این مورد را در یک ویدیو دیده بود که sql server بین دو تا سه برابر در واکنشی داده‌ها سریعتر عمل کرده بود.