عنوان: **بدست آوردن نام پروسه‌ای که Clipboard را قفل کرده است**
نویسنده: وحید نصیری
تاریخ: ۱۳۸۸/۰۲/۲۰ ۲۱:۰۷:۰۰
آدرس: www.dotnettips.info
برچسب‌ها: #C

امروز Clipboard سیستم عمل نمی‌کرد و عملیات حیاتی copy/paste از کار افتاده بود! پس از کمی جستجو مشخص شد که به صورت زیر می‌توان نام پروسه‌ای که Clipboard را باز و قفل کرده و مانع عملکرد سایر برنامه‌ها می‌شود، بدست آورد:

```
using System;
using System.Runtime.InteropServices;
using System.Windows.Forms;
using System.Diagnostics;

namespace testWinForms87
{
  class CTestClipboard
  {
      [DllImport("user32.dll", SetLastError = true)]
      private static extern IntPtr GetOpenClipboardWindow();

      [DllImport("user32.dll", SetLastError = true)]
      private static extern uint GetWindowThreadProcessId(
          IntPtr hWnd,
          out uint lpdwProcessId);

      public static void TrySetData()
      {
          try
          {
              Clipboard.SetData(DataFormats.Text, "وحید");
          }
          catch
          {
              IntPtr hwnd = GetOpenClipboardWindow();
              if (hwnd == IntPtr.Zero) return;
              uint pid;
              GetWindowThreadProcessId(hwnd, out pid);
              MessageBox.Show(string.Format("clipboard is locked by: {0}",
                              Process.GetProcessById((int)pid).Modules[0].FileName));
          }
      }
  }
}
```

با استفاده از تابع GetOpenClipboardWindow دستگیره پنجره‌ای که این‌کار را کرده یافت می‌شود و سپس با استفاده از GetWindowThreadProcessId می‌توان id آن پروسه را یافت. سپس با کمک متد Process.GetProcessById امکان بدست آوردن اطلاعات بیشتری از آن پروسه میسر می‌گردد.

به نظر این یک باگ در VPC است.

اگر از MS Virtual PC استفاده می‌کنید و این اتفاق رخ داد، داخل سیستم عاملی که توسط VPC در حال اجرا است، یک متن ساده را کپی کنید. سپس به منوی برنامه VPC ، گزینه edit مراجعه کرده و در ادامه گزینه Paste را انتخاب کنید. به این صورت بدون نیاز به بستن برنامه یا هر عملیات دیگری مشکل برطرف می‌شود.

عنوان: finally در سی شارپ چه زمانی اجرا نمی‌شود؟
نویسنده: وحید نصیری
تاریخ: ۱۳۸۸/۰۳/۰۵ ۲۱:۵۴:۰۰
آدرس: www.dotnettips.info
برچسب‌ها: #C

عموما از کدهای قرار گرفته در قطعه finally جهت آزاد سازی منابع استفاده می‌شود و تضمین شده است که این قطعه همواره اجرا می‌گردد، صرفنظر از اینکه آیا در قطعه try استثنایی رخ داده است یا خیر.

اما مثال زیر را در نظر بگیرید:

```
using System;

namespace testWinForms87
{
  class CTestFinally
  {
      public static void Run()
      {
          try
          {
              TryAndTry();
          }
          catch (Exception exError)
          {
              Console.WriteLine(exError.Message);
          }
          finally
          {
              Console.WriteLine("Finally...!");
          }
          Console.ReadKey();
      }

      static void TryAndTry()
      {
          try
          {
              TryAndTry();
          }
          catch (Exception exError)
          {
              Console.WriteLine(exError.Message);
          }
          finally
          {
              Console.WriteLine("Try: Finally...!");
          }
      }
  }

}
```

در این کد به علت بروز stack overflow هیچگاه به finally نخواهیم رسید.

نظرات خوانندگان

نویسنده: علیرضا
تاریخ: ۰۱:۴۲:۱۸ ۱۳۸۸/۰۳/۰۶

وحید جان، این بار دیگه زیادی باحالش کردی

دهن بنده خدا رو سرویس میکنی تا بالا بیاره، بعد میگی کار نمیکنه؟ :)

نویسنده: پرهام
تاریخ: ۰۹:۵۶:۴۰ ۱۳۸۸/۰۳/۰۶

من زیادی آماتورم ولی با علیرضا موافق هستم. شما سیستم رو دور زدی، پیچوندی!! Finally باید برای کد نویسی استاندارد تست بشه.

نویسنده: salar
تاریخ: ۱۰:۲۱:۱۴ ۱۳۸۸/۰۳/۰۶

در Environment.Exit
هم به علت خروج ناگهانی finally اجرا نمی شه.

نویسنده: مهرزاد قاسمی
تاریخ: ۲۲:۲۹:۵۱ ۱۳۸۸/۰۳/۰۶

جالب بود ، ممنون .

نویسنده: maysam
تاریخ: ۱۵:۴۷:۰۴ ۱۳۸۹/۰۶/۰۱

اصلا جالب نبود : توی Ruby هزار تا tip جالب تر در مورد SEH هستش

عنوان: **رسم گراف**
نویسنده: وحید نصیری
تاریخ: ۱۳۸۸/۰۳/۰۶ ۱۹:۳۴:۰۰
آدرس: www.dotnettips.info
برچسب‌ها: #C

کتابخانه‌های زیادی برای رسم گراف وجود دارند منجمله mxGraph که برای استفاده غیرتجاری رایگان و سورس باز است.
mxGraph نگارش‌های Java ، PHP و JavaScript ایی نیز دارد که به همراه بسته مربوطه ارائه می‌شوند.
پس از دریافت آن، در فولدری به نام dotnet می‌توانید سورس کتابخانه مربوط به دات نت فریم ورک آن‌را دریافت کنید.
فایل پروژه‌ی VS.Net را در آن فولدر نخواهید یافت. حتی آن‌را کامپایل هم نکرده‌اند. (احتمالا به این دلیل که کسی نپرسد این پروژه با چه محصولی تولید شده و آیا لایسنس استفاده از آن را دارید یا خیر. این هم یک روش است ...)
برای کامپایل آن، یک پروژه library جدید را در VS.Net آغاز کرده و پوشه‌های موجود در پوشه‌ی dotnet را به آن افزوده و سپس آن‌را کامپایل کنید تا فایل mxGraph.dll تولید شود.

یک مثال ساده از نحوه‌ی استفاده‌ی آن به صورت زیر است که فایل test.png را تولید خواهد کرد.

```
using System;
using System.Drawing;
using System.Windows.Forms;
using com.mxgraph;
using System.Drawing.Imaging;

       void Test1()
       {
           // Creates graph with model
           mxGraph graph = new mxGraph();
           Object parent = graph.GetDefaultParent();

           // Adds cells into the graph
           graph.Model.BeginUpdate();
           try
           {
               Object v1 = graph.InsertVertex(parent, null, "30 ,80 ,20 ,20 ,"سلام,
"strokeColor=#FFCF8A;fillColor=#FFCF8A;gradientColor=white;fontBold=true;fontFamily=tahoma;rounded=true
;shadow=true;shape=ellipse");
               Object v2 = graph.InsertVertex(parent, null, "!30 ,80 ,150 ,200 ,"دنیای ظالم,
"rounded=true;shadow=true;fontFamily=tahoma");
               Object e1 = graph.InsertEdge(parent, null, "e1", v1, v2, "fontFamily=tahoma");
           }
           finally
           {
               graph.Model.EndUpdate();
           }

           mxCellRenderer.CreateImage(graph, null, 1,
                                      Color.White, true, null).Save("test.png", ImageFormat.Png);
       }
```

و یا اگر قصد داشته باشید که از آن در ASP.Net استفاده کنید، یک generic handler را به پروژه خود افزوده (مثلا ImageHandler.ashx) و کد آن‌را برای مثال به صورت زیر تغییر دهید:

```
using System;
using System.Web;
using com.mxgraph;
using System.Drawing;
using System.Web.Services;
using System.IO;
using System.Drawing.Imaging;

namespace test
{
   [WebService(Namespace = "http://tempuri.org/")]
   [WebServiceBinding(ConformsTo = WsiProfiles.BasicProfile1_1)]
   public class ImageHandler : IHttpHandler
   {
       public void ProcessRequest(HttpContext context)
       {
           // Creates graph with model
           mxGraph graph = new mxGraph();
           Object parent = graph.GetDefaultParent();

           // Adds cells into the graph
           graph.Model.BeginUpdate();
           try
           {
               Object v1 = graph.InsertVertex(parent, null, "30 ,80 ,20 ,20 ,"سلام,
"strokeColor=#FFCF8A;fillColor=#FFCF8A;gradientColor=white;fontBold=true;fontFamily=tahoma;rounded=true
;shadow=true;shape=ellipse");
               Object v2 = graph.InsertVertex(parent, null, "!30 ,80 ,150 ,200 ,"دنیای ظالم,
"rounded=true;shadow=true;fontFamily=tahoma");
               Object e1 = graph.InsertEdge(parent, null, "e1", v1, v2, "fontFamily=tahoma");
           }
           finally
           {
               graph.Model.EndUpdate();
           }

           Image image = mxCellRenderer.CreateImage(graph, null, 1, Color.White, true, null);

           // Render BitMap Stream Back To Client
           MemoryStream memStream = new MemoryStream();
           image.Save(memStream, ImageFormat.Png);

           memStream.WriteTo(context.Response.OutputStream);
       }

       public bool IsReusable
       {
           get
           {
               return false;
           }
       }
```

```
    }
}
```

اکنون نحوه استفاده از این handler در یک صفحه وب به صورت زیر است:

```
<img src="ImageHandler.ashx" />
```

نظرات خوانندگان

نویسنده: وحید نصیری
تاریخ: ۱۳۸۸/۱۲/۲۲ ۱۱:۴۸:۱۳

برای رسم گراف کتابخانه‌های سورس باز دیگری هم موجود است:

graphsharp.codeplex.com

wpfgraph.codeplex.com

www.codeplex.com/quickgraph

و ...

عنوان: **پیدا کردن MAC Address از روی IP**
نویسنده: وحید نصیری
تاریخ: ۱۳۸۸/۰۴/۱۴ ۲۲:۲۵:۰۰
آدرس: www.dotnettips.info
برچسب‌ها: #C

دستگاه کارتخوان تحت شبکه‌ای را تصور کنید که تا دیروز در شبکه دیده می‌شد اما امروز ناپدید شده! دیگر نمی‌شود با آن ارتباط برقرار کرد و توسط برنامه نوشته شده آن‌را تخلیه کرد. به صورت پیش فرض یک IP ثابت به این دستگاه‌ها انتساب داده می‌شود مثلا 192.168.1.100 . حال اگر در شبکه این IP رزرو نشود ممکن است DHCP این IP را به کامپیوتر دیگری بدهد یا روتری به اشتباه از آن استفاده کند. الان چطور باید تشخیص داد که این IP توسط چه وسیله‌ای در حال استفاده است؟
برای بدست آوردن MAC Address یک دیوایس در شبکه از دستور خط فرمان ARP می‌توان استفاده کرد. برای مثال:

```
arp -a
```

و یا :

```
arp -a findstr 192.168.1.100
```

اگر در این لیست‌ها MAC Address آی پی مورد نظر شما یافت نشد، ابتدا آن IP را ping کنید و بعد دستورات فوق را استفاده نمائید. علت هم این است که دستورات فوق تنها ARP table ایی را بر اساس ارتباطات ایستگاه کاری فعلی با شبکه نمایش می‌دهند و اگر کامپیوتر فعلی هیچ ارتباطی با IP مورد نظر نداشته باشد، خروجی خاصی را نمایش نخواهد داد.

حال اگر با برنامه نویسی بخواهیم این‌کار را انجام دهیم با استفاده از تابع API زیر نیز می‌توان به همین نتیجه رسید:

```
[System.Runtime.InteropServices.DllImport("Iphlpapi.dll", EntryPoint = "SendARP")]
        internal extern static Int32 SendArp(Int32 destIpAddress, Int32 srcIpAddress, byte[]
macAddress, ref Int32 macAddressLength);

        public static System.Net.NetworkInformation.PhysicalAddress GetMacFromIP(System.Net.IPAddress
IP)
        {
            if (IP.AddressFamily != System.Net.Sockets.AddressFamily.InterNetwork)
                throw new ArgumentException("supports just IPv4 addresses");

            Int32 addrInt = IpToInt(IP);
            Int32 srcAddrInt = 0;
            byte[] mac = new byte[6]; // 48 bit

            int length = mac.Length;
            int reply = SendArp(addrInt, srcAddrInt, mac, ref length);

            byte[] emptyMac = new byte[12];

            if (reply != 0)
            {
                //No MAC Address found for the IP Address
                return new System.Net.NetworkInformation.PhysicalAddress(emptyMac);
            }
            return new System.Net.NetworkInformation.PhysicalAddress(mac);
        }

        private static Int32 IpToInt(System.Net.IPAddress IP)
        {
            byte[] bytes = IP.GetAddressBytes();
            return BitConverter.ToInt32(bytes, 0);
        }
```

مک آدرسی که در حال استفاده از IP ما بود:

```
00:0F:8F:F1:EE:40
```

مک آدرس واقعی دستگاه کارتخوان:

```
00.64.00.00.5E.7A
```

این تداخل رخ داده، باعث شده بود که ارتباط با دستگاه کارتخوان قطع شود. همانطور که می‌دانید سه بایت اول هر مک آدرس می‌تواند بیانگر شرکت سازنده‌ای باشد. 00:0F:8F مربوط به روترهای سیسکو است.

نظرات خوانندگان

نویسنده: ali
تاریخ: ۱۳۹۱/۰۴/۰۹ ۱۵:۳۷

khob bod mamnon az shoma

عنوان: **رمزگشایی عنوان یک ایمیل فارسی دریافت شده**
نویسنده: وحید نصیری
تاریخ: ۱۳۸۸/۰۴/۲۰ ۱۸:۴۴:۰۰
آدرس: www.dotnettips.info
برچسب‌ها: #C

گوگل اجازه‌ی فعال کردن POP3 را روی اکانت‌ها GMail می‌دهد. فرض کنید با استفاده از یکی از کلاینت‌های POP3 دات نت می‌خواهیم ایمیل‌ها را با برنامه نویسی دریافت کنیم (و مثلا از Outlook استفاده نکنیم). اکنون به نظر شما عنوان دریافت شده زیر چه معنایی دارد؟

```
=?UTF-8?B?QW5hbHl0aWNzIHZhaGlkbmFzaXJpLmJsb2dzcG90LmNvbSAyMDA4MTIyNiAo2KLZhdin?= =?UTF-
8?B?2LEg2LPYp9mK2Kop?=
```

برای درک اتفاق رخ داده باید به RFC های مربوطه مراجعه کرد (RFC-2822 و RFC-2047). مطابق استانداردهای ذکر شده، هدر ارسالی یک ایمیل همواره باید از حروف اسکی تشکیل شود. حال اگر عنوان ایمیل که جزئی از هدر را تشکیل می‌دهد از حروف غیر اسکی تشکیل شد، حتما باید یک لایه encoding روی آن‌ها صورت گیرد. دو حالت تعریف شده در این‌جا مطابق استاندارد میسر است:

الف) Quoted Printable : در این حالت عنوان با =?utf-8?Q شروع می‌شود.

ب) Base64 : در این روش عنوان با =?utf-8?B شروع خواهد شد.

روش متداول، روش ب است که نسبت به روش الف فشرده‌تر می‌باشد. در این حالت برای درک معنای قسمت‌های مختلف رشته دریافت شده باید به الگوی زیر مراجعه کرد:

```
=?charset?encoding?
```

EncodedText

```
?=
```

در این‌جا charset بیانگر نحوه encoding متن اصلی است که بر روی آن الگوریتم base64 اعمال شده.

در رشته طولانی فوق که در ابتدای مقاله به آن اشاره شده، عنوان به دو قسمت تجزیه شده. یا به عبارتی دوبار الگوی فوق در آن تکرار شده است که باید EncodedText های آن‌ها را یافت و سپس آن‌ها را با توجه به charset مربوطه از حالت base64 به یک رشته معمولی تبدیل نمود.

```
//using System.Text;
public static string Base64ToString(string charset, string encodedString)
{
  //تبدیل بیس 64 به آرایه‌ای از بایت‌ها
  byte[] buffer = Convert.FromBase64String(encodedString);
  //تبدیل آرایه‌ای از بایت‌ها به رشته با توجه به انکدینگ مربوطه
  return Encoding.GetEncoding(charset).GetString(buffer);
}
```

اکنون عنوان صحیح ایمیل فوق به صورت زیر قابل دریافت خواهد بود:

```
string subject = Base64ToString("utf-8",
"QW5hbHl0aWNzIHZhaGlkbmFzaXJpLmJsb2dzcG90LmNvbSAyMDA4MTIyNiAo2KLZhdin2LEg2LPYp9mK2Kop");
```

عنوان: **فشرده سازی با فرمت 7z**
نویسنده: وحید نصیری
تاریخ: ۱۳۸۸/۰۵/۰۶ ۲۱:۱۴:۰۰
آدرس: www.dotnettips.info
برچسب‌ها: #C

جی‌میل هر ایمیلی را که به همراه آن یک فایل اجرایی پیوست شده باشد برگشت می‌زند. Zip کردن آن هم فایده ندارد چون محتویات فایل‌های zip را هم بررسی می‌کند! فقط به نظر فرمت rar و همچنین 7z را بررسی نمی‌کند (احتمالا با مجوز آن مشکل دارد).
قوی‌ترین برنامه سورس بازی که این فرمت را پشتیبانی می‌کند، برنامه 7zip است و خوشبختانه محصور کننده‌هایی نیز جهت کار با کتابخانه‌های این برنامه برای دات نت فریم ورک موجود است. برای مثال:

SevenZipSharp

مزیت استفاده از این کتابخانه این است که اغلب فرمت‌های پر کاربرد را نیز پشتیبانی می‌کند (شامل rar ، gz ، zip و ...).
برای استفاده از آن به فایل‌های 7z.dll و SevenZipSharp.dll نیاز خواهید داشت. 7z.dll از برنامه 7zip گرفته شده و SevenZipSharp.dll هم محصور کننده دات نتی آن است.

مثالی در مورد فشرده سازی با فرمت 7z با کمک کتابخانه‌های نامبرده شده:

```
using SevenZip;
using System.Windows.Forms;
using System;

   class C7Z
   {
       public static void Compress7Z(string filePath, string outPath)
       {
           SevenZipCompressor.SetLibraryPath(String.Format(@"{0}\7z.dll", Application.StartupPath));
           SevenZipCompressor cmp = new SevenZipCompressor
                                       {
                                           ArchiveFormat = OutArchiveFormat.SevenZip,
                                           CompressionMethod = CompressionMethod.Lzma,
                                           CompressionMode = CompressionMode.Create,
                                           CompressionLevel = CompressionLevel.High,
                                           VolumeSize = 0
                                       };
           cmp.CompressFiles(outPath, filePath);
       }

   }
```

```
C7Z.Compress7Z(@"C:\test\test.txt", @"C:\test\test.7z");
```

مثال‌های بیشتری را با دریافت سورس SevenZipSharp می‌توانید مشاهده کنید.

نظرات خوانندگان

نویسنده: SirAsad
تاریخ: ۱۳۸۸/۰۵/۰۷ ۱۴:۱۹:۴۲

آقای نصیری ببخشید که این سوال ربطی به پست شما نداره.
آقا ما یه صفحه داریم که توش ما یدونه کمبو , 3 تا repeater و یدونه هم Detail view رو Bind می کنیم . با این حتما متوجه شدین که سرعت لود خیلی کمه .

به نظر شما برای performance چی کار کنیم ؟
آدرس صفحه هم اینه : http://iranjarchi.com/AdverShow.aspx?id=93

موفق باشید

نویسنده: وحید نصیری
تاریخ: ۱۳۸۸/۰۵/۰۷ ۱۵:۱۱:۱۸

از ترکیب این دو مطلب استفاده کنید:
http://www.dotnettips.info/2009/07/blog-post_06.html
http://www.dotnettips.info/2009/07/blog-post_13.html

هم حالت اجکسی پیدا می‌کنه و هم فشرده سازی اعمال میشه.

عنوان: **استفاده از کلاس‌های My* وی بی در C#**
نویسنده: وحید نصیری
تاریخ: ۱۳۸۸/۰۵/۱۵ ۲۱:۱۸:۰۰
آدرس: www.dotnettips.info
برچسب‌ها: C#

یک سری قابلیت در فضای نام Microsoft.VisualBasic وجود دارد که به ظاهر سایر برنامه نویسان دات نت از آن محروم هستند. برای مثال My.Computer.Network.IsAvailable برای بررسی اینکه آیا اتصال به شبکه برقرار است یا My.Computer.Audio.Play جهت نواختن یک فایل صوتی، کلاس‌های My.Webservices، My.User، My.Computer، My.Application و My.DataSources و امثال آن.

از این فضای نام در C# یا تمامی زبان‌های دیگر دات نت نیز می‌توان استفاده کرد. تنها کافی است ارجاعی را به Microsoft.VisualBasic.dll اضافه کنید، در ادامه using Microsoft.VisualBasic.MyServices و سپس معادل‌های آن‌ها به صورت زیر خواهند بود:

```
'VB code
Me.cbNetworked.Checked = My.Computer.Network.IsAvailable

// C# code
MyComputer mc = new MyComputer();
cbNetworked.Checked = mc.Network.IsAvailable;

'VB code
Me.cbAltKey.Checked = My.Computer.Keyboard.AltKeyDown
Me.cbCapsLock.Checked = My.Computer.Keyboard.CapsLock
Me.cbCtrlKey.Checked = My.Computer.Keyboard.CtrlKeyDown
' etc...

// C# code
MyComputer mc = new MyComputer();
this.cbAltKey.Checked = mc.Computer.Keyboard.AltKeyDown;
this.cbCapsLock.Checked = mc.Computer.Keyboard.CapsLock;
this.cbCtrlKey.Checked = mc.Computer.Keyboard.CtrlKeyDown;
' etc...

'VB code
My.Computer.Audio.Play(lbClips.SelectedItem)

// C# code
MyAudio ma = new MyAudio();
ma.Play(lbClips.SelectedItem);

'VB code
My.Computer.Info.TotalPhysicalMemory
'etc...

// C# code
MyComputer mc = new MyComputer();
mc.Info.TotalPhysicalMemory;
// etc...
```

هر چند در واقعیت این فضای نام تنها محصور کننده‌ی یک سری از کلاس‌های دیگر دات نت است. برای مثال اگر به سورس دات نت فریم ورک مراجعه کنید، My.Computer.Network.IsAvailable آن دقیقا محصور کننده‌ی متد NetworkInterface.GetIsNetworkAvailable واقع شده در فضای نام استاندارد System.Net.NetworkInformation است.

نظرات خوانندگان

نویسنده: وحید نصیری
تاریخ: ۱۳۸۸/۰۵/۱۵ ۲۲:۵۰:۳۱

به روز رسانی!

مطلب فوق در مورد VS2005 صحیح است. در VS2008 به صورت زیر باید عمل کرد:

```
using Microsoft.VisualBasic.Devices;
Computer mc = new Computer();
bool isAvailable = mc.Network.IsAvailable;
```

نویسنده: Anonymous
تاریخ: ۱۳۸۸/۰۵/۱۶ ۰۰:۲۰:۰۱

مثل همیشه عالی بود، خسته نباشی

عنوان: **کامپایل پویای کد در دات نت**
نویسنده: وحید نصیری
تاریخ: ۱۳۸۸/۰۶/۲۵ ۱۶:۱۷:۰۰
آدرس: www.dotnettips.info
گروه‌ها: #C

در دات نت فریم ورک امکان کامپایل پویای یک قطعه کد دریافت شده از یک رشته، توسط فضای نام CodeDom مهیا است که قدرت قابل توجهی را در اختیار برنامه نویس قرار می‌دهد.

مثال یک:
رشته زیر را کامپایل کرده و تبدیل به یک فایل exe کنید:

```
string source =
           @"
               namespace Foo
               {
                    public class Bar
                    {
                        static void Main(string[] args)
                        {
                            Bar.SayHello();
                        }

                        public static void SayHello()
                        {
                            System.Console.WriteLine(""Hello World"");
                        }
                    }
               }
            ";
```

روش انجام کار به همراه توضیحات مربوطه به صورت کامنت:

```
using System;
using System.Collections.Generic;
//دو فضای نامی که برای این منظور اضافه شده‌اند
using Microsoft.CSharp;
using System.CodeDom.Compiler;

namespace compilerTest
{
   class Program
   {
       static void compileIt1()
       {
           //سورس کد ما جهت کامپایل
           string source =
           @"
               namespace Foo
               {
                    public class Bar
                    {
                        static void Main(string[] args)
                        {
                            Bar.SayHello();
                        }

                        public static void SayHello()
                        {
                            System.Console.WriteLine(""Hello World"");
                        }
                    }
               }
            ";

           //تعیین نگارش کامپایلر مورد استفاده
           Dictionary<string, string> providerOptions = new Dictionary<string, string>
                {
                    {"CompilerVersion", "v3.5"}
```

```
                };
            //تعیین اینکه کد ما سی شارپ است
            CSharpCodeProvider provider = new CSharpCodeProvider(providerOptions);

            //تعیین اینکه خروجی یک فایل اجرایی است بعلاوه مشخص سازی محل ذخیره سازی فایل نهایی
            CompilerParameters compilerParams = new CompilerParameters
            {
                OutputAssembly = "D:\\Foo.EXE",
                GenerateExecutable = true
            };

            //عملیات کامپایل در اینجا صورت می‌گیرد
            CompilerResults results = provider.CompileAssemblyFromSource(compilerParams, source);

            //اگر خطایی وجود داشته باشد نمایش داده خواهد شد
            Console.WriteLine("Number of Errors: {0}", results.Errors.Count);
            foreach (CompilerError err in results.Errors)
            {
                Console.WriteLine("ERROR {0}", err.ErrorText);
            }
        }

        static void Main(string[] args)
        {
            compileIt1();

            Console.WriteLine("Press a key...");
            Console.ReadKey();
        }
    }
}
```

مثال 2:

کد مورد نظر را به صورت یک فایل dll کامپایل کنید.

برای این منظور تمامی مراحل مانند قبل است فقط GenerateExecutable ذکر شده به false تنظیم شده و نام خروجی نیز به foo.dll باید تنظیم شود.

مثال 3:

کد مورد نظر را در حافظه کامپایل کرده (خروجی dll یا exe نمی‌خواهیم)، سپس متد SayHello آن را به صورت پویا فراخوانی نموده و خروجی را نمایش دهید.

در این حالت روش کار همانند مثال 1 است با این تفاوت که GenerateInMemory = true و GenerateExecutable = false تنظیم می‌شوند. همچنین جهت دسترسی به متد کلاس ذکر شده، از قابلیت‌های ریفلکشن موجود در دات نت فریم ورک استفاده خواهد شد.

```
using System;
using System.Collections.Generic;
using Microsoft.CSharp;
using System.CodeDom.Compiler;
using System.Reflection;

namespace compilerTest
{
   class Program
   {
       static void compileIt2()
       {
           //سورس کد ما جهت کامپایل
           string source =
           @"
               namespace Foo
               {
                    public class Bar
                    {
                        static void Main(string[] args)
                        {
                            Bar.SayHello();
                        }

                        public static void SayHello()
                        {
```

```
                            System.Console.WriteLine(""Hello World"");
                        }
                    }
                }
            ";

            //تعیین نگارش کامپایلر مورد استفاده
            Dictionary<string, string> providerOptions = new Dictionary<string, string>
                 {
                     {"CompilerVersion", "v3.5"}
                 };
            //تعیین اینکه کد ما سی شارپ است
            CSharpCodeProvider provider = new CSharpCodeProvider(providerOptions);

            //نحوه تعیین مشخص سازی کامپایل در حافظه
            CompilerParameters compilerParams = new CompilerParameters
            {
                GenerateInMemory = true,
                GenerateExecutable = false
            };

            //عملیات کامپایل در اینجا صورت می‌گیرد
            CompilerResults results = provider.CompileAssemblyFromSource(compilerParams, source);

            // اگر خطایی در کامپایل وجود نداشت متد دلخواه را فراخوانی می‌کنیم
            if (results.Errors.Count == 0)
            {
                //استفاده از ریفلکشن برای دسترسی به متد و فراخوانی آن
                Type type = results.CompiledAssembly.GetType("Foo.Bar");
                MethodInfo method = type.GetMethod("SayHello");
                method.Invoke(null, null);
            }
        }


        static void Main(string[] args)
        {
            compileIt2();

            Console.WriteLine("Press a key...");
            Console.ReadKey();
        }
    }
}
```

نکته: نحوه‌ی استفاده از اسمبلی‌های دیگر در رشته سورس کد خود

مثال:

اگر رشته سورس ما به صورت زیر بوده و از اسمبلی System.Drawing.Dll نیز کمک گرفته باشد،

```
string source =
            @"
              namespace Foo
              {

                  public class Bar
                  {
                      static void Main(string[] args)
                      {
                          Bar.SayHello();
                      }

                      public static void SayHello()
                      {
                          System.Console.WriteLine(""Hello World"");
                          var r = new System.Drawing.Rectangle(0,0,100,100);
                          System.Console.WriteLine(r);
                      }
                  }
              }
            ";
```

هنگام کامپایل آن توسط روش مثال یک، با خطای زیر مواجه خواهیم شد.

```
Number of Errors: 1
```

```
ERROR The type or namespace name 'Drawing' does not exist in the namespace 'System' (are you missing an assembly reference?)
```

برای رفع این مشکل و معرفی این اسمبلی، سطر زیر باید پس از تعریف compilerParams اضافه شود.

```
compilerParams.ReferencedAssemblies.Add("System.Drawing.Dll");
```

اکنون کد کامپایل شده و مشکلی نخواهد داشت.
نمونه‌ای دیگر از این دست، استفاده از LINQ می‌باشد. در این حالت اسمبلی System.Core.Dll نیز به روش ذکر شده باید معرفی گردد تا مشکلی در کامپایل کد رخ ندهد.

کاربردها:
1- استفاده در ابزارهای تولید کد (برای مثال در برنامه Linqer از این قابلیت استفاده می‌شود)
2- استفاده‌های امنیتی (ایجاد روش‌های تولید یک سریال به صورت پویا و کامپایل پویای کد مربوطه در حافظه‌ای محافظت شده)
3- استفاده جهت مقاصد محاسباتی پیشرفته
4- دادن اجازه‌ی کد نویسی به کاربران برنامه‌ی خود (شبیه به سیستم‌های ماکرو و اسکریپت نویسی موجود)
و ...

نظرات خوانندگان

نویسنده: میثم جوادی
تاریخ: ۲۰:۲۳:۵۹ ۱۳۸۸/۰۶/۲۵

تو VB هم همچین چیزی به اسم eval بود و به خاطر این من دنبال یه همچین معادلی تو #C بودم که پیدا نکردم!(البته تو برنامه نویس یکی از قدیمی ها گفت قراره تو 4 بیاد دیگه دنبالش نگشتم.)

ممنون.

نویسنده: وحید نصیری
تاریخ: ۲۰:۳۳:۴۹ ۱۳۸۸/۰۶/۲۵

البته این فراتر است از eval وی بی و شما کد کامل رو می‌تونید توسط آن کامپایل کنید.
برای کامپایل سورس از نوع VB.Net هم VBCodeProvider بجای CSharpCodeProvider در مثال‌های بالا قابل استفاده است.

نویسنده: Mohammad Shams Javi
تاریخ: ۲۲:۰۹:۱۴ ۱۳۸۸/۰۶/۲۵

سلام

خیلی مفید و جالب بود، خصوصا که قابلیتها و امکانات Reflection و CodeDom، بحث روز امنیت نرم‌افزارهای .Net مثل انواع روشهای SelfObfuscation و SelfPatching و ... هستند.

نویسنده: Kianoosh
تاریخ: ۰۴:۴۹:۵۶ ۱۳۸۸/۰۶/۲۶

با سلام

مقاله خیلی مفید و زیبائی بود. به زیبائی در این مقله در باره تولید خودکار کد و استفاده از reflection صحبت کرده بودید.

مثالهای انتهای مقاله هم بسیار جالب بودند.
لطفا باز هم درباره reflection و CodDom مطالب بیشتری بنویسید.
ممنون.

نویسنده: ..::A-3BT::..
تاریخ: ۲۱:۱۰:۵۷ ۱۳۸۸/۰۹/۰۳

ممنون ، جناب نصیری من یه مشکلی دارم اینکه یه اسمبلی که بصورت جداست می خوام رفرنس کنم بهش ولی نتونستم میشه یه راهنمایی بکنید؟
بسیار ممنون

نویسنده: وحید نصیری
تاریخ: ۱۰:۴۳:۴۹ ۱۳۸۸/۰۹/۰۴

سلام
شما در حین کامپایل اولیه در قسمت ReferencedAssemblies.Add مسیر کامل اسمبلی مورد نظر را ذکر کنید تا عملیات کامپایل با موفقیت به پایان برسد.
هنگام اجرای پویای کد، اسمبلی مورد نظر یا باید در GAC باشد یا کنار فایل اجرایی اصلی یا سایر مسیرهای استانداردی که دات

نت فریم ورک در حین اجرا به دنبال اسمبلی‌ها می‌گردد.

نویسنده: reza
تاریخ: ۱۸:۳۰:۳۹ ۱۳۸۸/۰۹/۱۷

سلام
آقای نصیری من چندتا مشکل با این موضوع دارم اگه کمکم کنید ممنون میشم.
1- من زمان اجرای فایل تولید شده یه فرم ظاهر بشه در حالی که اضافه بر فرم یه صفحه Command هم باز میشه! چطور میشه کاری کرد که این صفحه باز نشه؟
2- چطور میشه برای فایل های اجرایی تولید شده آیکون در نظر گرفت.
3- توی کدی که قرار است توسط CodeDom کامپایل شود یه متغیر از نوع آرایه ای از Byte دارم که باید توسط برنامه اصلی مقدار دهی شود روشی که استفاده کردم به اینصورته
";{" + [byte[] a = new byte[]{"+MyByteArray[0] + "," + MyByteArray[1"
ولی این روش هم سرعت برنامه رو پایین میاره هم محدودیت برای تعداد کاراکتری که باید کامپایل شوند ایجاد میکند. شما چه روشی رو برای مقدار دهی این آرایه پیشنهاد میکنید.

نویسنده: وحید نصیری
تاریخ: ۱۹:۴۶:۳۰ ۱۳۸۸/۰۹/۱۷

1- احتمالا برای اجرا از کلاس Process مربوط به فضای نام System.Diagnostics کمک می‌گیرد. اگر اینطور است باید خاصیت process.StartInfo.CreateNoWindow به true تنظیم شود.
2- متد parameters.EmbeddedResources.Add هم موجود است. کمی در مورد آن تحقیق کنید.
3- سرعت این روش فقط در حد زمان انجام کامپایل کامل، کند است؛ مابقی آن تفاوتی با اجرای یک برنامه واقعی ندارد. ضمنا محدودیتی هم من ندیدم. محدودیت‌های آن همان محدودیت‌های برای مثال کامپایلر سی شارپ است. مثلا یک سطر نباید از 16777214 کاراکتر بیشتر باشد و امثال آن.
ضمنا استفاده از + هنگام چسباندن رشته‌ها به هم در حجم کم تاثیر آنچنانی روی کارآیی ندارد. ولی اگر تعداد زیاد است بهتر است از StringBuilder استفاده شود.

نویسنده: ..::A-3BT::..
تاریخ: ۱۷:۵۲:۴۲ ۱۳۸۸/۰۹/۱۸

راستی گزینه هایی که برای Optimize کردن کد کامپایل شده هست ، تو این روش هم قابل استفاده است؟

نویسنده: وحید نصیری
تاریخ: ۱۸:۵۷:۰۳ ۱۳۸۸/۰۹/۱۸

بله. خاصیت CompilerOptions آن‌را مساوی "/optimize" قرار دهید و یا سایر موارد مورد نظر.

نویسنده: علی یگانه مقدم
تاریخ: ۱۵:۳۱ ۱۳۹۳/۰۹/۲۲

اوایل کارم با سی شارپ بود ، یک پروژه توی codeproject قرار گرفته بود که یک برنامه برای ساخت slideshow با تمامی امکانات لازم ساخته بود که خروجیش هم exe بود
وارد کردن و ترتیب تصاویر و موسیقی و تکرار و حرکت خودکار یا با کلیک ماوس تصاویر و تنظیمات دیگه
ذخیره کار به صورت پروژه و بازیابی اون به صورت serialization
و همینطور کد خروجی exe
نحوه کدنویسی شکیل و ساخت یافتش به قدری کامل و شیوا بود که باعث شد بیش از پیش به این زبان هم علاقه مند بشم و هم به برنامه نویسی
خدا طرف رو خیر بده ، یه زندگی رو با این کدش دگرگون کرد

عنوان: **بررسی واژه کلیدی static**
نویسنده: وحید نصیری
تاریخ: ۱۳۸۸/۰۶/۲۸ ۱۶:۲۴:۰۰
آدرس: www.dotnettips.info
برچسب‌ها: #C

تفاوت بین یک کلاس استاتیک، متدی استاتیک و یا متغیر عضو استاتیک چیست؟ چه زمانی باید از آن‌ها استفاده کرد و لزوم بودن آن‌ها چیست؟

برای پاسخ دادن به این سؤالات باید از نحوه‌ی تقسیم بندی حافظه شروع کرد.

RAM برای هر نوع پروسه‌ای که در آن بارگذاری می‌شود به سه قسمت تقسیم می‌گردد: Heap ، Stack و Static (استاتیک در دات نت در حقیقت قسمتی از Heap است که به آن High Frequency Heap نیز گفته می‌شود).

این قسمت استاتیک حافظه، محل نگهداری متدها و متغیرهای استاتیک است. آن متدها و یا متغیرهایی که نیاز به وهله‌ای از کلاس برای ایجاد ندارند، به صورت استاتیک ایجاد می‌گردند. در سی شارپ از واژه کلیدی static برای معرفی آن‌ها کمک گرفته می‌شود. برای مثال:

```
class MyClass
{
    public static int a;
    public static void DoSomething();
}
```

در این مثال برای فراخوانی متد DoSomething نیازی به ایجاد یک وهله جدید از کلاس MyClass نمی‌باشد و تنها کافی است بنویسیم:

```
MyClass.DoSomething(); // and not -> new MyClass().DoSomething();
```

نکته‌ی مهمی که در اینجا وجود دارد این است که متدهای استاتیک تنها قادر به استفاده از متغیرهای استاتیک تعریف شده در سطح کلاس هستند. علت چیست؟

به مثال زیر دقت نمائید:

```
class MyClass
{
 // non-static instance member variable
 private int a;
 //static member variable
 private static int b;
 //static method
 public static void DoSomething()
 {
  //this will result in compilation error as “a” has no memory
  a = a + 1;
  //this works fine since “b” is static
  b = b + 1;
 }
}
```

در این مثال اگر متد DoSomething را فراخوانی کنیم، تنها متغیر b تعریف شده، در حافظه حضور داشته (به دلیل استاتیک معرفی شدن) و چون با روش فراخوانی MyClass.DoSomething هنوز وهله‌ای از کلاس مذکور ایجاد نشده، به متغیر a نیز حافظه‌ای اختصاص داده نشده است و نامعین می‌باشد.

بر این اساس کامپایلر نیز از کامپایل شدن این کد جلوگیری کرده و خطای لازم را گوشزد خواهد کرد.

اکنون تعریف یک کلاس به صورت استاتیک چه اثری را خواهد داشت؟

با تعریف یک کلاس به صورت استاتیک مشخص خواهیم کرد که این کلاس تنها حاوی متدها و متغیرهای استاتیک می‌باشد. امکان ایجاد یک وهله از آن‌ها وجود نداشته و نیازی نیز به این امر ندارند. این کلاس‌ها امکان داشتن instance variables را نداشته و به

صورت پیش فرض از نوع sealed به حساب خواهند آمد و امکان ارث بری از آن‌ها نیز وجود ندارد. علت این امر هم این است که یک کلاس static هیچ نوع رفتاری را تعریف نمی‌کند.

پس با این تفاسیر چرا نیاز به یک کلاس static ممکن است وجود داشته باشد؟
همانطور که عنوان شد یک کلاس استاتیک هیچ نوع رفتاری را تعریف نمی‌کند بنابراین بهترین مکان است برای تعریف متدهای کمکی که به سایر اعضای کلاس‌های ما وابستگی نداشته، عمومی بوده، مستقل و متکی به خود هستند. عموما متدهای کمکی در یک برنامه به صورت مکرر فراخوانی شده و نیاز است تا به سرعت در دسترس قرار داشته باشند و حداقل یک مرحله ایجاد وهله کلاس در اینجا برای راندمان بیشتر حذف گردد.
برای مثال متدی را در نظر بگیرید که بجز اعداد، سایر حروف یک رشته را حذف می‌کند. این متد عمومی است، وابستگی به سایر اعضای یک کلاس یا کلاس‌های دیگر ندارد. بنابراین در گروه متدهای کمکی قرار می‌گیرد. اگر از افزونه‌ی ReSharper استفاده نمائید، این نوع متدها را به صورت خودکار تشخیص داده و راهنمایی لازم را جهت تبدیل آن‌ها به متدهای استاتیک ارائه خواهد داد.

با کلاس‌های استاتیک نیز همانند سایر کلاس‌های یک برنامه توسط JIT compiler رفتار می‌شود، اما با یک تفاوت. کلاس‌های استاتیک فقط یکبار هنگام اولین دسترسی به آن‌ها ساخته شده و در قسمت High Frequency Heap حافظه قرار می‌گیرند. این قسمت از حافظه تا پایان کار برنامه از دست garbage collector در امان است (بر خلاف garbage-collected heap یا object heap که جهت instance classes مورد استفاده قرار می‌گیرد)

نکته:
در برنامه‌های ASP.Net از بکارگیری متغیرهای عمومی استاتیک برحذر باشید (از static fields و نه static methods). این متغیرها بین تمامی کاربران همزمان یک برنامه به اشتراک گذاشته شده و همچنین باید مباحث قفل‌گذاری و امثال آن‌را در محیط‌های چند ریسمانی هنگام کار با آن‌ها رعایت کرد (thread safe نیستند).

نظرات خوانندگان

نویسنده: نیما
تاریخ: ۱۳۸۸/۰۶/۲۸ ۲۲:۰۵:۰۸

با سلام خدمت استاد نصیری
جناب نصیری طبق نکت آخر پس اینکه ما میام برای استفاده از jQuery Ajax از متدهای استاتیک استفاده میکنیم کار خوبی نمیکنیم؟آخه این تنها راه هست.
یه سوال دیگه هم داشتم که MS Ajax چطور میتونه متدهای غیر استاتیک صفحه رو هم صدا بزنه در حالی که ما در jQuery فقط قادر به صدا زدن متدهای استاتیک هستیم؟
سپاسگزارم

نویسنده: وحید نصیری
تاریخ: ۱۳۸۸/۰۶/۲۸ ۲۳:۰۵:۲۹

سلام،
- همانطور که تاکید کردم فیلد استاتیک و نه متد استاتیک که مشکل همزمانی را ندارد.
- برای jQuery Ajax چون بسیاری از ملاحظاتی که توسط MS Ajax انجام می‌شود مانند کار با ViewState و ارسال و مدیریت آن صورت نمی‌گیرد، امکان ایجاد وهله‌ای از کلاس استاندارد صفحه ASP.Net توسط آن میسر نیست. بنابراین باید این متد را استاتیک تعریف کرد تا وابستگی آن‌را از شیء صفحه قطع کرد. به همین جهت jQuery Ajax بسیار بهینه‌تر از MS Ajax عمل می‌کند (چون اساسا درکی از ViewState ندارد)

نویسنده: محمد امین شریفی
تاریخ: ۱۳۸۸/۰۶/۲۸ ۲۳:۳۳:۰۴

همانطور که گفته شد از متد های استاتیک برای کارهایی که زیاد به آن ها نیاز داریم استفاده می شود.
نکته های خوبی را بیان کردید.
لطفا درباره کلاس های sealed بیشتر بنویسید.

پیروز باشید.

نویسنده: وحید نصیری
تاریخ: ۱۳۸۸/۰۶/۲۸ ۲۳:۵۹:۵۴

الزاما خیر. اگر متد شما صرفا یک متد utility است و مستقل است یا اگر کلا متد شما وابستگی به کلاس جاری ندارد یا نیاز است متد مورد نظر از کلاس جاری گسسته شود، می‌توان آن‌را استاتیک تعریف کرد.

نویسنده: محمد امین شریفی
تاریخ: ۱۳۸۸/۰۶/۲۹ ۱۵:۳۶:۳۹

درباره کلاس های static و sealed هم می شود توضیح دهید؟
در دات نت میکرو برخی کلاس ها را که در رابطه با پورت ها بودند و همچنین برای ساخت اکستنشن برای html helper،دات نت آنها را بصورت استاتیک تعریف کرده است.چگونه می توان از GC در کلاس ها،متغییر ها و تابع های استاتیک سود برد؟

ممنون،

نویسنده: وحید نصیری
تاریخ: ۱۳۸۸/۰۶/۲۹ ۱۶:۰۴:۰۵

- همانطور که عنوان شد یک کلاس sealed قابل ارث بری نیست. توضیحات بیشتر: http://msdn.microsoft.com/en-us/library/88c54tsw(VS.71).aspx
- من جایی عنوان نکردم اشیاء غیر استاتیک داخل یک متد استاتیک garbage collected نمی‌شوند. فقط عنوان کردم اگر فیلدی استاتیک بود در آن ناحیه از حافظه تا پایان کار برنامه باقی می‌ماند. یا اگر متدی استاتیک بود، کار ساخت جهت دسترسی سریع به آن فقط یکبار انجام می‌شود.

نویسنده: میثم جوادی
تاریخ: ۱۳۸۸/۰۷/۰۱ ۰۸:۴۰:۱۸

در واقع زمانیکه ما یک کلاس رو سینگلتون پیاده سازی میکنیم، همواره از یک نقطه آبجکت رو میخونیم؟ کی این آبجکت از حافظه خالی میشود؟(بعد از اتمام برنامه؟) اگر اینطور است در Web-App مشکل ساز نخواهد بود؟(مانند فیلد استاتیک؟)

نویسنده: وحید نصیری
تاریخ: ۱۳۸۸/۰۷/۰۱ ۱۱:۲۷:۴۱

لطفا به آخرین لینک مقاله فوق مراجعه کنید. قسمت "thread safe نیستند" . یک مقاله دو قسمتی است که این مباحث همزمانی رو بررسی کرده.

نویسنده: DotNetCoders
تاریخ: ۱۳۸۸/۰۷/۰۱ ۱۸:۱۷:۵۸

ممنون جالب بود.فقط جای این نکته خالی موند که معادل static در vb.net کلمه کلیدی share هست.

/

جناب نصیری جای یک مقاله کامل با موضوع event و delegate ها در وبلاگ شما خالیه.

الان همه پروژه های اپن سورس رو که باز می کنی حتمی توش از ایونت ها استفاده شده ولی مفهوم و کاراییش برای خیلی ها (مثل خودم) گنگه...

ممنون میشم این مورد هم مثل همه مورد های دیگه کامل بررسی کنید.

ممنون و خسته نباشید.

عنوان: **رهنمودهای کد نویسی سی شارپ 3**
نویسنده: وحید نصیری
تاریخ: ۱۳۸۸/۰۷/۱۱ ۱۷:۵۰:۰۰
آدرس: www.dotnettips.info
برچسب‌ها: #C

دو فایل زیر مقاله و خلاصه مقاله‌ای در مورد روش‌های بهتر کد نویسی با سی شارپ 3 هستند.

رهنمودها
+ خلاصه

این رهنمودها (و نه استانداردها) جهت بالا بردن کیفیت کدهای تهیه شده، یک دست شدن آن‌ها در یک سازمان، تهیه مستندات بهتر و امکان نگهداری ساده‌تر آن‌ها، بسیار مؤثرند.
تعدادی از آن‌ها را در مقاله‌ی " زیباتر کد بنویسیم " دیده‌اید. مقالات فوق گردآوری و به روز رسانی اینگونه نکات جهت پوشش دادن سی شارپ 3 می‌باشند.

ماخذ

نظرات خوانندگان

نویسنده: میثم
تاریخ: ۱۳۸۸/۰۷/۱۱ ۲۱:۵۰:۳۱

یه کم هم خودتون در مورد refactoring بنویسید عالی میشه.

عنوان: **دریافت فایل از یوتیوب**
نویسنده: وحید نصیری
تاریخ: ۱۳۸۹/۰۱/۰۱ ۱۶:۰۲:۰۰
آدرس: www.dotnettips.info
برچسب‌ها: #C

سلام

سال نو مبارک! به امید سالی بهتر از پارسال!

این روزها با هزینه‌ای معادل هزینه‌ی تهیه‌ی یک هاست اشتراکی سالیانه برای بالاگذاری یک سایت معمولی در 5 سال قبل، می‌توان یک VPS تهیه کرد و به این صورت قفل و کلید یک نیمچه سرور را (با 200 و خرده‌ای مگ رم، 30 گیگ فضا، سرعت CPU نزدیک به 700 MHz و ویندوز سرور 2003 یا 2008) در اختیار شما قرار می‌دهند (البته به قول معروف هر چقدر پول بدهید همانقدر هم سخت افزار در اختیار شما قرار می‌دهند) بجای صرفا یک دایرکتوری مجازی محدود با 100 مگ فضای هاست که هر احدی در آن هاست اشتراکی می‌تواند سر مبارک را اندکی چرخانده و تمام زندگی شما را مرور کند و غیره!
استفاده‌ی مفیدی هم که این VPS برای من داشته، ترنس لود کردن یک سری فایل است (با توجه به سرعت‌های نجومی دریافت فایل این سرورها). برای مثال دریافت فایل از یوتیوب و انتقال به یک هاست دیگر برای دریافت ساده‌تر خودم و یا دیگران.
برای نمونه سایت dotnet-tv.com را در نظر بگیرید. تعدادی از ویدیوهای این سایت در یوتیوب هاست شده و از این دست زیاد هستند. خیلی‌ها برای فرار از مشکلات کمبود پهنای باند از یوتیوب استفاده می‌کنند. یوتیوب هم که از این طرف بسته است. خوب، من الان می‌خواهم ویدیوی مربوط به ASP.Net MVC آن را مشاهده کنم، چکار باید کرد؟!
یک برنامه‌ی ساده‌ی کنسول را تهیه کرده‌ام که این کار را برای VPS داران تسهیل می‌کند.
- دریافت فایل از یوتیوب
- آپلود خودکار آن به رپیدشیر

دریافت سورس برنامه

یک نمونه خروجی آن: (فایل‌های یوتیوب سایت ذکر شده که به رپیدشیر منتقل شده)
دریافت

در سورس این برنامه موارد زیر پیاده سازی شده است:
- یافتن لینک‌های یوتیوب سایت dotnet-tv.com با استفاده از regular expressions
- یافتن لینک دانلود مستقیم این فایل‌ها از سایت یوتیوب که شامل استفاده از regular expressions برای استخراج قسمت‌های مفید از صفحات و همچنین استفاده از امکانات Json دات نت فریم ورک سه و نیم برای parse قسمت‌های استخراج شده است.
- ایجاد یک thread pool سفارشی که هر بار 7 لینک مستقیم را به صورت همزمان از یوتیوب دریافت می‌کند. (thread pool پیش فرض دات نت تمام تردها را به یکباره شروع می‌کند که برای اینکار مفید نیست. به همین جهت از این thread pool سفارشی شده استفاده شد)
پیش فرض فایلی که از سایت یوتیوب دریافت می‌شود MP4 با کیفیت بالا است که با fmt=18 در فایل Youtube.cs مشخص شده. فرمت‌های دیگر را می‌توانید از این فایل ایده بگیرید.
- آپلود فایل دریافتی از یوتیوب به یک اکانت رایگان کالکتور در رپیدشیر. (ماخذ این مورد در سایت code projects)
مشخصات این اکانت رایگان کالکتور در فایل app.config باید ذکر شود.

این سورس می‌تونه ایده‌ی ابتدایی بسیاری از کارهای مشابه باشد. برای مثال ایجاد یک وب سرویس، یک وب سایت، یک سرویس ایمیلی و غیره.

پ.ن.
کار انجام شده فعالیت وارز محسوب نمی‌شود زیرا مجوز ویدیوهای سایت یوتیوب این امکان توزیع (و بسیاری موارد دیگر) را به شما می‌دهد.

نظرات خوانندگان

نویسنده: Afshar Mohebbi
تاریخ: ۱۳۸۹/۰۱/۰۱ ۱۶:۳۴:۴۳

سال نو مبارک، ایده جالبی بود.

نویسنده: نیما
تاریخ: ۱۳۸۹/۰۱/۰۱ ۲۳:۴۷:۴۸

سلام آقای نصیری
سال نو مبارک
این همون Leech هست؟

نویسنده: وحید نصیری
تاریخ: ۱۳۸۹/۰۱/۰۲ ۰۰:۵۷:۵۰

این هم یک نوع از آن است.

نویسنده: zamanphp
تاریخ: ۱۳۸۹/۰۱/۰۳ ۱۲:۲۳:۳۴

چند وقت پیشپ منم یه vps داشتم که os اون centos بود ، هر فایلی که از این ور یا از اون ور بسته بود رو مث آب خوردن روی vps دانلود می کردم ، چیزی ه همیشه تو کف بدم سرعت باور نکردنی دانلود بود در عرض چند ثانیه چه فایلهای حجیمی رو که من دانلود نمی کردم.

نویسنده: iMAN
تاریخ: ۱۳۸۹/۰۱/۰۳ ۲۰:۳۱:۲۳

عالی بود مثل همیشه، ولی این پرشین گیگ همیشه خدا داون هست!
سال نو هم مبارک آقای نصیری، امیدوارم موفق تر از همیشه باشید.

نویسنده: reza
تاریخ: ۱۳۸۹/۰۱/۰۵ ۱۳:۵۶:۱۹

چه جوری این vps رو باید تهیه کرد

نویسنده: وحید نصیری
تاریخ: ۱۳۸۹/۰۱/۰۵ ۱۴:۰۵:۲۷

سؤال خوبی نبود

نویسنده: reza
تاریخ: ۱۳۸۹/۰۱/۰۵ ۱۶:۱۳:۲۰

آقای نصیری شرمنده ولی خوب بی سوادیم در ضمن قانونیه یا مثل vpn فردا خفتمون نکنند

نویسنده: وحید نصیری
تاریخ: ۱۳۸۹/۰۱/۰۵ ۱۶:۵۱:۲۳

خرید و فروش VPS تا جایی که می‌دونم منع قانونی ندارد. مثل خرید و فروش سرور معمولی است.

VPN هم اساسا هدفش ارتباط دفاتر شرکت‌های مختلف است. در همین کشور خودمون خیلی از شرکت‌ها از طریق VPN به دفاتر مختلف خودشون وصل می‌شوند یا کارمندانشون امکان کار از راه دور را پیدا می‌کنند.
بنابراین در حالت کلی عنوان غیرقانونی دادن به VPN کار صحیحی نیست.

نویسنده: Mohammad Shams Javi
تاریخ: ۱۳۸۹/۰۱/۰۶ ۰۹:۲۷:۳۱

سلام

مطلب بسیار جالب و مفیدی بود.
امیدوارم سال بسیار خوب و پر برکتی (پر از سفارش کار)داشته باشید.

موفق و پیروز باشید

عنوان: **سری فیبوناچی و دات نت 4 !**
نویسنده: وحید نصیری
تاریخ: ۱۳۸۹/۰۲/۰۹ ۲۱:۲۵:۰۰
آدرس: www.dotnettips.info
برچسب‌ها: #C

سری معروف فیبوناچی که معرف حضور شما هست. سری از اعداد است که هر عدد آن مساوی حاصل جمع دو عدد ماقبل آن است. دو عدد اول این سری هم 0 و 1 هستند.

اگر بخواهیم این الگوریتم را به صورت یک متد بازگشتی نمایش دهیم به صورت زیر خواهد بود:

```
public static int Fibonacci(int x)
{
    if (x <= 1)
          return 1;
    return Fibonacci(x - 1) + Fibonacci(x - 2);
}
```

این الگوریتم چند مشکل دارد:

الف) برای اعداد بزرگ حتی با بکارگیری Int64 و یا double و امثال آن هم باز به جواب نخواهیم رسید (برای مثال 1500 را بررسی کنید).

ب) بسیار کند است.

در دات نت 4 برای کار با اعداد بزرگ، فضای نام System.Numerics معرفی شده است که حاوی نوع جدیدی از اعداد به نام BigInteger است.

اکنون اگر الگوریتم سری فیبوناچی را بر اساس این نوع داده جدید بازنویسی کنیم خواهیم داشت:

```
using System;
using System.Collections.Generic;
using System.Numerics; //needs a ref. to this assembly

namespace Fibonaci
{
   public class CFibonacci
   {
       public static int Fibonacci(int x)
       {
           if (x <= 1)
               return 1;
           return Fibonacci(x - 1) + Fibonacci(x - 2);
       }

       public static IEnumerable<BigInteger> BigFib(Int64 toNumber)
       {
           BigInteger previous = 0;
           BigInteger current = 1;

           for (Int64 y = 1; y <= toNumber; y++)
           {
               var auxiliar = current;
               current += previous;
               previous = auxiliar;
               yield return current;
           }
       }
   }
}
```

و مثالی در مورد نحوه استفاده از آن:

```
using System;
using System.Linq;
```

```
namespace Fibonaci
{
   class Program
   {
       static void Main()
       {
           foreach (var i in CFibonacci.BigFib(10))
           {
               Console.WriteLine("{0}", i);
           }

           var num = 12000;
           var fib = CFibonacci.BigFib(num).Last();
           Console.WriteLine("fib({0})={1}", num, fib);

           Console.WriteLine("Press a key...");
           Console.ReadKey();
       }
   }
}
```

برای نمونه با عدد 12000 خروجی برنامه در کسری از ثانیه (و نه چند دقیقه یا ساعت) به شرح زیر خواهد بود:

```
fib(12000)=5142634249113365925793965792899545208268344435268296004358738632486 22
6541402071401389255147626107001009927557114405957916735603943724208942713632368 9
0220795622156962279145089144790590766825123267598809824638290242678314854666540 4
4737238404316460094524991127385787834667936287635749920429028506944204244447120 0
5229232934910367230242866231728501552588821039758370707148017884077297269235705 4
7182399886189676168711943464625099170269110089476956181083454209957733682149390 5
4165165893786050606701121522243585979767174851402346263457587711254126585701172 3
3145399041523160872953478172038112296589987101853200373528455934237255262713230 0
6389582539601208794805085509523363363844566868744023292625362045745997388951083 8
2354278515937123638990947097473859916672061135190356878184540942562466655979191 2
0221228971083887333477383511839128795672550442615046142191484419181052325765877 0
9988549275792703440923434006592840076974180213220388892946370234232414834360527 5
2892828047209449335968266251912720358181340410454297218123107622489140473061145 9
0332194269322506603898748316370940260123046705494434911105585034877998905851706 9
9608762679570920521572784344305457768002450765067824024074242127042267490747692 7
2242273394545076032364061910002166367508087042929904089184088075364647433006933 2
7232021833458226821990676346326138716131850050397049131478110055649436106334137 1
4357778796118315412553837120429675202849608463310347678307117777960404258101788 8
2825778492065967108236317115728966890438125408067685581552498755337265706369597 0
3966810916144914070724071127985942791991244387240528430589136680295476342190597 0
1520631145818744942011883877570743585799931087019958576080768017925827346100046 0
9752706492956452847434954703817837004382362894467092660195553765742719481589336 5
8849486310166754789679872814022492158480935533437970715634262057049683408635869 2
3094646720333067620626504796007239299163445638199847941146318217181637965012068 4
3508239978813709046016781904184551195129693427398875916987783953249229443033432 8
4697290519813153022428892283412515421124815984360962946905188903308536054077048 0
2563345120170537044758617754657777775930041014416619743935590363177308881251521 5
0963837791859529474788797003420902801949021039439242230240368705911940700585837 9
5213709899445723629000574573542080375885372320699213464299770501094058138616842 7
4738297367281671001465263250988895885167589422311742182943472894287860556997151 2
6529178338491015720367977945835424557984697383047259337016097752370790257512980 3
072039857524154149354311250529579592001
```

نظرات خوانندگان

نویسنده: حسین مرادی نیا
تاریخ: ۱۳۸۹/۰۲/۰۹ ۲۲:۰۳:۲۳

استاد نمی دونم چقدر این سوال به موضوع ربط داره،ولی در بعضی جاها نوشته شده که حلقه foreach سنگین تر از حلقه for هستش و حلقه for سریعتر اجرا میشه.
بعضی ها هم میگن که اینطور نیست.می خواستم بدونم نظر شما در این مورد چی هست.

نویسنده: وحید نصیری
تاریخ: ۱۳۸۹/۰۲/۰۹ ۲۲:۵۳:۱۵

بله. درسته. ولی من دراکثر موارد خوانایی کد را به چند میلی ثانیه بهبود کارآیی ترجیح می‌دم.

نویسنده: Meysam
تاریخ: ۱۳۸۹/۰۲/۱۰ ۱۰:۲۳:۱۴

مشکلی که روش اول داره فقط تو نوع داده ای نیست!

نویسنده: Nasser Hajloo
تاریخ: ۱۳۸۹/۰۲/۱۱ ۱۱:۵۱:۴۸

بخوبی یادم میاد که در درس طراحی الگوریتم خونده بودیم که نوشتن دنباله فیبوناچی بصورت بازگشتی صحیح نیست و بهتره که از روش های ساده یعنی استفاده از یک حلقه ساده برای بدست آوردن جواب استفاده کرد. که شما هم درواقع برنامه رو بصورت یک حلقه ساده نوشتید و با yield مقدار نهایی رو برگردوندید. کار بسیار خوبیه و از شما سپاسگذارم که این موارد رو یادآوری می کنید. اینها نشون میده که نوشتن یک متد بصورت اشتباه چقدر کارایی برنامه رو پایین میاره. امیدوارم در آینده الگوریتم های دیگری رو هم بررسی کنید.

نویسنده: ramin
تاریخ: ۱۳۸۹/۰۲/۱۲ ۱۳:۰۸:۵۹

سلام
من VS2010 رو نصب کردم ولی فضای نام System.Numerics رو نمیشناسه.
مشکل از چی میتونه باشه؟

نویسنده: وحید نصیری
تاریخ: ۱۳۸۹/۰۲/۱۲ ۱۴:۲۷:۲۵

نوشتم که ... needs a ref. to this assembly
از منوی پروژه گزینه‌ی add reference ، ارجاعی را به اسمبلی استاندارد System.Numerics اضافه کنید.

نویسنده: MHM5000
تاریخ: ۱۳۸۹/۰۲/۱۳ ۱۶:۳۹:۰۸

سلام استاد...
همونطوری که به ویکیپدیا لینک داده بودید، برای روش دنباله فیبوناچی روش زیر هم وجود دارد:
http://albums.kimag.es/albums/mhm5000/71051717.jpg
اگه از این فرمول استفاده بشه، جواب سریعتر از حالتی که الآن شما توضیح دادید، بدست میاد؟
برای اعداد بزرگتر عرض می کنم...

جدیدا VB کار شدم اونم از ورژن ۶!
می خواستم بدونم بهتره شروع به یادگیری 2010 بکنم(یک ضرب) یا همون 6 خوبه؟ اصلا فرق محسوسی داره؟
چون من قابلیتی مثل برنامه حاظر توی vb6 ندیدم که بعد از عدد شانزدهم رو نمایش بده!
ممنون
MHM5000 | Erudite.ir

نویسنده: وحید نصیری
تاریخ: ۱۳۸۹/۰۲/۱۳ ۱۹:۲۴:۰۵

سلام
- تمرکز این مطلب روی بهینه سازی الگوریتم نبود. بیشتر قصد داشتم اعداد عظیم الجثه رو معرفی کنم که برای مثال در این یک نمونه کاربرد دارد یا مباحث رمزنگاری اطلاعات و الگوریتم RSA
- با VB.Net شروع کنید. ساپورت VB6 احتمالا در محصولات سری بعد مایکروسافت به پایان می‌رسد یا بسیار محدود خواهد بود:
http://msdn.microsoft.com/en-us/vbrun/ms788708.aspx

نویسنده: SpEEdY
تاریخ: ۱۳۸۹/۰۶/۰۸ ۱۰:۱۹:۰۲

من یه سوال دارم. امکان اینکه از bigInteger تو c++ استفاده بشه نیست؟

نویسنده: وحید نصیری
تاریخ: ۱۳۸۹/۰۶/۰۸ ۱۰:۲۹:۰۷

از کتابخانه OpenSSL در CPP استفاده کنید:
http://www.openssl.org/docs/crypto/bn.html
dll یا حتی lib آماده آن برای ویندوز هم هست:
http://www.slproweb.com/products/Win32OpenSSL.html

نویسنده: SpEEdY
تاریخ: ۱۳۸۹/۰۶/۰۸ ۱۰:۵۰:۳۰

از کتابخانه OpenSSL در CPP استفاده کنید:

ممنون بابت جوابتون. راستش من خیلی مبتدی هستم. یه الگوریتم دارم که بر طبق اون باید اعداد تصادفی خیلی بزرگ تولید کنم، اونها رو جمع و ضرب کنم. اینکه چطوری باید از dll یا lib استفاده کنم رو بلد نیستم. از VS2008 استفاده میکنم. اگر لطف کنین و بیشتر توضیح بدین که چیکار کنم ممنون میشم.

نویسنده: وحید نصیری
تاریخ: ۱۳۸۹/۰۶/۰۸ ۱۵:۳۴:۵۴

پاسخ:
http://www.dotnettips.info/2010/08/openssl.html

عنوان: **Tuple در دات نت 4**
نویسنده: وحید نصیری
تاریخ: ۱۳۸۹/۰۲/۱۷ ۱۳:۰۸:۰۰
آدرس: www.dotnettips.info
برچسب‌ها: #C

نوع جدیدی در دات نت 4 به نام Tuple اضافه شده است که در این مطلب به بررسی آن خواهیم پرداخت.
در ریاضیات، Tuple به معنای لیست مرتبی از اعضاء با تعداد مشخص است. Tuple در زبان‌های برنامه نویسی Dynamic مانند اف شارپ، LISP ، Perl و بسیاری موارد دیگر مطلب جدیدی نیست. در زبان‌های dynamic برنامه نویس‌ها می‌توانند متغیرها را بدون معرفی نوع آن‌ها تعریف کنند. اما در زبان‌های Static مانند سی شارپ ، برنامه نویس‌ها موظفند نوع متغیرها را پیش از کامپایل آن‌ها معرفی کنند که هر چند کار کد نویسی را اندکی بیشتر می‌کند اما به این صورت شاهد خطاهای کمتری نیز خواهیم بود (البته سی شارپ 4 این مورد را با معرفی واژه‌ی کلیدی dynamic تغییر داده است).
برای مثال در اف شارپ داریم:

```
let data = (“John Doe”, 42)
```

که سبب ایجاد یک tuple که المان اول آن یک رشته و المان دوم آن یک عدد صحیح است می‌شود. اگر data را بخواهیم نمایش دهیم خروجی آن به صورت زیر خواهد بود:

```
printf “%A” data
// Output: (“John Doe”,42)
```

در دات نت 4 فضای نام جدیدی به نام System.Tuple معرفی شده است که در حقیقت ارائه دهنده‌ی نوعی جنریک می‌باشد که توانایی در برگیری انواع مختلفی را دارا است :

```
public class Tuple<T1>
up to:
public class Tuple<T1, T2, T3, T4, T5, T6, T7, TRest>
```

همانند آرایه‌ها، اندازه‌ی Tuples نیز پس از تعریف قابل تغییر نیستند (immutable). اما تفاوت مهم آن با یک آرایه در این است که اعضای آن می‌توانند نوع‌های کاملا متفاوتی داشته باشند. همچنین تفاوت مهم آن با یک ArrayList یا آرایه‌ای از نوع Object، مشخص بودن نوع هر یک از اعضاء آن است که type safety بیشتری را به همراه خواهد داشت و کامپایلر می‌تواند در حین کامپایل دقیقا مشخص نماید که اطلاعات دریافتی از نوع صحیحی هستند یا خیر.

یک مثال کامل از Tuples را در کلاس زیر ملاحظه خواهید نمود:

```
using System;
using System.Linq;
using System.Collections.Generic;

namespace TupleTest
{
   class TupleCS4
   {
  #region Methods (4)

  // Public Methods (4)

       public static Tuple<string, string> GetFNameLName(string name)
       {
           if (string.IsNullOrWhiteSpace(name))
```

```
                throw new NullReferenceException("name is empty.");

            var nameParts = name.Split(',');

            if (nameParts.Length != 2)
                throw new FormatException("name must contain ','");

            return Tuple.Create(nameParts[0], nameParts[1]);
        }

        public static void PrintSelectedTuple()
        {
            var list = new List<Tuple<string, int>>
                           {
                               new Tuple<string, int>("A", 1),
                               new Tuple<string, int>("B", 2),
                               new Tuple<string, int>("C", 3)
                           };

            var item = list.Where(x => x.Item2 == 2).SingleOrDefault();
            if (item != null)
                Console.WriteLine("Selected Item1: {0}, Item2: {1}",
                    item.Item1, item.Item2);
        }

        public static void PrintTuples()
        {
            var tuple1 = new Tuple<int>(12);
            Console.WriteLine("tuple1 contains: item1:{0}", tuple1.Item1);

            var tuple2 = Tuple.Create("Item1", 12);
            Console.WriteLine("tuple2 contains: item1:{0}, item2:{1}",
                tuple2.Item1, tuple2.Item2);

            var tuple3 = Tuple.Create(new DateTime(2010, 5, 6), "Item2", 20);
            Console.WriteLine("tuple3 contains: item1:{0}, item2:{1}, item3:{2}",
                tuple3.Item1, tuple3.Item2, tuple3.Item3);
        }

        public static void Tuple8()
        {
            var tup =
                new Tuple<int, int, int, int, int, int, int, Tuple<int, int>>
                    (1, 2, 3, 4, 5, 6, 7, new Tuple<int, int>(8, 9));

            Console.WriteLine("tup.Rest Item1: {0}, Item2: {1}",
                    tup.Rest.Item1,tup.Rest.Item2);
        }

  #endregion Methods
   }
}
```

```
using System;

namespace TupleTest
{
   class Program
   {
       static void Main()
       {
           var data = TupleCS4.GetFNameLName("Vahid, Nasiri");
           Console.WriteLine("Data Item1:{0} & Item2:{1}",
                   data.Item1, data.Item2);

           TupleCS4.PrintTuples();

           TupleCS4.PrintSelectedTuple();

           TupleCS4.Tuple8();

           Console.WriteLine("Press a key...");
           Console.ReadKey();
       }
   }
}
```

توضیحات :

- روش‌های متفاوت ایجاد Tuples را در متد PrintTuples می‌توانید ملاحظه نمائید. همچنین نحوه‌ی دسترسی به مقادیر هر کدام از اعضاء نیز مشخص شده است.

- کاربرد مهم Tuples در متد GetFNameLName نمایش داده شده است؛ زمانیکه نیاز است تا چندین خروجی از یک تابع داشته باشیم. به این صورت دیگر نیازی به تعریف آرگومان‌هایی به همراه واژه کلیدی out نخواهد بود یا دیگر نیازی نیست تا یک شیء جدید را ایجاد کرده و خروجی را به آن نسبت دهیم. به همان سادگی زبان‌های dynamic در اینجا نیز می‌توان یک tuple را ایجاد و استفاده کرد.

- بدیهی است از Tuples در یک لیست جنریک و یا حالات دیگر نیز می‌توان استفاده کرد. مثالی از این دست را در متد PrintSelectedTuple ملاحظه خواهید نمود. ابتدا یک لیست جنریک از Tuple ایی با دو عضو تشکیل شده است. سپس با استفاده از امکانات LINQ ، عضوی که آیتم دوم آن مساوی 2 است یافت شده و سپس المان‌های آن نمایش داده می‌شود.

- نکته‌ی دیگری را که حین کار با Tuples می‌توان در نظر داشت این است که اعضای آن حداکثر شامل 8 عضو می‌توانند باشند که عضو آخر باید یک Tuple تعریف گردد و بدیهی است این Tuple نیز می‌تواند شامل 8 عضو دیگر باشد و الی آخر که نمونه‌ای از آن را در متد Tuple8 می‌توان مشاهده کرد.

نظرات خوانندگان

نویسنده: Afshar Mohebbi
تاریخ: ۱۳۸۹/۰۲/۱۷ ۱۶:۱۸:۳۲

جالب و مفید بود. ممنون.

نویسنده: علی اقدم
تاریخ: ۱۳۸۹/۱۱/۰۳ ۲۲:۵۷:۱۰

البته در استفاده از Tuple ها باید به این نکته توجه نمود که بعد از نمونه سازی یک Tuple دیگر امکان تغییر خصوصیات اون نمونه وجود نداره که بدلیل عدم وجود setter است،
همچنین می توان از خصوصیت Rest در Tuple های تودرتو استفاده نمود، مثلا

```
tup.Rest.Rest.Item1
```

عنوان: **دات نت 4 و کلاس Lazy**
نویسنده: وحید نصیری
تاریخ: ۱۳۸۹/۰۲/۲۰ ۲۱:۵۸:۰۰
آدرس: www.dotnettips.info
برچسب‌ها: #C

یکی از الگوهای برنامه نویسی شیء گرا، Lazy Initialization Pattern نام دارد که دات نت 4 پیاده سازی آن‌را سهولت بخشیده است.

در دات نت 4 کلاس جدیدی به فضای نام System اضافه شده است به نام Lazy و هدف از آن lazy initialization است؛ من ترجمه‌اش می‌کنم وهله سازی با تاخیر یا به آن on demand construction هم گفته‌اند (زمانی که به آن نیاز هست ساخته خواهد شد).

فرض کنید در برنامه‌ی خود نیاز به شیءایی دارید و ساخت این شیء بسیار پرهزینه است. نیازی نیست تا بلافاصله پس از تعریف، این شیء ساخته شود و تنها زمانیکه به آن نیاز است باید در دسترس باشد. کلاس Lazy جهت مدیریت اینگونه موارد ایجاد شده است. تنها کاری که در اینجا باید صورت گیرد، محصور کردن آن شیء هزینه‌بر توسط کلاس Lazy است:

```
Lazy<ExpensiveResource> ownedResource = new Lazy<ExpensiveResource>();
```

در این حالت برای دسترسی به شیء ساخته شده از ExpensiveResource ، می‌توان از خاصیت Value استفاده نمود (ownedResource.Value). تنها در حین اولین دسترسی به ownedResource.Value ، شیء ExpensiveResource ساخته خواهد شد و نه پیش از آن و نه در اولین جایی که تعریف شده است. پس از آن این حاصل cache شده و دیگر وهله سازی نخواهد شد.

ownedResource دارای خاصیت IsValueCreated نیز می‌باشد و جهت بررسی ایجاد آن شیء می‌تواند مورد استفاده قرار گیرد.

برای مثال قصد داریم اطلاعات ExpensiveResource را ذخیره کنیم اما تنها در حالتیکه یکبار مورد استفاده قرار گرفته باشد.

کلاس Lazy دارای دو متد سازنده‌ی دیگر نیز می‌باشد:

```
public Lazy(bool isThreadSafe);
public Lazy(Func<T> valueFactory, bool isThreadSafe);
```

و هدف از آن استفاده‌ی صحیح از این متد در محیط‌های چند ریسمانی است. بدیهی است در این نوع محیط ها علاقه‌ای نداریم که در یک لحظه توسط چندین ترد مختلف، سبب ایجاد وهله‌های ناخواسته‌ای از ExpensiveResource شویم و تنها یک مورد از آن کافی است یا به قولی thread safe, lazy initialization of expensive objects

بدیهی است اگر برنامه‌ی شما چند ریسمانی نیست می‌توانید این مکانیزم را کنسل کرده و اندکی کارآیی برنامه را با حذف قفل‌های همزمانی این کلاس بالا ببرید.

مثال اول:

```
using System;
using System.Threading;

namespace LazyExample
{
 class Program
 {
     static void Main()
     {
         Console.WriteLine("Before assignment");
         var slow = new Lazy<Slow>();
         Console.WriteLine("After assignment");

         Thread.Sleep(1000);

         Console.WriteLine(slow);
         Console.WriteLine(slow.Value);
```

```
        Console.WriteLine("Press a key...");
        Console.Read();
    }
 }


 class Slow
 {
     public Slow()
     {
         Console.WriteLine("Start creation");
         Thread.Sleep(2000);
         Console.WriteLine("End creation");
     }
 }
}
```

خروجی این برنامه به شرح زیر است:

```
Before assignment
After assignment
Value is not created.
Start creation
End creation
LazyExample.Slow
Press a key...
```

همانطور که ملاحظه می‌کنید تنها در حالت دسترسی به مقدار Value شیء slow ، عملا وهله‌ای از آن ساخته خواهد شد.

مثال دوم:

شاید نیاز به مقدار دهی خواص کلاس پرهزینه وجود داشته باشد. برای مثال علاقمندیم خاصیت SomeProperty کلاس ExpensiveClass را مقدار دهی کنیم. برای این منظور می‌توان به شکل ذیل عمل کرد (یک Func<t>را می‌توان به سازنده‌ی آن ارسال نمود):

```
using System;

namespace LazySample
{
 class Program
 {
     static void Main()
     {
         var expensiveClass =
             new Lazy<ExpensiveClass>
             (
                 () =>
                 {
                     var fobj = new ExpensiveClass
                                    {
                         SomeProperty = 100
                     };
                     return fobj;
                 }
             );

         Console.WriteLine("expensiveClass has value yet {0}",
             expensiveClass.IsValueCreated);

         Console.WriteLine("expensiveClass.SomeProperty value {0}",
             (expensiveClass.Value).SomeProperty);

         Console.WriteLine("expensiveClass has value yet {0}",
             expensiveClass.IsValueCreated);

         Console.WriteLine("Press a key...");
         Console.Read();
     }
 }
```

```
class ExpensiveClass
{
    public int SomeProperty { get; set; }

    public ExpensiveClass()
    {
        Console.WriteLine("ExpensiveClass constructed");
    }
}
}
```

کاربردها

:

- علاقمندیم تا ایجاد یک شیء هزینه‌بر تنها پس از انجام یک سری امور هزینه‌بر دیگر صورت گیرد. برای مثال آیا تابحال شده است که با یک سیستم ماتریسی کار کنید و نیاز به چند گیگ حافظه برای پردازش آن داشته باشید؟! زمانیکه از کلاس Lazy استفاده نمائید، تمام اشیاء مورد استفاده به یکباره تخصیص حافظه پیدا نکرده و تنها در زمان استفاده از آن‌ها کار تخصیص منابع صورت خواهد گرفت.

- ایجاد یک شیء بسیار پر هزینه بوده و ممکن است در بسیاری از موارد اصلا نیازی به ایجاد و یا حتی استفاده از آن نباشد. برای مثال یک شیء کارمند را درنظر بگیرید که یکی از خواص این شیء، لیستی از مکان‌هایی است که این شخص قبلا در آنجاها کار کرده است. این اطلاعات نیز به طور کامل از بانک اطلاعاتی دریافت می‌شود. اگر در متدی، استفاده کننده از شیء کارمند هیچگاه اطلاعات مکان‌های کاری قبلی او را مورد استفاده قرار ندهد، آیا واقعا نیاز است که این اطلاعات به ازای هر بار ساخت وهله‌ای از شیء کارمند از دیتابیس دریافت شده و همچنین در حافظه ذخیره شود؟

نظرات خوانندگان

نویسنده: مهدی پایروند
تاریخ: ۱۳۸۹/۰۲/۲۱ ۰۹:۵۹:۵۱

کتاب مفیدی برای شروع کار و یا یادگیری برای بکار بردن C#4 بنظرتون میاد:

(Addison-Wesley Essential C# 4.0 (978 Pages

(Wrox Professional C# 4 and .NET 4 (1852 Pages

(OReilly C# 4.0 in a Nutshell 4th Edition Feb 2010 (1056 Pages

(Sams C# 4.0 How To Feb 2010 (669 Pges

(Effective C# 50 Specific Ways to Improve Your C# Second Edition (342 Pges

نویسنده: مهدی پایروند
تاریخ: ۱۳۸۹/۰۲/۲۱ ۱۰:۰۱:۵۵

راستی فکر کنم یواش یواش باید عنوان وبتون رو عوض کنید به "Net Tips and .Net 4.0."

نویسنده: وحید نصیری
تاریخ: ۱۳۸۹/۰۲/۲۱ ۱۵:۰۹:۲۱

تنها کتاب اختصاصی برای C#4

C# Language Specification 4.0

این کتاب‌هایی که نام بردید تشکیل شده از تمام مؤلفه‌های دات نت. مثلا اون کتاب 1852 صفحه‌ای از WCF تا WPF را هم دارد. خود WPF جامع حدود 800 صفحه است. یا WCF راحت 600 صفحه است.

نویسنده: Pantee
تاریخ: ۱۳۸۹/۰۲/۲۴ ۲۰:۱۰:۲۳

آقا مطلبتو کپی کردن، محض اطلاع:

http://www.zabet.ir/lazy-initialization-pattern-dotnet-4.html

نویسنده: وحید نصیری
تاریخ: ۱۳۸۹/۰۲/۲۵ ۰۱:۰۵:۳۶

سلام، موفق باشید.

عنوان: SortedSet در دات نت 4
نویسنده: وحید نصیری
تاریخ: ۱۳۸۹/۰۲/۲۶ ۱۸:۱۶:۰۰
آدرس: www.dotnettips.info
برچسب‌ها: #C

SortedSet

قرار گرفته در فضای نام System.Collections.Generic دات نت 4، لیستی از اشیاء به صورت خودکار مرتب شده را ارائه می‌دهد. SortedSet نیز همانند HashSet از اعضای منحصربفردی تشکیل خواهد شد اما اینبار به شکلی مرتب شده. برای پیاده سازی آن از red-black tree data structure استفاده شده است که مهم‌ترین مزیت آن امکان افزودن و یا حذف اشیاء به آن بدون کاهش قابل توجه کارآیی برنامه است.

مثال اول:

```
using System;
using System.Collections.Generic;

namespace SortedSetTest
{
   class Program
   {
       static void sample1()
       {
           var setRange = new SortedSet<int> { 2, 5, 6, 2, 1, 4, 8 };

           foreach (var i in setRange)
           {
               Console.WriteLine(i);
           }
       }

       static void Main()
       {
           sample1();
       }
   }
}
```

در این مثال با نحوه‌ی ایجاد این لیست جنریک خود مرتب شونده‌ی تکراری نپذیر (!) آشنا می‌شوید. اگر این مثال را اجرا نمائید، خروجی آن مرتب شده است و همچنین تنها شامل یک عدد 2 است (اعضای تکراری را حذف می‌کند).

مثال دوم:

```
using System;
using System.Collections.Generic;

namespace SortedSetTest
{
   class Program
   {
       static void sample2()
       {
           var setRange = new SortedSet<int>();
           var random = new Random();

           for (int counter = 0; counter < 100; counter++)
           {
               var rnd = random.Next(-180, 181);
               if (!setRange.Add(rnd))
               {
                   Console.WriteLine("Couldn't add {0}", rnd);
               }
           }

           Console.WriteLine("Result set:");
           foreach (var item in setRange)
           {
```

```
                Console.WriteLine(item);
            }
        }

        static void Main()
        {
            sample2();
        }
    }
}
```

در این مثال نحوه‌ی افزودن اعضای مختلف به این لیست ویژه، توسط متد Add آن بیان شده است. اگر آیتمی در این لیست موجود باشد، مجددا اضافه نشده و حاصل متد Add آن، False خواهد بود.

مثال سوم:

اگر از سایر انواع سفارشی تعریف شده استفاده نمائید، باید روش مقایسه‌ی آن‌ها را نیز با پیاده سازی اینترفیس استاندارد IComparable ارائه دهید؛ در غیر اینصورت با خطای At least one object must implement IComparable متوقف خواهید شد.

```
using System;
using System.Collections;
using System.Collections.Generic;

namespace SortedSetTest
{
   class FileInfo
   {
       public string Name { set; get; }
       public long Size { set; get; }
   }

   class FileInfoComparer : IComparer<FileInfo>
   {
       public int Compare(FileInfo x, FileInfo y)
       {
           var caseiComp = new CaseInsensitiveComparer();
           return caseiComp.Compare(x.Name, y.Name);
       }
   }


   class Program
   {

       static void sample3()
       {
           var setRange = new SortedSet<FileInfo>(new FileInfoComparer())
                               {
                                   new FileInfo
                                       {
                                           Name = "file1.txt",
                                           Size = 100
                                       },
                                   new FileInfo
                                       {
                                           Name = "file2.txt",
                                           Size = 10
                                       },
                                       new FileInfo
                                           {
                                           Name = "file3.txt",
                                           Size = 300
                                       }
                               };

           foreach (var item in setRange)
           {
               Console.WriteLine(item.Name);
           }
       }

       static void Main()
       {
           sample3();
```

```
            Console.WriteLine("Press a key...");
            Console.ReadKey();
        }
    }
}
```

در این مثال اشیایی از نوع کلاس FileInfo به لیست ویژه‌ی ما اضافه شده‌اند. برای اینکه امکان مقایسه‌ی آن‌ها فراهم باشد ، کلاس FileInfoComparer با پیاده سازی اینترفیس IComparer ، روش مقایسه دو شیء از این دست را ارائه می‌دهد.

عنوان: **منسوخ شده‌ها در دات نت 4**
نویسنده: وحید نصیری
تاریخ: ۱۳۸۹/۰۳/۰۴ ۰۹:۰۹:۰۰
آدرس: www.dotnettips.info
برچسب‌ها: #C

برای کاهش حجم دات نت در نگارش‌های بعدی، کلاس‌هایی که توسط تیم‌های مختلف پیشتر توسعه یافته بودند، اکنون با هم تلفیق شده و نظم بهتری پیدا کرده‌اند. برای مثال کلیه کلاس‌های مرتبط با زبان‌های اسکریپتی اکنون به فضای نام System.CodeDom.Compiler.ICodeCompiler منتقل شده‌اند و غیره. مرجع نسبتا مفصلی در مورد منسوخ شده‌ها در دات نت 4 اخیرا منتشر شده است:

نوع‌های منسوخ شده
کلاس‌ها و متدهای منسوخ شده

و حتما می‌دانید که چگونه یک متد یا نوع را باید به صورت منسوخ شده معرفی کرد:

The Obsolete attribute

در این حالت هنگام کامپایل برنامه، پیغام اخطاری توسط کامپایلر صادر خواهد شد.

نظرات خوانندگان

نویسنده: Amin
تاریخ: ۱۳۸۹/۰۳/۰۴ ۱۱:۵۴:۳۶

ممنون آقای نصیری.

نویسنده: Mohammad Shams Javi
تاریخ: ۱۳۸۹/۰۴/۲۹ ۱۱:۵۳:۵۳

مطلب مفیدی بود.

عنوان: آرگومان‌های نامگذاری شده (named arguments/parameters) در C#4 :
نویسنده: وحید نصیری
تاریخ: ۱۳۸۹/۰۳/۱۶ ۱۸:۵۸:۰۰
آدرس: www.dotnettips.info
برچسب‌ها: C#

سؤال: آیا شما بدون مراجعه به مستندات شیء FormsAuthenticationTicket می‌توانید پاسخ دهید هر کدام از آرگومان‌های ذکر شده در سازنده‌ی این کلاس چه کاربرد و معنایی دارند؟

```
var ticket = new FormsAuthenticationTicket
               (
                 1,
                 principal.Identity.Name,
                 DateTime.Now,
                 DateTime.Now.AddMinutes(30),
                 true,
                 string.Empty,
                 FormsAuthentication.FormsCookiePath
               );
```

الان چطور؟!

```
//using named parameters of C# 4.0
           var ticket = new FormsAuthenticationTicket
               (
                 version: 1,
                 name: principal.Identity.Name,
                 issueDate: DateTime.Now,
                 expiration: DateTime.Now.AddMinutes(30),
                 isPersistent: true,
                 userData: string.Empty,
                 cookiePath: FormsAuthentication.FormsCookiePath
               );
```

گاهی از اوقات ویژگی‌هایی که به یک زبان برنامه نویسی اضافه می‌شوند راه و روش برنامه نویسی را ممکن است کاملا تحت تاثیر قرار دهند؛ برای مثال استفاده از Generics . اما گاهی از اوقات این ویژگی‌های جدید syntax sugar یا nice to have هستند(!) مانند Named arguments .

Named arguments یا آرگومان‌های نامگذاری شده خوانایی کدهای شما را با بیان اینکه هر آرگومان چه معنایی دارد، افزایش می‌دهند. برای مثال به نظر شما کدام روش فراخوانی متد Copy در ذیل خواناتر است؟

```
File.Copy("source.txt", "destination.txt", true);
```

و یا؟

```
File.Copy("source.txt", "destination.txt", overwrite: true);
```

هر دو یک کار را انجام می‌دهند اما درک عملکرد روش اول حتما نیاز به بررسی مستندات متد Copy را خواهد داشت.

نکات تکمیلی:

متد ساده زیر را در نظر بگیرید:

```
static void Foo(int x, int y) {}
```

الف) در حالت معمولی و بدون استفاده از named argument ، پارامترهای یک متد، positional parameters هم نامیده می‌شوند. به این معنا که محل ذکر یک پارامتر مهم است و برای مثال در روش متداول فراخوانی ذیل، پارامتر دوم همیشه همان y است و پارامتر اول همواره همان x درنظر گرفته می‌شود:

```
Foo(1,2);
```

اما زمانیکه از named parameters استفاده می‌کنید، این ترتیب دیگر اهمیتی نداشته و هر دو فراخوانی ذیل مجاز می‌باشند و به یک معنا هستند:

```
Foo(x:4, y:2);
Foo(y:2, x:4);
```

ب) همواره positional parameters باید پیش از named parameters ذکر شوند. برای مثال فراخوانی ذیل مجاز است:

```
Foo(4, y:2);
```

اما فراخوانی زیر مجاز نبوده و برنامه کامپایل نخواهد شد:

```
Foo(x:4, 2);     // Compile time error
```

نظرات خوانندگان

نویسنده: Amin
تاریخ: ۱۳۸۹/۰۳/۱۷ ۰۸:۵۲:۱۰

جالب بود. ممنون

عنوان: **به روز رسانی کتاب Threading in CS**
نویسنده: وحید نصیری
تاریخ: ۱۳۸۹/۰۶/۲۷ ۱۲:۰۸:۰۰
آدرس: www.dotnettips.info
برچسب‌ها: #C

آقای Albahari (نویسنده برنامه معروف LINQPad) کتاب رایگان خودشون رو در مورد برنامه نویسی چند ریسمانی در سی شارپ به روز کرده‌اند که از آدرس ذیل قابل دریافت است. این به روز رسانی‌ها شامل مباحث اضافه شده در دات نت 4 مانند tasks و غیره که از مزایای پردازش موازی بهره می‌برند نیز می‌شوند.

Threading in C#, Joseph Albahari

عنوان: **مدیریت ساده‌تر امور Async (غیر همزمان) در نسخه‌ی بعدی زبان‌های دات نتی**
نویسنده: وحید نصیری
تاریخ: ۱۳۸۹/۰۸/۱۹ ۱۱:۰۴:۰۰
آدرس: www.dotnettips.info
برچسب‌ها: #C

چندی قبل مطلبی را در این سایت در مورد معرفی الگویی که توسط آن می‌توان اعمال غیر همزمان را به صورت پی در پی انجام داد، مطالعه کردید:

انجام پی در پی اعمال Async به کمک Iterators - قسمت اول
انجام پی در پی اعمال Async به کمک Iterators - قسمت دوم

و بحث اصلی مطالب فوق هم این است:
"در برنامه نویسی متداول همیشه عادت داریم که اعمال به صورت A -> B -> C انجام شوند. اما در Async programming ممکن است ابتدا C انجام شود، سپس A و بعد B یا هر حالت دیگری صرفنظر از تقدم و تاخر آن‌ها در حین معرفی متدهای مرتبط در یک قطعه کد. همچنین میزان خوانایی این نوع کدنویسی نیز مطلوب نیست...."

خبر خوش آن است که پشتیبانی از این نوع مدل پی در پی برنامه نویسی در نگارش‌های بعدی سی شارپ و VB.NET اضافه شده است.

لیستی از مقالات منتشر شده در این مورد را در ادامه ملاحظه خواهید کرد:

What is Visual Studio Async?

Visual Studio Async CTP for the rest of us...

Making Asynchronous Programming Easy

C# 5 Async Exception Handling

C# 5.0 Asynchrony – A Simple Intro and A Quick Look at async/await Concepts in the Async CTP

C# 5 Async, Part 1: Simplifying Asynchrony – That for which we await

Asynchrony in C# 5, Part One

Async, Await and C# vNext

C# 5.0 Asynchronous Made Easy

What's Next in C#? Get Ready for Async!

علاوه بر آن یک سری ویدیوی مرتبط با این بحث نیز منتشر شده است:

مصاحبه با تیم طراح

مصاحبه با Anders Hejlsberg

نظرات خوانندگان

نویسنده: محمود رمضانی
تاریخ: ۱۳۸۹/۰۸/۱۹ ۱۱:۵۰:۰۸

راستش اگه بخوام نظر شخصیم رو بگم باید بگم از سیاست جدید ماکروسافت مبنی بر تمرکز بر روی زبان برنامه نویسی به جای API ها زیاد خوشم نمیاد.
درسته که نتیجش سرعت بیشتر در برنامه نویسی اما یک مشکل بزرگ داره.تا قبل از این اگه شما یک زبان برنامه نویسی رو یاد میگرفتید می تونستید خیلی آسون و سریع اون رو در زبان های دیگه هم یاد بگیرید و ازش استفاده کنید چون بیشتر API ها با زبان های C/C++ نوشته شدن بنابرین زبان ها معمولا یک API رو فراخوانی می کردن فقط نحوه فراخوانی اون ها در زبان ها فرق می کرد .
به عنوان مثال برنامه نویسی سوکت در Java,CSharp,CPlusPlus ... کاملا شبیه هم هستش.MultiThreading هم به همین صورت.اگه شما با یکی از این زبان ها این Concept
ها رو یاد بگیرید می تونید همون Concept رو در بقیه زبان ها هم سریع یاد بگیرید.
اما با این سیاست جدید اگه شما مثلا برنامه نویسی ASync رو در CSharp یاد بگیرید نمی تونید معلومات خودتون رو در این زمینه به زبان های دیگه ببرید و فقط میتونید ازش در دات نت استفاده کنید.
خوشحال می شم نظر بقیه اساتید رو در این زمینه بدونم .

نویسنده: وحید نصیری
تاریخ: ۱۳۸۹/۰۸/۱۹ ۱۲:۰۲:۲۵

- اتفاقا اخیرا مایکروسافت تمرکز کمتری روی زبان‌ها داشته و تمرکز بیشتر اون بر روی کتابخانه‌ها و افزونه‌های مرتبط با دات نت فریم ورک بوده مثل افزونه‌های برنامه نویسی موازی و بهبودهای زیادی که در زمینه برنامه نویسی چند ریسمانی به دات نت 4 اضافه شده. این مبحث ساده سازی async هم در جهت تکمیل این بحث است.
- هنوز هم با تمام این اضافات، حجم مستندات زبان سی شارپ در حد یک دانشنامه‌ی قطور مثل مستندات زبان CPP در نیامده است.
- اینکه مثلا زبان جاوا در چند سال اخیر درجا زده و مثلا در مورد LINQ یا قابلیت‌های پویای زبان یا همین مبحث ساده سازی Async کاری نکرده آیا به نظر شما دلیل مناسبی است برای کم کاری دیگران؟

عنوان: **تغییرات دسترسی به کدها در دات نت 4**
نویسنده: وحید نصیری
تاریخ: ۱۳۸۹/۰۸/۳۰ ۱۷:۴۱:۰۰
آدرس: www.dotnettips.info
برچسب‌ها: #C

دو پروژه‌ی سورس باز XML RPC و Log4Net برای اجرا شدن در برنامه‌های دات نت 4 نیاز به اندکی تغییر در هر دو برنامه‌ی فراخوان و اسمبلی‌های آن‌ها دارند که در ادامه توضیحات مربوطه ارائه خواهند شد.

اگر یک پروژه‌ی جدید دات نت 4 را آغاز کنید و سپس ارجاعی را به یکی از اسمبلی‌های ذکر شده اضافه نمائید، اولین خطایی را که حین استفاده مشاهده خواهید نمود، مورد زیر است:

```
Could not resolve assembly "System.Web".
The assembly is not in the currently targeted framework ".NETFramework,Version=v4.0,Profile=Client".
Please remove references to assemblies not in the targeted framework or consider retargeting your
project.
```

علت هم اینجا است که در تنظیمات پروژه‌های جدید مبتنی بر دات نت 4، پیش فرض Target framework انتخابی توسط VS.NET 2010 از نوع Client profile است؛ که صرفا جهت کاهش حجم دات نت فریم ورک مورد نیاز این نوع برنامه‌ها طراحی شده است. در این پروفایل ساده شده، اسمبلی System.Web وجود ندارد. بنابراین جهت استفاده از کتابخانه‌های XML RPC و یا Log4Net نیاز است تا در خواص پروژه، Target framework را بر روی دات نت فریم ورک 4 کامل قرار داد تا خطای فوق برطرف شود.

خطای دومی که حین کار با کتابخانه‌های XML RPC و یا Log4Net در یک برنامه‌ی دات نت 4 حتما با آن مواجه خواهید شد در ادامه ذکر گردیده است:

```
Inheritance security rules violated while overriding member:
GetObjectData(System.Runtime.Serialization.SerializationInfo,
System.Runtime.Serialization.StreamingContext),
Security accessibility of the overriding method must match the security accessibility of the method
being overriden.
```

متد ISerializable.GetObjectData با ویژگی SecurityCritical در دات نت فریم ورک مزین شده است. با تغییرات امنیتی صورت گرفته در دات نت 4، متدی که این متد را تحریف می‌کند نیز باید با همان سطح دسترسی متد virtual اصلی معرفی گردد و گرنه برنامه اجرا نخواهد شد. البته این مشکل ما نیست؛ مشکل سازندگان کتابخانه‌های ذکر شده است! ولی خوب تا این لحظه برای مثال کتابخانه XML RPC برای دات نت 4 به روز نشده است ولی سورس کامل آن در دسترس است.
برای رفع این مشکل ابتدا سورس این کتابخانه‌ها را دریافت کرده و سپس در فایل AssemblyInfo.cs آن‌ها یک سطر زیر را حذف نموده و پروژه را مجددا کامپایل کنید:

```
[assembly: AllowPartiallyTrustedCallers]
```

علت وجود این ویژگی در کتابخانه‌های ذکر شده این است که بتوان از آن‌ها در محیط‌های اصطلاحا partially trusted (برای مثال هر برنامه‌ای که در internet zone یا intranet zone اجرا می‌شود) استفاده کرد. در دات نت 4 با تغییرات انجام شده معنای AllowPartiallyTrustedCallers به security-transparency تغییر کرده است. بنابراین با قید آن یا باید هر جایی که متد GetObjectData ذکر شده در این کتابخانه‌ها تحریف می‌شود، ویژگی SecurityCritical را صریحا اعمال کرد یا اینکه می‌توان AllowPartiallyTrustedCallers را حذف کرده و وظیفه‌ی انجام آن‌را به CLR محول نمود.

برای مطالعه بیشتر:
Using Libraries from Partially Trusted Code
Security Changes in the .NET Framework 4

TypeLoadException based on Security-Transparent Code, Level 2

Making log4net run on .NET 4.0

نظرات خوانندگان

نویسنده: Afshar Mohebbi
تاریخ: ۱۳۸۹/۰۸/۳۰ ۲۰:۰۰:۲۶

پیشنهاد می‌کنم مطالب این مدلی را به انگلیسی بنویسید نه فارسی. چون مخاطب فارسی زبان آن کم و مخاطب انگلیسی زبان آن به شدت زیاد است.

عنوان: **فلسفه وجودی Path.Combine**
نویسنده: وحید نصیری
تاریخ: ۱۳۹۰/۰۳/۱۳ ۱۸:۲۹:۰۰
آدرس: www.dotnettips.info
برچسب‌ها: #C

عموما اکثر کدهای موجود از روش زیر برای ساخت یک مسیر استفاده می‌کنند:

```
string path = somePath + "\\" + filename;
```

اما اگر همین برنامه تحت Mono در لینوکس اجرا شود به مشکل بر می‌خورد زیرا در لینوکس مسیرها این‌بار به صورت زیر هستند:

```
/somepath/filename
```

به همین جهت توصیه شده است برای ساخت مسیرها در برنامه‌ی خود، از متد Path.Combine موجود در فضای نام System.IO استفاده کنید زیرا این متد از مقادیر Path.DirectorySeperatorChar و Path.VolumeSeparatorChar جهت تهیه مسیر نهایی استفاده می‌کند. این مقادیر در ویندوز (\) و لینوکس (/) متفاوت بوده و به صورت خودکار در زمان اجرا توسط فریم ورک مورد استفاده مدیریت خواهند شد.
همچنین مزیت دیگر استفاده از Path.Combine ، تعیین اعتبار ورودی است؛ به این معنا که اگر از کاراکترهای غیرمجاز استفاده شود، یک استثناء صادر خواهد شد.

یک مورد دیگر هم شاید بد نباشد همینجا اضافه شود و آن هم فلسفه وجودی Environment.NewLine است. مطابق معمول رسم بر این است که سطر جدید با \n در انتهای یک رشته مشخص شود اما این همیشه صحیح نیست و در پلتفرم‌های مختلف متفاوت است. Environment.NewLine در ویندوز مساوی \r\n است و در سیستم‌های مبتنی بر Unix مساوی \n خواهد بود. به همین جهت بهتر است از این پس بجای \n از Environment.NewLine جهت مشخص سازی سطر جدید استفاده کنید.

نظرات خوانندگان

نویسنده: مهران کی آرش
تاریخ: ۱۳۹۰/۰۵/۱۰ ۰۱:۴۰:۴۲

عالی بود

عنوان: **سرعت ایجاد اشیاء CLR**
نویسنده: وحید نصیری
تاریخ: ۱۳۹۰/۰۳/۲۲ ۱۱:۲۷:۰۰
آدرس: www.dotnettips.info
برچسب‌ها: #C

به نظر شما چه تعداد شیء CLR را می‌توان در یک ثانیه ایجاد کرد؟
برنامه کنسول زیر دو نسخه معمولی و نسخه پردازش موازی یک آزمایش ساده را برای اندازه گیری این مطلب ارائه می‌دهد:

```
using System;
using System.Collections.Generic;
using System.Diagnostics;
using System.Threading;
using System.Threading.Tasks;

namespace ObjectInitSpeedTest
{
   class Program
   {
       //Note: don't forget to build it in Release mode.
       static void Main()
       {
           normalSpeedTest();
           parallelSpeedTest();

           Console.ForegroundColor = ConsoleColor.White;
           Console.WriteLine("Press a key ...");
           Console.ReadKey();
       }

       private static void parallelSpeedTest()
       {
           Console.ForegroundColor = ConsoleColor.Yellow;
           Console.WriteLine("parallelSpeedTest");

           long totalObjectsCreated = 0;
           long totalElapsedTime = 0;

           var tasks = new List<Task>();
           var processorCount = Environment.ProcessorCount;

           Console.WriteLine("Running on {0} cores", processorCount);

           for (var t = 0; t < processorCount; t++)
           {
               tasks.Add(Task.Factory.StartNew(
               () =>
               {
                   const int reps = 1000000000;
                   var sp = Stopwatch.StartNew();
                   for (var j = 0; j < reps; ++j)
                   {
                       new object();
                   }
                   sp.Stop();

                   Interlocked.Add(ref totalObjectsCreated, reps);
                   Interlocked.Add(ref totalElapsedTime, sp.ElapsedMilliseconds);
               }
               ));
           }

           // let's complete all the tasks
           Task.WaitAll(tasks.ToArray());

           Console.WriteLine("Created {0:N} objects in 1 sec\n", (totalObjectsCreated /
(totalElapsedTime / processorCount)) * 1000);
       }

       private static void normalSpeedTest()
       {
           Console.ForegroundColor = ConsoleColor.Green;
           Console.WriteLine("normalSpeedTest");
```

```
            const int reps = 1000000000;
            var sp = Stopwatch.StartNew();
            sp.Start();
            for (var j = 0; j < reps; ++j)
            {
                new object();
            }
            sp.Stop();

            Console.WriteLine("Created {0:N} objects in 1 sec\n", (reps / sp.ElapsedMilliseconds) *
1000);
        }
    }
}
```

هنگام اجرای برنامه فراموش نکنید که حالت Build را بر روی Release قرار دهید.

نظرات خوانندگان

نویسنده: Faraz Fallahi
تاریخ: ۱۳۹۰/۰۳/۲۲ ۱۳:۲۴:۳۹

حالا چه نتیجه ای باید گرفت ؟!؟

http://img.288.ir/images/objectinitspeedtest.png

نویسنده: Faraz Fallahi
تاریخ: ۱۳۹۰/۰۳/۲۲ ۱۳:۳۷:۵۹

http://img.288.ir/images/objectinitspeedtest2.png

نویسنده: وحید نصیری
تاریخ: ۱۳۹۰/۰۳/۲۲ ۱۴:۰۹:۰۶

wow! این 32 هسته رو از کجا آوردید؟!

نویسنده: Faraz Fallahi
تاریخ: ۱۳۹۰/۰۳/۲۲ ۱۴:۵۶:۳۱

int processorCount = 32

نویسنده: Faraz Fallahi
تاریخ: ۱۳۹۰/۰۳/۲۲ ۱۵:۲۰:۴۰

مسئله اینجاس که با چهار برابر کردن مقداز processorCount واقعی نتیجه هم چهار برابر بهتر شده !
(ولی کردم 64 دیگه نتیجه زیاد با 32 تغییری نکرد.)

نویسنده: Hossein Moradinia
تاریخ: ۱۳۹۰/۰۳/۲۳ ۱۸:۳۰:۱۹

من که نفهمیدم چرا با تغییر processorcount به 32 زمان کاهش پیدا کرد!!!
اگه کسی میدونه بگه!!!

نویسنده: وحید نصیری
تاریخ: ۱۳۹۰/۰۳/۲۳ ۲۱:۴۹:۱۱

نسخه پردازش موازی فوق به صورت پیش فرض بر اساس تعداد هسته‌های CPU تنظیم شده یعنی تصور کنید که روی هر هسته واقعا نسخه‌های normalSpeedTest به موازات هم در حال اجرا هستند. اگر 8 هسته باشد، 8 tasks خواهیم داشت.
زمانیکه دستی تعداد هسته رو تنظیم کنید (بیشتر از تعداد واقعی هسته‌ها البته)، به همان نسخه‌ی threading سابق بر می‌گردیم یعنی اینبار task های اضافه شده بر اساس زمان آزاد پردازشی هسته‌ها، بین آن‌ها مرتبا سوئیچ خواهند شد و هر کدام که زودتر پایان یابد زمان پردازشی بیشتری را در اختیار مابقی تردهای ایجاد شده قرار می‌دهد. بدیهی است افزایش این عدد زمانیکه CPU به حد اشباع رسیده است (احتمالا شاید بوی سوختن آن به مشام رسیده باشد ...!)، تاثیر آنچنانی نخواهد داشت. علت اینکه عموما از تعداد هسته برای ایجاد parallel tasks استفاده می‌کنند هم همین مورد است. چون فقط برنامه شما نیست که قرار است از تمام توانایی‌های CPU استفاده کند.

نویسنده: وحید نصیری
تاریخ: ۱۳۹۰/۰۳/۲۳ ۲۲:۰۴:۱۷

یک مطلب دیگر رو هم اضافه کنم. ThreadPool در دات نت 4 به 64 logical processors محدود شده. به عبارتی مثلا حین استفاده از Parallel.For/ForEach این محدودیت وجود دارد و پس از آزاد شدن یک task ، task بعدی (پس از 64 البته) وارد عمل خواهد شد و همینطور الی آخر

+

ویندوز سرور 2008 نگارش R2 فقط تا 256 logical processors رو پشتیبانی می‌کنه.

نویسنده: وحید نصیری
تاریخ: ۱۳۹۰/۰۳/۳۰ ۰۱:۱۲:۵۵

Head-to-head benchmark: C++ vs .NET

نویسنده: A. Karimi
تاریخ: ۱۳۹۰/۰۴/۰۳ ۱۴:۱۳:۵۶

ظاهرا پردازنده‌های نسل i مثل i5 ،i3 و i7 دارای مکانیزم موازی سازی متفاوتی هستند که در این موارد تاثیر زیادی دارند برای مثال از یک پردازنده یک هسته‌ای نسل i می‌توان توان دو هسته‌ای گرفت به شرط دادن توان الکتریکی کافی به پردازنده. به علاوه بسیاری از سخت افزارهای جدید از Overclocking به صورت پویا پشتیبانی می‌کنند و ریختن بار زیادتر بر روی CPU جواب بهتری می‌دهد. غیر از Overclocking وقتی از CPU های مدرن کمتر از حد توانشان استفاده شود بخش‌هایی از خودشان را موقتاً غیر فعال می‌کنند تا از مصرف برق بکاهند وقتی بار CPU زیاد شود این بخش‌های خاموش، روشن شده و قدرت CPU بیشتر خواهد شد.

شبیه همین مکانیزم در صورتی که سخت‌افزار مدرن باشد، درباره RAM نیز اتفاق می‌افتد. فکر می‌کنم به همین دلیل ریختن بار بیشتر بر CPU نتایجی دارد که گاهی عجیب به نظر می‌رسد.

عنوان: **Static Reflection**
نویسنده: وحید نصیری
تاریخ: ۱۳۹۰/۰۵/۰۹ ۲۰:۵۲:۰۰
آدرس: www.dotnettips.info
برچسب‌ها: #C

قابلیت Dynamic reflection یا به اختصار همان reflection متداول، از اولین نگارش‌های دات نت فریم در دسترس است و امکان دسترسی به اطلاعات مرتبط با کلاس‌ها، متدها، خواص و غیره را در زمان اجرا مهیا می‌سازد. تابحال به کمک این قابلیت، امکان تهیه‌ی ابزارهای پیشرفته‌ی زیر مهیا شده است:

انواع و اقسام

- فریم ورک‌های آزمون واحد
- code generators
- ORMs
- ابزارهای آنالیز کد
- و ...

برای مثال فرض کنید که می‌خواهید برای یک کلاس به صورت خودکار، متدهای آزمون واحد تهیه کنید (تهیه یک code generator ساده). اولین نیاز این برنامه، دسترسی به امضای متدها به همراه نام آرگومان‌ها و نوع آن‌ها است. برای حل این مساله باید برای مثال یک parser زبان سی شارپ یا اگر بخواهید کامل‌تر کار کنید، به ازای تمام زبان‌های قابل استفاده در دات نت فریم ورک باید parser تهیه کنید که ... کار ساده‌ای نیست. اما با وجود reflection به سادگی می‌توان به این نوع اطلاعات دسترسی پیدا کرد و نکته‌ی مهم آن هم این است که مستقل است از نوع زبان مورد استفاده. به همین جهت است که این نوع ابزارها را در فریم ورک‌هایی که فاقد امکانات reflection هستند، کمتر می‌توان یافت. برای مثال کیفیت کتابخانه‌های آزمون واحد CPP در مقایسه با آنچه که در دات نت مهیا هستند، اصلا قابل مقایسه نیستند. برای نمونه به یکی از معظم‌ترین فریم ورک‌های آزمون واحد CPP که توسط گوگل تهیه شده مراجعه کنید : (+)

قابلیت Reflection ، مطلب جدیدی نیست و برای مثال زبان جاوا هم سال‌ها است که از آن پشتیبانی می‌کند. اما نگارش سوم دات نت فریم ورک با معرفی LINQ ، lambda expressions و Expressions در یک سطح بالاتر از این Dynamic reflection متداول قرار گرفت.

تعریف Static Reflection :

استفاده از امکانات Reflection API بدون بکارگیری رشته‌ها، به کمک قابلیت اجرای به تعویق افتاده‌ی LINQ، جهت دسترسی به متادیتای المان‌های کد، مانند خواص، متدها و غیره.

برای مثال کد زیر را در نظر بگیرید:

```
//dynamic reflection
PropertyInfo property = typeof (MyClass).GetProperty("Name");
MethodInfo method = typeof (MyClass).GetMethod("SomeMethod");
```

این کد، یک نمونه از دسترسی به متادیتای خواص یا متدها را به کمک Reflection متداول نمایش می‌دهد. مهم‌ترین ایراد آن استفاده از رشته‌ها است که تحت نظر کامپایلر نیستند و تنها زمان اجرا است که مشخص می‌شود آیا MyClass واقعا خاصیتی به نام Name داشته است یا خیر.

چقدر خوب می‌شد اگر این قابلیت بجای dynamic بودن (مشخص شدن در زمان اجرا)، استاتیک می‌بود و در زمان کامپایل قابل بررسی می‌شد. این امکان به کمک lambda expressions و expression trees دات نت سه بعد، میسر شده است. کلیدهای اصلی Static Reflection کلاس‌های Func و Expression هستند. با استفاده از کلاس Func می‌توان lambda expression ایی را تعریف کرد که مقداری را بر می‌گرداند و توسط کلاس Expression می‌توان به محتوای یک delegate دسترسی یافت. ترکیب این دو، قدرت دستیابی به اطلاعاتی مانند PropertyInfo را در زمان طراحی کلاس‌ها، می‌دهد؛ با توجه به اینکه:

- کاملا توسط intellisense موجود در VS.NET پشتیبانی می‌شود.

- با استفاده از ابزارهای refactoring قابل کنترل است.
- از همه مهم‌تر، دیگری خبری از رشته‌ها نبوده و همه چیز تحت کنترل کامپایلر قرار می‌گیرد.

و شاید هیچ قابلیتی به اندازه‌ی Static Reflection در این چندسال اخیر بر روی اکوسیستم دات نت فریم ورک تاثیرگذار نبوده باشد. این روزها کمتر کتابخانه یا فریم ورکی را می‌توانید پیدا کنید که از Static Reflection استفاده نکند. سرآغاز استفاده گسترده از آن به Fluent NHibernate بر می‌گردد؛ سپس در انواع و اقسام ORMs ، mocking frameworks و غیره استفاده شد و مدتی است که در ASP.NET MVC نیز مورد استفاده قرار می‌گیرد (برای مثال TextBoxFor معروف آن):

```
public string TextBoxFor<T>(Expression<Func<T,object>> expression);
```

به این ترتیب حین استفاده از آن دیگری نیازی نخواهد بود تا نام خاصیت مدل مورد نظر را به صورت رشته وارد کرد:

```
<%= this.TextBoxFor(model => model.FirstName); %>
```

یک مثال ساده از تعریف و بکارگیری Static Reflection :

```
public PropertyInfo GetProperty<T>(Expression<Func<T, object>> expression)
{
 var memberExpression = expression.Body as MemberExpression;

 if (memberExpression == null)
     throw new InvalidOperationException("Not a member access.");

  return memberExpression.Member as PropertyInfo;
}
```

همانطور که عنوان شد کلیدهای اصلی بهره‌گیری از امکانات Static reflection ، استفاده از کلاس‌های Expression و Func هستند که در آرگومان متد فوق بکارگرفته شده‌اند و در حقیقت یک expression of a delegate است که به آن Lambdas as Data نیز گفته می‌شود. این delegate پارامتری از نوع T را دریافت کرده و سپس مقداری از نوع object را بر می‌گرداند. اما زمانیکه از کلاس Expression در اینجا استفاده می‌شود، این Func دیگر اجرا نخواهد شد، بلکه از آن به عنوان قطعه کدی که اطلاعاتش قرار است استخراج شود (Lambdas as Data) استفاده می‌شود.

برای نمونه Fluent NHibernate در پشت صحنه متد Map ، به کمک متدی شبیه به GetProperty فوق، a => a.Address1 را به رشته متناظر خاصیت Address1 تبدیل کرده و جهت تعریف نگاشت‌ها مورد استفاده قرار می‌دهد:

```
public class AddressMap : DomainMap<Address>
{
 public AddressMap()
 {
    Map(a => a.Address1);
 }
}
```

جهت اطلاع؛ قابلیت استفاده از «کد به عنوان اطلاعات» هم مفهوم جدیدی نیست و برای مثال زبان Lisp چند دهه است که آن‌را ارائه داده است!

برای مطالعه بیشتر:

Expression Tree Basics

Functional Programming for Everyday .NET Development

Introduction to static reflection

The basics behind static reflection

Dynamic reflection versus static reflection

Static Reflection of property names

Lisp is sin

نظرات خوانندگان

نویسنده: afsharm
تاریخ: ۱۳۹۰/۰۵/۱۰ ۰۸:۳۷:۵۸

سلام،

من همیشه اینجا چیزهای جدید یاد می‌گیرم.

نویسنده: Nima
تاریخ: ۱۳۹۰/۰۵/۱۰ ۱۱:۳۳:۰۲

سلام آقای نصیری
باز هم انگار در این پست مشکل اخیر من رو آموزش دادین.یه جورایی خیلی جالبه
من یه قطعه کد دیده بودم برای پیاده سازی INotifyPropertyChanged که در اینجا پرسیدم :
http://stackoverflow.com/questions/6829099/how-this-code-works-for-handling-inotifypropertychanged
البته آقای مارک گراول گفته که این روش سرعتش پایینه. البته من همچنان از کد خیلی سر در نیاوردم. یعنی سلسله مراتبی که انجام داده رو متوجه نمیشم از کجا نشات میگیره

ممنون و موفق باشی

نویسنده: وحید نصیری
تاریخ: ۱۳۹۰/۰۵/۱۰ ۱۲:۱۱:۰۱

بله. این هم یکی از کاربردهای static reflection در عمل است که در WPF و سیلورلایت می‌تونه مورد استفاده قرار بگیره.
هدف هم حذف رشته ذکر شده در متدهای متداول و اجباری PropertyChanged است که باید به ازای هر خاصیت نوشته شود.
این رشته‌ها (آرگومان‌های PropertyChanged) چون دقیقا همان نام خاصیت‌های تعریف شده در کلاس جاری هستند، بنابراین با استفاده از lambda به عنوان داده (توسط کلاس expression و func) به صورت strongly typed و همچنین قابل تشخیص توسط intellisense می‌توانند تفسیر و قابل دسترسی شوند. زمانیکه Expression Func of T را بجای آرگومان رشته‌ای تعریف کردید، خواص این T توسط intellisense و lambda expression ظاهر می‌شوند. تا اینجا یک مرحله پیشرفت است (شما دیگر رشته ننوشته‌اید و کد هست به عنوان داده). مرحله بعد ترجمه این کد هست به همان رشته. نهایتا متد PropertyChanged نیاز به رشته دارد. اینجا است که کلاس Expression وارد عمل می‌شود و کد را به داده مورد نظر ترجمه می‌کند.

نویسنده: وحید نصیری
تاریخ: ۱۳۹۰/۰۵/۱۰ ۱۲:۲۰:۵۸

جهت تکمیل بحث این کتاب هم اخیرا چاپ شده و بگردید می‌تونید پیداش کنید ....
Functional Programming in C#: Classic Programming Techniques for Modern Projects

نویسنده: Nima
تاریخ: ۱۳۹۰/۰۵/۱۰ ۱۲:۵۳:۳۸

بسیار عالی بود متوجه شدم.بسیار لطف کردین

عنوان: **کاربردهای Static reflection - قسمت اول**
نویسنده: وحید نصیری
تاریخ: ۱۳۹۰/۰۵/۲۰ ۱۹:۰۸:۰۰
آدرس: www.dotnettips.info
برچسب‌ها: #C

در مورد static reflection مقدمه‌ای پیشتر در این سایت قابل مطالعه است (^) و پیشنیاز بحث جاری است. در ادامه قصد داریم یک سری از کاربردهای متداول آن‌را که این روزها در گوشه و کنار وب یافت می‌شود، به زبان ساده بررسی کنیم.

بهبود کدهای موجود

از static reflection در دو حالت کلی می‌توان استفاده کرد. یا قرار است کتابخانه‌ای را از صفر طراحی کنیم یا اینکه خیر؛ کتابخانه‌ای موجود است و می‌خواهیم کیفیت آن‌را بهبود ببخشیم. هدف اصلی هم «حذف رشته‌ها» و «استفاده از کد بجای رشته‌ها» است.
برای مثال قطعه کد زیر یک مثال متداول مرتبط با WPF و یا Silverlight است. در آن با پیاده سازی اینترفیس INotifyPropertyChanged و استفاده از متد raisePropertyChanged ، به رابط کاربری برنامه اعلام خواهیم کرد که لطفا خودت را بر اساس اطلاعات جدید تنظیم شده در قسمت set خاصیت Name ، به روز کن:

```
using System.ComponentModel;

namespace StaticReflection
{
    public class User : INotifyPropertyChanged
    {
        string _name;
        public string Name
        {
            get { return _name; }
            set
            {
                if (_name == value) return;
                _name = value;
                raisePropertyChanged("Name");
            }
        }

        public event PropertyChangedEventHandler PropertyChanged;
        void raisePropertyChanged(string propertyName)
        {
            var handler = PropertyChanged;
            if (handler == null) return;
            handler(this, new PropertyChangedEventArgs(propertyName));
        }
```

```
    }
}
```

تعاریف قسمت PropertyChangedEventArgs این پیاده سازی، خارج از کنترل ما است و در دات نت فریم ورک تعریف شده است. حتما هم نیاز به رشته دارد؛ آن هم نام خاصیتی که تغییر کرده است. چقدر خوب می‌شد اگر می‌توانستیم این رشته را حذف کنیم تا کامپایلر بتواند صحت بکارگیری اطلاعات وارد شده را دقیقا پیش از اجرای برنامه بررسی کند. الان فقط در زمان اجرا است که متوجه خواهیم شد، مثلا آیا به روز رسانی مورد نظر صورت گرفته‌است یا خیر؛ اگر نه، یعنی احتمالا یک اشتباه تایپی جایی وجود دارد.

برای بهبود این کد همانطور که در قسمت قبل نیز گفته شد، از ترکیب کلاس‌های Expression و Func استفاده خواهیم کرد. در اینجا Func قرار نیست چیزی را اجرا کند، بلکه از آن به عنوان قطعه کدی که اطلاعاتش قرار است استخراج شود (Lambdas as Data) استفاده می‌شود. این استخراج اطلاعات هم توسط کلاس Expression انجام می‌شود. بنابراین قسمت اول بهبود کد به صورت زیر شروع می‌شود:

```
void raisePropertyChanged(Expression<Func<object>> expression)
```

الان اگر متد raisePropertyChanged بکارگرفته شده در خاصیت Name را بخواهیم اصلاح کنیم، حداقل با دو واقعه‌ی مطلوب زیر مواجه خواهیم شد:

Intellisense به صورت خودکار کار می‌کند:

حتی بدوی‌ترین ابزارهای Refactoring موجود (منظور همان ابزار توکار VS.NET است!) هم امکان Refactoring را در اینجا فراهم خواهند ساخت:

در پایان کد تکمیل شده فوق به شرح زیر خواهد بود که در آن از کلاس Expression جهت استخراج Member.Name استفاده شده است:

```
using System;
using System.ComponentModel;
using System.Linq.Expressions;

namespace StaticReflection
{
    public class User : INotifyPropertyChanged
    {
        string _name;
        public string Name
        {
            get { return _name; }
            set
            {
                if (_name == value) return;
                _name = value;
                raisePropertyChanged(() => Name);
            }
        }

        public event PropertyChangedEventHandler PropertyChanged;
        void raisePropertyChanged(Expression<Func<object>> expression)
        {
            var memberExpression = expression.Body as MemberExpression;
            if (memberExpression == null)
                throw new InvalidOperationException("Not a member access.");

            var handler = PropertyChanged;
            if (handler == null) return;
            handler(this, new PropertyChangedEventArgs(memberExpression.Member.Name));
        }
    }
}
```

در اینجا باز هم نهایتا به همان PropertyChangedEventArgs استاندارد و موجود، برمی‌گردیم؛ اما آرگومان رشته‌ای آن‌را به کمک ترکیب کلاس‌های Expression و Func تامین خواهیم کرد.

نظرات خوانندگان

نویسنده: Meysam Javadi
تاریخ: ۱۳۹۰/۰۵/۲۳ ۱۰:۱۳:۲۲

این روش مشکل کارایی داره، INPC ها بسیار بیشتر از اون چیزی که ما انتظارش رو داریم اجرا میشن!
به جاش Resharper 6.0 رو نصب بکنید(بایندینگ های XAML رو هم اصلاح میکنه یا میگه تو این DataContext نیست و...)

نویسنده: وحید نصیری
تاریخ: ۱۳۹۰/۰۵/۲۳ ۱۱:۱۴:۲۱

- یک بررسی علمی (بدون علامت تعجب احساسی در انتهای جمله) اینجا هست: (+)
در «یک میلیون بار» اجرا، حدودا 10 ثانیه تفاوت اجرا است نسبت به حالت بکارگیری رشته‌ها.
البته شما در عمل، نه در محیط آزمایشگاهی، پیدا کنید برنامه‌ای را که یک میلیون بار بخواهد خواصی را مرتبا به روز کند.
- زمانیکه LINQ هم ارائه شد، اولین مقالاتی که در این مورد ... در مورد نقد آن منتشر شد، تمرکز را گذاشتند روی کارآیی؛ که این کمی کند است! البته الان کمتر کسی است که در پروژه‌هایش حداقل از LINQ to Objects استفاده نکند. به این دلایل:
- هدف استفاده از LINQ اصلا مسابقه‌ی سرعت نیست.
- هدف تولید کدهای Strongly typed که این اهمیت‌ها را دارند: تحت نظر کامپایلر هستند، قابلیت refactoring دارند و intellisense خودکاری را به همراه خواهند داشت. تمام این‌ها نگهداری یک پروژه را (که اصل زمان اختصاص داده شده به توسعه یک نرم افزار هم همین قسمت نگهداری است)، ساده‌تر و قابل تحمل‌تر می‌کند.
- کاهش حجم کدهای نوشته شده. شما می‌تونید حجم بالایی از if-else و for و حلقه‌ها و غیره رو با یک سطر LINQ نمایش بدید. این هم در بالابردن خوانایی و همچنین نگهداری ساده‌تر برنامه مؤثر است.
- تبدیل ساده‌تر اطلاعات خام به اشیاء (LINQ to xyz ها)
و ...

شما خیلی از مزایا رو بدست خواهید آورد اما خوب مسلما این‌ها هزینه هم دارند. اما نه آنچنان که کسی بخواهد از آن‌ها صرفنظر کند.

نویسنده: Meysam Javadi
تاریخ: ۱۳۹۰/۰۵/۲۳ ۱۱:۲۷:۰۸

کامنتم بیشتر لحن آموزشی داشت تا انتقاد. تو یکی از پست های شما در مورد پیاده سازی INPC پیشنهادم رو داده بودم فکر کنم
http://justinangel.net/AutomagicallyImplementingINotifyPropertyChanged

نویسنده: وحید نصیری
تاریخ: ۱۳۹۰/۰۵/۲۳ ۱۲:۰۷:۳۵

هدف من از این بحث، بحث در مورد refactoring متدی بود که رشته‌ای را که دقیقا نام یکی از خاصیت‌های یک کلاس است را قبول می‌کند. می‌تونست یک مثال دیگر باشد. می‌تونست اصلا ربطی به این INPC نداشته باشد.

عنوان: **ایجاد لیستی از کلاسی جنریک**
نویسنده: وحید نصیری
تاریخ: ۱۳۹۰/۰۵/۲۹ ۲۱:۲۰:۰۰
آدرس: www.dotnettips.info
برچسب‌ها: #C

کلاس جنریک زیر را در نظر بگیرید:

```
public class Column<T>
{
     public string Name { set; get; }
     public T Data { set; get; }
}
```

مشکلی که با این نوع کلاس‌ها وجود دارد این است که نمی‌توان مثلا لیست زیر را در مورد آن‌ها تعریف کرد:

```
IList<Column<T>> myList = new List<Column<T>>();
```

به عبارتی می‌خواهیم یک لیست از کلاسی جنریک داشته باشیم. راه حل انجام آن به صورت زیر است:

```
using System.Collections;

namespace Tests
{
    public interface IColumn
    {
        string Name { set; get; }
        object Data { set; get; }
    }

    public class Column<T> : IColumn
    {
        public string Name { set; get; }

        public T Data { set; get; }

        object IColumn.Data
        {
            get { return this.Data; }
```

```
            set {  this.Data = (T)value; }
        }
    }
}
```

ابتدا یک اینترفیس عمومی را همانند اعضای کلاس Column تعریف می‌کنیم که در آن بجای T از object استفاده شده است. سپس یک پیاده سازی جنریک از این اینترفیس را ارائه خواهیم داد؛ با این تفاوت که اینبار خاصیت Data مربوط به اینترفیس، به صورت خصوصی و صریح با استفاده از IColumn.Data تعریف می‌شود و نمونه‌ی جنریک هم نام آن، عمومی خواهد بود.
اکنون می‌توان نوشت:

```
var myList = new List<IColumn>();
```

برای مثال در این حالت تعریف لیست زیر که از تعدادی وهله‌ی کلاسی جنریک ایجاد شده، کاملا مجاز می‌باشد:

```
var myList = new List<IColumn>
{
    new Column<int> { Data = 1, Name = "Col1"},
    new Column<double> { Data = 1.2, Name = "Col2"}
};
```

خوب، تا اینجا یک مرحله پیشرفت است.اکنون اگر بخواهیم در این لیست، Data مثلا عنصری را که نامش Col1 است، دریافت کنیم چه باید کرد؟ آن هم نه به شکل object بلکه از نوع T مشخص:

```
static T GetColumnData<T>(IList<IColumn> list, string name)
{
    var column = (Column<T>)Convert.ChangeType(list.Single(s => s.Name.Equals(name)),
typeof(Column<T>), null);
    return column.Data;
}
```

و نمونه‌ای از استفاده آن:

```
int data = GetColumnData<int>(myList, "Col1");
```

نظرات خوانندگان

نویسنده: رضا عرب
تاریخ: ۱۳۹۰/۰۵/۳۱ ۰۳:۰۳:۱۴

ممنون از مطلب مفیدتون، به نظر شما این کد خواناتر نیست؟

```
(static T GetColumnData(IList list, string name
}
return (T)list.Single(p => p.Name == name).Data
}
```

نویسنده: وحید نصیری
تاریخ: ۱۳۹۰/۰۵/۳۱ ۰۸:۳۱:۲۷

بله. این روش هم خوبه.
بیشتر هدفم طرح نکته وجود Convert.ChangeType در مورد تبدیل نوع‌های جنریک بود.

عنوان: **تعریف نوع جنریک به صورت متغیر**
نویسنده: وحید نصیری
تاریخ: ۱۳۹۰/۱۲/۱۲ ۱۱:۳۹:۰۰
آدرس: www.dotnettips.info
برچسب‌ها: #C

در تهیه مثال Auto Mapping به کمک امکانات توکار NH 3.2 به این مورد نیاز پیدا کردم:
بتوان نوع متد جنریک را به صورت متغیر تعریف کرد و این نوع در زمان کامپایل برنامه مشخص نباشد. مثلا چیزی شبیه به این مثال:

```
using System;

namespace GenericsSample
{
    class TestGenerics
    {
        public static void Print<T>(T data)
        {
            Console.WriteLine("Print<T>");
        }
    }

    class Program
    {
        static void Main(string[] args)
        {
            var type = typeof(Nullable<int>);
            TestGenerics.Print<type>(1);
        }
    }
}
```

این نوع فراخوانی متد Print در دات نت به صورت پیش فرض غیرمجاز است و نوع جنریک را نمی‌توان به صورت متغیر معرفی کرد.
که البته این هم راه حل دارد و به کمک Reflection قابل حل است:

```
using System;

namespace GenericsSample
{
    class TestGenerics
    {
        public static void Print<T>(T data)
        {
            Console.WriteLine("Print<T>");
        }
    }

    class Program
    {
        static void Main(string[] args)
        {
            var nullableIntType = typeof(Nullable<>).MakeGenericType(typeof(int));
            var method = typeof(TestGenerics).GetMethod("Print");
            var genericMethod = method.MakeGenericMethod(new[] { nullableIntType });
            genericMethod.Invoke(null, new object[] { 1 });
        }
    }
}
```

دو متد MakeGenericType و MakeGenericMethod برای ساخت پویای نوع‌های جنریک و همچنین ارسال آن‌ها به متدهای جنریک در دات نت وجود دارند که مثالی از نحوه استفاده از آن‌ها را در بالا ملاحظه می‌کنید.

مثال دوم:
اگر کلاس TestGenerics نسخه غیرجنریک متد Print را هم داشت، چطور؟ مثلا:

```
class TestGenerics
{
    public static void Print<T>(T data)
    {
        Console.WriteLine("Print<T>");
    }

    public static void Print(object data)
    {
        Console.WriteLine("Print");
    }
}
```

اینبار اگر برنامه فوق را اجرا کنیم، پیغام Ambiguous match found را حین فراخوانی GetMoethod دریافت خواهیم کرد؛ چون دو متد با یک نام در کلاس یاد شده وجود دارند. برای حل این مشکل باید به نحو زیر عمل کرد:

```
using System;
using System.Linq;

namespace GenericsSample
{
    class TestGenerics
    {
        public static void Print<T>(T data)
        {
            Console.WriteLine("Print<T>");
        }

        public static void Print(object data)
        {
            Console.WriteLine("Print");
        }
    }

    class Program
    {
        static void Main(string[] args)
        {
            var nullableIntType = typeof(Nullable<>).MakeGenericType(typeof(int));
            var method = typeof(TestGenerics).GetMethods()
                          .First(x => x.Name == "Print" &&
(x.GetParameters()[0]).ParameterType.IsGenericParameter);
            var genericMethod = method.MakeGenericMethod(new[] { nullableIntType });
            genericMethod.Invoke(null, new object[] { 1 });
        }
    }
}
```

GetMethods تمام متدها را بازگشت داده و سپس بر اساس متادیتای متدها، می‌توان تشخیص داد که کدام یک جنریک است.

نظرات خوانندگان

نویسنده: ZB
تاریخ: ۱۳۹۰/۱۲/۱۲ ۲۲:۰۲:۲۱

سلام آقای نصیری .
ممنون از مطلب مفیدتون. من این کد رو نوشتم برای اینکه یک کلاس رو به متد جنریک بفرستم ولی نمیدونم چطور میتونم T رو به کلاسی که میخوام Cast کنم. ممنون میشم من رو راهنمایی بفرمایید:

var clsType = typeof(MyClass);var method = typeof(TestGenerics).GetMethods().First(x => x.Name == "Print" &&
(x.GetParameters()[0]).ParameterType.IsGenericParameter);genericMethod = method.MakeGenericMethod(new[] {
clsType });genericMethod.Invoke(null, new object[] { new MyClass{P1=100,P2="Name"} });Console.Read();var
clsType = typeof(MyClass);var method = typeof(TestGenerics).GetMethods().First(x => x.Name == "Print" &&
(x.GetParameters()[0]).ParameterType.IsGenericParameter);genericMethod = method.MakeGenericMethod(new[] {
clsType });genericMethod.Invoke(null, new object[] { new MyClass{P1=100,P2="Name"} });Console.Read();var
method = typeof(TestGenerics).GetMethods().First(x => x.Name == "Print" &&
(x.GetParameters()[0]).ParameterType.IsGenericParameter);genericMethod = method.MakeGenericMethod(new[] {
clsType });genericMethod.Invoke(null, new object[] { new MyClass{P1=100,P2="Name"} });Console.Read();new[] {
clsType });genericMethod.Invoke(null, new object[] { new MyClass{P1=100,P2="Name"} });Console.Read();null, new
public static void Print(T data) { و;()object[] { new MyClass{P1=100,P2="Name"} });Console.Read();Console.Read
static void Print(T data) { که از خط آخر ایراد میگیره {{;if (typeof(T) == typeof(MyClass)){var cls = (MyClass)data
if (typeof(T) == که از خط آخر ایراد میگیره {{;if (typeof(T) == typeof(MyClass)){var cls = (MyClass)data
که از خط آخر ایراد میگیره {{;typeof(MyClass)){var cls = (MyClass)data که از خط آخر ایراد میگیره {{;var cls = (MyClass)data
میگیره

نویسنده: وحید نصیری
تاریخ: ۱۳۹۰/۱۲/۱۲ ۲۲:۳۰:۴۲

لطفا از سایت http://pastebin.com/ برای ارسال کد طولانی استفاده کنید. به نظر می‌رسه قسمتی از کد شما توسط بلاگر حذف شده.

نویسنده: ZB
تاریخ: ۱۳۹۰/۱۲/۱۲ ۲۳:۰۹:۳۸

بله ببخشید چون پست بلافاصله نمایش پیدا نمیکنه متوجه نشدم که بهم ریخته.
در این آدرس کد رو قرار دادم : http://pastebin.com/Q4bu1VV1
ولی از این خط 14 ارور میگیرم. در کل میخواستم بدونم چطور T رو به هر کلاسی Cast کنم. ممنون

نویسنده: ZB
تاریخ: ۱۳۹۰/۱۲/۱۲ ۲۳:۲۹:۱۶

آقای نصیری اون مشکل حل شد . کد رو در این آدرس قرار دادم http://pastebin.com/jKcnp6eb

نویسنده: وحید نصیری
تاریخ: ۱۳۹۰/۱۲/۱۲ ۲۳:۴۹:۲۴

این دو روش هم هست:

var cls = data as MyClass

Or

((var cls = (MyClass)Convert.ChangeType(data, typeof(T

نویسنده: ZB
تاریخ: ۱۳۹۰/۱۲/۱۳ ۰۸:۳۱:۱۰

سلام آقای نصیری. ببخشید من یه چیزی رو متوجه نشدم. در مثالی که فرمودین در قسمت اول که گفتین ایراد داره اگر ما اون قسمت رو به این صورت بنویسیم:

;(TestGenerics.Print(1

برنامه کار میکنه. من حتی با کلاس هم امتحان کردم جواب میگیرم. در چه مواردی شما جواب نگرفتین و از رفلکشن استفاده کردین؟ با تشکر

نویسنده: وحید نصیری
تاریخ: ۱۳۹۰/۱۲/۱۳ ۱۰:۱۷:۳۰

موردی که من بهش برخورد کردم این است:
فرض کنید یک متد جنریک دارید که T آن باید به صورت متغیر ساخته شود. مثلا نوع آن EnumStringType of T است.
این T هم هر بار در زمان اجرا بر اساس Enum های تعریف شده در اسمبلی فرق می‌کنه. متد مورد نظر هم آرگومانی نداره.
Map.Type EnumStringType of T است. اینجا است که نیاز خواهید داشت مراحلی که عنوان شد طی شود. یک مرحله خاصیت جنریک پویا تولید شود. یک مرحله متد پویای جنریک تولید شود و بعد هم فراخوانی آن.

نویسنده: Iman Abidi Ashtiani
تاریخ: ۱۳۹۰/۱۲/۱۳ ۱۱:۰۳:۵۸

اگر شما ;(TestGenerics.Print(1
رو با ;(TestGenerics.Print(1
عوض کنید که صورت مسئله هم عوض میشه و تبدیل میشه به یک مطلب آموزشی ابتدایی تویه جنریک ها.
اصلا اینجا دنبال اینیم که مشکل اینکه نوع جنریک را نمی‌توان به صورت متغیر معرفی کرد رو حل کنیم.و یکمم رفلکشن یاد بگیریم و حواسمون به امکانات عالیش باشه

نویسنده: ZB
تاریخ: ۱۳۹۰/۱۲/۱۳ ۱۴:۳۷:۱۵

من منظورم این بود که بهر حال این متغیر یک نوع داره. بجای اینکه ما اول مقدار رو بریزیم داخل یک متغیر و بعد ازش استفاده کنیم مستقیم نوع متغیر رو استفاده کنیم.

نویسنده: Amgh_12
تاریخ: ۱۳۹۰/۱۲/۱۷ ۱۵:۲۱:۲۲

سلام آقای نصیری

ممنون از سایت مفیدتون

می خواستم خواهش کنم برای نرم افزار subtitle tools امکان تصحیح یونیکد گروهی زیرنویس ها را هم فراهم کنید

البته می خواستم تو پست مربوطه نظر بگذارم ولی نظرات بسته بود

ممنون

نویسنده: وحید نصیری

تاریخ: ۱۳۹۰/۱۲/۱۸ ۱۰:۳۶:۰۶

سلام، امکان انتخاب یک پوشه را هم به همان قسمت تبدیل encoding اضافه کردم: (^)

پ.ن.
ایمیل من پایین صفحه هست. همچنین هر پروژه موجود در CodePlex یک قسمت discussion هم دارد برای طرح این نوع موارد.

عنوان: **Data Access Layer based on ADO.NET - Mapper Class**
نویسنده: محمد باقر سیف اللهی
تاریخ: ۱۳۹۱/۰۳/۲۸ ۲۳:۰
آدرس: www.dotnettips.info
برچسب‌ها: C#, DataAccessLayer

مدتی بود هنگام کار کردن با لایه SqlDataProvider در دات نت نیوک ، به این فکر می کردم که چطور میشه بدون استفاده از کلاس های خود دات نت نیوک ، از قابلیت Mapping ی که داره استفاده کرد یا به زبان دیگر یه کلاس Mapper ایجاد کرد (شبیه EF که نیاز به تنظیمات EF نداشته باشه ولی Mapping را انجام دهد. (برای آن دسته از دوستانی که با DotNetNuke آشنایی ندارند باید عرض کنم که DNN یک معماری سه لایه دارد شامل Control ، SqlDataProvider ، DataProvider و Info که در SqlDataProvider شما اطلاعات مربوط به دیتابیس را تنظیم کرده و می توانید آن را با کلاس Info که یک Common Layer محسوب می شود Map کنید) . برای همین شروع به نوشتن یک Data Access Layer کردم که میتواند این نگاشت را بین خروجی کوئری و Property های کلاس ایجاد کند. البته این اسمبلی بیشتر مناسب خروجی با ستون های زیاد و سطر های کم می باشد.
این اسمبلی را برای دانلود قرار می دهم و از شما می خواهم که آن را آزمایش کرده و نظر خود را با بنده درمیان بگذارید .
با تشکر و آرزوی موفقیت

دانلود
Version 1.2.0.0 + Help File

عنوان: **اصول برنامه نویسی موازی درNET. نسخه 4 بخش اول - 1**
نویسنده: اشکان 56
تاریخ: ۱۳۹۱/۰۳/۳۱ ۱۲:۷
آدرس: www.dotnettips.info
برچسب‌ها: C#, Task Programming Language, Threading, Parallel Extensions

بدون هیچ مطلب اضافی به سراغ اولین مثال می‌رویم. قطعه کد زیر را در نظر بگیرید :

```
using System;
using System.Threading.Tasks;

namespace Listing_01 {

class Listing_01 {

static void Main(string[] args) {
  Task.Factory.StartNew(() => {
         Console.WriteLine("Hello World");
  });

  // wait for input before exiting
  Console.WriteLine("Main method complete. Press enter to finish.");
  Console.ReadLine();
 }
}
```

در کد بالا کلاس Task نقش اصلی را بازی می‌کند.این کلاس قلب کتابخانه برنامه نویسی Task یا Task Programming Library می‌باشد.

در این بخش با موارد زیر در مورد Taskها آشنا می‌شویم:

- ایجاد و به کار انداختن انواع مختلف Task ها.
- کنسل کردن Task ها.
- منتظر شدن برای پایان یک Task.
- دریافت خروجی یا نتیجه از یک Task پایان یافته.
- مدیریت خطا در طول انجام یک Task

خب بهتر است به شرح کد بالا بپردازیم:

رای استفاده از کلاس Task باید فضای نام System.Threading.Tasks را بصورت ریر مورد استفاده قرار دهیم.

```
using System.Threading.Tasks;
```

این فضای نام نقش بسیار مهمی در برنامه نویسی Taskها دارد . فضای نام بعدی معروف است : System.Threading . اگر با برنامه نویسی تریدها بروش مرسوم وکلاسیک آشنایی دارید قطعاً با این فضای نام آشنایی دارید. اگر بخواهیم با چندین Task بطور همزمان کار کنیم به این فضای نام نیاز مبرم داریم. پس :

```
using System.Threading;
```

خب رسیدیم به بخش مهم برنامه :

```
Task.Factory.StartNew(() => {
   Console.WriteLine("Hello World");
});
```

متد استاتیک Task.Factory.StartNew یک Task جدید را ایجاد و شروع می‌کند که متن Hello Word را در خروجی کنسول نمایش می‌دهد. این روش ساده‌ترین راه برای ایجاد و شروع یک Task است.

در بخش‌های بعدی چگونگی ایجاد Taskهای پیچیده‌تر را بررسی خواهیم کرد . خروجی برنامه بالا بصورت زیر خواهد بود:

```
Main method complete. Press enter to finish.

Hello World
```

روشهای مختلف ایجاد یک Task ساده :

- ایجاد کلاس Task با استفاده از یک متد دارای نام که در داخل یک کلاس Action صدا زده می‌شود. مثال :

```
Task task1 = new Task(new Action(printMessage));
```

استفاده از یک delegate ناشناس (بدون نام). مثال :

```
Task task2 = new Task(delegate {
   printMessage();
});
```

- استفاده از یک عبارت لامبدا و یک متد دارای نام . مثال :

```
Task task3 = new Task(() => printMessage());
```

- استفاده از یک عبارت لامبدا و یک متد ناشناس (بدون نام). مثال :

```
Task task4 = new Task(() => {
   printMessage();
});
```

قطعه کد زیر مثال خوبی برای چهار روشی که در بالا شرح دادیم می‌باشد:

```
using System;
using System.Threading.Tasks;

namespace Listing_02 {

class Listing_02 {

static void Main(string[] args) {

   // use an Action delegate and a named method
   Task task1 = new Task(new Action(printMessage));

   // use a anonymous delegate
   Task task2 = new Task(delegate {
   printMessage();
});

  // use a lambda expression and a named method
  Task task3 = new Task(() => printMessage());

  // use a lambda expression and an anonymous method
  Task task4 = new Task(() => {
    printMessage();
  });
  task1.Start();
  task2.Start();
  task3.Start();
  task4.Start();

  // wait for input before exiting
```

```
  Console.WriteLine("Main method complete. Press enter to finish.");
  Console.ReadLine();
 }

 static void printMessage() {
  Console.WriteLine("Hello World");
  }
 }
}
```

خروجی برنامه بالا بصورت زیر است :

```
Main method complete. Press enter to finish.

Hello World

Hello World

Hello World

Hello World
```

نکته 1 : از مند استاتیک Task.Factory.StartNew برای ایجاد Task هایی که رمان اجرای کوتاه دارند استفاده می‌شود.

نکته 2 : اگر یک Taskدر حال اجرا باشد نمی‌توان آنرا دوباره استارت نمود باید برای یک نمونه جدید از آن Task ایجاد نمود و آنرا استارت کرد.

نظرات خوانندگان

نویسنده: رحمت رضایی
تاریخ: ۱۳۹۱/۰۳/۳۱ ۱۷:۵۱

از مند استاتیک Task.Factory.StartNew برای ایجاد Task هایی که زمان اجرای طولانی هم دارند استفاده می‌شود :

```
Task.Factory.StartNew(() =>
            {
                Thread.Sleep(1000);
            }, TaskCreationOptions.LongRunning);
```

نویسنده: ali
تاریخ: ۱۳۹۱/۰۳/۳۱ ۱۸:۳۷

سلام
مرسی از آموزشتون

این روش چه برتری نسبت به شیوه کلاسیک موازی کاری داره ؟
آیا همه امکانات شیوه کلاسیک رو پوشش می‌ده ؟

بی صبرانه منتظر ادامه آموزش هستم.
پیروز باشید.

نویسنده: حسین مرادی نیا
تاریخ: ۱۳۹۱/۰۴/۰۱ ۱:۵۹

اگر منظور شما از روش‌های کلاسیک استفاده از Threadهاست باید بدانید که آن روش‌ها برای CPUهای تک هسته ای در نظر گرفته شده بودند. همانطور که می‌دانید در CPUهای تک هسته ای ، CPU تنها قادر به اجرای یک وظیفه در یک واحد زمان می‌باشد. در این CPUها برای اینکه بتوان چندین وظیفه را همراه با هم انجام داد CPU بین کارهای در حال انجام در بازه‌های زمانی مختلف سوییچ میکند و برای ما اینطور به نظر می‌آید که CPU در حال انجام چند وظیفه در یک زمان است.
اما در CPUها چند هسته ای امروزی هر هسته قادر به اجرای یک وظیفه به صورت مجزا می‌باشد و این CPUها برای انجام کارهای همزمان عملکرد بسیار بسیار بهتری نسبت به CPUهای تک هسته ای دارند.
با توجه به این موضوع برای اینکه بتوان از قابلیتهای چند هسته ای CPUهای امروزی استفاده کرد باید برنامه نویسی موازی (Parallel Programming) انجام داد و روش‌های کلاسیک مناسب این کار نمی‌باشند.

نویسنده: محمد
تاریخ: ۱۳۹۱/۰۴/۰۱ ۹:۴۸

ممنون

نویسنده: saleh
تاریخ: ۱۳۹۱/۰۴/۱۷ ۰:۱۲

شما در کد خودتون taskها را قبل از دستور چاپ متن main method ... نوشته و استارت داده بودید ولی در خروجی برعکس این موضوع اتفاق افتاده! میشه درموردش توضیح بدید؟

نویسنده: وحید نصیری
تاریخ: ۱۳۹۱/۰۴/۱۷ ۰:۳۰

توضیحات بیشتر در اینجا

عنوان: اصول برنامه نویسی موازی درNET. نسخه 4 بخش اول - 2
نویسنده: اشکان 56
تاریخ: ۱۳۹۱/۰۴/۰۱ ۱:۱۹
آدرس: www.dotnettips.info
برچسب‌ها: C#, Task Programming Language, Threading, Parallel Extensions

تنظیم وضعیت برای یک Task

در مثال ذکر شده در قسمت قبل هر چهار Task یک عبارت را در خروجی نمایش دادند حال می‌خواهیم هر Task پیغام متفاوتی را نمایش دهد.برای این کار از کلاس زیر استفاده می‌کنیم :

```
System.Action<object>
```

تنظیم وضعیت برای یک Task این امکان را فراهم می‌کند که بر روی اطلاعات مختلفی یک پروسه مشابه را انجام داد.

مثال :

```
namespace Listing_03 {
class Listing_03 {
  static void Main(string[] args) {
   // use an Action delegate and a named method
   Task task1 = new Task(new Action<object>(printMessage),"First task");

   // use an anonymous delegate
   Task task2 = new Task(delegate (object obj) {
     printMessage(obj);
   }, "Second Task");

   // use a lambda expression and a named method
   // note that parameters to a lambda don't need
   // to be quoted if there is only one parameter
   Task task3 = new Task((obj) => printMessage(obj), "Third task");

   // use a lambda expression and an anonymous method
   Task task4 = new Task((obj) => {
    printMessage(obj);
   }, "Fourth task");

  task1.Start();
  task2.Start();
  task3.Start();
  task4.Start();

  // wait for input before exiting
  Console.WriteLine("Main method complete. Press enter to finish.");
  Console.ReadLine();
}

 static void printMessage(object message) {
   Console.WriteLine("Message: {0}", message);
  }
 }
}
```

کد بالا را بروش دیگری هم می‌توان نوشت :

```
using System;
using System.Threading.Tasks;

namespace Listing_04 {
class Listing_04 {
  static void Main(string[] args) {
   string[] messages = { "First task", "Second task",
    "Third task", "Fourth task" };

   foreach (string msg in messages) {
     Task myTask = new Task(obj => printMessage((string)obj), msg);
```

```
      myTask.Start();
   }

   // wait for input before exiting
   Console.WriteLine("Main method complete. Press enter to finish.");
   Console.ReadLine();
}

   static void printMessage(string message) {
    Console.WriteLine("Message: {0}", message);
   }
  }
}
```

نکته مهم در کد بالا تبدیل اطلاعات وضعیت Task به رشته کاراکتری است که در عبارت لامبدا مورد استفاده قرار می‌گیرد. System.Action فقط با داده نوع object کار می‌کند.

خروجی برنامه بالا بصورت زیر است :

```
Main method complete. Press enter to finish.

Message: Second task

Message: Fourth task

Message: First task

Message: Third task
```

البته این خروجی برای شما ممکن است متفاوت باشد چون در سیستم شما ممکن است Taskها با ترتیب متفاوتی اجرا شوند.با کمک Task Scheduler برا حتی می‌توان ترتیب اجرای Taskها را کنترل نمود

نظرات خوانندگان

نویسنده: حسین مرادی نیا
تاریخ: ۱۳۹۱/۰۴/۰۱ ۳:۳۲

در برنامه بالا ابتدا Taskها را Start کرده و سپس کد زیر اجرا می‌شود:

```
Console.WriteLine("Main method complete. Press enter to finish.");
```

سوال من اینه که چرا عبارت Main Method Complete.Press Enter to finish اول از همه در خروجی نمایش داده می‌شود؟!

نویسنده: وحید نصیری
تاریخ: ۱۳۹۱/۰۴/۰۱ ۹:۴۴

نوشتن متد Start به این معنا نیست که همین الان باید Start صورت گیرد. بعد Start دوم و بعد مورد سوم و الی آخر. پردازش موازی به همین معنا است و قرار است این موارد به موازات هم اجرا شوند و نه ترتیبی و پشت سر هم .
در یک برنامه کنسول، متد Main یعنی کدهایی که در ترد اصلی برنامه اجرا می‌شوند. زمان اجرای تمام taskهای تعریف شده، با زمان اجرای ترد اصلی برنامه بسیار بسیار نزدیک است اما ممکن است یک تاخیر چند میلی ثانیه‌ای اینجا وجود داشته باشد و آن هم وهله سازی و در صف قرار دادن taskها و اجرای آن‌ها است.
Task در دات نت 4 از thread pool مخصوص CLR استفاده می‌کند که همان thread pool ایی است که توسط متد ThreadPool.QueueUserWorkItem موجود در نگارش‌های قبلی دات نت، مورد استفاده قرار می‌گیرد؛ با این تفاوت که جهت کارکرد با Tasks بهینه سازی شده است (جهت استفاده بهتر از CPUهای چند هسته‌ای).
همچنین باید توجه داشت که استفاده از یک استخر تردها به معنای درصف قرار دادن کارها نیز هست. بنابراین یک زمان بسیار کوتاه جهت در صف قرار دادن کارها و سپس ایجاد تردهای جدید برای اجرای آن‌ها در اینجا باید درنظر گرفت.

یک منبع بسیار عالی برای مباحث پردازش موازی به همراه توضیحات لازم:
http://www.albahari.com/threading/part5.aspx#_Task_Parallelism

نویسنده: حسین مرادی نیا
تاریخ: ۱۳۹۱/۰۴/۰۱ ۱۶:۵۳

مرسی
خیلی مفید بود
اینطور که من فهمیدم CLR همه Taskهای Start شده را جمع آوری کرده و جهت اجرا درون یک صف قرار می‌دهد.
اما شما گفتید که قرار نیست کارها به ترتیب و پشت سر هم اجرا شوند! حال سوال اینجاست که هدف از درون صف قرار دادن Taskها چیست؟! مگر به صورت موازی اجرا نمیشوند؟!

نویسنده: وحید نصیری
تاریخ: ۱۳۹۱/۰۴/۰۱ ۱۷:۲۱

برای اینکه CPUها از لحاظ پردازش موازی دارای توانمندی‌های نامحدودی نیستند و لازم است مکانیزم صف وجود داشته باشد و همچنین برنامه شما تنها برنامه‌ای نیست که حق استفاده از توان پردازشی مهیا را دارد.

عنوان: **OpenID چیست؟**
نویسنده: صابر فتح الهی
تاریخ: ۱۳۹۱/۰۴/۰۲ ۱۹:۲۳
آدرس: www.dotnettips.info
برچسب‌ها: ASP.Net, C#, others, OpenID

چند وقتی میشه که دنبال روش‌های OpenID هستم که ببینم چطوری کار می‌کنند، خودم هم تازه شروع کردم خوب قبل از هر چیزی اول ببینیم مفهوم OpenID چی هست؟ و کم کم جلو میریم و مثال هایی معرفی می‌کنیم.

OpenID به شما اجازه می‌دهد با استفاده از اکانت (نام کاربری) که در یک سایت دارید بتوانید به سایت‌های متفاوتی وارد شوید (لاگین کنید) بدون این که نیاز به ثبت نام دوباره در آن سایت‌ها داشته باشید.

نمونه بارز آن می‌توان به سایت هایی مانند StackOverflow اشاره کرد.

OpenID به شما اجازه می‌دهد شما در یک نام کاربری که برای خود ایجاد کرده اید اطلاعاتی را که دارید با دیگر وبسایت‌ها به اشتراک بگذارید.
با OpenID کلمه عبور شما فقط توسط سرویس دهنده گرفته میشود و سرویس دهنده هویت شما را برای بازید از سایت دیگر تایید می‌کند.از طرف دیگر سرویس دهنده شما تا شما اجازه ندهید هیچ وب سایتی کلمه عبور شما را به هیچ وب سایتی نمی‌بیند. بنابراین نیازی نیست در مورد کلمه عبور خود نگرانی به خود راه دهید.

OpenID به سرعت در حال گسترش بروی وب استف در حال حاضر بیش از یک میلیارد نام کاربری (اکانت) وجود دارد و بیش از 50000 سایت OpenID را پذیرفته و با آن کار می‌کنند. چندین سازمان بزرگ موضوع OpenID را پذیرفته اند، سازمان هایی مانند Google, FaceBook,Yahoo!,Microsoft,AOL,MySpace,Sears, Universal Music Group, France Telecom, Novell, Sun, Telecom Italia
و بسیاری از سازمان‌های دیگر.

در کل می‌توان گفت ما در یک سایت خاص مانند یاهو ، گوگل، و... یک نام کاربری خواهیم داشت سپس برای ارتباط سایت مقصد (مثلا همین سایت) با نام کاربری ما در گوگل به این صورت عمل می‌شود که ابتدا از طریق این سایت ما به سایت گوگل هدایت می‌شویم در آنجا از ما یک تاییدیه جهت استفاده از سرویس OpenID از کاربر میگیرد. در صورت تایید کاربر سایت گوگل از این لحظه جهت احراز هویت کاربر برای ورود به سایت مقصد استفاده می‌کند .
زمانی که کاربر می‌خواهد به این سایت لاگین کند سایت نام کاربری و کلمه عبور او را (در صورتی که قبلا به گوگل لاگین کرده باشد نیازی نیست و وارد سایت می‌شود) به گوگل می‌فرستد و پس از تایید هویت در صورت صحیح بودن اجازه می‌دهد کاربر به آسانی وارد سایت مقصد شود.
تصویر زیر نمایانگر این روش می‌باشد.

در پست‌های آینده مثال‌ها و کامپوننت‌های سورس بازی جهت این کار به شما معرفی خواهیم کرد. جهت مطالعات بیشتر می‌توانید به این لینک‌ها مراجعه کنین (^ و ^ و ^).

نظرات خوانندگان

نویسنده: ashi mashi
تاریخ: ۱۳۹۱/۰۴/۱۸ ۱۱:۳۵

سلام،
این چیزی که شما میگید ، آیا مشابه single sign on هستش ؟
همون کار رو انجام میده ؟
آیا با Active directory قابل اتصال هست یا نه ؟

با تشکر....

نویسنده: صابر فتح اللهی
تاریخ: ۱۳۹۱/۰۴/۱۸ ۱۱:۳۸

فکر نمی‌کنم بشه
چون Active Directory با Domain تنظیم میشه
اما این حالت احراز هویت روی فرم تنظیم میشه
برای SSO مطلب توی همین سایت هست در واقع توی اون روش ما با یک اکانت که توی یک سایت داریم از سایت‌های دیگه روی همون سرویس استفاده میکنیم و میشه گفت حالت اختصاصی داره
اما توی این حالت شما با هر سایتی که این قابلیت داشته باشه میتونی کار کنی

نویسنده: Hamid NCH
تاریخ: ۱۳۹۲/۰۹/۱۴ ۱۲:۳۱

نقطه شروع یادگیری من در برنامه نویسی وب، پروژه رایگان شما جناب آقای استاد صابر فتح الهی بود.
بعد از اون هم که با سایت آقای دلشاد آشنا شدم و اینک سایت فوق العاده پربار جاری به همت تمام عزیزان.
و باز هم الان دارم از شما آقای فتح الهی چیز یاد میگیرم.

عنوان: **StringBuilder**
نویسنده: یوسف نژاد
تاریخ: ۱۳۹۱/۰۴/۰۶ ۱۶:۴۲
آدرس: www.dotnettips.info
برچسب‌ها: C#, StringBuilder

بهترین روش برای تولید و دستکاری یک رشته (string) طولانی و یا دستکاری متناوب و تکراری یک رشته استفاده از کلاس StringBuilder است. این کلاس در فضای نام System.Text قرار داره. شی String در دات‌نت‌فریمورک تغییرناپذیره (immutable)، بدین معنی که پس از ایجاد نمی‌توان محتوای اونو تغییر داد. برای مثال اگر شما بخواین محتوای یک رشته رو با اتصال به رشته‌ای دیگه تغییر بدین، اجازه اینکار را به شما داده نمی‌شه. درعوض به‌صورت خودکار رشته‌ای جدید در حافظه ایجاد میشه و محتوای دو رشته موجود پس از اتصال به هم درون اون قرار می‌گیره. این کار درصورتی‌که تعداد عملیات مشابه زیاد باشه می‌تونه تاثیر منفی بر کارایی و حافظه خالی در دسترس برنامه بگذاره.

کلاس StringBuilder با استفاده از آرایه‌ای از کاراکترها، راه‌حل مناسب و بهینه‌ای رو برای این مشکل فراهم کرده. این کلاس در زمان اجرا به شما اجازه می‌ده تا بدون ایجاد نمونه‌های جدید از کلاس String، محتوای یک رشته رو تغییر بدین. شما می‌تونید نمونه‌ای از این کلاس رو به‌صورت خالی و یا با یک رشته اولیه ایجاد کنید، سپس با استفاده از متدهای متنوع موجود، محتوای رشته رو با استفاده از انواع داده مختلف و به‌صورت دلخواه دستکاری کنید. هم‌چنین با استفاده از متد معروف ()ToString این کلاس می‌تونید در هر لحظه دلخواه رشته تولیدی رو بدست بیارین. دو پراپرتی مهم کلاس StringBuilder رفتارش رو درهنگام افزودن داده‌های جدید کنترل می‌کنن:

Capacity , Length

پراپرتی Capacity اندازه بافر کلاس StringBuilder را تعیین می‌کنه و Length طول رشته جاری موجود در این بافر رو نمایش می‌ده. اگر پس از افزودن داده جدید، طول رشته از اندازه بافر موجود بیشتر بشه، StringBuilder باید یه بافر جدید با اندازه‌ای مناسب ایجاد کنه تا رشته جدید رو بتونه تو خودش نگه داره. اندازه این بافر جدید به‌صورت پیش‌فرض دو برابر اندازه بافر قبلی درنظر گرفته می‌شه. بعد تمام رشته قبلی رو تو این بافر جدید کپی میکنه.

از برنامه ساده زیر میتونین برای بررسی این مسئله استفاده کنین:

```
using System.IO;
using System.Text;

class Program
{
  static void Main()
  {
    using (var writer = new StreamWriter("data.txt"))
    {
      var builder = new StringBuilder();
      for (var i = 0; i <= 256; i++)
      {
        writer.Write(builder.Capacity);
        writer.Write(",");
        writer.Write(builder.Length);
        writer.WriteLine();
        builder.Append('1'); // <-- Add one character
      }
    }
  }
}
```

دقت کنین که برای افزودن یک کاراکتر استفاده از دستور Append با نوع داده char (همونطور که در بالا استفاده شده) بازدهی بهتری نسبت به استفاده از نوع داده string (با یک کاراکتر) داره. خروجی کد فوق به صورت زیره:

```
16, 0
16, 1
16, 2
...
16,14
16,15
16,16
32,17
```

```
...
```

استفاده نامناسب و بی‌دقت از این کلاس می‌تونه منجر به بازسازی‌های متناوب این بافر شده که درنهایت فواید کلاس StringBuilder رو تحت تاثیر قرار میده. درهنگام کار با کلاس StringBuilder اگر از طول رشته موردنظر و یا حد بالایی برای Capacity آن آگاهی حتی نسبی دارین، می‌تونید با مقداردهی مناسب این پراپرتی از این مشکل پرهیز کنید.

نکته : مقدار پیش‌فرض پراپرتی Capacity برابر 16 است.

هنگام مقداردهی پراپرتی‌های Capacity یا Length به موارد زیر توجه داشته باشید:

- مقداردهی Capacity به میزانی کمتر از طول رشته جاری (پراپرتی Length)، منجر به خطای زیر می‌شه:

System.ArgumentOutOfRangeException

خطای مشابهی هنگام مقداردهی پراپرتی Capacityبه بیش از مقدار پراپرتی MaxCapacity رخ می‌دهه.البته این مورد تنها درصورتی‌که بخواین اونو به بیش از حدود 2 گیگابایت (Int32.MaxValue) مقداردهی کنید پیش میاد!

- اگر پراپرتی Length را به مقداری کمتر از طول رشته جاری تنظیم کنید، رشته به اندازه طول تنظیمی کوتاه (truncate) میشه.

- اگر مقدار پراپرتی Length را به میزانی بیشتر از طول رشته جاری تنظیم کنید، فضای خالی موجود در بافر با space پر میشه.

- تنظیم مقدار Length بیشتر از Capacity، منجر به مقداردهی خودکار پراپرتی Capacity به مقدار جدید تنظیم شده برای Length میشه.

در ادامه به یک مثال برای مقایسه کارایی تولید یک رشته طولانی با استفاده از این کلاس میپردازیم. تو این مثال از دو روش برای تولید رشته‌های طولانی استفاده میشه. روش اول که همون روش اتصال رشته‌ها (Concat) به هم هستش و روش دوم هم که استفاده از کلاس StringBuilder است. در قطعه کد زیر کلاس مربوط به عملیات تست رو مشاهده میکنین:

```
namespace StringBuilderTest
{
  internal class SbTest1
  {
    internal Action<string> WriteLog;
    internal int Iterations { get; set; }
    internal string TestString { get; set; }

    internal SbTest1(int iterations, string testString, Action<string> writeLog)
    {
      Iterations = iterations;
      TestString = testString;
      WriteLog = writeLog;
    }

    internal void StartTest()
    {
      var watch = new Stopwatch();

      //StringBuilder
      watch.Start();
      var sbTestResult = SbTest();
      watch.Stop();
      WriteLog(string.Format("StringBuilder time: {0}", watch.ElapsedMilliseconds));

      //Concat
      watch.Start();
      var concatTestResult = ConcatTest();
      watch.Stop();
      WriteLog(string.Format("ConcatTest time: {0}", watch.ElapsedMilliseconds));

      WriteLog(string.Format("Results are{0} the same", sbTestResult == concatTestResult ? string.Empty
```

```
: " NOT"));
    }

    private string SbTest()
    {
      var sb = new StringBuilder(TestString);
      for (var x = 0; x < Iterations; x++)
      {
        sb.Append(TestString);
      }
      return sb.ToString();
    }

    private string ConcatTest()
    {
      string concat = TestString;
      for (var x = 0; x < Iterations; x++)
      {
        concat += TestString;
      }
      return concat;
    }
  }
}
```

دو روش بحث‌شده در کلاس مورد استفاده قرار گرفته و مدت زمان اجرای هر کدوم از عملیات‌ها به خروجی فرستاده میشه. برای استفاده از این کلاس هم میشه از کد زیر در یک برنامه کنسول استفاده کرد:

```
do
{
  Console.Write("Iteration: ");
  var iterations = Convert.ToInt32(Console.ReadLine());
  Console.Write("Test String: ");
  var testString = Console.ReadLine();
  var test1 = new SbTest1(iterations, testString, Console.WriteLine);
  test1.StartTest();
  Console.WriteLine("-----------------------------------------------------------------");
} while (Console.ReadKey(true).Key == ConsoleKey.C); // C = continue
```

برای نمونه خروجی زیر در لپ‌تاپ من (Corei7 2630QM) بدست اومد:

```
C:\windows\system32\cmd.exe
Iteration: 100
Test String: salam
StringBuilder time: 0
ConcatTest time: 0
Results are the same
-----------------------------------------------------------------
Iteration: 1000
Test String: salam
StringBuilder time: 0
ConcatTest time: 1
Results are the same
-----------------------------------------------------------------
Iteration: 1000
Test String: kjasdfjasgdfagdsfkjhagsdfhagsdkfhgasdfgakjsdfgashdgfkajsdgfkjasdgfk
jasgdfkjasgdfkjasgdfkhagsdfkjagsdfkhagsdkfjagsdkjfgaskdjfgaskjdfgafgakjshdfgkajs
dgfkjasdgfkjagfkjasdgfkjasgdfkjasgdfkjagsdfkjagsdkfjgasdkjfgashdfgafgaksfgaksjdh
fgakjsdfgkasjdhgfkajsdgfkj
StringBuilder time: 0
ConcatTest time: 89
Results are the same
-----------------------------------------------------------------
Iteration: 10000
Test String: salam
StringBuilder time: 0
ConcatTest time: 115
Results are the same
-----------------------------------------------------------------
Iteration: 100000
Test String: salam
StringBuilder time: 2
ConcatTest time: 18661
Results are the same
-----------------------------------------------------------------
```

تنظیم خاصیت Capacity به یک مقدار مناسب میتونه تو کارایی تاثیرات زیادی بگذاره. مثلا در مورد مثال فوق میشه یه متد دیگه برای آزمایش تاثیر این مقداردهی به صورت زیر به کلاس برناممون اضافه کنیم:

```
private string SbCapacityTest()
{
  var sb = new StringBuilder(TestString) { Capacity = TestString.Length * Iterations };
  for (var x = 0; x < Iterations; x++)
  {
    sb.Append(TestString);
  }
  return sb.ToString();
}
```

تو این متد قبل از ورود به حلقه مقدار خاصیت Capacity به میزان موردنظر تنظیم شده و نتیجه بدست اومده:

```
C:\windows\system32\cmd.exe
Iteration: 100000
Test String: .NET Tips is awesome!
StringBuilder Capacity time: 5
StringBuilder time: 12
ConcatTest time: 89098
Results are the same
----------------------------------------------------------------
```

مشاهده میشه که روش concat خیلی کنده (دقت کنین که طول رشته اولیه هم بیشتر شده) و برای ادامه کار مقایسه اون رو کامنت کردم و نتایج زیر بدست اومد:

```
C:\windows\system32\cmd.exe
Iteration: 1000000
Test String: .NET Tips is awesome!
StringBuilder Capacity time: 37
StringBuilder time: 115
----------------------------------------------------------------
Iteration: 10000000
Test String: .NET Tips is awesome!
StringBuilder Capacity time: 372
StringBuilder time: 1151
----------------------------------------------------------------
Iteration: 15000000
Test String: .NET Tips is awesome!
Exception of type 'System.OutOfMemoryException' was thrown.
----------------------------------------------------------------
```

می‌بینین که استفاده مناسب از مقداردهی به خاصیت Capacity میتونه تا حدود 300 درصد سرعت برنامه ما رو افزایش بده. البته همیشه اینطوری نخواهد بود. ما در این مثال مقدار دقیق طول رشته نهایی رو میدونستیم که باعث میشه عملیات افزایش بافر کلاس StringBuilder هیچوقت اتفاق نیفته. این امر در واقعیت کمتر پیش میاد.

مقاله موجود در سایت dotnetperls شکل زیر رو به عنوان نتیجه تست بازدهی ارائه میده:

- در مواقعی که عملیاتی همچون مثال بالا طولانی و حجیم ندارین بهتره که از این کلاس استفاده نکنین چون عملیات‌های داخلی این کلاس در عملیات کوچک و سبک (مثل ابتدای نمودار فوق) موجب کندی عملیات میشه. همچنین استفاده از اون نیاز به کدنویسی بیشتری داره.

- این کلاس فشار کمتری به حافظه سیستم وارد میکنه. درمقابل استفاده از روش concat موجب اشغال بیش از حد حافظه میشه که خودش باعث اجرای بیشتر و متناوب‌تر GC میشه که در نهایت کارایی سیستم رو کاهش میده.

- استفاده از این کلاس برای عملیات Replace (و یا عملیات مشابه) در حلقه‌ها جهت کار با رشته‌های طولانی و یا تعداد زیادی رشته میتونه بسیار سریعتر و بهتر عمل کنه چون این کلاس برخلاف کلاس string اشیای جدید تولید نمیکنه.

- یه اشتباه بزرگ در استفاده از این کلاس استفاده از "+" برای اتصال رشته‌های درون StringBuilder هست. هرگز از این کارها نکنین. (فکر کنم واضحه که چرا)

نظرات خوانندگان

نویسنده: پویان
تاریخ: ۱۳۹۱/۰۴/۰۷ ۱۰:۲۲

بسیار مفید بود این اطلاعات ممنون

نویسنده: بهروز راد
تاریخ: ۱۳۹۱/۰۴/۰۷ ۱۳:۱۹

بهتره بعد از اتمام کار با شی ایجاد شده از کلاس StringBuilder، متد Clear اون رو فراخوانی کنید تا حافظه‌ی اشغال شده سریعتر توسط GC آزاد بشه.

نویسنده: یوسف نژاد
تاریخ: ۱۳۹۱/۰۴/۰۷ ۱۵:۸

دقیقا (با تشکر بابت یادآوری).
اما این متد از دات نت 4.0 به بعد اضافه شده. برای نسخه‌های قدیمی میشه از مقداردهی متغیر موردنظر به یک اینستنس جدید از کلاس (برای از بین بردن تمام ریفرنسهای اولیه به آبجکت قدیمی تا GC این نمونه قدیمی رو garbage تشخیص بده) استفاده کرد. یا از راه حل ساده‌تر مقداردهی پراپرتی Length به صفر بهره برد. (یا ^)

نویسنده: hossein101211
تاریخ: ۱۳۹۲/۰۳/۲۳ ۱۴:۱۸

با تشکر از مطلب خوبتون
میخواستم ببینم چه تفاوتی با stringWriter داره؟
اگه امکان داره دلایل استفاده از هر کدوم رو بیان نمایید
آیا این درسته که زمانی از string Writer استفاده میکنیم که فقط قصد نوشتن متن رو در memory داریم بدون اینکه مثلا عملیات اضافی مثل append داشته باشیم؟
وزمانی از StringBuilder استفاده میکنیم که میخوایم عملیات‌های اضافی‌تری و طولانی‌تری انجام بدیم؟

نویسنده: یوسف نژاد
تاریخ: ۱۳۹۲/۰۳/۲۴ ۲۰:۴۲

StringWriter یک TextWriter است (درواقع از آن مشتق شده است) که برای نوشتن متن درون حافظه موقت سیستم طراحی شده است. بنابراین هرجا که نیاز به یک TextWriter باشد میتوان از آن استفاده کرد تا رشته تولیدی درون حافظه نگهداری شود. مثل این مثال درباره HtmlTextWriter .
درضمن StringWriter برای تولید تکست، از StringBuilder استفاده میکند و حتی متدی برای برگرداندن نمونه داخلی از StringBuilder خود دارد.
بنابراین برای تولید رشته با تعداد زیاد عملیات چسباندن رشته‌ها (معمولا بیش از 5 بار) از StringBuilder استفاده میشود و درصورت نیاز به یک نمونه از TextWriter بدون اینکه مکان ذخیره تکست نهایی برای ما مهم باشد از StringWriter استفاده میشود (چون این تکست به صورت موقت درون خافظه سیستم نگهداری میشود). اطلاعات بیشتر
و بیشترتر

عنوان: **استفاده از OpenID در وب سایت جهت احراز هویت کاربران**
نویسنده: صابر فتح الهی
تاریخ: ۱۳۹۱/۰۴/۰۸ ۲۱:۲
آدرس: www.dotnettips.info
برچسب‌ها: ASP.Net, C#, MVC, Security, OpenID

قبلا شرح مختصری در زمینه OpenID در اینجا گفته شد.
حال می‌خواهیم این امکان را در پروژه خود بکار ببریم، جهت این کار باید ابتدا یک پروژه ایجاد کرده و از کتابخانه‌های سورس باز موجود استفاده کرد.
1- ابتدا در ویژوال استودیو یا هر نرم افزار دیگر یک پروژه MVC ایجاد نمایید.

2- نوع Internet Application و برای View Engine سایت Razor را انتخاب نمایید.

3- کتابخانه DotNetOpenId سورس باز را می‌توانید مستقیما از این آدرس دانلود نموده یا از طریق Package Manager Console و با نوشتن Install-Package DotNetOpenAuth به صورت آنلاین این کتابخانه را نصب نمایید.

4- مدل‌های برنامه را مانند زیر ایجاد نمایید

```
using System;
using System.Collections.Generic;
using System.ComponentModel.DataAnnotations;
using System.Globalization;
using System.Security.Cryptography;
using System.Text;
using System.Web.Mvc;
using System.Web.Security;

namespace OpenIDExample.Models
{
    #region Models

    public class ChangePasswordModel
    {
        [Required]
```

```
        [DataType(DataType.Password)]
        [Display(Name = "Current password")]
        public string OldPassword { get; set; }

        [Required]
        [ValidatePasswordLength]
        [DataType(DataType.Password)]
        [Display(Name = "New password")]
        public string NewPassword { get; set; }

        [DataType(DataType.Password)]
        [Display(Name = "Confirm new password")]
        [Compare("NewPassword", ErrorMessage = "The new password and confirmation password do not
match.")]
        public string ConfirmPassword { get; set; }
    }

    public class LogOnModel
    {
        [Display(Name = "OpenID")]
        public string OpenID { get; set; }

        [Required]
        [Display(Name = "User name")]
        public string UserName { get; set; }

        [Required]
        [DataType(DataType.Password)]
        [Display(Name = "Password")]
        public string Password { get; set; }

        [Display(Name = "Remember me?")]
        public bool RememberMe { get; set; }
    }

    public class RegisterModel
    {
        [Display(Name = "OpenID")]
        public string OpenID { get; set; }

        [Required]
        [Display(Name = "User name")]
        public string UserName { get; set; }

        [Required]
        [DataType(DataType.EmailAddress)]
        [Display(Name = "Email address")]
        public string Email { get; set; }

        [Required]
        [ValidatePasswordLength]
        [DataType(DataType.Password)]
        [Display(Name = "Password")]
        public string Password { get; set; }

        [DataType(DataType.Password)]
        [Display(Name = "Confirm password")]
        [Compare("Password", ErrorMessage = "The password and confirmation password do not match.")]
        public string ConfirmPassword { get; set; }
    }

    #endregion Models

    #region Services

    // The FormsAuthentication type is sealed and contains static members, so it is difficult to
    // unit test code that calls its members. The interface and helper class below demonstrate
    // how to create an abstract wrapper around such a type in order to make the AccountController
    // code unit testable.

    public interface IMembershipService
    {
        int MinPasswordLength { get; }

        bool ValidateUser(string userName, string password);

        MembershipCreateStatus CreateUser(string userName, string password, string email, string
OpenID);

        bool ChangePassword(string userName, string oldPassword, string newPassword);
```

```
        MembershipUser GetUser(string OpenID);
    }

    public class AccountMembershipService : IMembershipService
    {
        private readonly MembershipProvider _provider;

        public AccountMembershipService()
            : this(null)
        {
        }

        public AccountMembershipService(MembershipProvider provider)
        {
            _provider = provider ?? Membership.Provider;
        }

        public int MinPasswordLength
        {
            get
            {
                return _provider.MinRequiredPasswordLength;
            }
        }

        public bool ValidateUser(string userName, string password)
        {
            if (String.IsNullOrEmpty(userName)) throw new ArgumentException("Value cannot be null or
empty.", "userName");
            if (String.IsNullOrEmpty(password)) throw new ArgumentException("Value cannot be null or
empty.", "password");

            return _provider.ValidateUser(userName, password);
        }

        public Guid StringToGUID(string value)
        {
            // Create a new instance of the MD5CryptoServiceProvider object.
            MD5 md5Hasher = MD5.Create();
            // Convert the input string to a byte array and compute the hash.
            byte[] data = md5Hasher.ComputeHash(Encoding.Default.GetBytes(value));
            return new Guid(data);
        }

        public MembershipCreateStatus CreateUser(string userName, string password, string email, string
OpenID)
        {
            if (String.IsNullOrEmpty(userName)) throw new ArgumentException("Value cannot be null or
empty.", "userName");
            if (String.IsNullOrEmpty(password)) throw new ArgumentException("Value cannot be null or
empty.", "password");
            if (String.IsNullOrEmpty(email)) throw new ArgumentException("Value cannot be null or
empty.", "email");

            MembershipCreateStatus status;
            _provider.CreateUser(userName, password, email, null, null, true, StringToGUID(OpenID), out
status);
            return status;
        }

        public MembershipUser GetUser(string OpenID)
        {
            return _provider.GetUser(StringToGUID(OpenID), true);
        }

        public bool ChangePassword(string userName, string oldPassword, string newPassword)
        {
            if (String.IsNullOrEmpty(userName)) throw new ArgumentException("Value cannot be null or
empty.", "userName");
            if (String.IsNullOrEmpty(oldPassword)) throw new ArgumentException("Value cannot be null or
empty.", "oldPassword");
            if (String.IsNullOrEmpty(newPassword)) throw new ArgumentException("Value cannot be null or
empty.", "newPassword");

            // The underlying ChangePassword() will throw an exception rather
            // than return false in certain failure scenarios.
            try
            {
                MembershipUser currentUser = _provider.GetUser(userName, true /* userIsOnline */);
                return currentUser.ChangePassword(oldPassword, newPassword);
            }
```

```
            catch (ArgumentException)
            {
                return false;
            }
            catch (MembershipPasswordException)
            {
                return false;
            }
        }

        public MembershipCreateStatus CreateUser(string userName, string password, string email)
        {
            throw new NotImplementedException();
        }
    }

    public interface IFormsAuthenticationService
    {
        void SignIn(string userName, bool createPersistentCookie);

        void SignOut();
    }

    public class FormsAuthenticationService : IFormsAuthenticationService
    {
        public void SignIn(string userName, bool createPersistentCookie)
        {
            if (String.IsNullOrEmpty(userName)) throw new ArgumentException("Value cannot be null or
empty.", "userName");

            FormsAuthentication.SetAuthCookie(userName, createPersistentCookie);
        }

        public void SignOut()
        {
            FormsAuthentication.SignOut();
        }
    }

    #endregion Services

    #region Validation

    public static class AccountValidation
    {
        public static string ErrorCodeToString(MembershipCreateStatus createStatus)
        {
            // See http://go.microsoft.com/fwlink/?LinkID=177550 for
            // a full list of status codes.
            switch (createStatus)
            {
                case MembershipCreateStatus.DuplicateUserName:
                    return "Username already exists. Please enter a different user name.";

                case MembershipCreateStatus.DuplicateEmail:
                    return "A username for that e-mail address already exists. Please enter a different
e-mail address.";

                case MembershipCreateStatus.InvalidPassword:
                    return "The password provided is invalid. Please enter a valid password value.";

                case MembershipCreateStatus.InvalidEmail:
                    return "The e-mail address provided is invalid. Please check the value and try
again.";

                case MembershipCreateStatus.InvalidAnswer:
                    return "The password retrieval answer provided is invalid. Please check the value
and try again.";

                case MembershipCreateStatus.InvalidQuestion:
                    return "The password retrieval question provided is invalid. Please check the value
and try again.";

                case MembershipCreateStatus.InvalidUserName:
                    return "The user name provided is invalid. Please check the value and try again.";

                case MembershipCreateStatus.ProviderError:
                    return "The authentication provider returned an error. Please verify your entry and
try again. If the problem persists, please contact your system administrator.";

                case MembershipCreateStatus.UserRejected:
```

```
                    return "The user creation request has been canceled. Please verify your entry and
try again. If the problem persists, please contact your system administrator.";

                default:
                    return "An unknown error occurred. Please verify your entry and try again. If the
problem persists, please contact your system administrator.";
            }
        }
    }

    [AttributeUsage(AttributeTargets.Field | AttributeTargets.Property, AllowMultiple = false,
Inherited = true)]
    public sealed class ValidatePasswordLengthAttribute : ValidationAttribute, IClientValidatable
    {
        private const string _defaultErrorMessage = "'{0}' must be at least {1} characters long.";
        private readonly int _minCharacters = Membership.Provider.MinRequiredPasswordLength;

        public ValidatePasswordLengthAttribute()
            : base(_defaultErrorMessage)
        {
        }

        public override string FormatErrorMessage(string name)
        {
            return String.Format(CultureInfo.CurrentCulture, ErrorMessageString,
                name, _minCharacters);
        }

        public override bool IsValid(object value)
        {
            string valueAsString = value as string;
            return (valueAsString != null && valueAsString.Length >= _minCharacters);
        }

        public IEnumerable<ModelClientValidationRule> GetClientValidationRules(ModelMetadata metadata,
ControllerContext context)
        {
            return new[]{
                new
ModelClientValidationStringLengthRule(FormatErrorMessage(metadata.GetDisplayName()), _minCharacters,
int.MaxValue)
            };
        }
    }

    #endregion Validation
}
```

5- در پروژه مربوطه یک Controller به نام AccountController ایجاد نمایید. و کدهای زیر را برای آنها وارد نمایید.

```
using System.Web.Mvc;
using System.Web.Routing;
using System.Web.Security;
using DotNetOpenAuth.Messaging;
using DotNetOpenAuth.OpenId;
using DotNetOpenAuth.OpenId.RelyingParty;
using OpenIDExample.Models;

namespace OpenIDExample.Controllers
{
    public class AccountController : Controller
    {
        private static OpenIdRelyingParty openid = new OpenIdRelyingParty();

        public IFormsAuthenticationService FormsService { get; set; }

        public IMembershipService MembershipService { get; set; }

        protected override void Initialize(RequestContext requestContext)
        {
            if (FormsService == null) { FormsService = new FormsAuthenticationService(); }
            if (MembershipService == null) { MembershipService = new AccountMembershipService(); }

            base.Initialize(requestContext);
        }

        // **************************************
        // URL: /Account/LogOn
```

```
        // ************************************

        public ActionResult LogOn()
        {
            return View();
        }

        [HttpPost]
        public ActionResult LogOn(LogOnModel model, string returnUrl)
        {
            if (ModelState.IsValid)
            {
                if (MembershipService.ValidateUser(model.UserName, model.Password))
                {
                    FormsService.SignIn(model.UserName, model.RememberMe);
                    if (Url.IsLocalUrl(returnUrl))
                    {
                        return Redirect(returnUrl);
                    }
                    else
                    {
                        return RedirectToAction("Index", "Home");
                    }
                }
                else
                {
                    ModelState.AddModelError("", "The user name or password provided is incorrect.");
                }
            }

            // If we got this far, something failed, redisplay form
            return View(model);
        }

        // ************************************
        // URL: /Account/LogOff
        // ************************************

        public ActionResult LogOff()
        {
            FormsService.SignOut();

            return RedirectToAction("Index", "Home");
        }

        // ************************************
        // URL: /Account/Register
        // ************************************

        public ActionResult Register(string OpenID)
        {
            ViewBag.PasswordLength = MembershipService.MinPasswordLength;
            ViewBag.OpenID = OpenID;
            return View();
        }

        [HttpPost]
        public ActionResult Register(RegisterModel model)
        {
            if (ModelState.IsValid)
            {
                // Attempt to register the user
                MembershipCreateStatus createStatus = MembershipService.CreateUser(model.UserName,
model.Password, model.Email, model.OpenID);

                if (createStatus == MembershipCreateStatus.Success)
                {
                    FormsService.SignIn(model.UserName, false /* createPersistentCookie */);
                    return RedirectToAction("Index", "Home");
                }
                else
                {
                    ModelState.AddModelError("", AccountValidation.ErrorCodeToString(createStatus));
                }
            }

            // If we got this far, something failed, redisplay form
            ViewBag.PasswordLength = MembershipService.MinPasswordLength;
            return View(model);
        }
```

```
        // ************************************
        // URL: /Account/ChangePassword
        // ************************************

        [Authorize]
        public ActionResult ChangePassword()
        {
            ViewBag.PasswordLength = MembershipService.MinPasswordLength;
            return View();
        }

        [Authorize]
        [HttpPost]
        public ActionResult ChangePassword(ChangePasswordModel model)
        {
            if (ModelState.IsValid)
            {
                if (MembershipService.ChangePassword(User.Identity.Name, model.OldPassword,
model.NewPassword))
                {
                    return RedirectToAction("ChangePasswordSuccess");
                }
                else
                {
                    ModelState.AddModelError("", "The current password is incorrect or the new password
is invalid.");
                }
            }

            // If we got this far, something failed, redisplay form
            ViewBag.PasswordLength = MembershipService.MinPasswordLength;
            return View(model);
        }

        // ************************************
        // URL: /Account/ChangePasswordSuccess
        // ************************************

        public ActionResult ChangePasswordSuccess()
        {
            return View();
        }

        [ValidateInput(false)]
        public ActionResult Authenticate(string returnUrl)
        {
            var response = openid.GetResponse();
            if (response == null)
            {
                //Let us submit the request to OpenID provider
                Identifier id;
                if (Identifier.TryParse(Request.Form["openid_identifier"], out id))
                {
                    try
                    {
                        var request = openid.CreateRequest(Request.Form["openid_identifier"]);
                        return request.RedirectingResponse.AsActionResult();
                    }
                    catch (ProtocolException ex)
                    {
                        ViewBag.Message = ex.Message;
                        return View("LogOn");
                    }
                }

                ViewBag.Message = "Invalid identifier";
                return View("LogOn");
            }

            //Let us check the response
            switch (response.Status)
            {
                case AuthenticationStatus.Authenticated:
                    LogOnModel lm = new LogOnModel();
                    lm.OpenID = response.ClaimedIdentifier;
                    //check if user exist
                    MembershipUser user = MembershipService.GetUser(lm.OpenID);
                    if (user != null)
                    {
                        lm.UserName = user.UserName;
                        FormsService.SignIn(user.UserName, false);
```

```
                    }

                    return View("LogOn", lm);

                case AuthenticationStatus.Canceled:
                    ViewBag.Message = "Canceled at provider";
                    return View("LogOn");
                case AuthenticationStatus.Failed:
                    ViewBag.Message = response.Exception.Message;
                    return View("LogOn");
            }

            return new EmptyResult();
        }
    }
}
```

-6 سپس برای Action به نام LogOn یک View می‌سازیم، برای Authenticate نیازی به ایجاد View ندارد چون قرار است درخواست کاربر را به آدرس دیگری Redirect کند. سپس کدهای زیر را برای View ایجاد شده وارد می‌کنیم.

```
@model OpenIDExample.Models.LogOnModel
@{
    ViewBag.Title = "Log On";
}
<h2>
    Log On</h2>
<p>
    Please enter your username and password. @Html.ActionLink("Register", "Register")
    if you don't have an account.
</p>
<script src="@Url.Content("~/Scripts/jquery.validate.min.js")" type="text/javascript"></script>
<script src="@Url.Content("~/Scripts/jquery.validate.unobtrusive.min.js")"
type="text/javascript"></script>
<form action="Authenticate?ReturnUrl=@HttpUtility.UrlEncode(Request.QueryString["ReturnUrl"])"
method="post" id="openid_form">
<input type="hidden" name="action" value="verify" />
<div>
    <fieldset>
        <legend>Login using OpenID</legend>
        <div class="openid_choice">
            <p>
                Please click your account provider:</p>
            <div id="openid_btns">
            </div>
        </div>
        <div id="openid_input_area">
            @Html.TextBox("openid_identifier")
            <input type="submit" value="Log On" />
        </div>
        <noscript>
            <p>
                OpenID is service that allows you to log-on to many different websites using a single
                indentity. Find out <a href="http://openid.net/what/">more about OpenID</a> and
                <a href="http://openid.net/get/">how to get an OpenID enabled account</a>.</p>
        </noscript>
        <div>
            @if (Model != null)
            {
                if (String.IsNullOrEmpty(Model.UserName))
                {
                <div class="editor-label">
                    @Html.LabelFor(model => model.OpenID)
                </div>
                <div class="editor-field">
                    @Html.DisplayFor(model => model.OpenID)
                </div>
                <p class="button">
                    @Html.ActionLink("New User ,Register", "Register", new { OpenID = Model.OpenID })
                </p>
                }
                else
                {
                    //user exist
                <p class="buttonGreen">
                    <a href="@Url.Action("Index", "Home")">Welcome , @Model.UserName, Continue..." </a>
                </p>
```

```
                }
            }
        </div>
    </fieldset>
</div>
</form>
@Html.ValidationSummary(true, "Login was unsuccessful. Please correct the errors and try again.")
@using (Html.BeginForm())
{
    <div>
        <fieldset>
            <legend>Or Login Normally</legend>
            <div class="editor-label">
                @Html.LabelFor(m => m.UserName)
            </div>
            <div class="editor-field">
                @Html.TextBoxFor(m => m.UserName)
                @Html.ValidationMessageFor(m => m.UserName)
            </div>
            <div class="editor-label">
                @Html.LabelFor(m => m.Password)
            </div>
            <div class="editor-field">
                @Html.PasswordFor(m => m.Password)
                @Html.ValidationMessageFor(m => m.Password)
            </div>
            <div class="editor-label">
                @Html.CheckBoxFor(m => m.RememberMe)
                @Html.LabelFor(m => m.RememberMe)
            </div>
            <p>
                <input type="submit" value="Log On" />
            </p>
        </fieldset>
    </div>
}
```

پس از اجرای پروژه صفحه ای شبیه به پایین مشاهده کرده و سرویس دهنده OpenID خاص خود را می‌توانید انتخاب نمایید.

7- برای فعال سازی عملیات احراز هویت توسط FormsAuthentication در سایت باید تنظیمات زیر را در فایل web.config انجام دهید.

```
<authentication mode="Forms">
      <forms loginUrl="~/Account/LogOn" timeout="2880" />
</authentication>
```

خوب تا اینجا کار تمام است و کاربر در صورتی که در سایت OpenID نام کاربری داشته باشد می‌تواند در سایت شما Login کند.
جهت مطالعات بیشتر ودانلود نمونه کدهای آماده می‌توانید به لینک‌های (^ و ^ و ^ و ^ و ^ و ^ و ^) مراجعه کنید.
کد کامل پروژه را می‌توانید از اینجا دانلود نمایید.

منبع

نظرات خوانندگان

نویسنده: ahmadalli
تاریخ: ۱۳۹۱/۰۴/۰۸ ۲۳:۱۲

خوب این کدهای شما برای نسخه‌های قدیمی MVC هست. من نصفش رو درست متوجه نشدم. اگر میشه برای نسخه ۴ یا ۳ هم مثال بزارید.

نویسنده: امیرحسین جلوداری
تاریخ: ۱۳۹۱/۰۴/۰۹ ۱:۲۵

اگه میشه کد برنامه را ضمیمه کنید...

نویسنده: صابر فتح اللهی
تاریخ: ۱۳۹۱/۰۴/۰۹ ۲:۴

کدی که ابتدا گذاشته بودم ظاهرا خیلی قدیمی بود اون با کدهای جدید تعویض کردم و لینک دانلود کد هم برای دوستان قرار دادم

نویسنده: صابر فتح اللهی
تاریخ: ۱۳۹۱/۰۴/۰۹ ۲:۲۱

اصلاح شد

نویسنده: saleh
تاریخ: ۱۳۹۱/۰۴/۱۷ ۰:۳۳

فرضاً اگه شخصی که با OID لوگین کرده بخواد در فروم سایت من فعالیت کنه چه جوری به اطلاعات پستها و اعمالش پی ببرم؟ چون این شخص در سایت ثبت نام نکرده و طبیعتاً اکانتی در سایت نداره!

در قسمت اول مقاله به این موضوعات اشاره نکردید.

نویسنده: وحید نصیری
تاریخ: ۱۳۹۱/۰۴/۱۷ ۱۱:۴۴

بحث فوروم نوشته شده شما با طراحی یک سیستم جدید (که در اینجا مد نظر است) متفاوت است. سورس فوق را مطالعه کنید تا با نحوه یکپارچگی آن با سیستم membership دات نت آشنا شوید.

نویسنده: احمدعلی شفیعی
تاریخ: ۱۳۹۱/۰۴/۱۸ ۱۲:۳۸

ممنونم

نویسنده: کامران
تاریخ: ۱۳۹۲/۰۲/۱۲ ۲۳:۲۱

سلام دوست عزیز.
قربان برای ASP.NET Webforms هم میتونید مقاله ای رو آماده کنید؟

نویسنده: محسن خان
تاریخ: ۱۳۹۲/۰۲/۱۳ ۰:۴

یک پروژه دیگر هم در سایت در مورد پروتکل OAuth هست.

نویسنده: کامران
تاریخ: ۱۳۹۲/۰۲/۱۳ ۰:۹

ممنون از جوابتون، دوست عزیز این هم سمپل و کدهاش با MVC هست، مقالاتی برای Webforms پیدا کردم به زبان لاتین اما درست توضیح داده نشده متاسفانه

نویسنده: صابر فتح الهی
تاریخ: ۱۳۹۲/۰۲/۱۳ ۰:۲۵

سلام
دوست گلم یه مثال هست گذاشتم توی این آدرس

Google OpenID Authentication در ASP.NET با استفاده از DotNetOpenAuth

عنوان: **Extension Methods**
نویسنده: نیلوفر نوروزی
تاریخ: ۲۲:۵۸ ۱۳۹۱/۰۴/۰۸
آدرس: www.dotnettips.info
برچسب‌ها: C#, Extension Method

Extension methods شما را قادر می‌سازند تا به typeهای موجود بدون اینکه کلاس جدیدی ایجاد کنید که از آن‌ها به ارث رفته باشند، متدهای جدیدی اضافه نمائید و بیشترین استفاده آن‌ها در System.Collections.IEnumerable است.

به طور مثال این امکان وجود ندارد که بتوان بر روی IEnumerableها از دستور Foreach استفاده کرد.

برای نمونه من برای اینکه foreach داشته باشم، آن‌را به لیست تبدیل می‌کردم و سپس از امکان foreach بهره‌مند می‌شدم که شاید کار درستی نباشد.

اما با Extension Method افزودن متد foreach به IEnumerable به نحو زیر میسر است:

```
public static void ForEach<T>(this IEnumerable<T> collection, Action<T> action)
{
   foreach (var item in collection)
           action(item);
}
```

نظرات خوانندگان

نویسنده: بهروز راد
تاریخ: ۱۳۹۱/۰۴/۰۹ ۱۱:۱۲

من فکر نمی‌کنم این روش نوشتن مطلب صحیح باشه. شما خیلی کلی می‌نویسید. بهتره با جزئیات بیشتری به مطلب بپردازید. مثلاً لزوم ذکر نوع داده ای که قصد دارید اون رو توسعه بدید بعد از کلمه‌ی کلیدی this در پارامتر اول یا الزام وجود متد در یک کلاس static و تعریف static خود متد. اینها موضوعاتی هستند که برای فردی که با Extension Methodها آشنا نیست باید گفته بشه.

نویسنده: Amin
تاریخ: ۱۳۹۱/۰۴/۰۹ ۱۲:۱۰

من هم با نظر آقای راد موافقم. کلی گویی بدرد کسی که آشنایی زیادی نداره، نمیخوره.
شرمنده اما حسم از این پست این بود که یه جوری موضوع از سر خودتوت باز کردین.

نویسنده: وحید نصیری
تاریخ: ۱۳۹۱/۰۴/۰۹ ۱۲:۱۳

لطفا این بحث رو ادامه ندید. در یک سری از دسترسی‌های نویسنده‌ها به زودی تجدید نظر خواهد شد. چه کسانی که ابراز تمایل کرده بودند و فقط وقت من رو جهت ثبت آن‌ها نصف روزی تلف کردند؛ چه کسانی که مطالب سطحی ارسال می‌کنند.
با تشکر

ضمن اینکه یک مطلب را هم مد نظر داشته باشید. اینجا هدف بیشتر ذکر یک سری نکته است به همین جهت اسم سایت **tips** دارد (نکات ریز).
اسم سایت **encyclopedia** نیست که برای هر مطلبی قرار باشد کتاب نوشته شود. اگر نوشته شد، چقدر خوب؛ اگر نه ... یک نکته ریز جدید یاد گرفتید. این هم خوب.
همچنین هدف از این سایت خواننده عام با سطح مطالعه و اطلاعات صفر نیست و نبوده. از روز اولش اینطور نبوده و نخواهد بود.

یک مطالب رو هم فراموش نکنید. به قول مولوی: «گر تو بهتر میزنی بستان بزن»
در طی یک ماهی که این سایت در حالت تعلیق بود ... من هر چقدر سایت‌های فارسی زبان فنی برنامه نویسی رو گشتم که دوتا مطلب به درد بخور پیدا کنم ... چیزی نیافتم. هیچی! واقعا دریغ از یک مطلب فنی به درد بخور که منتشر شده باشد. حالا همین عده منتظرند سریع بریزند سر یک نفر شروع کنند به داد و قال.
باز تکرار می‌کنم: «گر تو بهتر میزنی بستان بزن»

نویسنده: نیلوفر نوروزی
تاریخ: ۱۳۹۱/۰۴/۰۹ ۱۴:۵۴

من اصلا قصد این را نداشتم که کلی گویی کنم
اگر مطلب درست نوشته نشده معذرت می‌خوام
هدفم فقط این بود که مطلبی که به خودم خیلی کمک کرده در اختیار دوستان قرار دهم.

نویسنده: peyman
تاریخ: ۱۳۹۱/۰۴/۰۹ ۱۸:۱۴

والا تا جایی که من یادم هست پایه این وبلاگ بر این بوده که افراد تا حدی آشنایی با مفاهیم ابتدایی رو دارند . اما در هر صورت ساده نویسی در بیان مطالب پیچیده میتونه این وبلاگ رو بطور خیلی گسترده‌تر مفید وافع کنه حتی برای افرادی که مبتدی هستند .
در پایان هم باید بگم که من دست و پای اون فردی رو میبوسم که به من حتی یک کلمه یاد بده ... چه آقای راد در وب سایت برنامه نویس . چه آقای نصیری و دوستان گرامیشون در این وبلاگ که مدت زیادی هست وبلاگ مفیدشون رو مطالعه میکنم ...

عنوان: **آشنایی با JSON؛ ساده - خوانا - کم حجم**
نویسنده: سروش ترک زاده
تاریخ: ۱۳۹۱/۰۴/۱۰ ۱۵:۱۹
آدرس: www.dotnettips.info
گروه‌ها: C#, JSON

JSON (JavaScript Object Notation) یک راه مناسب برای نگهداری اطلاعات است و از لحاظ ساختاری شباهت زیادی به XML ، رقیب قدیمی خود دارد.

وب سرویس و آجاکس برای انتقال اطلاعات از این روش استفاده می‌کنند و بعضی از پایگاه‌های داده مانند RavenDB بر مبنای این تکنولوژی پایه گذاری شده اند.

هیچ چیزی نمی‌تواند مثل یک مثال؛ خوانایی ، سادگی و کم حجم بودن این روش را نشان دهد :

اگر یک شئ با ساختار زیر در سی شارپ داشته باشید :

```
class Customer
    {
        public int Id { get; set; }
        public string FirstName { get; set; }
        public string LastName { get; set; }
    }
```

ساختار JSON متناظر با آن (در صورت این که مقدار دهی شده باشد) به صورت زیر است:

```
{
   "Id":1,
   "FirstName":"John",
   "LastName":"Doe"
}
```

و در یک مثال پیچیده‌تر :

```
class Customer
{
        public int Id { get; set; }
        public string FirstName { get; set; }
        public string LastName { get; set; }
        public Car Car { get; set; }
        public IEnumerable<Location> Locations { get; set; }
}

class Location
{
        public int Id { get; set; }
        public string Address { get; set; }
        public int Zip { get; set; }
}

class Car
{
        public int Id { get; set; }
        public string Model { get; set; }
}
```

```
{
      "Id":1,
      "FirstName":"John",
      "LastName":"Doe",
      "Car": {
                      "Id":1,
                      "Model":"Nissan GT-R"
                },
      "Locations":[
```

```
                            {
                                  "Id":1,
                                  "Address":"30 Mortensen Avenue, Salinas",
                                  "Zip":93905
                            },
                            {
                                  "Id":2,
                                  "Address":"65 West Alisal Street, #210, Salinas",
                                  "Zip":95812
                            }
                      ]
}
```

ساختار JSON را مجموعه ای از (نام - مقدار) تشکیل می‌دهد. ساختار مشابه آن در زبان سی شارپ KeyValuePair است.

مشاهده این تصاویر ، بهترین درک را از ساختار JSON به شما می‌دهد.

Json.net یکی از بهترین کتابخانه هایی است که برای کار با این تکنولوژی در net. ارائه شده است. بهترین روش اضافه نمودن آن به پروژه NuGet است.برای این کار دستور زیر را در Package Manager Console وارد کنید.

```
PM> Install-Package Newtonsoft.Json
```

با استفاده از کد زیر می‌توانید یک Object را به فرمت JSON تبدیل کنید.

```
var customer = new Customer
                              {
                                  Id = 1,
                                  FirstName = "John",
                                  LastName = "Doe",
                                  Car = new Car
                                            {
                                                Id = 1,
                                                Model = "Nissan GT-R"
                                            },
                                  Locations = new[]
                                                  {
                                                      new Location
                                                          {
                                                              Id = 1,
                                                              Address = "30 Mortensen Avenue,
Salinas",
                                                              Zip = 93905
                                                          },
                                                      new Location
                                                          {
                                                              Id = 2,
                                                              Address = "65 West Alisal Street, #210,
Salinas",
                                                              Zip = 95812
                                                          },
                                                  }
                              };





 var data = Newtonsoft.Json.JsonConvert.SerializeObject(customer);
```

خروجی تابع SerializeObject رشته ای است که محتوی آن را در چهارمین بلاک کد که در بالاتر آمده است، می‌توانید مشاهده کنید.

برای Deserialize کردن (Cast اطلاعات با فرمت JSON به کلاس موردنظر) از روش زیر بهره می‌گیریم :

```
var customer = Newtonsoft.Json.JsonConvert.DeserializeObject<Customer>(data);
```

آشنایی با این تکنولوژی، پیش درآمدی برای چشیدن طعم NoSQL و معرفی کارآمدترین روش‌های آن است که در آینده خواهیم آموخت...

خوشحال می‌شوم اگر نظرات شما را در باره این موضوع بدانم.

نظرات خوانندگان

نویسنده: احمدعلی شفیعی
تاریخ: ۱۳۹۱/۰۴/۱۰ ۱۵:۳۳

خیلی ممنون. واقعا کاربردی بود.

نویسنده: مهدی پایروند
تاریخ: ۱۳۹۱/۰۴/۱۰ ۱۶:۰

بهترین کتابخانه برای سریالایز کردن دیتا به سمت کلاینت است. و هم سرعتی تقریبا 8 برابر سرعت System.Web.Script.Serialization.JavaScriptSerializer دارد

نویسنده: سروش ترک زاده
تاریخ: ۱۳۹۱/۰۴/۱۰ ۱۷:۴۸

در تصدیق حرف شما، دیدن این مقایسه خالی از لطف نیست. (پایین صفحه : Performance Comparison)

نویسنده: ssm
تاریخ: ۱۳۹۱/۰۴/۱۰ ۲۰:۱۷

با سلام
بسیار عالی بود مشتاقانه منتظر مطالب بعدی شما هستم
موفق باشید

نویسنده: مهدی ترابی
تاریخ: ۱۳۹۱/۰۴/۱۰ ۲۳:۴

بسیار خوب.
لطفن JavaScript Object ~~Nation~~ تصحیح شود به JavaScript Object Notation

نویسنده: سروش ترک زاده
تاریخ: ۱۳۹۱/۰۴/۱۱ ۱۱:۴۶

ممنون از دقت نظر شما. تصحیح شد.

نویسنده: رضا.ب
تاریخ: ۱۳۹۱/۰۴/۱۲ ۳:۷

با درود،

علت کمرنگ شدن نقش XML (اگر کمرنگ شده؟) بخاطر همین سادگی و سبکی JSON هست؟

نویسنده: سروش ترک زاده
تاریخ: ۱۳۹۱/۰۴/۱۲ ۱۰:۳۳

سلام
JSON رقیبی قوی برای XML محسوب می‌شه، اما به نظر هنوز زود هست که بگیم XML در حال حذف شدن هست. چون هنوز سیستم‌های قدرتمندی مثل Microsoft Workflow Generator یا همین RSS که زیاد در طول روز با اون سروکار داریم، بر مبنای این تکنولوژی کار می‌کنند.

نویسنده: رضا
تاریخ: ۱۳۹۳/۰۳/۲۶ ۹:۳۵

با سلام و تبریک به خاطر مقاله مفیدتون
یه سوال برام پیش اومده ، سرعت عمل و کارایی JSON بهتره یا XML ؟
برای انجام یه برنامه لازم هست که بدونم از کدوم بهتره استفاده کنم.
ممنون و متشکر

نویسنده: سروش ترک زاده
تاریخ: ۱۳۹۳/۰۳/۲۶ ۱۱:۵۲

سلام
سرعت عمل، به ابزاری بستگی دارد که به وسیله آن اطلاعات serialize و deserialize می‌شود.
بهترین ابزاری که برای کار با XML معرفی شده است،(البته تا جایی که من خبر دارم) LinqToXML است. کار کردن با آن ساده است اما دردسرهای خاص خودش رو داره.
از طرفی فرمت JSON نسبت به XML، حجم کمتری دارد (حداقل به این دلیل که نیاز به باز و بسته کردن tag نیست)
در مجموع من JSON رو پیشنهاد می‌کنم.

عنوان: **خودمیزبانی ماژول های Nancy**
نویسنده: سلمان عرب عامری
تاریخ: ۱۳۹۱/۰۴/۱۴ ۱۱:۵
آدرس: www.dotnettips.info
برچسب‌ها: C#, Nancy, Self-Hosting

در ادامه بررسی پروژه Nancy، در این مطلب به میزبانی پروژه‌های Nancy بدون نیاز به Asp.net می‌پردازیم. به این معنی که برنامه اجرایی که شما می‌نویسید خود یک سرور ایجاد می‌کند و کاربر با وارد کردن آدرس دستگاه شما در مرورگر خود، صفحات و ماژول‌های طراحی شده توسط شما را مشاهده می‌کند.
از کاربردهای چنین سیستمی به سایت‌های قابل حمل، و یا ارائه خدمات یک نرم افزار بر روی صفحات html می‌توان اشاره کرد. مثل گوگل دسکتاپ و یا گزارشات برخی سرویس‌های ویندوزی و یا حتی تنظیم یک سخت افزار متصل به سیستم از روی شبکه. یک ایده جالب می‌تواند ارسال اس ام اس از طریق شبکه و با جی اس ام مودم باشد. که به عنوان مثال کاربران با ورود به یک صفحه و ثبت پیام بتوانند از طریق جی اس ام مودم متصل به سرور آن را ارسال کنند. با یک مثال ساده ادامه می‌دهیم.

برای شروع یک پروژه از نوع Console بسازید و در Package manager کتابخانه Nancy.Hosting.Self را نصب کنید.
حالا یک ماژول جدید به نام TestModule.cs به پروژه اضافه می‌کنیم.

```
public class TestModule:NancyModule
{
 public TestModule()
 {
 Get["/"] = x=> { return "It is a test for nancy self hosting."; };

 }
}
```

حالا وارد program.cs شده و در متدMain کد زیر را می‌نویسیم:

```
var selfHost = new NancyHost(new Uri("http://localhost:12345"));
selfHost.Start();
Console.ReadKey();
selfHost.Stop();
```

در خط اول پورتی که منتظر دریافت درخواست‌های کاربران است را برابر 12345 قرار می‌دهیم. بنابراین برای تست این کد باید در مرورگر آدرس

http://localhost:12345 را تایپ کنید. اگر بخواهیم کاربر عدد انتهایی را وارد نکند باید از پورت 80 استفاده کنیم که پیش فرض http است ولی اکثرا در سیستم برنامه نویس‌ها توسط IIS مشغول می‌باشد.
در خط بعد سرور را اجرا کرده ایم و برنامه را به حالت انتظار برای فشرده شدن کلیدی در کنسول برده ایم.
وقتی کلیدی در کنسول فشرده شود سرور به حالت توقف می‌رود و اجرای برنامه پایان می‌یابد.
Nancy امکانات دیگری هم دارد. به عنوان مثال می‌توان برای طراحی نمای ماژول‌ها از موتورهای دید استفاده کرد (ViewEngines). موتورهایی مثل Razor و ... . در صورت علاقمندی دوستان، در این باره هم خواهم نگاشت.

نظرات خوانندگان

نویسنده: شهروز جعفری
تاریخ: ۱۳۹۱/۰۴/۱۶ ۱۸:۹

لطفا ادامه بدید تشکر میکنم

عنوان: RavenDB؛ تجربه متفاوت از پایگاه داده
نویسنده: سروش ترک زاده
تاریخ: ۱۳۹۱/۰۴/۱۵ ۱۹:۳۱
آدرس: www.dotnettips.info
برچسب‌ها: C#, JSON, NoSQL, RavenDB

" به شما خواننده گرامی پیشنهاد می‌کنم مطلب قبلی را مطالعه کنید تا پیش زمینه مناسبی در باره این مطلب کسب کنید. "

ماهیت این پایگاه داده وب سرویسی مبتنی بر REST است و فرمت اطلاعاتی که از سرور دریافت می‌شود، JSON است.

گام اول: باید آخرین نسخه RavenDB را دریافت کنید. همان طور که مشاهده می‌کنید، ویرایش‌های مختلف کتابخانه هایی که برای نسخه Client و همچنین Server طراحی شده است، دراین فایل قرار گرفته است.

Name	Date modified	Type	Size
Backup	۲۰۱۲/۰۲/۰۶ ۰۴:۲۰ ...	File folder	
Bundles	۲۰۱۲/۰۲/۰۶ ۰۴:۲۰ ...	File folder	
Client	۲۰۱۲/۰۲/۰۶ ۰۴:۲۰ ...	File folder	
Client-3.5	۲۰۱۲/۰۲/۰۶ ۰۴:۲۰ ...	File folder	
EmbeddedClient	۲۰۱۲/۰۲/۰۶ ۰۴:۲۰ ...	File folder	
Samples	۲۰۱۲/۰۲/۰۶ ۰۴:۲۰ ...	File folder	
Server	۲۰۱۲/۰۲/۰۶ ۰۴:۲۰ ...	File folder	
Silverlight	۲۰۱۲/۰۲/۰۶ ۰۴:۲۰ ...	File folder	
Silverlight-4	۲۰۱۲/۰۲/۰۶ ۰۴:۲۰ ...	File folder	
Smuggler	۲۰۱۲/۰۲/۰۶ ۰۴:۲۰ ...	File folder	
Web	۲۰۱۲/۰۲/۰۶ ۰۴:۲۰ ...	File folder	
acknowledgments	۲۰۱۲/۰۲/۰۶ ۰۴:۱۸ ...	Text Document	
license	۲۰۱۲/۰۲/۰۶ ۰۴:۱۸ ...	Text Document	
Raven-GetBundles	۲۰۱۲/۰۲/۰۶ ۰۴:۱۸ ...	PS1 File	
Raven-UpdateBundles	۲۰۱۲/۰۲/۰۶ ۰۴:۱۸ ...	PS1 File	
readme	۲۰۱۲/۰۲/۰۶ ۰۴:۱۸ ...	Text Document	
Start	۲۰۱۲/۰۲/۰۶ ۰۴:۱۸ ...	Windows Comma...	

برای راه اندازی Server باید فایل Start را اجرا کنید، چند ثانیه بعد محیط مدیریتی آن را در مرورگر خود مشاهده می‌کنید. در بالای صفحه روی لینک Databases کلیک کنید و در صفحه باز شده گزینه New Database را انتخاب کنید. با دادن یک نام دلخواه حالا شما یک پایگاه داده ایجاد کرده اید. تا همین جا دست نگه دارید و اجازه دهید با این محیط دوست داشتنی و قابلیت‌های آن بعدا آشنا شویم.

در گام دوم به Visual Studio می‌رویم و نحوه ارتباط با پایگاه داده و استفاده از دستورات آن را فرا می‌گیریم.

گام دوم:

با یک پروژه Test شروع می‌کنیم که در هر گام تکمیل می‌شود و می‌توانید پروژه کامل را در پایان این پست دانلود کنید.

برای استفاده از کتابخانه‌های مورد نیاز دو راه وجود دارد:
استفاده از NuGet : با استفاده از دستور زیر Package مورد نیاز به پروژه شما افزوده می‌شود.

```
PM> Install-Package RavenDB -Version 1.0.919
```

اضافه کردن کتابخانه‌ها به صورت دستی : کتابخانه‌های مورد نیاز شما در همان فایلی که دانلود شده بود و در پوشه Client قرار دارند.

کتابخانه هایی را که NuGet به پروژه من اضافه کرد، در تصویر زیر مشاهده می‌کنید :

با Newtonsoft.Json در اولین بخش بحث آشنا شدید. NLog هم یک کتابخانه قوی و مستقل برای مدیریت Log است که این پایگاه داده از آن بهره برده است.

" دلیل اینکه از پروژه تست استفاده کردم ؛ تمرکز روی کدها و مشاهده تاثیر آن‌ها ، مستقل از UI و لایه‌های دیگر نرم افزار است. بدیهی است که استفاده از آن‌ها در هر پروژه امکان پذیر است. "

برای شروع نیاز به آدرس Server و نام پایگاه داده داریم که می‌توانید در App.config به عنوان تنظیمات نرم افزار شما ذخیره شود و هنگام اجرای نرم افزار مقدار آن‌ها را خوانده و در متغییرهای readonly ذخیره شوند.

```
<appSettings>
    <add key="ServerName" value="http://SorousH-HP:8080/"/>
    <add key="DatabaseName" value="TestDatabase" />
</appSettings>
```

هنگامی که صفحه Management Studio در مرورگر باز است، می‌توانید از نوار آدرس مرورگر خود آدرس سرور را به دست آورید.

```
[TestClass]
    public class BeginnerTest
    {
        private readonly string serverName;
        private readonly string databaseName;

        public BeginnerTest()
        {
            serverName = ConfigurationManager.AppSettings["ServerName"];
            databaseName = ConfigurationManager.AppSettings["DatabaseName"];
        }
    }
```

برای برقراری ارتباط با پایگاه داده نیاز به یک شئ از جنس DocumentStore و جهت انجام عملیات مختلف (ذخیره، حذف و ...) نیاز به یک شئ از جنس IDocumentSession است. کد زیر، نحوه کار با آن‌ها را به شما نشان می‌دهد :

```
[TestClass]
    public class BeginnerTest
    {
        private readonly string serverName;
        private readonly string databaseName;

        private DocumentStore documentStore;
        private IDocumentSession session;

        public BeginnerTest()
        {
            serverName = ConfigurationManager.AppSettings["ServerName"];
            databaseName = ConfigurationManager.AppSettings["DatabaseName"];
        }

        [TestInitialize]
        public void TestStart()
        {
            documentStore = new DocumentStore { Url = serverName };
            documentStore.Initialize();
            session = documentStore.OpenSession(databaseName);

        }

        [TestCleanup]
        public void TestEnd()
        {
             session.SaveChanges();
             documentStore.Dispose();
             session.Dispose();
        }
    }
```

در طراحی این پایگاه داده از اگوی Unit Of Work استفاده شده است. به این معنی که تمام تغییرات در حافظه ذخیره می‌شوند و به محض اجرای دستور ;()session.SaveChanges ارتباط برقرار شده و تمام تغییرات ذخیره خواهند شد.

هنگام شروع (تابع : TestStart) متغییر session مقدار دهی می‌شود و در پایان کار (تابع : TestEnd) تغییرات ذخیره شده و منابعی که توسط این دو شئ در حافظه استفاده شده است، رها می‌شود.

البته بر مبنای طراحی شما، دستور ;()session.SaveChanges می‌تواند پس از انجام هر عملیات اجرا شود.

برای آشنا شدن با نحوه ذخیره کردن اطلاعات، به کد زیر دقت کنید:

```
class User
    {
        public int Id { get; set; }
        public string Name { get; set; }
        public string Address { get; set; }
        public int Zip { get; set; }اهی
    }
```

```
[TestMethod]
        public void Insert()
        {
            var user = new User
                            {
                                Id = 1,
                                Name = "John Doe",
                                Address = "no-address",
                                Zip = 65826
                            };
            session.Store(user);
        }
```

اگر همه چیز درست پیش رفته باشد، وقتی به محیط RavenDB Studio که هنوز در مرورگر شما باز است، نگاهی می‌اندازید، یک سند جدید ایجاد شده است که با کلیک روی آن، اطلاعات آن قابل مشاهده است.
لحظه‌ی لذت بخشی است...
یکی از روش‌های خواندن اطلاعات هم به صورت زیر است:

```
[TestMethod]
        public void Select()
        {
            var user = session.Load<User>(1);
        }
```

نتیجه خروجی این دستور هم یک شئ از جنس کلاس User است.

تا این جا، ساده‌ترین مثال‌های ممکن را مشاهده کردید و حتما در بحث بعد مثال‌های جالب‌تر و دقیق‌تری را بررسی می‌کنیم و همچنین نگاهی به جزئیات طراحی و قراردادهای از پیش تعیین شده می‌اندازیم.

" به شما پیشنهاد می‌کنم که منتظر بحث بعدی نباشید! همین حالا دست به کار شوید... "

نسخه بدون کتابخانه‌های موردنیاز (2 مگابایت) : RavenDBTest_Small.zip
نسخه کامل (15 مگابایت) : RavenDBTest.zip

نظرات خوانندگان

نویسنده: ناصر طاهری
تاریخ: ۱۳۹۱/۰۴/۱۶ ۱۷:۱۲

سلام.
مقوله‌ی جالبیه برای من.
منتظر ادامه هستم. موفق باشید.

نویسنده: حسین
تاریخ: ۱۳۹۱/۰۴/۱۶ ۱۹:۱۳

بسیار ممنون که کاربردی پیش میرین.من پروژمو با همین روش پیاده سازی می‌کنم و از اطلاعات خوبتون استفاده کردم.

نویسنده: ramin_rp
تاریخ: ۱۳۹۱/۰۴/۱۷ ۱۰:۴۳

سلام
وقتی ravendb رو استارت میکنم و محیط مدیریت اون تو browser اجرا میشه برای انجام عملیاتی مثل ایجاد دیتابیس user,pass میخواد که هرچی میدم قبول نمیکنه
حتی raven رو به عنوان service هم نصب کردم ولی مشکل حل نشد
جستجو تو نت هم نتیجه نداد
مشکل از چیه؟

(مستندات رسمی ravendb خوب نیست، مستندات mongodb واقعا کامل و جامع هست)

نویسنده: وحید نصیری
تاریخ: ۱۳۹۱/۰۴/۱۷ ۱۱:۳۷

توضیحات بیشتر در اینجا

```
By default RavenDB allow anonymous access only for read requests (HTTP GET),
and since we creating data, we need to specify a username and password.
You can control this by changing the AnonymousAccess setting in the server configuration file.
Enter your username and password of your Windows account and a sample data will be generated for you.
```

نویسنده: سروش ترک زاده
تاریخ: ۱۳۹۱/۰۴/۱۷ ۲۱:۱۸

سلام
ببخشید که دیر به سوال شما پاسخ دادم...
یه راه دیگه، علاوه بر راهی که توسط جناب آقای نصیری ارائه شده است، وجود دارد.
در پوشه Server فایل Raven.Server.exe را با Notepad باز کنید، سپس مقدار تنظیمات با کلید "Raven/AnonymousAccess" را به "All" تغییر دهید.توجه کنید که به بزرگ و کوچک بودن حروف حساس است.

در ضمن RavenDB از نظر سابقه و تعداد کاربران، قابل مقایسه با پایگاه داده هایی مثل SQL نیست و حق با شماست...

نویسنده: صابر

تاریخ: ۱۳۹۱/۰۴/۲۱ ۱۱:۱۹

سلام
نمی‌دانم مشکل از چیه ؟ ولی وقتی من سعی می‌کنم که بسته‌ی RavenDB رو از طریق Nuget دریافت کنم Error زیر رو میده .

```
Install-Package : The element 'metadata' in namespace
'http://schemas.microsoft.com/packaging/2010/07/nuspec.xsd' has invalid child element
'frameworkAssemblies' in namespace
'http://schemas.microsoft.com/packaging/2010/07/nuspec.xsd'. List of possible elements expected:
'summary' in namespace
'http://schemas.microsoft.com/packaging/2010/07/nuspec.xsd'.
At line:1 char:1
+ Install-Package RavenDB -Version 1.0.919
+ ~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
+ CategoryInfo : NotSpecified: (:) [Install-Package], InvalidOperationException
+ FullyQualifiedErrorId : NuGet.VisualStudio.Cmdlets.InstallPackageCmdlet
```

نویسنده: وحید نصیری
تاریخ: ۱۳۹۱/۰۴/۲۱ ۱۱:۵۸

به احتمال زیاد VS.NET شما دسترسی به اینترنت ندارد (^ و ^).

نویسنده: peyman
تاریخ: ۱۳۹۱/۰۵/۰۸ ۹:۱۸

آقا سروش کی شروع میکنی سری جدید رو منتظریم قربان !

نویسنده: سروش ترک زاده
تاریخ: ۱۳۹۱/۰۵/۰۸ ۱۸:۵۹

سلام
ببخشید که دیر شد، به احتمال زیاد پنجشنبه ادامه آموزش را روی سایت قرار خواهم داد...

نویسنده: پژمان
تاریخ: ۱۳۹۱/۰۶/۰۱ ۲۳:۲۵

ravendb هم مثل اینکه برای جستجو از لوسین استفاده می‌کنه.

نویسنده: فرزاد
تاریخ: ۱۳۹۱/۰۷/۱۰ ۲۳:۵۰

سلام
میخوام یه نرم افزار تحت ویندوز بنویسم که نیاز دارم از بانک اطلاعاتی استفاده کنم از طرفی هم نمی‌خوام با Sql server و یا access کار کنم چون نیاز به نزم افزار هایی با حجم زیاد هست که برای کاربر دردسر میشه.
میخواستم اگه میشه بفرمایید که به نظرتون از چی استفاده کنم بهتره؟
ممنون

نویسنده: حسین مرادی نیا
تاریخ: ۱۳۹۱/۰۷/۱۱ ۵:۴۸

نکته اینکه وقتی بانک اطلاعات Access رو استفاده کنین ، حتما نیازی نیست که Access روی کامپیوتر کاربر نصب باشه تا بتونه از برنامه شما استفاده کنه.
به هر حال میتونید از Sql Server CE استفاده کنید: http://www.dotnettips.info/search/label/SQL%20Server%20CE

عنوان: **آشنایی با مفهوم Indexer در C#.NET**
نویسنده: علیرضا اسم‌رام
تاریخ: ۱۳۹۱/۰۴/۲۰ ۱۹:۵۶
آدرس: www.dotnettips.info
گروه‌ها: #C

زمانی که صحبت از Indexer می‌شود، بطور ناخوداگاه ذهنمان به سمت آرایه‌ها می‌رود. آرایه‌ها در واقع ساده‌ترین اشیاء ی هستند که مفهوم Index در آنها معنا دار است.
اگر با آرایه‌ها کار کرده باشید با عملگر **[]** در سی شارپ آشنایی دارید. یک Indexer در واقع نوع خاصی از خاصیت (property) است که در بدنه کلاس تعریف می‌شود و به ما امکان استفاده از عملگر [] را برای نمونه کلاس فراهم می‌کند.
همانطور که به شباهت Indexer و Property اشاره شد نحوه تعریف Indexer بصورت زیر می‌باشد:

```
public type this [type identifier]
{
     get{ ... }
     set{ ... }
}
```

در پیاده سازی Indexer به نکات زیر دقت کنید:
از کلمه کلیدی **this** برای نعریف Indexer استفاده می‌شود و یک Indexer نام ندارد.

از دستیاب get برای برگرداندن مقدار و از دستیاب set جهت مقدار دهی و انتساب استفاده می‌شود.

الزامی به پیاده سازی Indexer با مقادیر صحیح عددی نیست و شما می‌توانید Indexer ی با پارامترهایی از نوع string یا double داشته باشید.

Indexerها قابلیت سربارگذاری (overloaded) دارند.

Indexerها می‌توانند چندین پارامتری باشند مانند آرایه‌های دو بعدی که دو پارامتری هستند.

به مثالی جهت پیاده سازی کلاس ماتریس توجه کنید:

```
public class Matrix
{
    // فیلدها
    private int _row, _col;
    private readonly double[,] _values;

    // تعداد ردیف‌های ماتریس
    public int Row
    {
        get { return _row; }
        set
        {
            _row = value > 0 ? value : 3;
        }
    }

    // تعداد ستون‌های ماتریس
    public int Col
    {
        get { return _col; }
        set
        {
            _col = value > 0 ? value : 3;
        }
    }

    // نعریف یک ایندکسر
    public double this[int r, int c]
```

```
    {
        get { return Math.Round(_values[r, c], 3); }
        set { _values[r, c] = value; }
    }

    // سازنده 1
    public Matrix()
    {
        _values = new double[_row,_col];
    }

    // سازنده 2
    public Matrix(int row, int col)
    {
        _row = row;
        _col = col;
        _values = new double[_row,_col];
    }

}
```

نحوه استفاده از کلاس ایجاد شده:

```
public class UseMatrixIndexer
{
    // ایجاد نمونه از شیء ماتریس
    private readonly Matrix m = new Matrix(5, 5);

    private double item;

    public UseMatrixIndexer()
    {
        // دسترسی به عنصر واقع در ردیف چهار و ستون سه
        item = m[4, 3];
    }
}
```

نظرات خوانندگان

نویسنده: وحید نصیری
تاریخ: ۱۳۹۱/۰۴/۲۰ ۲۱:۶

ممنون. Indexer مطلب جالبی است؛ ولی یک مورد کلا از زبان سی شارپ از قلم افتاده به نام indexed property
در حالت عادی می‌شود بر روی یک وهله از کلاس، Index اعمال کرد. اگر نخواهیم روی کل وهله اعمال شود چطور؟ برای مثال فقط روی یک خاصیت خاص.
پیاده سازی آن یک نکته کوچک دارد که به شرح زیر است:

```
using System;

namespace IndexedProperties
{
    public class Data
    {
        private int[] _localArray;
        private ArrayIndexer _arrayIndexer;

        public Data()
        {
            _localArray = new int[10];
            for (int i = 0; i < 10; i++)
                _localArray[i] = i + 1;
            _arrayIndexer = new ArrayIndexer(this);
        }

        public ArrayIndexer Number
        {
            get { return _arrayIndexer; }
        }

        public class ArrayIndexer
        {
            private Data _arrayOwner;

            public ArrayIndexer(Data arrayOwner)
            {
                _arrayOwner = arrayOwner;
            }

            public int this[int index]
            {
                get { return _arrayOwner._localArray[index]; }
            }

            public int Length
            {
                get { return _arrayOwner._localArray.Length; }
            }
        }
    }


    class Program
    {
        static void Main(string[] args)
        {
            var data = new Data();
            for (int i = 0; i < 10; i++)
                Console.WriteLine(data.Number[i]);
        }
    }
}
```

در اینجا از کلاس‌های تودرتو برای پیاده سازی Indexed property استفاده شده است.
به این ترتیب تعاریف آرایه مورد استفاده نیازی نیست مستقیما در معرض دید قرار گیرد و همچنین read-only هم تعریف شده است. به علاوه اینکه indexerها محدود به int نیستند و برای مثال می‌توان از string و غیره نیز استفاده کرد.

نویسنده: میثم هوشمند
تاریخ: ۱۳۹۱/۰۴/۲۱ ۰:۳۶

از هر دو نفر خیلی خیلی ممنونم
خب جناب نصیری خود این یک پست بود :) تشکر

نویسنده: Max
تاریخ: ۱۳۹۱/۰۴/۲۱ ۲۱:۲۴

سلام .
ممنون از این مطلب جالب و پر استفاده . اما من یه چیزی فهمیدم ، اونم اینکه چقدر چیز نمیدونم :)
در پناه خدا موفق و پیروز باشید

عنوان: **Debugging برنامه هایی که از خط فرمان پارامتر می گیرند!**
نویسنده: علیرضا اسم‌رام
تاریخ: ۱۳۹۱/۰۴/۲۹ ۱۸:۱۰
آدرس: www.dotnettips.info
برچسب‌ها: C#, Debugging, VisualStudio.NET

بسیاری از برنامه‌ها وجود دارند که در زمان فراخوانی از خط فرمان (Command Line) پارامترهایی دریافت می‌کنند و نسبت به آن پارامترها رفتار مشخصی را از خود نشان می‌دهند.
یکی از کاربردهای پارامتر ورودی **args** که از نوع آرایه ای از رشته‌ها در متد **Main** برنامه‌های **کنسول** بطور پیش فرض تعریف شده است همین موضوع است. شما می‌توانید از طریق کنترل مقدار این پارامتر برنامه‌ی خود را توسعه دهید.
برای مثال برنامه ای جهت چاپ مجذور اعدادی که از خط فرمان خوانده می‌شوند را در نظر بگیرید. کد مورد استفاده در این برنامه به شکل زیر خواهد بود:

```
static void Main(string[] args)
{
    foreach (var arg in args)
    {
        var x = Convert.ToInt32(arg);
        Console.WriteLine("{0} ^ 2 = {1}", x, x * x);
    }
    Console.ReadKey();
}
```

اما نکته اصلی این مطلب در مورد **Debugging** این گونه برنامه‌ها است.

جهت دیباگ این قبیل برنامه‌ها در ویژوال استادیو از قسمت **Solution Explorer** بر روی نام پروژه کلیک راست کرده و گزینه **Properties** را انتخاب کنید.
در پنجره باز شده به زبانه **Debug** بروید و در قسمت **Command Line Arguments** پارامترهای ورودی خود را بطوریکه با فضای خالی (Space) از هم جدا شده اند وارد کنید.

حال می‌توانید پس از ذخیره کردن تنظیمات، کلید F5 را بزنید.

نظرات خوانندگان

نویسنده: مسعود زیانی
تاریخ: ۱۳۹۱/۰۵/۲۱ ۱۴:۳۷

دوست عزیز ممنون بسیار کاربردی بود مطلبت.

عنوان: **مروری بر سازنده‌ها - سازنده‌های ایستا (static)**
نویسنده: مجتبی چنانی
تاریخ: ۱۳۹۱/۰۴/۳۱ ۱۳:۲۴
آدرس: www.dotnettips.info
برچسب‌ها: #C

مروری بر سازنده‌ها
سازنده‌های ایستا (static)

همانطور که می‌دانید در هنگام ساخت کلاس‌ها وجود یک سازنده الزامی و واجب است، حتی در زمانی که شما برای کلاستان سازنده‌ای تعریف نکنید یک سازنده پیشفرض برای کلاس، توسط دات نت و زبان صورت می‌گیرد.

اما چند نکته درباره سازنده‌های ایستا وجود دارد:

- سازنده ایستا هیچگونه صفتی (public, private, protected, internal,...) را نمی‌پذیرد.
- سازنده استاتیک نمی‌تواند پارامتر ورودی بپذیرد.
- سازنده استاتیک را نمی‌توان به صورت مستقیم فراخوانی کرد.
- برنامه نویس و کاربر هیچ کنترلی بر زمان فراخوانی این نوع سازنده ندارد.

کاربرد این نوع سازنده‌ها بیشتر در زمانهایی می‌باشد که برنامه نویس بخواهد قبل از استفاده کلاس یک سری پیش نیازها بررسی شود و یا یک سری تنظیمات ایجاد شود یا تخصیص حافظه‌ای صورت بگیرد. البته فقط یک بار نیاز است این ملزومات برطرف یا بررسی گردد. زیرا سازنده‌های استاتیک قبل از فراخوانی سازنده معمولی در هنگام ساخت اولین شیء از کلاس فراخوانی می‌شوند و دیگر فراخوانی نمی‌شوند

```
class test
    {
        static string fname, lname;
        static test()
        {
            console.write("first name:");
            fname = console.readline();
            console.write("last name:");
            lname = console.readline();
            console.writeline("thank you");
        }
        public test()
        {
            //some other code
        }
    }
```

در مثال بالا همانطور که مشاهده می‌شود (یک مثال ساده ساده) در قبل از استفاده از کلاس یکبار نام و نام خانوادگی از کاربر پرسیده می‌شود که می‌توان نمونه‌ای از این کد را به ورود کاربران تعمیم داد تا مطمئن شویم این کار پیش از هرچیزی صورت گرفته است.

نظرات خوانندگان

نویسنده: احمدعلی شفیعی
تاریخ: ۱۳۹۱/۰۴/۳۱ ۱۵:۴۵

آیا میشه در کلاس‌های ایستا از سازنده‌های ایستا استفاده کرد؟

نویسنده: مجتبی چنانی
تاریخ: ۱۳۹۱/۰۴/۳۱ ۱۶:۱

با سلام

بله در یک کلاس ایستا می‌توان سازنده‌ی ایستا داشت، البته این نکته را فراموش نکنید که در کلاس‌های ایستا تمامی متد ها، خصوصیات، اشیاء و... باید ایستا باشند لذا شما نمی‌توانید سازنده معمولی داشته باشید.

عنوان: **آشنایی با مفاهیم نوع داده Enum و توسعه آن - قسمت یکم**
نویسنده: علیرضا اسم‌رام
تاریخ: ۱۳۹۱/۰۵/۰۲ ۰:۸
آدرس: www.dotnettips.info
برچسب‌ها: C#, Enum

نوع داده شمارشی یا Enum، جهت تعاریف مقادیر ثابت و قابل شمارش در برنامه، بسیار کاربرد دارد. مقادیری که در این نوع داده تعریف می‌شوند بطور خودکار از عدد 0 شماره گذاری می‌شوند و به ترتیب یکی به آن‌ها اضافه می‌شود. برای مثال حالت زیر را در نظر بگیرید:

```
public enum Grade
    {
        Failing,        // = 0
        BelowAverage,   // = 1
        Average,        // = 2
        VeryGood,       // = 3
        Excellent       // = 4
    }
```

در این حالت متد ()ToString نوع داده Enum عنوان مقادیر ثابت را بر می‌گرداند.

جهت برگشت مقدار عددی و شماره مقادیر ثابت‌های تعریف شده از متد ()ToString با فرمت D (شماره مقدار را بصورت Decimal نشان می‌دهد) و فرمت X جهت نمایش بصورت هگزا می‌توان استفاده کرد.

روش عرف برای نمایش مقدار عددی استفاده از تبدیل نوع صریح به int است.

به منظور درک بهتر موضوع، از یک برنامه کنسول استفاده می‌کنیم تا این نوع داده شمارشی را در آن استفاده کنیم.

```
static void Main(string[] args)
{
    Grade grade = Grade.Average;
    Console.WriteLine(grade.ToString());    // Print Avarage
    Console.WriteLine(grade.ToString("D")); // Print 2
    Console.WriteLine(grade.ToString("X")); // Print 00000002
    Console.WriteLine((int) grade);         //Print 2
    Console.ReadKey();
}
```

تغییر شماره (اندیس) مقادیر ثابت تعریف شده:

جهت تغییر شماره مقادیر کافیست بصورت زیر عمل کنیم:

```
public enum Grade
{
    Failing = 5,
    BelowAverage = 10,
    Average = BelowAverage + 5, // = 15
    VeryGood = 18,
    Excellent = 20
}
```

همانطور که در بالا می‌بینید برای مقدار Average بصورت ترکیبی عمل شده است.
بصورت پیش فرض کامپایلر سی شارپ از Int32 جهت نگهداری اعضای یک Enum استفاده می‌کند. هر چند غیر معقول به نظر می‌رسد اما شما می‌توانید این نوع را به byte - sbyte - short - ushort - uint - long تغییر دهید.

```
public enum Grade : byte
{
    Failing = 5,
    BelowAverage = 10,
    Average = BelowAverage + 5, // = 15
    VeryGood = 18,
    Excellent = 20
```

```
}
```

بدیهی است در این حالت خروجی دستور زیر 0F خواهد بود:

```
Console.WriteLine(grade.ToString("X")); // Print 0F
```

همچنین به خروجی دستورات زیر در حالت فوق توجه کنید:

```
Console.WriteLine("Underlying type: {0}", Enum.GetUnderlyingType(grade.GetType())); // Print
System.Byte

Console.WriteLine("Type Code      : {0}", grade.GetTypeCode()); // Print Byte
```

و البته این:

```
Console.WriteLine("Value : {0}", (int)grade); // Print 15
```

در قسمت دوم این مطلب با استفاده از فضای نام System.Reflection و Extension Methodها و Custom Attribute کمی مقادیر Enum را توسعه خواهیم داد.

نظرات خوانندگان

نویسنده: احمد احمدی
تاریخ: ۱۳۹۱/۰۵/۰۲ ۲:۴۶

سلام - بحث Enum خیلی بحث جالبی است .
فقط فراموش کردید برچسب Enum را هم به این قسمت از مقالتون اضافه کنید .
با تشکر ...

نویسنده: علیرضا اسم‌رام
تاریخ: ۱۳۹۱/۰۵/۰۲ ۹:۳۲

سلام. با تشکر از شما. این موضوع را در قسمت سوم و باز هم با کمک متدهای الحاقی اجرا می‌کنیم. در نهایت امیدوارم یک کلاس از متدهای الحاقی جهت کار با Enumها داشته باشیم.
ممنون از یادآوری شما.

نویسنده: KishIsland
تاریخ: ۱۳۹۱/۱۲/۲۷ ۱۵:۲۰

سپاس.مفید بود

عنوان: **آشنایی با مفاهیم نوع داده Enum و توسعه آن - قسمت دوم**
نویسنده: علیرضا اسم‌رام
تاریخ: ۱۳۹۱/۰۵/۰۲ ۰:۳۲
آدرس: www.dotnettips.info
برچسب‌ها: C#, Extension Method, Enum

اگر با نوع داده Enum آشنایی ندارید قسمت یکم این مطلب را بخوانید.

```
public enum Grade
{
    Failing = 5,
    BelowAverage = 10,
    Average = BelowAverage + 5, // = 15
    VeryGood = 18,
    Excellent = 20
}
```

بازنویسی متد ()ToString: امکان بازنویسی متد ()ToString در نوع Enum وجود ندارد. بنابراین برای چاپ عبارت Very Good به جای VeryGood تکنیک زیر جالب به نظر می‌رسد. هر چند استفاده از آرایه و ترکیب اندیس آن با Enum و یا استفاده از HashTable راه هایی است که در ابتدا به ذهن ما خطور می‌کند اما لطفاً به ادامه مطلب توجه فرمایید!
با در نظر گرفتن مثال قبل، یک Custom Attribute به نوع داده شمارشی اضافه می‌کنیم. برای این منظور بصورت زیر عمل می‌کنیم.

1. ایجاد کلاس Description که از کلاس Attribute مشتق شده است و تعریف خصوصیت Text:

```
class Description : Attribute
{
    public string Text;
    public Description(string text)
    {
        Text = text;
    }
}
```

2. به سراغ نوع Enum تعریف شده رفته و جهت استفاده از صفت جدید که در مرحله قبل پیاده سازی کردیم، تغییرات را به شکل زیر اعمال می‌کنیم:

```
public enum Grade
{
    [Description("Mardood")]
    Failing = 5,

    [Description("Ajab Shansi")]
    BelowAverage = 10,

    [Description("Bad Nabood")]
    Average = BelowAverage + 5,

    [Description("Khoob Bood")]
    VeryGood = 18,

    [Description("Gol Kashti")]
    Excellent = 20
}
```

تنها کاری که باقی مانده یاری گرفتن از متدهای الحاقی (Extension Methods) جهت خواندن مقدار Description است:

```
public static class ExtensionMethodCls
{
    public static string GetDescription(this Enum enu)
    {

        Type type = enu.GetType();
```

```
        MemberInfo[] memInfo = type.GetMember(enu.ToString());

        if (memInfo != null && memInfo.Length > 0)
        {

            object[] attrs = memInfo[0].GetCustomAttributes(typeof(Description), false);

            if (attrs != null && attrs.Length > 0)
                return ((Description)attrs[0]).Text;
        }

        return enu.ToString();
    }
}
```

حال نوع Enum ما کمی توسعه یافته است و توسط متد GetDescription می توان متن دلخواه و متناسب با مقدار را نمایش داد:

```
Console.WriteLine(grade.GetDescription());  // Print Bad Nabood
```

کد کامل مثال بررسی شده نیز بصورت زیر خواهد بود:

```
using System;
using System.Reflection;

namespace CSharpEnum
{

    class Description : Attribute
    {
        public string Text;
        public Description(string text)
        {
            Text = text;
        }
    }

    public enum Grade
    {
        [Description("Mardood")]
        Failing = 5,

        [Description("Ajab Shansi")]
        BelowAverage = 10,

        [Description("Bad Nabood")]
        Average = BelowAverage + 5,

        [Description("Khoob Bood")]
        VeryGood = 18,

        [Description("Gol Kashti")]
        Excellent = 20
    }

    public static class ExtensionMethodCls
    {
        public static string GetDescription(this Enum enu)
        {

            Type type = enu.GetType();

            MemberInfo[] memInfo = type.GetMember(enu.ToString());

            if (memInfo != null && memInfo.Length > 0)
            {

                object[] attrs = memInfo[0].GetCustomAttributes(typeof(Description), false);

                if (attrs != null && attrs.Length > 0)
                    return ((Description)attrs[0]).Text;
            }

            return enu.ToString();
        }
```

```
    }

    class Program
    {
        static void Main(string[] args)
        {
            const Grade grade = Grade.Average;
            Console.WriteLine("Underlying type: {0}", Enum.GetUnderlyingType(grade.GetType()));
            Console.WriteLine("Type Code      : {0}", grade.GetTypeCode());
            Console.WriteLine("Value          : {0}", (int)grade);
            Console.WriteLine("-------------------------------------");
            Console.WriteLine(grade.ToString()); // name of the constant
            Console.WriteLine(grade.ToString("G")); // name of the constant
            Console.WriteLine(grade.ToString("F")); // name of the constant
            Console.WriteLine(grade.ToString("x")); // value is hex
            Console.WriteLine(grade.ToString("D")); // value in decimal
            Console.WriteLine("-------------------------------------");
            Console.WriteLine(grade.GetDescription());  // Print Bad Nabood
            Console.ReadKey();
        }
    }
}
```

با استفاده از این تکنیک (مخصوصاً ما فارسی زبان ها) به راحتی می‌توانیم از مقادیر Enum استفاده بهتری ببریم. برای مثال اگر بخواهیم یک مقدار Enum را بصورت فارسی در یک Drop Down List نمایش دهیم این تکنیک بسیار مفید خواهد بود.

نظرات خوانندگان

نویسنده: حسین
تاریخ: ۱۳۹۱/۰۵/۰۲ ۱۲:۱۹

سلام
کدهایی که به این صورت نوشته میشن چی بهشون گفته میشه؟ چی هستن؟
خیلی جاها این کدها رو دیدم به صورت‌های مختلف
کمی تو گوگل سرچ کردم نتیجه مطلوبی نگرفتم یعنی نمیدونم دنبال چی بگردم
لطفا راهنمایی کنید.

نویسنده: وحید نصیری
تاریخ: ۱۳۹۱/۰۵/۰۲ ۱۲:۲۳

به Attribute ایی که در اینجا توسعه داده شده (یا از آن استفاده شده)، اصطلاحا data annotation هم گفته می‌شود. یک سری از فریم ورک‌ها به صورت توکار قادر به استفاده از آن‌ها هستند مانند ASP.NET MVC برای نمایش توضیحات مرتبط یا نمایش برچسب‌ها به صورت خودکار.
مطالب فوق رو می‌تونید پایه طراحی این نوع کتابخانه‌ها درنظر بگیرید.

نویسنده: حسین
تاریخ: ۱۳۹۱/۰۵/۰۲ ۱۲:۳۰

کدهایی که تو linq هم استفاده میشن از همین دسته اند؟
مثلا

```
[global::System.Data.Linq.Mapping.ColumnAttribute(Storage="_ID", AutoSync=AutoSync.Always, DbType="Int
NOT NULL IDENTITY", IsDbGenerated=true)]
```

نویسنده: وحید نصیری
تاریخ: ۱۳۹۱/۰۵/۰۲ ۱۲:۴۶

بله. به این‌ها Attribute، meta-data و یا data annotation گفته می‌شود.

نویسنده: ایمان محمدی
تاریخ: ۱۳۹۱/۰۵/۰۶ ۱۴:۹

نیازی به تعریف کلاس Description نیست.
کافیه از فضای نام System.ComponentModel استفاده کنید و در مقدار بازگشتی متد GetDescription بجای

```
return ((Description)attrs[0]).Text;
```

بنوسید

```
return ((DescriptionAttribute)attrs[0]).Description;
```

نویسنده: وحید نصیری
تاریخ: ۱۳۹۱/۰۵/۰۶ ۱۴:۲۰

هدف این مطلب بررسی زیر ساخت این نوع طراحی بوده نه صرفا مصرف کننده محض بودن. تشابه اسمی اهمیتی نداره. روش انجام کار مهم بوده در اینجا.

نویسنده: ایمان محمدی
تاریخ: ۱۳۹۱/۰۵/۰۶ ۱۶:۵

وقتی کلاس Description در فضای نام System.ComponentModel وجود داره دلیلی نداره کلاس مشابه ای تعریف کنیم.
بخاطر اینکه مصرف کننده محض نباشیم یک متد الحاقی به نام GetEnumList() اضافه کردم که لیست اعضای یک Enum رو برای استفاده در کمبو باکس و ... بر می‌گردونه :
ابتدا کلاس زیر به کلاس ExtensionMethodCls اضافه می‌کنیم :

```
public class EnumObject
        {
            public Enum ValueMember { get; set; }
            public int intValueMember
            {
                get { return int.Parse(ValueMember.ToString("D")); }
            }
            public string stringValueMember
            {
                get { return ValueMember.ToString(""); }
            }
            public string DisplayMember
            {
                get { return ValueMember.GetDescription(); }
            }
        }
```

و متد الحاقی زیر رو برای گرفتن لیست تعریف می‌کنیم:

```
public static List<EnumObject> GetEnumList(this Enum enu)
        {
            List<EnumObject> li = new List<EnumObject>();
            foreach (var item in enu.GetType().GetEnumValues())
            {
                li.Add(new EnumObject { ValueMember = (Enum)item });
            }
            return li;
        }
```

نحوه استفاده :

```
comboBox1.DataSource = Grade.VeryGood.GetEnumList();
          comboBox1.DisplayMember = "DisplayMember";
          comboBox1.ValueMember = "ValueMember";
```

همون طوری که در کد بالا می‌بینید برای گرفتن لیست مجبور شدیم یکی از اعضای enum رو انتخاب کنیم (Grade. **VeryGood** .GetEnumList()) شاید انتخاب یکی از اعضا و بعد درخواست لیست اعضا رو کردن کار قشنگی نباشه به همین دلیل متد زیر رو تعریف کردیم :

```
public static List<EnumObject> EnumToList<T>()
        {
            Type enumType = typeof(T);
            if (enumType.BaseType != typeof(Enum))
                throw new ArgumentException("T must be of type System.Enum");

            List<EnumObject> li = new List<EnumObject>();
            foreach (var item in enumType.GetEnumValues())
            {
                li.Add(new EnumObject { ValueMember = (Enum)item });
            }
```

```
            return li;
        }
```

نحوه استفاده :

```
comboBox1.DataSource =ExtensionMethodCls.EnumToList<Grade>();
          comboBox1.DisplayMember = "DisplayMember";
          comboBox1.ValueMember = "ValueMember";
```

کد کامل :

```
public static class ExtensionMethodCls
    {
        public class EnumObject
        {
            public Enum ValueMember { get; set; }
            public int intValueMember
            {
                get { return int.Parse(ValueMember.ToString("D")); }
            }
            public string stringValueMember
            {
                get { return ValueMember.ToString(""); }
            }
            public string DisplayMember
            {
                get { return ValueMember.GetDescription(); }
            }
        }



        public static List<EnumObject> EnumToList<T>()
        {
            Type enumType = typeof(T);
            if (enumType.BaseType != typeof(Enum))
                throw new ArgumentException("T must be of type System.Enum");

            List<EnumObject> li = new List<EnumObject>();
            foreach (var item in enumType.GetEnumValues())
            {
                li.Add(new EnumObject { ValueMember = (Enum)item });
            }
            return li;
        }

        public static List<EnumObject> GetEnumList(this Enum enu)
        {
            List<EnumObject> li = new List<EnumObject>();
            foreach (var item in enu.GetType().GetEnumValues())
            {
                li.Add(new EnumObject { ValueMember = (Enum)item });
            }
            return li;
        }

        public static string GetDescription(this Enum enu)
        {

            Type type = enu.GetType();

            MemberInfo[] memInfo = type.GetMember(enu.ToString());

            if (memInfo != null && memInfo.Length > 0)
            {

                object[] attrs = memInfo[0].GetCustomAttributes(typeof(DescriptionAttribute), false);

                if (attrs != null && attrs.Length > 0)
                    return ((DescriptionAttribute)attrs[0]).Description;
            }

            return enu.ToString();

        }
```

```
    }
```

نویسنده: علیرضا اسم‌رام
تاریخ: ۱۳۹۱/۰۵/۰۷ ۹:۴۵

با سلام خدمت شما دوست عزیز. همانطور که آقای نصیری اشاره کردند هدف این مطلب و البته این سری از مطالب آشنایی قدم به قدم با مفاهیم این نوع داده بوده است.
انقیاد کنترل‌ها با اشیاء از نوع Enum موضوع بعدی این سری از مطالب بود. البته باز هم از شما تشکر می‌کنم بخاطر این نظر.
اگر این قابلیت در سایت ایجاد شود که بتوان به نظرات لینک داد بهتر است.

نویسنده: وحید نصیری
تاریخ: ۱۳۹۱/۰۵/۰۷ ۱۰:۰

علامت آبی رنگ # کنار هر نظر، لینک مستقیم به همان نظر است.

نویسنده: علیرضا اسم‌رام
تاریخ: ۱۳۹۱/۰۵/۰۷ ۲۲:۸

مرسی. خیلی خیلی خوب...

نویسنده: سعید یزدانی
تاریخ: ۱۳۹۱/۱۱/۱۷ ۲۰:۵۴

باسلام؛ لطفا یه توضیحی در باره‌ی MemberInfo بدید.

نویسنده: وحید نصیری
تاریخ: ۱۳۹۱/۱۱/۱۸ ۹:۱۱

مراجعه کنید به MSDN و مثال‌های آن در همان صفحه.

عنوان: Google OpenID Authentication در ASP.NET با استفاده از DotNetOpenAuth
نویسنده: صابر فتح الهی
تاریخ: ۱۳۹۱/۰۵/۰۴ ۵:۴۷
آدرس: www.dotnettips.info
گروه‌ها: ASP.Net, C#, Security, OpenID, DotNetOpenAuth

در پست‌های قبلی OpenID چیست؟
استفاده از OpenID در وب سایت جهت احراز هویت کاربران با مفاهیم OpenID تاحدودی آشنا شدیم
در این پست می‌خواهیم با یک مثال ساده نشان دهیم که سایت ما چگونه با استفاده از OpenID Authentication می تواند از اکانت گوگل استفاده کرده و کاربر در وب سایت ما شناسایی شود.
برای اینکار ابتدا
1- کتابخانه DotNetOpenAuth را از طریق NuGet به لیست رفرنس‌های پروژه وب خود اضافه نمایید
2- یک صفحه بنام Login.aspx یا هر نام دلخواهی را به پروژه خود اضافه نمایید. در نهایت کد Html صفحه شما به ید به صورت زیر باشد.

```
<%@ Page Language="C#" AutoEventWireup="true" CodeBehind="Login.aspx.cs" Inherits="OAuthWebTest.Login"
%>

<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Transitional//EN" "http://www.w3.org/TR/xhtml1/DTD/xhtml1-
transitional.dtd">
<html xmlns="http://www.w3.org/1999/xhtml">
<head runat="server">
    <title></title>
</head>
<body>
    <form id="form1" runat="server">
    <div id="loginform">
        <div id="NotLoggedIn" runat="server">
            <asp:ImageButton ID="ButtonLoginWithGoogle" runat="server" ImageUrl="Google.JPG"
                OnCommand="ButtonLoginWithGoogle_Click"
CommandArgument="https://www.google.com/accounts/o8/id" />
            <p />
            <asp:Label runat="server" ID="LabelMessage" />
        </div>
    </div>
    </form>
</body>
</html>
```

در کد Html بالا به خصوصیت CommandArgument از کنترل ImageButton دقت نمایید که دارای مقدار "https://www.google.com/accounts/o8/id" می باشد. در واقع این آدرس باید برای اکانت گوگل جهت احراز هویت توسط OpenID استفاده شود. (آدرس API گوگل برای استفاده از این سرویس)
3- در قسمت کد نویسی صفحه کدهای زیر را وارد نمایید.

```
using System;
using System.Web.UI;
using System.Web.UI.WebControls;
using DotNetOpenAuth.OpenId.RelyingParty;

namespace OAuthWebTest
{
    public partial class Login : Page
    {
        protected void Page_Load(object sender, EventArgs e)
        {
            OpenIdRelyingParty openIdRelyingParty = new OpenIdRelyingParty();
            var response = openIdRelyingParty.GetResponse();
            if (response == null) return;
            switch (response.Status)
            {
                case AuthenticationStatus.Authenticated:
                    NotLoggedIn.Visible = false;
                    Session["GoogleIdentifier"] = response.ClaimedIdentifier.ToString();
                    Response.Redirect("Default.aspx");
                    break;
                case AuthenticationStatus.Canceled:
                    LabelMessage.Text = "Cancelled.";
                    break;
```

```
                case AuthenticationStatus.Failed:
                    LabelMessage.Text = "Login Failed.";
                    break;
            }
        }

        protected void ButtonLoginWithGoogle_Click(object src, CommandEventArgs e)
        {
            string discoveryUri = e.CommandArgument.ToString();
            OpenIdRelyingParty openId = new OpenIdRelyingParty();
            var returnToUrl = new UriBuilder(Request.Url) { Query = "" };
            var request = openId.CreateRequest(discoveryUri, returnToUrl.Uri, returnToUrl.Uri);
            request.RedirectToProvider();
        }
    }
}
```

همانگونه که مشاهده می‌کنید در رویداد کلیک دکمه لوگین ابتدا آدرس یکتا گوگل (مقدار ImageButton اختصاص داده شده به کنترل CommandArgument) به همراه آدرس صفحه جاری وب سایت توسط متد CreateRequest از شی openId ترکیب شده و یک درخواست (Request) ساخته شده و در نهایت برای سرویس دهنده گوگل ارسال می‌شود. با اجرای پروژه با تصویر زیر روبرو می‌شوید بروی کلید لوگین کلیک نمایید.

در واقع با کلیک روی دکمه مورد نظر تکه کدی که در بالا شرح داده شد اجرا شده و ما را به صفحه ای شبیه تصویر پایین هدایت می‌کند

در صورتی که کلید Sign In انتخاب شود شما به سایت (همین برنامه) اجازه داده اید که از اطلاعات حساب کاربری شما استفاده کند. پس از کلیک به برنامه اصلی (طبق آدرس بازگشت تعیین شده در رویداد ButtonLoginWithGoogle_Click) در رویداد PageLoad صفحه لاگین اطلاعات بازگشتی از سایت سرویس دهنده (در اینجا گوگل) چک می‌شود و با توجه با پاسخ مورد نظر عملیاتی انجام می‌شود، مثلا در صورتی که شما تایید کرده باشید اطلاعات شناسایی شما توسط گوگل در کلیدی به نام GoogleIdentifier در سشن ذهیره شده و شما به صفحه اصلی سایت هدایت می‌شوید.

دموی پروژه در این آدرس قرار داده شده است.

سورس پروژه از این آدرس قابل دسترسی است.

توجه: در این آدرس شرح داده شده که چگونه دسترسی سایت‌های دیگر به اکانت گوگل خود را قطع کنید

در پست‌های آینده اتصال به تویتر و ف ی س بوک و سایت‌های دیگر توضیح داده خواهد شد.

نظرات خوانندگان

نویسنده: مجتبی چنانی
تاریخ: ۱۳۹۱/۰۵/۰۴ ۹:۰

با سلام و تشکر از پست خوبتون

در زمانی که ما از سیستم‌های ورود و ثبت کاربران شرکت‌های دیگر استفاده می‌کنیم آیا می‌توانیم لاگ گرفته یا اینکه برای خودمان یک صفحه داشته باشیم تا ورود و خروج‌های اکانت‌های درون وبسایتمان را بررسی کنیم یا اینکه خودمان باید این بخش را کدنویسی کنیم؟

یک سئوال دیگر این است که زمانی که از openid های شرکت‌های دیگه استفاده می‌کنیم فقط احراز هویت را از این سرویس‌ها دریافت می‌کنیم یا اینکه در همه صفحات و دیگر کارهای کاربر نظارت به صورت خودکار انجام می‌شود یا اینکه باز هم باید کدنویسی کنیم؟

نویسنده: صابر فتح اللهی
تاریخ: ۱۳۹۱/۰۵/۰۴ ۱۴:۹

در صورتی که بخواهید لاگ بندازید باید خود کد نویسی کرده و در دیتابیس ذخیره کنید
بله می‌توان در هر صفحه با استفاده از کلیدی که در سشن کاربر ذخیره کرده اید (با توجه به اینکه سشن کاربران جداست) می‌توانید وجود کاربر را چک کنید

نویسنده: سینا علیزاده
تاریخ: ۱۳۹۲/۰۹/۰۳ ۱۳:۲۹

آقا عالی بود ممنون
میشه برای facebook و توییتر هم بزارید ، واقعا بهش نیاز دارم ممنون میشم
کارتون عالیه موفق باشید

نویسنده: محسن خان
تاریخ: ۱۳۹۲/۰۹/۰۳ ۱۳:۳۵

نگاهی هم به پروژه DotNetAuth داشته باشید.

نویسنده: دادخواه
تاریخ: ۱۳۹۲/۱۰/۱۶ ۱۱:۳۱

سلام.

این سرویس چطوری برای اکانت‌های یاهو کار می‌کنه؟

عنوان: **استفاده از "," و ";" اضافه در C#**
نویسنده: بهروز راد
تاریخ: ۱۳۹۱/۰۵/۰۹ ۹:۱
آدرس: www.dotnettips.info
برچسب‌ها: C#, Tips, Tricks

در C# می‌تونید در انتهای تعریف آخرین آیتم یک Enum یا هنگام استفاده از سینتکس Object Initializer یا Collection Initializer، یک کامای اضافی قرار بدید.

اون طور که گفته شده ، این رفتار بدین دلیل است که Code Generatorها راحت‌تر بتوانند کد تولید کنند. مطمئناً اگر در یک حلقه‌ی تکرار برای ایجاد آیتم های یک Enum، در انتهای آیتم‌های اون، کاراکتر "," قرار می‌دید، حذف نکردن آخرین کاما از حذف کردن اون کار راحت‌تری است! همچنین Comment کردن آخرین آیتم نیز راحت‌تر صورت می‌پذیرد.

```
public enum MyEnum
{
    Item1 = 1,
    Item2 = 2,
    Item3 = 4,
    // Item4
}

MyViewModel viewModel = new MyViewModel()
{
    Property1 = "Value1",
    Property2 = "Value2",
    Property3 = "Value3",
};
```

و البته، در هنگام فراخوانی یک متد میشه به تعداد دلخواه در انتهای اون، کاراکتر ";" قرار داد.

```
myBusiness business = new myBusiness();
business.DoWork(); ; ; ; ; ; ;
```

نظرات خوانندگان

نویسنده: پژمان
تاریخ: ۱۳۹۱/۰۵/۱۲ ۱۸:۲۲

این رو میشه اسمش رو گذاشت forgiveness in language design

عنوان: **اشیاء Enumerable و Enumerator و استفاده از قابلیت‌های yield (قسمت اول)**
نویسنده: ابراهیم بیاگوی
تاریخ: ۱۳۹۱/۰۵/۱۷ ۱۸:۲۵
آدرس: www.dotnettips.info
برچسب‌ها: C#, LINQ, Enumerator, Enumerable

در این مقاله می‌خواهیم نحوهٔ ساخت اشیایی با خصوصیات Enumerable را بررسی کنیم. بررسی ویژگی این اشیاء دارای اهمیت است حداقل به این دلیل که پایهٔ یکی از قابلیت مهم زبانی سی‌شارپ یعنی LINQ هستند. برای یافتن پیش‌زمینه‌ای در این موضوع خواندن این مقاله‌های بسیار خوب (۱ و ۲) نیز توصیه می‌شود.

Enumerableها

اشیاء Enumerable یا به‌عبارت دیگر اشیائی که اینترفیس IEnumerable را پیاده‌سازی می‌کنند، دامنهٔ گسترده‌ای از Collectionهای CLI را شامل می‌شوند. همانطور که در نمودار زیر نیز می‌توانید مشاهده کنید IEnumerable (از نوع غیر Generic آن) در بالای سلسله مراتب اینترفیس‌های Collectionهای CLI قرار دارد:

IEnumerable
Interface
Methods
GetEnumerator
ICollection
Interface
IEnumerable
Properties
Count
IsSynchronized
SyncRoot
Methods
CopyTo
IEnumerable<T>
Generic Interface
IEnumerable
Methods
GetEnumerator
ICollection<T>
Generic Interface
IEnumerable<T>
IEnumerable
Properties
Count
IsReadOnly
Methods
Add
Clear
Contains
CopyTo
Remove
IDictionary
Interface
ICollection
IEnumerable
Properties
IsFixedSize
IsReadOnly
Keys
this
Values
Methods
Add
Clear
Contains
GetEnumerator
Remove
IList
Interface
ICollection
IEnumerable
Properties
IsFixedSize
IsReadOnly
this
Methods
Add
Clear
Contains
IndexOf
Insert
Remove
RemoveAt
IList<T>
Generic Interface
ICollection<T>
IEnumerable<T>
IEnumerable
Properties
this
Methods
IndexOf
Insert
RemoveAt
IDictionary<TKe...
Generic Interface
ICollection<KeyValue...
IEnumerable<KeyValu...
IEnumerable
Properties
Keys
this
Values
Methods
Add
ContainsKey
Remove
TryGetValue

درخت اینترفیس‌های Collectionها در CLI منبع

IEnumerableها همچنین دارای اهمیت دیگری نیز هستند؛ قابلیت‌های LINQ که از دات‌نت ۳.۵ به دات‌نت اضافه شدند به‌عنوان Extensionهای این اینترفیس تعریف شده‌اند و پیاده‌سازی Linq to Objects را می‌توانید در کلاس استاتیک System.Linq.Enumerable در System.Core مشاهده کنید. (می‌توانید برای دیدن آن را با ILDasm یا Reflector باز کنید یا پیاده‌سازی آزاد آن در پروژهٔ Mono را اینجا مشاهده کنید که برای شناخت بیشتر LINQ واقعاً مفید است.)

همچنین این Enumerableها هستند که foreach را امکان‌پذیر می‌کنند. به عبارتی دیگر هر شئ‌ای که قرار باشد در foreach (var x in object) قرار بگیرد و بدین طریق اشیاء درونی‌اش را برای پیمایش یا عملی خاص قرار دهد باید Enumerable باشد.

همانطور که قبلاً هم اشاره شد IEnumerable از نوع غیر Generic در بالای نمودار Collectionها قرار دارد و حتی IEnumerable از نوع Generic نیز باید آن را پشتیبانی کند. این موضوع به احتمال به این دلیل در طراحی لحاظ شد که مهاجرت به .NET 2.0 که قابلیت‌های Generic را افزوده بود ساده‌تر کند. IEnumerable همچنین قابلیت covariance که از قابلیت‌های جدید C# 4.0 هست را دارا است (در اصل IEnumerable دارای Generic از نوع out است).

Enumerableها همانطور که از اسم اینترفیس IEnumerable انتظار می‌رود اشیایی هستند که می‌توانند یک شئ Enumerator که IEnumerator را پیاده‌سازی کرده‌است را از خود ارائه دهند. پس طبیعی است برای فهم و درک دلیل وجودی Enumerable باید Enumerator را بررسی کنیم.

Enumeratorها

Enumerator شئ است که در یک پیمایش یا به‌عبارت دیگر گذر از روی تک‌تک عضوها ایجاد می‌شود که با حفظ موقعیت فعلی و پیمایش امکان ادامهٔ پیمایش را برای ما فراهم می‌آورد. اگر بخواهید آن را در حقیقت بازسازی کنید شئ Enumerator به‌مانند کاغذ یا جسمی است که بین صفحات یک کتاب قرار می‌دهید که مکانی که در آن قرار دارید را گم نکنید؛ در این مثال، Enumerable همان کتاب است که قابلیت این را دارد که برای پیمایش به وسیلهٔ قرار دادن یک جسم در وسط آن را دارد.

حال برای اینکه دید بهتری از رابطهٔ بین Enumerable و Enumerator از نظر برنامه‌نویسی به این موضوع پیدا کنیم یک کد نمونهٔ عملی را بررسی می‌کنیم.

در اینجا نمونهٔ ساده و خوانایی از استفاده از یک List برای پیشمایش تمامی اعداد قرار دارد:

```
List<int> list = new List<int>();
list.Add(1);
list.Add(2);
list.Add(3);
foreach (int i in list)
{
    Console.WriteLine(i);
}
```

همانطور که قبلاً اشاره foreach نیاز به یک Enumerable دارد و List هم با پیاده‌سازی IList که گسترشی از IEnumerable هست نیز یک نوع Enumerable هست. اگر این کد را Compile کنیم و IL آن را بررسی کنیم متوجه می‌شویم که CLI در اصل چنین کدی را برای اجرا می‌بینید:

```
List<int> list = new List<int>();
list.Add(1);
list.Add(2);
list.Add(3);
IEnumerator<int> listIterator = list.GetEnumerator();
while (listIterator.MoveNext())
{
    Console.WriteLine(listIterator.Current);
```

```
}
listIterator.Dispose();
```

(می‌توان از using استفاده نمود که Dispose را خود انجام دهد که اینجا برای سادگی استفاده نشده‌است.)

همانطور که می‌بینیم یک Enumerator برای Enumerable ما (یعنی List) ایجاد شد و پس از آن با پرسش این موضوع که آیا این پیمایش امکان ادامه دارد، کل اعضا پیموده‌شده و عمل مورد نظر ما بر آن‌ها انجام شده‌است.

خب، تا اینجای کار با خصوصیات و اهمیت Enumeratorها و Enumerableها آشنا شدیم، حال نوبت به آن می‌رسد که بررسی کنیم آن‌ها را چگونه می‌سازند و بعد از آن با کاربردهای فراتری از آن‌ها نسبت به پیمایش یک List آشنا شویم.

ساخت Enumeratorها و Enumerableها

همانطور که اشاره شد ایجاد اشیاء Enumerable به اشیاء Enumerator مربوط است، پس ما در یک قطعه کد که پیمایش از روی یک آرایه را فراهم می‌آورد ایجاد هر دوی آن‌ها و رابطهٔ بینشان را بررسی می‌کنیم.

```
public class ArrayEnumerable<T> : IEnumerable<T>
    {
        private T[] _array;
        public ArrayEnumerable(T[] array)
        {
            _array = array;
        }


        public IEnumerator<T> GetEnumerator()
        {
            return new ArrayEnumerator<T>(_array);
        }

        System.Collections.IEnumerator System.Collections.IEnumerable.GetEnumerator()
        {
            return GetEnumerator();
        }
    }

    public class ArrayEnumerator<T> : IEnumerator<T>
    {
        private T[] _array;
        public ArrayEnumerator(T[] array)
        {
            _array = array;
        }

        public int index = -1;

        public T Current { get { return _array[index]; } }

        object System.Collections.IEnumerator.Current { get { return this.Current; } }

        public bool MoveNext()
        {
            index++;
            return index < _array.Length;
        }

        public void Reset()
        {
            index = 0;
        }

        public void Dispose() { }
    }
```

ادامه

نظرات خوانندگان

نویسنده: مرتضی
تاریخ: ۱۳۹۱/۰۵/۱۷ ۱۹:۲۰

درخت اینترفیس‌های Collectionها در سی‌شارپ منبع: http://www.mbaldinger.com/post/NET-Collection-Interface-Hierarchy.aspx
بجای سی‌شارپ به دانت‌نت تغییرش بدید
درخت اینترفیس‌های Collectionها در دانت‌نت

نویسنده: ابراهیم بیاگوی
تاریخ: ۱۳۹۱/۰۵/۱۷ ۱۹:۲۹

من شخصاً اطمینان ندارم که همهٔ زبان‌های CLI از همین Collectionها استفاده کنند و البته این نمودار با Syntax سی‌شارپ بود به همین دلیل سی‌شارپ نوشته بودم با این حال آن را به Collectionهای CLI تبدیل کردم.

عنوان: **اشیاء Enumerable و Enumerator و استفاده از قابلیت‌های yield (قسمت دوم)**
نویسنده: ابراهیم بیاگوی
تاریخ: ۱۳۹۱/۰۵/۱۸ ۱۱:۴۴
آدرس: www.dotnettips.info
برچسب‌ها: C#, Enumerator, Enumerable

در مطلب قبل متوجه شدیم که Enumerable و Enumerator چه چیزی هستند و آن‌ها را چگونه می‌سازند. در انتهای آن مطلب نیز قطعه کدی وجود داشت که در آن دیدیم چگونه یک شئ Enumerable می‌تواند در عملیاتی نسبتاً پیچیده یک شئ Enumerator ایجاد کند.

حال می‌خواهیم قابلیت زبانی‌ای را بررسی کنیم که در اصل مشابه همین کاری که ما انجام دادیم یعنی ایجاد شئ جداگانهٔ Enumerator و برگرداندن یک نمونه از آن در زمانی که ما GetEnumerator را از Enumerableمان فراخوانی می‌کنیم را انجام می‌دهد.

yield و نحوهٔ پیاده‌سازی آن

در اینجا قطعه کدی قرار دارد که در اصل جایگزین دو کلاسی‌است که در انتهای مطلب قبل قرار داشت که به کمک قابلیت yield آن را بازنویسی کرده‌ایم:

```
public class ArrayEnumerable<T> : IEnumerable<T>
    {
        T[] _array;
        public ArrayEnumerable(T[] array)
        {
            _array = array;
        }

        public IEnumerator<T> GetEnumerator()
        {
            int index = 0;
            while (index < _array.Length)
            {
                yield return _array[index];
                index++;
            }
            yield break;
        }

        System.Collections.IEnumerator System.Collections.IEnumerable.GetEnumerator()
        {
            return GetEnumerator();
        }
    }
```

(yield break در اینجا مانند return در یک تابع/متد با نوع خروجی void اضافی‌است و فقط برای آشنایی با syntax دومی که yield در سی‌شارپ پشتیبانی می‌کند قرار داده شده‌است)

همانطور که می‌بینیم کد قبلی ما به مقدار بسیاری ساده‌تر و خواناتر شد و برای فهم آن کافی است که مفهوم yield را بدانیم.

yield به معنای برآوردن یا ارائه‌کردن کلید واژه‌ای است که می‌توان آن را اینگونه تصور کرد که با هر با صدا زده‌شدن کد را متوقف می‌کند و نتیجه‌ای را برمی‌گرداند و با درخواست ما برای ادامهٔ کار (با MoveNext) کار خود را از همان جای متوقف شده ادامه می‌دهد.

حالا اگر کمی دقیقتر باشیم سوالی که باید برای ما پیش بیاید این است که آیا CLR خود yield را پشیبانی می‌کند؟
این قطعه کدی است که با کمک بازگردانی مجدد همین کلاس به زبان سی‌شارپ دیده می‌شود:

```
public class ArrayEnumerable<T> : IEnumerable<T>, IEnumerable
{
    // Fields
    private T[] _array;
```

```
    // Methods
    public ArrayEnumerable(T[] array)
    {
        this._array = array;
    }

    public IEnumerator<T> GetEnumerator()
    {
        return new <GetEnumerator>d__0(0);
    }

    IEnumerator IEnumerable.GetEnumerator()
    {
        return this.GetEnumerator();
    }

    // Nested Types
    [CompilerGenerated]
    private sealed class <GetEnumerator>d__0 : IEnumerator<T>, IEnumerator, IDisposable
    {
        // Fields
        private int <>1__state;
        private T <>2__current;
        public ArrayEnumerable<T> <>4__this;
        public int <index>5__1;

        // Methods
        [DebuggerHidden]
        public <GetEnumerator>d__0(int <>1__state)
        {
            this.<>1__state = <>1__state;
        }

        private bool MoveNext()
        {
            switch (this.<>1__state)
            {
                case 0:
                    this.<>1__state = -1;
                    this.<index>5__1 = 0;
                    while (this.<index>5__1 < ArrayEnumerable<T>._array.Length)
                    {
                        this.<>2__current = ArrayEnumerable<T>._array[this.<index>5__1];
                        this.<>1__state = 1;
                        return true;
                    Label_0050:
                        this.<>1__state = -1;
                        this.<index>5__1++;
                    }
                    break;

                case 1:
                    goto Label_0050;
            }
            return false;
        }

        [DebuggerHidden]
        void IEnumerator.Reset()
        {
            throw new NotSupportedException();
        }

        void IDisposable.Dispose()
        {
        }

        // Properties
        T IEnumerator<T>.Current
        {
            [DebuggerHidden]
            get
            {
                return this.<>2__current;
            }
        }

        object IEnumerator.Current
        {
            [DebuggerHidden]
```

```
            get
            {
                return this.<>2__current;
            }
        }
    }
}
```

(توجه: برای خواندن این کد، <...>ها را نادیده بگیرید، اینها هیچ وظیفهٔ خاصی ندارند و کار خاصی نمی‌کنند)
این کد را که البته چندان خوانا نیست اگر با کد انتهای مطلب قبل مقایسه کنید متوجه می‌شوید که دارای اشتراک‌هایی‌است. در آن مثال نیز شئ Enumerable یک شئ جداگانه بود (در اینجا یک کلاس درونی است) که هنگامی که GetEnumerator را صدا می‌زدیم نمونه‌ای از آن ایجاد می‌شد و بازگردانیده می‌شد.

در این کد کامپایلر وضعیت‌های مختلفی که برای توقف و ادامهٔ کار MoveNext که مهم‌ترین بخش کد هست را با کمک ترکیبی از switch case و goto پیاده‌سازی کرده‌است که با کمی دقت می‌توانید متوجه منطق آن شوید :)

ممکن است به نظرتان برسد که این قطعه کد از نظر (حداقل نامگذاری) در سی‌شارپ صحیح نیست. اینگونه نامگذاری‌ها که از نظر CLR (و زبان IL) درست ولی از نظر زبان سطح بالا نادرست هستند باعث می‌شوند که از هرگونه برخورد نامی احتمالی با نام‌های معتبر تعریف شده توسط کاربر جلوگیری شود.

احتمالاً اگر پیش‌زمینه نسبت به این مطلب داشته باشید با خود خواهید گفت که «این که واضح بود، اصلاً وظیفهٔ ماشین در سطح پایین نیست که چنین عملی را پشتیبانی کند». واضح‌بودن این موضوع برای شما شاید به این دلیل باشد که پیاده‌سازی yield را قبلاً جای دیگری ندیده‌اید. برای درک این مطلب در اینجا نحوهٔ پیاده‌سازی yield را در پایتون بررسی می‌کنیم.

```
def array_iterator(array):
    length = len(array)
    index = 0
    while index < length:
        yield array[index]
        index = index + 1
```

اگر کد مفسر پایتون را برای این generator بررسی کنیم متوجه می‌شویم که پایتون دارای عملگر خاصی در سطح ماشین برای yield است:

```
>>> import dis
>>> dis.dis(array_iterator)
  2           0 LOAD_GLOBAL              0 (len)
              3 LOAD_FAST                0 (array)
              6 CALL_FUNCTION            1
              9 STORE_FAST               1 (length)

  3          12 LOAD_CONST               1 (0)
             15 STORE_FAST               2 (index)

  4          18 SETUP_LOOP              35 (to 56)
        >>   21 LOAD_FAST                2 (index)
             24 LOAD_FAST                1 (length)
             27 COMPARE_OP               0 (<)
             30 POP_JUMP_IF_FALSE       55

  5          33 LOAD_FAST                0 (array)
             36 LOAD_FAST                2 (index)
             39 BINARY_SUBSCR
             40 YIELD_VALUE
             41 POP_TOP

  6          42 LOAD_FAST                2 (index)
             45 LOAD_CONST               2 (1)
             48 BINARY_ADD
             49 STORE_FAST               2 (index)
             52 JUMP_ABSOLUTE           21
        >>   55 POP_BLOCK
        >>   56 LOAD_CONST               0 (None)
             59 RETURN_VALUE
```

همانطور که می‌بینیم پایتون دارای عملگر خاصی برای پیاده‌سازی yield بوده و به مانند سی‌شارپ از قابلیت‌های قبلی ماشین برای پیاده‌سازی yield استفاده نکرده‌است.
yield و iteratorها قابلیت‌های زیادی را در اختیار برنامه‌نویسان قرار می‌دهند. برنامه‌نویسی async یکی از این قابلیت‌هاست.
پیوندهای ابتدای مقالهٔ اول را در این زمینه مطالعه کنید (البته با ورود دات‌نت ۴.۵ شیوهٔ دیگری نیز برای برنامه‌نویسی async ایجاد شده). از قابلیت‌های دیگر طراحی سادهٔ یک ماشین حالت است.
کد زیر ساده‌ترین حالت یک ماشین حالت را نمایش می‌دهد که به کمک قابلیت yield ساده‌تر پیاده‌سازی شده‌است:

```
public class SimpleStateMachine : IEnumerable<bool>
    {
        public IEnumerator<bool> GetEnumerator()
        {
            while (true)
            {
                yield return true;
                yield return false;
            }
        }

        System.Collections.IEnumerator System.Collections.IEnumerable.GetEnumerator()
        {
            return GetEnumerator();
        }
    }
```

(البته استفاده اینگونه از yield (در حلقهٔ بی‌نهایت) خطرناک است و ممکن است برنامه‌تان را در اثر بی‌دقتی قفل کنید، حداقل به همین دلیل بهتر است همیشه چنین اشیائی دارای محدودیت باشند.)
می‌توانید از SimpleStateMachine به این شکل استفاده کنید:

```
new SimpleStateMachine().Take(20).ToList().ForEach(x => Console.WriteLine(x));
```

که ۲۰ حالت از این ماشین حالت را چاپ خواهد کرد که البته اگر Take را قرار نمی‌دادیم برنامه را قفل می‌کرد.

نظرات خوانندگان

نویسنده: متفکر
تاریخ: ۱۳۹۱/۰۵/۱۹ ۱۰:۴۲

سلام... کلاس SimpleStateMachine (برخلاف نامش) Simple نیست و حتی Error-Prone هستش. اگر کلاس رو بدین شکل در نظر بگیریم:

```
public class ReallySimple<T> : IEnumerable<T>
{
    //blah blah blah...
}
```

در این صورت می‌تونیم با استفاده از Range همون کار رو انجام بدیم:

```
Enumerable.Range(1, 20).Select(r => new ReallySimple()).ToList().ForEach(x => Console.WriteLine(x));
```

نویسنده: ابراهیم بیاگوی
تاریخ: ۱۳۹۱/۰۵/۱۹ ۱۳:۴

به Error-Prone آن در خود مقاله هم اشاره شده. SimpleStateMachine که ما قصد پیاده‌سازی داشته‌ایم تقریباً به این حالت است: منبع
که می‌توان بین حرکت‌ها یک Action قرار داد. (که هدف ما بیشتر همین هست)
به طور کلی می‌گویند که State-Machine درست‌کردن با این قابلیت چندان درست نیست ، فقط امکان‌پذیر هست.

عنوان: **مروری بر کاربردهای Action و Func - قسمت اول**
نویسنده: وحید نصیری
تاریخ: ۱۳۹۱/۰۵/۲۵ ۱۳:۵
آدرس: www.dotnettips.info
گروه‌ها: C#, Refactoring

delegateها، نوع‌هایی هستند که ارجاعی را به یک متد دارند؛ بسیار شبیه به function pointers در C و CPP هستند، اما برخلاف آن‌ها، delegates شیءگرا بوده، به امضای متد اهمیت داده و همچنین کد مدیریت شده و امن به شمار می‌روند.
سیر تکاملی delegates را در مثال ساده زیر می‌توان ملاحظه کرد:

```
using System;

namespace ActionFuncSamples
{
    public delegate int AddMethodDelegate(int a);
    public class DelegateSample
    {
        public void UseDelegate(AddMethodDelegate addMethod)
        {
            Console.WriteLine(addMethod(5));
        }
    }

    public class Helper
    {
        public int CustomAdd(int a)
        {
            return ++a;
        }
    }

    class Program
    {
        static void Main(string[] args)
        {
            Helper helper = new Helper();

            // .NET 1
            AddMethodDelegate addMethod = new AddMethodDelegate(helper.CustomAdd);
            new DelegateSample().UseDelegate(addMethod);

            // .NET 2, anonymous delegates
            new DelegateSample().UseDelegate(delegate(int a) { return helper.CustomAdd(a); });

            // .NET 3.5
            new DelegateSample().UseDelegate(a => helper.CustomAdd(a));
        }
    }
}
```

معنای کلمه delegate، واگذاری مسئولیت است. به این معنا که ما در متد UseDelegate، نمی‌دانیم addMethod به چه نحوی تعریف خواهد شد. فقط می‌دانیم که امضای آن چیست.
در دات نت یک، یک وهله از شیء AddMethodDelegate ساخته شده و سپس متدی که امضایی متناسب و متناظر با آن را داشت، به عنوان متد انجام دهنده مسئولیت معرفی می‌شد. در دات نت دو، اندکی نحوه تعریف delegates با ارائه delegates بی‌نام، ساده‌تر شد و در دات نت سه و نیم با ارائه lambda expressions ، تعریف و استفاده از delegates باز هم ساده‌تر و زیباتر گردید.
به علاوه در دات نت 3 و نیم، دو Generic delegate به نام‌های Action و Func نیز ارائه گردیده‌اند که به طور کامل جایگزین تعریف طولانی delegates در کدهای پس از دات نت سه و نیم شده‌اند. تفاوت‌های این دو نیز بسیار ساده است:
اگر قرار است واگذاری قسمتی از کد را به متدی محول کنید که مقداری را بازگشت می‌دهد، از Func و اگر این متد خروجی ندارد از Action استفاده نمائید:

```
Action<int> example1 = x => Console.WriteLine("Write {0}", x);
example1(5);

Func<int, string> example2 = x => string.Format("{0:n0}", x);
Console.WriteLine(example2(5000));
```

در دو مثال فوق، نحوه تعریف inline یک Action و یا Func را ملاحظه می‌کنید. Action به متدی اشاره می‌کند که خروجی ندارد و در اینجا تنها یک ورودی int را می‌پذیرد. Func در اینجا به تابعی اشاره می‌کند که یک ورودی int را دریافت کرده و یک خروجی string را باز می‌گرداند.

پس از این مقدمه، در ادامه قصد داریم مثال‌های دنیای واقعی Action و Func را که در سال‌های اخیر بسیار متداول شده‌اند، بررسی کنیم.

مثال یک) ساده سازی تعاریف API ارائه شده به استفاده کنندگان از کتابخانه‌های ما

عنوان شد که کار delegates، واگذاری مسئولیت انجام کاری به کلاس‌های دیگر است. این مورد شما را به یاد کاربردهای interfaceها نمی‌اندازد؟

در interfaceها نیز یک قرارداد کلی تعریف شده و سپس کدهای یک کتابخانه، تنها با امضای متدها و خواص تعریف شده در آن کار می‌کنند و کتابخانه ما نمی‌داند که این متدها قرار است چه پیاده سازی خاصی را داشته باشند.

برای نمونه طراحی API زیر را درنظر بگیرید که در آن یک interface جدید تعریف شده که تنها حاوی یک متد است. سپس کلاس Runner از این interface استفاده می‌کند:

```
using System;

namespace ActionFuncSamples
{
    public interface ISchedule
    {
        void Run();
    }

    public class Runner
    {
        public void Exceute(ISchedule schedule)
        {
            schedule.Run();
        }
    }

    public class HelloSchedule : ISchedule
    {
        public void Run()
        {
            Console.WriteLine("Just Run!");
        }
    }

    class Program
    {
        static void Main(string[] args)
        {
            new Runner().Exceute(new HelloSchedule());
        }
    }
}
```

در اینجا ابتدا باید این interface را در طی یک کلاس جدید (مثلا HelloSchedule) پیاده سازی کرد و سپس حاصل را در کلاس Runner استفاده نمود.

نظر شما در مورد این طراحی ساده شده چیست؟

```
using System;

namespace ActionFuncSamples
{
    public class Schedule
    {
        public void Exceute(Action run)
        {
            run();
        }
    }

    class Program
```

```
    {
        static void Main(string[] args)
        {
            new Schedule().Exceute(() => Console.WriteLine("Just Run!"));
        }
    }
}
```

با توجه به اینکه هدف از معرفی interface در طراحی اول، واگذاری مسئولیت نحوه تعریف متد Run به کلاسی دیگر است، به همین طراحی با استفاده از یک Action delegate نیز می‌توان رسید. مهم‌ترین مزیت آن، حجم بسیار کمتر کدنویسی استفاده کننده نهایی از API تعریف شده ما است. به علاوه امکان inline coding نیز فراهم گردیده است و در همان محل تعریف Action، بدنه آن‌را نیز می‌توان تعریف کرد.

بدیهی است delegates نمی‌توانند به طور کامل جای interfaceها را پر کنند. اگر نیاز است قرارداد تهیه شده بین ما و استفاده کنندگان از کتابخانه، حاوی بیش از یک متد باشد، استفاده از interfaceها بهتر هستند.

از دیدگاه بسیاری از طراحان API، اشیاء delegate معادل interface ایی با یک متد هستند و یک وهله از delegate معادل وهله‌ای از کلاسی است که یک interface را پیاده سازی کرده‌است.

علت استفاده بیش از حد interfaceها در سایر زبان‌ها برای ابتدایی‌ترین کارها، کمبود امکانات پایه‌ای آن زبان‌ها مانند نداشتن anonymous methods، lambda expressions و anonymous delegates هستند. به همین دلیل مجبورند همیشه و در همه‌جا از interfaceها استفاده کنند.

ادامه دارد ...

نظرات خوانندگان

نویسنده: بهروز راد
تاریخ: ۱۳۹۱/۰۵/۲۵ ۱۷:۳۰

نمیشه همیشه اینطور گفت. بستگی به کاری داره که قرار هست انجام بشه. اینترفیس IComparable که فقط متد CompareTo رو داره، یک مثال نقض هست.

نویسنده: وحید نصیری
تاریخ: ۱۳۹۱/۰۵/۲۵ ۱۷:۳۷

طراحی IComparable مربوط به زمان دات نت یک است. اگر آن زمان امکانات زبان مثل امروز بود، می‌شد از طراحی ساده‌تری استفاده کرد.
یک نمونه از طراحی‌های اخیر تیم دات نت رو میشه در WebGrid دید. در این طراحی برای نمونه جهت دریافت فرمول فرمت کردن مقدار یک cell، از Func استفاده کردن. می‌شد این رو با اینترفیس هم نوشت (چون قرار است کاری به خارج از کلاس محول شود و هر بار اطلاعاتی به آن ارسال و نتیجه‌ای جدید اخذ گردد؛ پیاده سازی آن با شما، نتیجه را فقط در اختیار WebGrid ما قرار دهید). اما جدا استفاده از آن تبدیل می‌شد به عذاب برای کاربر که به نحو زیبایی با Func و امکانات جدید زبان حل شده.

نویسنده: بهروز راد
تاریخ: ۱۳۹۱/۰۵/۲۵ ۱۸:۵۱

فکر نمی‌کنم به خاطر دات نت 1 باشه. دلیلی فراتر از این وجود داره. با کمی جستجو، این لینک که بر اساس VS 2010 نوشته شده، در پاراگراف آخر دلیل منطقی‌تری رو ارائه میده. در مورد WebGrid که فرمودید، بحثش جداست. من از کامپوننت‌های متن باز Telerik در بستر ASP.NET MVC استفاده می‌کنم و از انعطاف پذیری Action و Func در متدهای اون لذت می‌برم. حرف من در مورد تعریف واجب استفاده از Predefined Delegates به جای اینترفیس‌های تک متدی است.

```
One good example of using a single-method interface instead of a delegate is
```

IComparable

```
or the generic version,
```

<IComparable<T

```
.
```

IComparable

```
declares the
```

CompareTo

```
method, which returns an integer that specifies a less than, equal to,
or greater than relationship between two objects of the same type.
```

IComparable

```
can be used as the basis of a sort algorithm. Although using a delegate
 comparison method as the basis of a sort algorithm would be valid, it
is not ideal. Because the ability to compare belongs to the class and
the comparison algorithm does not change at run time, a single-method
interface is ideal.
```

نویسنده: وحید نصیری

تاریخ: ۱۳۹۱/۰۵/۲۵ ۱۹:۱۱

- در مورد تعریف «واجب» کسی اینجا بحث نکرده. این هم یک دید طراحی است. آیا کسی می‌تونه بگه اولین طراحی مطرح شده در مطلب جاری اشتباه است؟ خیر. اما ضرورتی ندارد تا این اندازه صرفا جهت واگذاری مسئولیت انجام یک متد به کلاسی دیگر، اینقدر طراحی انجام شده زمخت و طولانی باشد.
- در متن MSDN فوق نوشته شده که استفاده از delegate در این حالت خاص نیز معتبر است؛ اما ایده‌آل نیست. دلیلی که آورده از نظر من ساختگی است. ضرورتی ندارد تعریف یک delegate معرفی شده در runtime عوض شود. یا عنوان کرده که IComparable پایه مرتب سازی یک سری از متدها است. خوب ... بله زمانیکه از روز اول اینطور طراحی کردید همه چیز به هم مرتبط خواهند بود.

پ.ن.
قسمت نظرات MSDN یک زمانی باز بود ولی ... بعد از مدتی پشیمان شدند و به نظر این قابلیت منسوخ شده در این سایت!

عنوان: مروری بر کاربردهای Action و Func - قسمت دوم
نویسنده: وحید نصیری
تاریخ: ۱۳۹۱/۰۵/۲۸ ۱۸:۱۰
آدرس: www.dotnettips.info
برچسب‌ها: C#, Refactoring

در قسمت قبل از Func و Actionها برای ساده سازی طراحی‌های مبتنی بر اینترفیس‌هایی با یک متد استفاده کردیم. این مورد خصوصا در حالت‌هایی که قصد داریم به کاربر اجازه‌ی فرمول نویسی بر روی اطلاعات موجود را بدهیم، بسیار مفید است.

مثال دوم) به استفاده کننده از API کتابخانه خود، اجازه فرمول نویسی بدهید

برای نمونه مثال ساده زیر را درنظر بگیرید که در آن قرار است یک سری عدد که از منبع داده‌ای دریافت شده‌اند، بر روی صفحه نمایش داده شوند:

```
public static void PrintNumbers()
{
    var numbers = new[] { 1,2,3,5,7,90 }; // from a data source
    foreach(var item in numbers)
    {
        Console.WriteLine(item);
    }
}
```

قصد داریم به برنامه نویس استفاده کننده از کتابخانه گزارش‌سازی خود، این اجازه را بدهیم که پیش از نمایش نهایی اطلاعات، بتواند توسط فرمولی که مشخص می‌کند، فرمت اعداد نمایش داده شده را تعیین کند.
روال کار اکثر ابزارهای گزارش‌سازی موجود، ارائه یک زبان اسکریپتی جدید برای حل این نوع مسایل است. اما با استفاده از Func و ... روش‌های Code first (بجای روش‌های Wizard first)، خیلی از این رنج و دردها را می‌توان ساده‌تر و بدون نیاز به اختراع و یا آموزش زبان جدیدی حل کرد:

```
public static void PrintNumbers(Func<int,string> formula)
{
    var numbers = new[] { 1,2,3,5,7,90 };  // from a data source
    foreach(var item in numbers)
    {
        var data = formula(item);
        Console.WriteLine(data);
    }
}
```

اینبار با استفاده از Func، امکان فرمول نویسی را به کاربر استفاده کننده از API ساده گزارش ساز فرضی خود داده‌ایم. Func تعریف شده در اینجا یک عدد int را در اختیار استفاده کننده قرار می‌دهد. در این بین، برنامه نویس می‌تواند هر نوع تغییر یا هر نوع فرمولی را که مایل است بر روی این عدد به کمک دستور زبان جاری مورد استفاده، اعمال کند و در آخر تنها باید نتیجه این عملیات را به صورت یک string بازگشت دهد. برای مثال:

```
PrintNumbers(number => string.Format("{0:n0}",number));
```

البته سطر فوق ساده شده فراخوانی زیر است:

```
PrintNumbers((number) =>{ return string.Format("{0:n0}",number); });
```

به این ترتیب اعداد نهایی با جدا کننده سه رقمی نمایش داده خواهند شد.
از این نوع طراحی، در ابزارها و کتابخانه‌های جدید گزارش سازی مخصوص ASP.NET MVC زیاد مشاهده می‌شوند.

مثال سوم) حذف کدهای تکراری برنامه

فرض کنید قصد دارید در برنامه وب خود مباحث caching را پیاده سازی کنید:

```
using System;
using System.Web;
using System.Web.Caching;
using System.Collections.Generic;

namespace WebToolkit
{
    public static class CacheManager
    {
        public static void CacheInsert(this HttpContextBase httpContext, string key, object data, int
durationMinutes)
        {
            if (data == null) return;
            httpContext.Cache.Add(
                key,
                data,
                null,
                DateTime.Now.AddMinutes(durationMinutes),
                TimeSpan.Zero,
                CacheItemPriority.AboveNormal,
                null);
        }
    }
}
```

در هر قسمتی از برنامه که قصد داشته باشیم اطلاعاتی را در کش ذخیره کنیم، الگوی تکراری زیر باید طی شود:

```
var item = httpContext.Cache[key];
if (item == null)
{
    item = ReadDataFromDataSource();
    if (item == null)
          return null;

    CacheInsert(httpContext, key, item, durationMinutes);
}
```

ابتدا باید وضعیت کش جاری بررسی شود؛ اگر اطلاعاتی در آن موجود نبود، ابتدا از منبع داده‌ای مورد نظر خوانده شده و سپس در کش درج شود.

می‌توان در این الگوی تکراری، خواندن اطلاعات را از منبع داده، به یک Func واگذار کرد و به این صورت کدهای ما به نحو زیر بازسازی خواهند شد:

```
using System;
using System.Web;
using System.Web.Caching;
using System.Collections.Generic;

namespace WebToolkit
{
    public static class CacheManager
    {
        public static void CacheInsert(this HttpContextBase httpContext, string key, object data, int
durationMinutes)
        {
            if (data == null) return;
            httpContext.Cache.Add(
                key,
                data,
                null,
                DateTime.Now.AddMinutes(durationMinutes),
                TimeSpan.Zero,
                CacheItemPriority.AboveNormal,
                null);
        }

        public static T CacheRead<T>(this HttpContextBase httpContext, string key, int durationMinutes,
Func<T> ifNullRetrievalMethod)
        {
            var item = httpContext.Cache[key];
            if (item == null)
            {
```

```
                item = ifNullRetrievalMethod();
                if (item == null)
                    return default(T);

                CacheInsert(httpContext, key, item, durationMinutes);
            }
            return (T)item;
        }
    }
}
```

و استفاده از آن نیز به نحو زیر خواهد بود:

```
var user = HttpContext.CacheRead(
                            "Key1",
                            15,
                            () => _usersService.FindUser(userId));
```

پارامتر سوم متد CacheRead به صورت خودکار تنها زمانیکه اطلاعات کش متناظری با کلید Key1 وجود نداشته باشند، اجرا شده و نتیجه در کش ثبت می‌گردد. در اینجا دیگر از if و else و کدهای تکراری بررسی وضعیت کش خبری نیست.

نظرات خوانندگان

نویسنده: علی علیار
تاریخ: ۱۳۹۱/۰۶/۱۵ ۲۰:۲

سلام میشه یه توضیحی درباره کد زیر بدید؟

```
public static T CacheRead<T>(this HttpContextBase httpContext, string key, int durationMinutes, Func<T>
ifNullRetrievalMethod)
        {
            var item = httpContext.Cache[key];
            if (item == null)
            {
                item = ifNullRetrievalMethod();
                if (item == null)
                    return default(T);

                CacheInsert(httpContext, key, item, durationMinutes);
            }
            return (T)item;
        }
```

نویسنده: وحید نصیری
تاریخ: ۱۳۹۱/۰۶/۱۵ ۲۰:۱۳

در متن توضیح دادم «... الگوی تکراری زیر باید طی شود ...».
برای خواندن اطلاعات از کش سیستم، این الگوی تکراری در هرجایی از برنامه باید انجام شود:
الف) ابتدا باید به شیء Cache مراجعه شود. شاید بر اساس یک key مفروض، اطلاعاتی در آن موجود باشد.
ب) اگر نبود (قطعه if تعریف شده)، باید به یک منبع داده مشخص، مراجعه و اطلاعات دریافت شود. سپس این اطلاعات در کش برای دفعات مراجعه بعدی ثبت گردد.
ج) اطلاعات نهایی بازگشت داده شود.

در اینجا قسمت مراجعه به منبع داده، توسط Func به استفاده کننده از متد CacheRead واگذار شده است. به این صورت ما فقط می‌دونیم که یک تابع در اختیار این متد قرار خواهد گرفت که در زمان مناسب می‌شود آن را فراخوانی کرد.
مثلا در مثالی که در انتهای بحث است یک نمونه از کاربرد آن‌را مشاهده می‌کنید.

عنوان: **مروری بر کاربردهای Action و Func - قسمت سوم**
نویسنده: وحید نصیری
تاریخ: ۱۳۹۱/۰۵/۲۹ ۲۰:۱۹
آدرس: www.dotnettips.info
گروه‌ها: C#, Refactoring

در ادامه مثال سوم قسمت قبل ، در مورد حذف کدهای تکراری توسط Action و Func، در این قسمت به یک مثال نسبتا پرکاربرد دیگر آن جهت ساده سازی try/catch/finally اشاره خواهد شد.
احتمالا هزاران بار در کدهای خود چنین قطعه کدی را تکرار کرده‌اید:

```
try {
        // code
} catch(Exception ex) {
        // do something
}
```

این مورد را نیز می‌توان توسط Actionها کپسوله کرد و پیاده سازی قسمت بدنه try آن‌را به فراخوان واگذار نمود:

```
void Execute(Action action) {
    try {
       action();
    } catch(Exception ex) {
       // log errors
    }
}
```

و برای نمونه جهت استفاده از آن خواهیم داشت:

```
Execute(() => {open a file});
```

یا اگر عمل انجام شده باید خروجی خاصی را بازگرداند (برخلاف یک Action که خروجی از آن انتظار نمی‌رود)، می‌توان طراحی متد Execute را با Func انجام داد:

```
public static class SafeExecutor
{
    public static T Execute<T>(Func<T> operation)
    {
        try
        {
            return operation();
        }
        catch (Exception ex)
        {
            // Log Exception
        }
        return default(T);
    }
}
```

در این حالت فراخوانی متد Execute به نحو زیر خواهد بود:

```
var data = SafeExecutor.Execute<string>(() =>
{
    // do something
    return "result";
});
```

و اگر در این بین استثنایی رخ دهد، علاوه بر ثبت جزئیات خطای رخ داده شده، نال را بازگشت خواهد داد.

از همین دست می‌توان به کپسوله سازی منطق «سعی مجدد» در انجام کاری اشاره کرد:

```
public static class RetryHelper
{
   public static void RetryOperation(Action action, int numRetries, int retryTimeout)
   {
       if( action == null )
           throw new ArgumentNullException("action");

       do
       {
          try {  action(); return;  }
          catch
          {
              if( numRetries <= 0 ) throw;
              else
                 Thread.Sleep( retryTimeout );
           }
       } while( numRetries-- > 0 );
   }
}
```

برای مثال فرض کنید برنامه قرار است اطلاعاتی را از وب دریافت کند. ممکن است در سعی اول آن، خطای اتصال یا در دسترس نبودن لحظه‌ای سایت رخ دهد. در اینجا نیاز خواهد بود تا این عملیات چندین بار تکرار شود؛ که نمونه‌ای از آن‌را در ذیل ملاحظه می‌کنید:

```
RetryHelper.RetryOperation(() => SomeFunction(), 3, 1000);
```

نظرات خوانندگان

نویسنده: مجتبی صحرائی
تاریخ: ۱۳۹۱/۰۵/۲۹ ۲۱:۶

بسیار زیبا

نویسنده: امیر
تاریخ: ۱۳۹۱/۰۵/۲۹ ۲۲:۴۶

واقعا بحث زیبا و پرکاربردی است.

نویسنده: رضا
تاریخ: ۱۳۹۱/۰۵/۳۰ ۲۲:۲۸

واقعاً مبحث فوق العاده ای هست و شما هم عالی توضیح میدید. حذف کدهای تکراری واقعاً کمک کننده هستش.
ممنون.

نویسنده: Hamid NCH
تاریخ: ۱۳۹۲/۰۹/۱۳ ۱۵:۵۱

در مورد این توضیح میدین.خیلی ممنون

```
return default (T);
```

نویسنده: وحید نصیری
تاریخ: ۱۳۹۲/۰۹/۱۳ ۱۷:۳۳

گاهی از اوقات حین کار با نوع‌های جنریک نیاز دارید که مثلا null بازگشت بدید. در این حالت کامپایلر شما را با خطای Cannot convert null to type parameter T متوقف می‌کند. به همین جهت مرسوم است در این حالت از default T استفاده شود که مقدار پیش فرض نوع را برمی‌گرداند. اگر reference type باشد (مثل کلاس‌ها) این مقدار پیش فرض null خواهد بود؛ اگر value type باشد مانند int صفر بازگشت داده می‌شود.

عنوان: **تبدیل تاریخ میلادی به شمسی در SSIS به کمک سی شارپ**
نویسنده: محمد باقر سیف اللهی
تاریخ: ۱۳۹۱/۰۵/۳۱ ۲۲:۲
آدرس: www.dotnettips.info
برچسب‌ها: C#, SSIS

برای تبدیل تاریخ میلادی به تاریخ شمسی در packageهای SSIS می‌توان از زبان سی شارپ استفاده کرد . بدین طریق می‌توان در طی عملیات ETL و هنگام transform کردن داده‌ها ، عملیات تبدیل از میلادی به شمسی را انجام داد . عملیات تبدیل داده در این مثال به کمک Script Component انجام می‌شود.

برای این کار از داده‌های موجود در پایگاه داده **[AdventureWorksLT2008R2].[SalesLT].[Address]** استفاده می‌کنم .
برای این کار یک package تعریف می‌کنم و برای استخراج داده‌ها یک OLEDB تعریف می‌کنم . برای تبدیل داده‌ها از Script Component و برای گرفتن خروجی آزمایشی از union استفاده می‌کنم :

دایره‌های قرمز رنگ مشخص شده در تصویر ، بیانگر data viewerهای تعریف شده است.

در تنظیمات dataSource مانند زیر عمل می‌کنیم :
دستور زیر فقط برای نمایش یک نمونه از کارکرد عملیات مورد نظر در این مثال می‌باشد

نمونه خروجی مانند زیر می‌باشد :

سپس برای تنظیمات Script Component به ترتیب زیر عمل می‌کنیم :

در قسمت Input column ستون هایی را که به عنوان پارامتر می‌توانیم با آنها کار کنیم ، تعریف می‌کنیم : (دقت شود که تغییر نام متغیرها ، در کدها اعمال می‌شوند .)

حال می‌خواهم ستون‌های تاریخ میلادی و شمسی را برای خروجی تعریف کنم :

همانطور که مشاهده می‌کنید ، نوع داده ای برای خروجی را رشته تعریف کردم.

سپس به قسمت script بر می‌گردیم (سمت چپ پنجره) و روی Edit Script کلیک می‌کنیم : (در صورتی که تمایل به کد نویسی با VB را دارید در همین قسمت می‌توانید آن را تنظیم کنید)

پنجره ای مانند زیر برای ویرایش کدها ، باز می‌شود

ssisscript - Integration Services Script Component (Administrator)

File Edit View Refactor Project Build Debug Data Tools Window Help

main.cs

ScriptMain | PreExecute()

```
/* Microsoft SQL Server Integration Services Script Component
 *  Write scripts using Microsoft Visual C# 2008.
 *  ScriptMain is the entry point class of the script.*/

using System;
using System.Data;
using Microsoft.SqlServer.Dts.Pipeline.Wrapper;
using Microsoft.SqlServer.Dts.Runtime.Wrapper;

[Microsoft.SqlServer.Dts.Pipeline.SSISScriptComponentEntryPointAttribute]
public class ScriptMain : UserComponent
{

    public override void PreExecute()...

    public override void PostExecute()...

    public override void Input0_ProcessInputRow(Input0Buffer Row)
    {
        /*
          Add your code here
        */

        string shamsi = MiladiToShamsi(Row.ModifiedDate);
        Row.Miladi = Row.ModifiedDate.ToShortDateString();
        Row.Shamsi = shamsi;

    }
    public string Get2Digits(string A)...
    private bool MiladiIsLeap(int tt)...
    public string MiladiToShamsi(int iMiladiMonth, int iMiladiDay, int iMiladiYear)...
    public string MiladiToShamsi(DateTime e)...

}
```

همانطور که مشاهده می‌کنید درون این کلاس 4 متد تبدیل تاریخ را پیاده کردم و از آنها در متد input0_processInputRow استفاده کردم . این کار نیازی به پیاده سازی حلقه ندارد و به راحتی می‌توان آنها را روی سطرهای دلخواه تعریف کرد . خروجی نمایش داده شده در data viewerها مانند زیر می‌باشند :

قبل از اعمال تبدیلات :

بعد از اعمال تبدیلات :

Task:
My DF 1
OLE DB Source
25 rows
Script Component
25 rows
Union All
Data Viewer 1 - AFTER at Script Component.Outp...
Detach
Copy Data
ModifiedDate Miladi Shamsi
2005-07-01 ... 07/01/2005 1384/04/10
2005-08-01 ... 08/01/2005 1384/05/10
2005-09-01 ... 09/01/2005 1384/06/10
2005-10-01 ... 10/01/2005 1384/07/09
2005-11-01 ... 11/01/2005 1384/08/10
2005-12-01 ... 12/01/2005 1384/09/10
2006-01-01 ... 01/01/2006 1384/10/11
2006-03-01 ... 03/01/2006 1384/12/10
2006-06-01 ... 06/01/2006 1385/03/11
2006-07-01 ... 07/01/2006 1385/04/10
2006-08-01 ... 08/01/2006 1385/05/10
2006-09-01 ... 09/01/2006 1385/06/10
2006-10-01 ... 10/01/2006 1385/07/09
2006-11-01 ... 11/01/2006 1385/08/10
Detached
Total rows: 25, bu
Rows displayed =
n Managers
AdventureWorksLT2008R2

موفق باشید

نظرات خوانندگان

نویسنده: پژمان
تاریخ: ۱۳۹۱/۰۵/۳۱ ۲۲:۱۰

سلام. ممنون از شما.

کاربرد ssis در محیط کاری برای شما بیشتر در چه مواردی است؟

نویسنده: محمد باقر سیف اللهی
تاریخ: ۱۳۹۱/۰۶/۰۱ ۷:۴۴

سلام ... SSIS کاربردهای زیادی داره مخصوصا وقتی که با تکنولوژی‌های دیگه ترکیب بشه .... اما نمونه ای که خودم با اون برخورد داشتم ، برای انتقال داده از پایگاه داده‌های قدیمی با فرمت هایی مثل Access به پایگاه داده‌های جدید در قالب SQL بود که خیلی مفید بود و سرعت اجرای عملیاتش فوق العاده است . موفق باشید

نویسنده: بهروز راد
تاریخ: ۱۳۹۱/۰۶/۰۱ ۱۳:۶

ما نیز با SSIS زیاد سر و کار داریم. چون از زبانی با نام Natural استفاده می‌کنیم که برای به اشتراک گذاری داده‌ها فقط می‌تونه با فایل‌های متنی ساده کار کنه. SSIS به ما کمک می‌کنه تا داده‌های ایجاد شده رو به SQL Server منتقل کنیم.

نویسنده: علیرضا
تاریخ: ۱۳۹۱/۰۸/۲۷ ۱۴:۴۹

http://learnbi.ir/1391/08/22/post-14/

نویسنده: وحید نصیری
تاریخ: ۱۳۹۱/۰۸/۲۷ ۱۸:۴۶

امکان به اشتراک گذاشتن مطالب برای کلیه اعضای سایت وجود دارد:
http://www.dotnettips.info/DailyLinks
لطفا برای بعد از این قسمت ویژه استفاده کنید.

عنوان: مروری بر کاربردهای Action و Func - قسمت چهارم
نویسنده: وحید نصیری
تاریخ: ۱۳۹۱/۰۶/۰۵ ۲۳:۵۳
آدرس: www.dotnettips.info
برچسب‌ها: C#, Refactoring

طراحی API برنامه توسط Actionها

روش مرسوم طراحی Fluent interfaces ، جهت ارائه روش ساخت اشیاء مسطح به کاربران بسیار مناسب هستند. اما اگر سعی در تهیه API عمومی برای کار با اشیاء چند سطحی مانند معرفی فایل‌های XML توسط کلاس‌های سی شارپ کنیم، اینبار Fluent interfaces آنچنان قابل استفاده نخواهند بود و نمی‌توان این نوع اشیاء را به شکل روانی با کنار هم قرار دادن زنجیر وار متدها تولید کرد. برای حل این مشکل روش طراحی خاصی در نگارش‌های اخیر NHibernate معرفی شده است به نام loquacious interface که این روزها در بسیاری از APIهای جدید شاهد استفاده از آن هستیم و در ادامه با پشت صحنه و طرز تفکری که در حین ساخت این نوع API وجود دارد آشنا خواهیم شد.

در ابتدا کلاس‌های مدل زیر را در نظر بگیرید که قرار است توسط آن‌ها ساختار یک جدول از کاربر دریافت شود:

```
using System;
using System.Collections.Generic;

namespace Test
{
    public class Table
    {
        public Header Header { set; get; }
        public IList<Cell> Cells { set; get; }
        public float Width { set; get; }
    }

    public class Header
    {
        public string Title { set; get; }
        public DateTime Date { set; get; }
        public IList<Cell> Cells { set; get; }
    }

    public class Cell
    {
        public string Caption { set; get; }
        public float Width { set; get; }
    }
}
```

در روش طراحی loquacious interface به ازای هر کلاس مدل، یک کلاس سازنده ایجاد خواهد شد. اگر در کلاس جاری، خاصیتی از نوع کلاس یا لیست باشد، برای آن نیز کلاس سازنده خاصی درنظر گرفته می‌شود و این روند ادامه پیدا می‌کند تا به خواصی از انواع ابتدایی مانند int و string برسیم:

```
using System;
using System.Collections.Generic;

namespace Test
{
    public class TableApi
    {
        public Table CreateTable(Action<TableCreator> action)
        {
            var creator = new TableCreator();
            action(creator);
            return creator.TheTable;
        }
    }

    public class TableCreator
    {
        readonly Table _theTable = new Table();
        internal Table TheTable
        {
```

```
            get { return _theTable; }
        }

        public void Width(float value)
        {
            _theTable.Width = value;
        }

        public void AddHeader(Action<HeaderCreator> action)
        {
            _theTable.Header = ...
        }

        public void AddCells(Action<CellsCreator> action)
        {
            _theTable.Cells = ...
        }
    }
}
```

نقطه آغازین API ایی که در اختیار استفاده کنندگان قرار می‌گیرد با متد CreateTable ایی شروع می‌شود که ساخت شیء جدول را به ظاهر توسط یک Action به استفاده کننده واگذار کرده است، اما توسط کلاس TableCreator او را مقید و راهنمایی می‌کند که چگونه باید اینکار را انجام دهد.

همچنین در بدنه متد CreateTable، نکته نحوه دریافت خروجی از Action ایی که به ظاهر خروجی خاصی را بر نمی‌گرداند نیز قابل مشاهده است.

همانطور که عنوان شد کلاس‌های xyzCreator تا رسیدن به خواص معمولی و ابتدایی پیش می‌روند. برای مثال در سطح اول چون خاصیت عرض از نوع float است، صرفا با یک متد معمولی دریافت می‌شود. دو خاصیت دیگر نیاز به Creator دارند تا در سطحی دیگر برای آن‌ها سازنده‌های ساده‌تری را طراحی کنیم.

همچنین باید دقت داشت که در این طراحی تمام متدها از نوع void هستند. اگر قرار است خاصیتی را بین خود رد و بدل کنند، این خاصیت به صورت internal تعریف می‌شود تا در خارج از کتابخانه قابل دسترسی نباشد و در intellisense ظاهر نشود.

مرحله بعد، ایجاد دو کلاس HeaderCreator و CellsCreator است تا کلاس TableCreator تکمیل گردد:

```
using System;
using System.Collections.Generic;

namespace Test
{
    public class CellsCreator
    {
        readonly IList<Cell> _cells = new List<Cell>();
        internal IList<Cell> Cells
        {
            get { return _cells; }
        }

        public void AddCell(string caption, float width)
        {
            _cells.Add(new Cell { Caption = caption, Width = width });
        }
    }

    public class HeaderCreator
    {
        readonly Header _header = new Header();
        internal Header Header
        {
            get { return _header; }
        }

        public void Title(string title)
        {
            _header.Title = title;
        }

        public void Date(DateTime value)
        {
            _header.Date = value;
        }

        public void AddCells(Action<CellsCreator> action)
```

```
        {
            var creator = new CellsCreator();
            action(creator);
            _header.Cells = creator.Cells;
        }
    }
}
```

نحوه ایجاد کلاس‌های Builder و یا Creator این روش بسیار ساده و مشخص است:
مقدار هر خاصیت معمولی توسط یک متد ساده void دریافت خواهد شد.
هر خاصیتی که اندکی پیچیدگی داشته باشد، نیاز به یک Creator جدید خواهد داشت.
کار هر Creator بازگشت دادن مقدار یک شیء است یا نهایتا ساخت یک لیست از یک شیء. این مقدار از طریق یک خاصیت internal بازگشت داده می‌شود.

البته عموما بجای معرفی مستقیم کلاس‌های Creator از یک اینترفیس معادل آن‌ها استفاده می‌شود. سپس کلاس Creator را internal تعریف می‌کنند تا خارج از کتابخانه قابل دسترسی نباشد و استفاده کننده نهایی فقط با توجه به متدهای void تعریف شده در interface کار تعریف اشیاء را انجام خواهد داد.

در نهایت، مثال تکمیل شده ما به شکل زیر خواهد بود:

```
using System;
using System.Collections.Generic;

namespace Test
{
    public class TableCreator
    {
        readonly Table _theTable = new Table();
        internal Table TheTable
        {
            get { return _theTable; }
        }

        public void Width(float value)
        {
            _theTable.Width = value;
        }

        public void AddHeader(Action<HeaderCreator> action)
        {
            var creator = new HeaderCreator();
            action(creator);
            _theTable.Header = creator.Header;
        }

        public void AddCells(Action<CellsCreator> action)
        {
            var creator = new CellsCreator();
            action(creator);
            _theTable.Cells = creator.Cells;
        }
    }

    public class CellsCreator
    {
        readonly IList<Cell> _cells = new List<Cell>();
        internal IList<Cell> Cells
        {
            get { return _cells; }
        }

        public void AddCell(string caption, float width)
        {
            _cells.Add(new Cell { Caption = caption, Width = width });
        }
    }

    public class HeaderCreator
    {
        readonly Header _header = new Header();
```

```
        internal Header Header
        {
            get { return _header; }
        }

        public void Title(string title)
        {
            _header.Title = title;
        }

        public void Date(DateTime value)
        {
            _header.Date = value;
        }

        public void AddCells(Action<CellsCreator> action)
        {
            var creator = new CellsCreator();
            action(creator);
            _header.Cells = creator.Cells;
        }
    }
}
```

نحوه استفاده از این طراحی نیز جالب توجه است:

```
var data = new TableApi().CreateTable(table =>
            {
                table.Width(1);
                table.AddHeader(header=>
                {
                    header.Title("new rpt");
                    header.Date(DateTime.Now);
                    header.AddCells(cells=>
                    {
                        cells.AddCell("cell 1", 1);
                        cells.AddCell("cell 2", 2);
                    });
                });
                table.AddCells(tableCells=>
                {
                    tableCells.AddCell("c 1", 1);
                    tableCells.AddCell("c 2", 2);
                });
            });
```

این نوع طراحی مزیت‌های زیادی را به همراه دارد:

الف) ساده سازی طراحی اشیاء چند سطحی و تو در تو

ب) امکان درنظر گرفتن مقادیر پیش فرض برای خواص

ج) ساده‌تر سازی تعاریف لیست‌ها

د) استفاده کنندگان در حین استفاده نهایی و تعریف اشیاء به سادگی می‌توانند کدنویسی کنند (مثلا سلول‌ها را با یک حلقه اضافه کنند).

ه) امکان بهتر استفاده از امکانات Intellisense . برای مثال فرض کنید یکی از خاصیت‌هایی که قرار است برای آن Creator درست کنید یک interface را می‌پذیرد. همچنین در برنامه خود چندین پیاده سازی کمکی از آن نیز وجود دارد. یک روش این است که مستندات قابل توجهی را تهیه کنید تا این امکانات توکار را گوشزد کند؛ روش دیگر استفاده از طراحی فوق است. در اینجا در کلاس Creator ایجاد شده چون امکان معرفی متد وجود دارد، می‌توان امکانات توکار را توسط این متدها نیز معرفی کرد و به این ترتیب Intellisense تبدیل به راهنمای اصلی کتابخانه شما خواهد شد.

نظرات خوانندگان

نویسنده: بهروز راد
تاریخ: ۱۳۹۱/۰۶/۰۶ ۱۷:۳۸

این تکنیک و مقاله، یکی از مطالب Must Read سال هست. به شخصه از این تکنیک در توسعه‌ی کامپوننت‌های ASP.NET MVC استفاده می‌کنم. کلاً تکنیک Fluent که برادر نصیری فعلاً در دو مقاله به اون پرداختند، انعطاف پذیری بسیاری به برنامه‌ها میده. مثلاً شبیه سازی روال RowDataBound کنترل GridView در Web Forms، در بستر MVC با استفاده از یک Action.

به نظر من کمبودی که ASP.NET MVC در حال حاضر داره، داشتن مجموعه ای غنی از کامپوننت‌های توکار هست که فکر می‌کنم در نسخه‌های آینده، مایکروسافت این نقیصه رو بر طرف می‌کنه، شاید با مشارکت شرکت‌های دیگه مثل Telerik.

عنوان: **نکات مربوط به گرد کردن اعداد در دات نت**
نویسنده: امین مصباحی
تاریخ: ۱۳۹۱/۰۶/۰۹ ۰:۰
آدرس: www.dotnettips.info
برچسب‌ها: C#, Round

متد System.Math.Round که برای گرد کردن اعداد اعشاری به کار می‌رود، دارای 8 نوعoverload می‌باشد، که عدم توجه به موارد مربوط به آن باعث بروز خطا در محاسبات خواهد شد. به طور مثال پیش بینی شما از گرد کردن عدد 3.45 عدد 3.5 است ولی گاهی 3.5 و گاهی 3.4 گرد خواهد شد.
پس بهتر است تا با نکات زیر به شکل دقیق آشنا باشید.

```
Round(Decimal)
Rounds a decimal value to the nearest integral value.
```

(ورودی: دسیمال) به نزدیکترین عدد کامل گرد میکند، 4.3 به 4 و 4.8 به 5 گرد می‌شود. ولی در صورتیکه فاصله تا عدد کامل قبل و بعد برابر باشد به نزدیکترین عدد زوج گرد میکند، 4.5 به 4 گرد می‌شود چرا که 5 عددی فرد است.
نکته 1: خروجی تابع از نوع دسیمال است نه عدد کامل.
نکته 2: این تابع بر طبق استاندارد IEEE Standard 754, section 4 پیاده سازی شده است که در اصطلاح rounding to nearest یا banker's roundingنیز گفته می‌شود. نتیجه برای به حداقل رساندن خطا است. نتیجه این حالت از متد با نتیجه Round(Decimal, MidpointRounding.ToEven) برابر است.

```
Round(Double)
Rounds a double-precision floating-point value to the nearest integral value.
```

(ورودی: double) به نزدیکترین عدد کامل گرد میکند، 4.3 به 4 و 4.8 به 5 گرد می‌شود. ولی در صورتیکه فاصله تا عدد کامل قبل و بعد برابر باشد به نزدیکترین عدد زوج گرد میکند، 4.5 به 4 گرد می‌شود چرا که 5 عددی فرد است.
نکته 1: خروجی تابع از نوع double است نه عدد کامل.
نکته 2: این تابع بر طبق استاندارد IEEE Standard 754, section 4 پیاده سازی شده است که در اصطلاح rounding to nearest یا banker's roundingنیز گفته می‌شود. نتیجه برای به حداقل رساندن خطا است. نتیجه این حالت از متد با نتیجه Round(Double, MidpointRounding.ToEven) برابر است.
نکته 3: گاهی اوقات به دلیل از دست دادن دقت، ناشی از استفاده از مقادیر دسیمال به جای ممیز شناور و یا انجام محاسبات ریاضی بر روی بخش ممیزی خواهد بود. مثلا: زمانی که 11.5 ماحصل جمع 1. و 11.4 باشد به جای 12 که عدد زوج است به 11 گرد می‌شود!

```
Round(Decimal, Int32)
Rounds a decimal value to a specified number of fractional digits.
```

عدد دسیمال ورودی خود را به صورتی گرد میکند که:
1: تعداد ارقام اعشاری بعد از ممیز به اندازه پارامتر دوم این نوع ورودی متد Round باشد (بین صفر تا 28).
2: استفاده از این متد همانند فراخوانی آن با ورودی‌های Round(Decimal, Int32, MidpointRounding.ToEven) است. یعنی اینکه اگر رقم آخر بعد از ممیز دقیقا وسط مقدار قبل و بعد باشد (3.75) در صورتی که رقم ماقبل آخر فرد باشد رو به بالا گرد خواهد شد (مثال: 3.75 به 3.8 گرد خواهد شد) و اگر رقم ما قبل آخر زوج باشد تغییر نخواهد کرد (مثال: 3.45 به 3.4 گرد خواهد شد)
نکته 1: این تابع بر طبق استاندارد IEEE Standard 754, section 4 پیاده سازی شده است که در اصطلاح rounding to nearest یا banker's roundingنیز گفته می‌شود.

```
Math.Round(3.44, 1); //Returns 3.4.
Math.Round(3.45, 1); //Returns 3.4.
Math.Round(3.46, 1); //Returns 3.5.
```

```
Math.Round(4.34, 1); // Returns 4.3
Math.Round(4.35, 1); // Returns 4.4
Math.Round(4.36, 1); // Returns 4.4
```

```
Round(Decimal, MidpointRounding)
Rounds a decimal value to the nearest integer.
A parameter specifies how to round the value if it is midway between two other numbers.
```

عدد دسیمال ورودی خود را به نزدیک‌ترین عدد integer گرد میکند، پارامتر اول عدد گرد نشده و پارامتر دوم مشخص میکند که در صورتیکه مقدار اعشاری عددی میانی (5و 50و 500و 5000و .... باشد) چگونه این گرد کردن صورت گیرد.

حالت‌های MidpointRounding:

MidpointRounding.ToEven: در صورتیکه مقدار اعشاری عددی میانی (5و 50و 500و 5000و ....) باشد، و رقم ماقبل آخر اعشار فرد باشد، رو به بالا گرد خواهد شد و در صورتیکه رقم ماقبل آخر اعشار زوج باشد، بدون تغییر باقی خواهد ماند (3.75 به 3.8 و 3.65 به 3.6 گرد می‌شود.)

نکته 1: این تابع بر طبق استاندارد IEEE Standard 754, section 4 پیاده سازی شده است که در اصطلاح rounding to nearest یا banker's roundingنیز گفته می‌شود.

MidpointRounding.AwayFromZero: در صورتیکه مقدار اعشاری عددی میانی (5و 50و 500و 5000و ....) باشد، در این حالت همواره عمل گرد کردن رو به رقم بعدی خواهد بود. این رایج‌ترین حالت گرد کردن است که به symmetric arithmetic rounding شناخته می‌شود.

```
Round(Double, Int32)
Rounds a double-precision floating-point value to a specified number of fractional digits.
```

یک عدد اعشاری از نوع Double (با دقت مضاعف) که تعداد مشخصی رقم بعد از ممیز دارد (به طور مثال 10 رقم اعشار)، به تعداد رقم اعشاری که کاربر به عنوان پارامتر دوم ذکر میکند(بین صفر تا 15 رقم)، (مثلا 4 رقم)گرد میکند. اگر تعداد رقم اعشار بیش از 15 تعیین شود، عدد 15 جایگزین خواهد شد. استفاده از این متد همانند فراخوانی آن به صورت Round(Double, Int32, MidpointRounding.ToEven) می‌باشد. یعنی اینکه اگر رقم آخر بعد از ممیز دقیقا وسط مقدار قبل و بعد باشد (3.75) در صورتی که رقم ماقبل آخر فرد باشد رو به بالا گرد خواهد شد (مثال: 3.75 به 3.8 گرد خواهد شد) و اگر رقم ما قبل آخر زوج باشد تغییر نخواهد کرد (مثال: 3.45 به 3.4 گرد خواهد شد)

نکته 1: این تابع بر طبق استاندارد IEEE Standard 754, section 4 پیاده سازی شده است که در اصطلاح rounding to nearest یا banker's roundingنیز گفته می‌شود.

```
Round(Double, MidpointRounding)
Rounds a double-precision floating-point value to the nearest integer.
A parameter specifies how to round the value if it is midway between two other numbers.
```

(عدد اعشاری ورودی: Double) عدد با دقت مضاعف ورودی خود را به نزدیک‌ترین عدد integer گرد میکند، پارامتر اول عدد گرد نشده و پارامتر دوم مشخص میکند در صورتیکه مقدار اعشاری عددی میانی (5و 50و 500و 5000و .... باشد) چگونه این گرد کردن صورت گیرد.

حالت‌های MidpointRounding:

MidpointRounding.ToEven: در صورتیکه مقدار اعشاری عددی میانی (5و 50و 500و 5000و ....) باشد، و رقم ماقبل آخر اعشار فرد باشد، رو به بالا گرد خواهد شد و در صورتیکه رقم ماقبل آخر اعشار زوج باشد، بدون تغییر باقی خواهد ماند (3.75 به 3.8 و 3.65 به 3.6 گرد می‌شود.)

نکته 1: این تابع بر طبق استاندارد IEEE Standard 754, section 4 پیاده سازی شده است که در اصطلاح rounding to nearest یا banker's roundingنیز گفته می‌شود.

MidpointRounding.AwayFromZero: در صورتیکه مقدار اعشاری عددی میانی (5و 50و 500و 5000و ....) باشد، در این حالت همواره عمل گرد کردن رو به رقم بعدی خواهد بود. این رایج‌ترین حالت گرد کردن است که به symmetric arithmetic rounding شناخته می‌شود.

```
Round(Decimal, Int32, MidpointRounding)
Rounds a decimal value to a specified number of fractional digits.
A parameter specifies how to round the value if it is midway between two other numbers.
```

عدد دسیمال ورودی خود را با تعداد اعشار اعلام شده و به صورتی گرد میکند که:

1: تعداد ارقام اعشاری بعد از ممیز به اندازه پارامتر دوم این نوع ورودی متد Round باشد (بین صفر تا 28).

حالت‌های MidpointRounding:

MidpointRounding.ToEven: در صورتیکه مقدار اعشاری عددی میانی (5و 50و 500و 5000و ....) باشد، و رقم ماقبل آخر اعشار فرد باشد، رو به بالا گرد خواهد شد و در صورتیکه رقم ماقبل آخر اعشار زوج باشد، بدون تغییر باقی خواهد ماند (3.75 به 3.8 و 3.65 به 3.6 گرد می‌شود.)

نکته 1: این تابع بر طبق استاندارد IEEE Standard 754, section 4 پیاده سازی شده است که در اصطلاح rounding to nearest یا banker's roundingنیز گفته می‌شود.

MidpointRounding.AwayFromZero: در صورتیکه مقدار اعشاری عددی میانی (5و 50و 500و 5000و ....) باشد، در این حالت همواره عمل گرد کردن به سمت رقم بعدی خواهد بود. این رایج‌ترین حالت گرد کردن است که به symmetric arithmetic rounding شناخته می‌شود.

```
3.4 = Math.Round( 3.45, 1, MidpointRounding.ToEven)
3.5 = Math.Round( 3.45, 1, MidpointRounding.AwayFromZero)

-3.4 = Math.Round(-3.45, 1, MidpointRounding.ToEven)
-3.5 = Math.Round(-3.45, 1, MidpointRounding.AwayFromZero)
```

```
Round(Double, Int32, MidpointRounding)
Rounds a double-precision floating-point value to the specified number of fractional digits.
A parameter specifies how to round the value if it is midway between two other numbers.
```

(عدد اعشاری ورودی: Double) عدد با دقت مضاعف ورودی خود را به نزدیک‌ترین عدد با تعداد رقم اعشار مشخص شده گرد میکند، پارامتر اول متد، عدد گرد نشده و پارامتر دوم تعداد رقم اعشار (بین صفر تا 15 رقم) تعیین شده گرد می‌کند. اگر تعداد رقم اعشار بیش از 15 تعیین شود، عدد 15 جایگزین خواهد شد. و پارامتر سوم مشخص میکند که در صورتیکه مقدار اعشاری عددی میانی (5و 50و 500و 5000و .... باشد) چگونه این گرد کردن صورت گیرد.

حالت‌های MidpointRounding:

MidpointRounding.ToEven: در صورتیکه مقدار اعشاری عددی میانی (5و 50و 500و 5000و ....) باشد، و رقم ماقبل آخر اعشار فرد باشد، رو به بالا گرد خواهد شد و در صورتیکه رقم ماقبل آخر اعشار زوج باشد، بدون تغییر باقی خواهد ماند (3.75 به 3.8 و 3.65 به 3.6 گرد می‌شود.)

نکته 1: این تابع بر طبق استاندارد IEEE Standard 754, section 4 پیاده سازی شده است که در اصطلاح rounding to nearest یا banker's roundingنیز گفته می‌شود.

MidpointRounding.AwayFromZero: در صورتیکه مقدار اعشاری عددی میانی (5و 50و 500و 5000و ....) باشد، در این حالت همواره عمل گرد کردن به سمت رقم بعدی خواهد بود. این رایج‌ترین حالت گرد کردن است که به symmetric arithmetic

rounding شناخته می‌شود.

```
// The example displays the following output:
// 2.125 --> 2.13
// 2.135 --> 2.13
// 2.145 --> 2.15
// 3.125 --> 3.13
// 3.135 --> 3.14
// 3.145 --> 3.15
This code example produces the following results:

 3.4 = Math.Round( 3.45, 1)
-3.4 = Math.Round(-3.45, 1)

 3.4 = Math.Round( 3.45, 1, MidpointRounding.ToEven)
 3.5 = Math.Round( 3.45, 1, MidpointRounding.AwayFromZero)

-3.4 = Math.Round(-3.45, 1, MidpointRounding.ToEven)
-3.5 = Math.Round(-3.45, 1, MidpointRounding.AwayFromZero)
```

نظرات خوانندگان

نویسنده: سعید
تاریخ: ۱۳۹۱/۰۶/۰۹ ۱۳:۳۸

جالب بود. هیچ وقت به این مساله تا این حد دقیق نشده بودم!

نویسنده: مسعود زیانی
تاریخ: ۱۳۹۱/۰۹/۱۹ ۱۷:۳۳

با سلام1

من اعداد دسیمالم رو به صورت زیر رند میکنم:

```
decimal.round(x,2)
```

حالا اگه عددم مثلا 12 باشه به صورت زیر نشون میده:

12.00

در صورتیکه نمیخوام دوتا صفر رو نشون بده...باید چیکار کنم؟

نویسنده: وحید نصیری
تاریخ: ۱۳۹۱/۰۹/۱۹ ۱۸:۴۲

```
dec.ToString("G29")
```

این مطلب را مطالعه کنید: « String Formatting in CS » و همچنین

نویسنده: مسعود زیانی
تاریخ: ۱۳۹۱/۰۹/۲۰ ۹:۱

ممنون آقای نصیری

عنوان: **مروری بر کاربردهای Action و Func - قسمت پنجم**
نویسنده: وحید نصیری
تاریخ: ۱۳۹۱/۰۶/۱۰ ۲۱:۴
آدرس: www.dotnettips.info
برچسب‌ها: C#, Refactoring

یکی دیگر از کاربردهای Action و Func، امکان حذف و بازنویسی switch statements بسیار حجیم و طولانی به نحوی شکیل است؛ و در ادامه این نوع Refactoring را بررسی خواهیم کرد.
در ابتدا مثال زیر را که از یک سوئیچ، برای انتخاب نوع حرکت و اعمال آن استفاده می‌کند، در نظر بگیرید:

```
using System;

namespace ActionFuncSamples
{
    public enum Direction
    {
        Up,
        Down,
        Left,
        Right
    }

    public class Sample5
    {
        public void Move(Direction direction)
        {
            switch (direction)
            {
                case Direction.Up:
                    MoveUp();
                    break;

                case Direction.Down:
                    MoveDown();
                    break;

                case Direction.Left:
                    MoveLeft();
                    break;

                case Direction.Right:
                    MoveRight();
                    break;
            }
        }

        private void MoveRight()
        {
            Console.WriteLine("MoveRight");
        }

        private void MoveLeft()
        {
            Console.WriteLine("MoveLeft");
        }

        private void MoveDown()
        {
            Console.WriteLine("MoveDown");
        }

        private void MoveUp()
        {
            Console.WriteLine("MoveUp");
        }
    }
}
```

یک کلاس ساده که بر اساس مقدار enum دریافتی، حرکت به چهار جهت را سبب خواهد شد. اکنون می‌خواهیم با استفاده از Actionها نحوه تعریف متدهای حرکت را به فراخوان واگذار (کلاسی بسته برای تغییر اما باز برای توسعه) و به علاوه switch را نیز کلا حذف کنیم:

```
public interface IMove
```

```
    {
        Dictionary<Direction, Action> MoveMap { get; }
    }

    public class Move : IMove
    {
        public Dictionary<Direction, Action> MoveMap
        {
            get
            {
                return new Dictionary<Direction, Action>
               {
                    { Direction.Up, MoveUp},
                    { Direction.Down, MoveDown},
                    { Direction.Left, MoveLeft},
                    { Direction.Right, MoveRight}
               };
            }
        }

        private void MoveRight()
        {
            Console.WriteLine("MoveRight");
        }

        private void MoveLeft()
        {
            Console.WriteLine("MoveLeft");
        }

        private void MoveDown()
        {
            Console.WriteLine("MoveDown");
        }

        private void MoveUp()
        {
            Console.WriteLine("MoveUp");
        }
    }

    public class Sample5
    {
        public void NewMove(IMove move, Direction dirction)
        {
            foreach (var item in move.MoveMap)
            {
                if (item.Key == dirction)
                {
                    item.Value(); //run action
                    return;
                }
            }
        }
```

توضیحات:

استفاده از Dictionary برای حذف سوئیچ بسیار مرسوم است. خصوصا زمانیکه توسط switch صرفا قرار است یک مقدار ساده بازگشت داده شود. در این حالت می‌توان کل سوئیچ را حذف کرد. caseهای آن تبدیل به keyهای یک Dictionary و مقادیری که باید بازگشت دهد، تبدیل به valueهای متناظر خواهند شد.

اما در اینجا مساله اندکی متفاوت است و خروجی خاصی مد نظر نیست؛ بلکه در هر قسمت قرار است کار مفروضی انجام شود. بنابراین اینبار کلیدهای Dictionary بازهم بر اساس مقادیر caseهای تعریف شده (در اینجا enum ایی به نام Direction) و مقادیر آن نیز Action تعریف خواهند شد. همچنین برای اینکه بتوان امکان تعریف قالبی (کلاسی) را برای تعاریف متدهای متناظر با اعضای enum نیز میسر کرد، این Dictionary را در یک interface قرار داده‌ایم تا کلاس‌های پیاده سازی کننده آن، تعریف متدها را نیز دربر داشته باشند.

همانطور که ملاحظه می‌کنید، اینبار تعاریف متدهای Move از کلاس Sample5 خارج شده‌اند، به علاوه برای دسترسی به آن‌ها نیز switch ایی تعریف نشده و از Action استفاده شده است.

نهایتا اینبار متد جدید Move تعریف شده، با اینترفیس IMove کار می‌کند که یک Dictionary حاوی متدهای متناظر با اعضای enum جهت را ارائه می‌دهد.

یک نکته تکمیلی :

متد NewMove را به شکل‌های زیر نیز می‌توان پیاده سازی کرد:

```
// or ...
move.MoveMap[dirction]();

// or ...
Action action;
if(move.MoveMap.TryGetValue(dirction, out action))
   action();
```

نظرات خوانندگان

نویسنده: ایمان عبیدی
تاریخ: ۱۳۹۱/۰۶/۱۱ ۱۵:۳۸

خیلی ممنون ، من که خیلی دوست دارم این سری مقالات رو و دنبال میکنم.

کد این قسمت رو که برای تمرین خودم انجامش دادم (با یکم تغییرات ناچیز) با اجازه از جناب نصیری اتچش کردم
Sample5.rar

نویسنده: حسین مرادی نیا
تاریخ: ۱۳۹۱/۰۶/۱۱ ۱۷:۲۸

مرسی
مطب جالبی هست و من هم دنبال میکنم.

عنوان: **روش نامگذاری Smurf ایی!**
نویسنده: وحید نصیری
تاریخ: ۱۳۹۱/۰۶/۱۱ ۱۱:۱
آدرس: www.dotnettips.info
برچسب‌ها: C#, Design patterns, Naming

اگر به یک سری از کتابخانه‌ها دقت کنید، تمام کلاس‌های آن‌ها دارای یک پیشوند تکراری هستند؛ مثلا **Smurf** XMLDataRow، **Smurf** XMLElement و الی آخر در مورد تمام کلاس‌های موجود در پروژه. به این رویه « Smurf Naming Convention » گفته می‌شود!
در این نوع کتابخانه‌ها زمانیکه کاربری بر روی دکمه‌ای کلیک می‌کند، **Smurf** AccountView اطلاعات **Smurf** AccountDTO را به **Smurf** AccountController منتقل می‌کند. در ادامه از خاصیت **Smurf** ID دریافتی، مقدار **Smurf** OrderHistory دریافت شده و به **Smurf** HistoryReportingView جهت نمایش ارسال خواهد شد. اگر استثنای **Smurf** ErrorEvent رخ دهد، توسط **Smurf** ErrorLogger در فایلی به نام **smurf** / **smurf** log.log/ ثبت خواهد شد.

کلمه Smurf هم از شخصیتی کارتونی به همین نام اخذ شده است که در زبان مخصوص آن‌ها اکثر افعال و نام‌ها از کلمه Smurf مشتق می‌شود! برای مثال در مورد ماهیگیری کردن در یک رودخانه عنوان می‌کنند «We're going **smurfing** on the River **Smurf** today».

خوب، چکار باید کرد؟ روش صحیح معرفی نام یک شرکت در حین طراحی و نامگذاری کلاس‌های یک کتابخانه چیست؟
در مطلب بسیار جامع و عالی « اصول و قراردادهای نام‌گذاری در دات‌نت » عنوان شده است که اساس نام‌گذاری فضاهای نام باید از قاعده زیر پیروی کند:

```
<Company>.<Technology|Produt|Project>[.<Feature>][.<SubNamespace>]
```

مثلا مایکروسافت یکبار فضای نام Microsoft.Reporting.WebForms را تعریف کرده است و ... همین! دیگر به ابتدای هر کلاسی در این کتابخانه، پیشوند Microsoft یا MS و امثال آن اضافه نشده است تا بر روی اعصاب و روان استفاده کننده تاثیر منفی داشته باشد.

نظرات خوانندگان

نویسنده: رحمت اله رضایی
تاریخ: ۱۳۹۱/۰۶/۱۱ ۱۸:۳۸

مشکل اینجاست که در پروژه ای، کلاسهایی هم اسم کلاسهای دات نت داشته باشیم. همیشه باید فضای نام را در ابتدای کلاسها نوشت.
مثلا فرض کنید کنترلهای TextBox و Button و ... را در یک پروژه وب فرم یا ویندوز فرم سفارشی کرده باشیم. در این حالت چکار باید بکنیم؟

نویسنده: وحید نصیری
تاریخ: ۱۳۹۱/۰۶/۱۱ ۱۹:۲۰

- برنامه FxCop می‌تونه اسمبلی‌های شما رو آنالیز کنه و دقیقا گزارش بده که چه مواردی هم نام کلاس‌های پایه دات نت هستند و بهتر است تغییر نام پیدا کنند. بنابراین به این صورت می‌تونید خیلی سریع حجم بالایی از کدها رو بررسی و رفع اشکال کنید.
- به علاوه زمانیکه طراح شما هستید، محدودیتی در نامگذاری نهایی وجود ندارد. مثلا نام کلاس مشتق شده را NumericTextBox قرار دهید و مواردی مانند این که بیانگر عملکرد سفارشی و ویژه کلاس مشتق شده جدید هستند:

```
public class RequiredTextBox : TextBox
```

عنوان: **عبارت using و نحوه استفاده صحیح از آن**
نویسنده: وحید نصیری
تاریخ: ۱۳۹۱/۰۶/۱۲ ۱۳:۳۱
آدرس: www.dotnettips.info
برچسب‌ها: C#, Refactoring, iTextSharp

مثال ساده زیر را که در مورد تعریف یک کلاس Disposable و سپس استفاده از آن توسط عبارت using است را به همراه سه استثنایی که در این متدها تعریف شده است، در نظر بگیرید:

```
using System;

namespace TestUsing
{
    public class MyResource : IDisposable
    {
        public void DoWork()
        {
            throw new ArgumentException("A");
        }

        public void Dispose()
        {
            throw new ArgumentException("B");
        }
    }

    public static class TestClass
    {
        public static void Test()
        {
            using (MyResource r = new MyResource())
            {
                throw new ArgumentException("C");
                r.DoWork();
            }
        }
    }
}
```

به نظر شما قطعه کد زیر چه عبارتی را نمایش می‌دهد؟ C یا B یا A؟

```
try
{
     TestClass.Test();
}
catch (Exception ex)
{
    Console.WriteLine(ex.Message);
}
```

پاسخ: برخلاف تصور (که احتمالا C است؛ چون قبل از فراخوانی متد DoWork سبب بروز استثناء شده است)، فقط B را در خروجی مشاهده خواهیم کرد!

و این دقیقا مشکلی است که در حین کار با کتابخانه iTextSharp برای اولین بار با آن مواجه شدم. روش استفاده متداول از iTextSharp به نحو زیر است:

```
using (var pdfDoc = new Document(PageSize.A4))
{
   //todo: ...
}
```

در این بین هر استثنایی رخ دهد، در لاگ‌های خطای سیستم شما تنها خطاهای مرتبط با خود iTextSharp را مشاهده خواهید کرد و نه مشکل اصلی را که در کدهای ما وجود داشته است. البته این یک مشکل عمومی است و اگر « using statement and suppressed exceptions » را در گوگل جستجو کنید به نتایج مشابه زیادی خواهید رسید.

و خلاصه نتایج هم این است:

اگر به ثبت جزئیات خطاهای سیستم اهمیت می‌دهید (یکی از مهم‌ترین مزیت‌های دات نت نسبت به بسیاری از فریم ورک‌های مشابه که حداکثر خطای 0xABC12EF را نمایش می‌دهند)، از using استفاده نکنید! using در پشت صحنه به try/finally ترجمه می‌شود و بهتر است این مورد را دستی نوشت تا اینکه کامپایلر اینکار را به صورت خودکار انجام دهد.
در اینجا باز هم به یک سری کد تکراری try/finally خواهیم رسید و همانطور که در مباحث کاربردهای Action و Func در این سایت ذکر شد، می‌توان آن‌را تبدیل به کدهایی با قابلیت استفاده مجدد کرد. یک نمونه از پیاده سازی آن‌را در این سایت « C# Using Blocks can Swallow Exceptions » می‌توانید مشاهده کنید که خلاصه آن کلاس زیر است:

```
using System;

namespace Guard
{
    public static class SafeUsing
    {
        public static void SafeUsingBlock<TDisposable>(this TDisposable disposable, Action<TDisposable>
action)
            where TDisposable : IDisposable
        {
            disposable.SafeUsingBlock(action, d => d);
        }

        internal static void SafeUsingBlock<TDisposable, T>(this TDisposable disposable, Action<T>
action, Func<TDisposable, T> unwrapper)
            where TDisposable : IDisposable
        {
            try
            {
                action(unwrapper(disposable));
            }
            catch (Exception actionException)
            {
                try
                {
                    disposable.Dispose();
                }
                catch (Exception disposeException)
                {
                    throw new AggregateException(actionException, disposeException);
                }

                throw;
            }

            disposable.Dispose();
        }
    }
}
```

برای استفاده از کلاس فوق مثلا در حالت بکارگیری iTextSharp خواهیم داشت:

```
new Document(PageSize.A4).SafeUsingBlock(pdfDoc =>
{
  //todo: ...
});
```

علاوه بر اینکه SafeUsingBlock یک سری از اعمال تکراری را کپسوله می‌کند، از AggregateException نیز استفاده کرده است (معرفی شده در دات نت 4). به این صورت چندین استثنای رخ داده نیز در سطحی بالاتر قابل دریافت و بررسی خواهند بود و استثنایی در این بین از دست نخواهد رفت.

نظرات خوانندگان

نویسنده: بهمن خلفی
تاریخ: ۱۳۹۱/۰۶/۱۳ ۷:۵۹

با سلام خدمت شما جناب نصیری
با توجه به این مطلب که زیاد هم قدیمی نیست به نظر شما استفاده از using پیشنهاد میشود یا استفاده از try catch ؟
چون بنده در برنامه‌های کاربردی تحت وب بیشتر از using استفاده میکنم و از Elmah هم برای ثبت خطاهای سیستم استفاده میکنم .

نویسنده: وحید نصیری
تاریخ: ۱۳۹۱/۰۶/۱۳ ۸:۴۲

مطلب جاری بیشتر به شبیه سازی try/ **finally** معادل using که توسط کامپایلر به صورت خودکار تولید می‌شود مرتبط است نه try/catch کلی. بحث dispose خودکار اشیاء disposable و اینکه استفاده از using به دلیلی که عنوان شد مناسب نیست. بنابراین بجای using از SafeUsingBlock استفاده کنید (شبیه سازی بهتر کاری است که کامپایلر در پشت صحنه جهت معادل سازی یا پیاده سازی using انجام می‌دهد؛ اما بدون از دست رفتن استثناهای رخ داده). مابقی را هم ELMAH انجام می‌دهد.
اگر از using استفاده کنید و ELMAH، فقط خطاهای مرتبط با مثلا iTextSharp رو در لاگ‌ها خواهید یافت؛ مثلا شیء document آن dispose شده، اما خطا و مشکل اصلی که به کدهای ما مرتبط بوده و نه iTextSharp، این میان گم خواهد شد. اما با استفاده از SafeUsingBlock ، دلیل اصلی نیز لاگ می‌شود.

نویسنده: مرادی
تاریخ: ۱۳۹۱/۰۶/۱۳ ۹:۹

البته از حق هم نمی‌شه گذشت که طراح‌های iText Sharp در این جا هم از Best Practiceها پیروی نکردند
این قاعده کلی کار هستش که در Dispose و متد Finalize خطایی ایجاد نشه
حتی چند بار فراخوانی Dispose هم نباید ایجاد خطا کنه، حتی خطای Object Disposed Exception
متاسفانه تفاوت‌های Java و .NET. تونسته رو این تیم که iText Sharp رو از Java به .NET. برده، رو به یک سری اشتباهات بندازه، مثل این مورد، و عدم استفاده از Enum و چند تا مورد دیگه
اگه فرض کنیم ما Dispose رو فراخونی نکنیم و این فراخونی توسط CLR و در فاینالایزر کلاس رخ بده، این خطایی موجود در Dispose می‌تونه مسائل بدتری رو هم در پی داشته باشه
به دوستان توصیه می‌کنم حتما با Best Practiceهای مدیریت خطا مخصوص .NET. آشنا بشن، که در اینترنت منابع زیادی برای این مهم موجوده

نویسنده: افشین
تاریخ: ۱۳۹۱/۰۶/۱۳ ۱۳:۱۰

واقا خسته نباشید.
هر روز بابات این سایت شما رو دعا میکنیم.
ببخشید این موضوع رو این جا مطرح میکنم چون راه دیگه ای پیدا نکردم

نویسنده: سید امیر سجادی
تاریخ: ۱۳۹۲/۰۲/۰۵ ۹:۴۹

سلام. من این مشکلی که گفتید رو توی VB.NET تست کردم مشکلی نداشت. آیا .NET این مشکل رو داره و یا فقط C#?

نویسنده: وحید نصیری

تاریخ: ۱۳۹۲/۰۲/۰۵ ۱۰:۱۴

این رفتار در VB.NET هم قابل مشاهده است:

```
Public Class MyResource
    Implements IDisposable
    Public Sub DoWork()
        Throw New ArgumentException("A")
    End Sub

    Public Overloads Sub Dispose() Implements System.IDisposable.Dispose
        Throw New ArgumentException("B")
    End Sub
End Class

Public NotInheritable Class TestClass
    Private Sub New()
    End Sub
    Public Shared Sub Test()
        Using r As New MyResource()
            Throw New ArgumentException("C")
            r.DoWork()
        End Using
    End Sub
End Class
Module Module1

    Sub Main()
        Try
            TestClass.Test()
        Catch ex As Exception
            Console.WriteLine(ex.Message)
        End Try
    End Sub

End Module
```

عبارت نمایش داده شده در اینجا هم B است.

عنوان: Highlight کردن لینک صفحه جاری در ASP.NET MVC
نویسنده: علیرضا اسم‌رام
تاریخ: ۱۳۹۱/۰۶/۱۹ ۲۲:۵۷
آدرس: www.dotnettips.info
برچسب‌ها: C#, ASP.Net MVC, Helper

راه حل‌های مختلفی جهت Highlight کردن لینک صفحه جاری وجود دارد و مهم‌ترین کاربرد آن در منوی اصلی سایت است.

در این مطلب سعی داریم با ارائه یک Helper راه حل مناسبی را برای این موضوع ارائه کنیم. مسئولیت این Helper ایجاد لینک است با در نظر گرفتن یک شرط: آیا لینک ایجاد شده به Action جاری اشاره دارد؟ اگر بله یک CSS Class با عنوان currentMenuItem به آن اضافه کن.

```
public static MvcHtmlString MenuLink(this HtmlHelper helper, string text, string action, string
controller)
{
    var routeData = helper.ViewContext.RouteData.Values;
    var currentController = routeData["controller"];
    var currentAction = routeData["action"];

    if(String.Equals(action, currentAction as string,
              StringComparison.OrdinalIgnoreCase)
        &&
       String.Equals(controller, currentController as string,
               StringComparison.OrdinalIgnoreCase))

    {
       return helper.ActionLink(
           text,action, controller, null,
           new { @class="currentMenuItem"}
           );
    }
    return helper.ActionLink(text, action, controller);
}
```

نحوه استفاده:

```
<li>@Html.MenuLink("Contact", "Contact", "Home")</li>
```

و البته کمی تغییرات در فایل CSS خود را فراموش نکنید:

```
ul#menu li a {
    background: none;
    color: #999;
    padding: 5px;
    border-radius: 15px;
    text-decoration: none;
}

ul#menu li a.currentMenuItem {
    background-color: black;
    color: white;
}
```

همچنین دوستانی که از Bootstrap و البته Navbar آن استفاده می‌کنند می‌توانند با کمی تغییرات در این Helper استفاده بهینه ای از آن داشته باشند:

```
public static MvcHtmlString MenuLinkBootstrap(this HtmlHelper helper, string text, string action,
string controller)
{

    var routeData = helper.ViewContext.RouteData.Values;
    var currentController = routeData["controller"];
    var currentAction = routeData["action"];

   if (String.Equals(action, currentAction as string, StringComparison.OrdinalIgnoreCase) &&
String.Equals(controller, currentController as string, StringComparison.OrdinalIgnoreCase))
   {
        return new MvcHtmlString("<li class=\"active\">" + helper.ActionLink(text, action, controller)
+ "</li>");
   }

    return new MvcHtmlString("<li>" + helper.ActionLink(text, action, controller) + "</li>");

}
```

نظرات خوانندگان

نویسنده: امین
تاریخ: ۱۳۹۱/۰۶/۲۰ ۹:۲۶

مطلب عالی بود مخصوصا بخش آخر که برای bootstrap رو هم گفتین.

نویسنده: وحید نصیری
تاریخ: ۱۳۹۱/۰۷/۰۶ ۱۲:۳۷

ممنون. مطلب جالبی است. یک راه حل عمومی دیگر مبتنی بر jQuery :

```
//--------------انتخاب خودکار لینک‌های بالای صفحه به ازای صفحه جاری
$(document).ready(function () {
    $("#headermenu a").each(function () {
        var $a = $(this);
        var href = $a.attr("href");
        if (href && (location.pathname.toLowerCase() == href.toLowerCase())) {
            //صفحه جاری را یافتیم
            $a.css({
                "color": "Yellow",
                "border-bottom": "1px solid"
             });
        }
    });
});
```

این روش بر اساس آدرس صفحه جاری و یافتن آن در ناحیه headermenu و سپس رنگی کردن آن کار می‌کند.

نویسنده: امیرحسین مرجانی
تاریخ: ۱۳۹۱/۰۷/۲۷ ۱۶:۲۳

ممنونم
نکته خوبی گفتید ... چندوقتی بود دنبال مطلب شما می‌گشتم که الان به چشمم خورد.

نویسنده: صابر فتح الهی
تاریخ: ۱۳۹۲/۰۱/۰۷ ۱۲:۴۸

سلام مهندس (تبریک سال نو با تاخیر)

روشی که گفتین به نظر من یه ایراد کوچک داره و اونم اینه که در صورتی لینک رنگی می‌شه که حتما آدرس مورد نظر با آدرس لینک برابر باشه، در صورتی که لینک موردنظر یک صفحه فرعی باشه و ان لینک در لیست منو وجود نداشته باشه هیچ لینکی رنگی نمی‌شه، مثلا در سایت دات نت تیپس در صورتی که کاربر روی لینک جزئیات یک پروژه کلیک کنه لینک پروژه‌ها رنگی میشه در صورتی که ادرسش توی منوها موجود نیست، من وقتی کدهای سایت مذکور بررسی کردم فهمیدم به جای اون شما از تابع startsWith (تعریف شده توسط شما) استفاده کردین، در این صورت لینکهایی که با قسمتی از آدرس مذکور شروع بشه تغییرات رنگ (استایل مورد نظر) رو اون اعمال میشه.
اگر اشتباه می‌کنم لطفا تصحیح بفرمایید.

نویسنده: وحید نصیری
تاریخ: ۱۳۹۲/۰۱/۰۷ ۱۲:۵۸

بله. هدف این بوده تا حد ممکن شخص بدونه در چه قسمتی از سایت است. مثلا اگر به جزئیات یک پروژه مراجعه کرده، باید بدونه در قسمت پروژه‌ها هست.
این کد جدید و اصلاح شده است:

```
if (typeof String.prototype.startsWith != 'function') {
    // see below for better implementation!
    String.prototype.startsWith = function (str) {
        return this.indexOf(str) == 0;
    };
}

$(document).ready(function () {
    $("#headermenu a").each(function () {
        var $a = $(this);
        var href = $a.attr("href");

        var path = $.trim(location.pathname.toLowerCase());
        var menuHref = $.trim(href.toLowerCase());

        if (href && (path == menuHref || (menuHref != '/' && path.startsWith(menuHref)))) {
            //صفحه جاری را یافتیم
            $a.css({
                "color": "Yellow",
                "border-bottom": "2px solid"
            });
        }
    });
});
```

نویسنده: مهدی موسوی
تاریخ: ۱۳۹۲/۰۱/۰۸ ۱۵:۱۱

سلام.
به گمانم می‌تونید با یه تغییر کوچک، کدتون رو بهینه کنید. شما می‌تونید پس از یافتن صفحه جاری، return false کنید تا each به سراغ مابقی anchorهای موجود نره (چون لینک مورد نظر پیدا شده و CSS مربوطه هم اعمال شده، بنابراین دیگه نیازی نیست این حلقه به کارش ادامه بده؛ مگر اینکه بگیم بیش از یک لینک در صفحه وجود داره که در اون صورت باید location.pathname رو خارج each یکبار cache کنیم و ...).

البته، می‌تونستیم ابتدا anchorهای مورد نظر رو filter کنیم، بعد با اضافه کردن کلاس (فرضا) highlight همون کارو انجام بدیم:

```
$(function(){
    var loc = location.pathname.toLowerCase();
    $('#headermenu a').filter(function(){
        var href = this.getAttribute('href');
        return href && href.toLowerCase() == loc;
    }).addClass('highlight');
});
```

طبیعتا اگر فرض کنیم که کلیه URLها از سمت سرور بصورت Lowercase رندر میشن، اونوقت می‌تونیم از کد فوق نیز پرهیز کنیم:

```
$('#headermenu a[href="' + location.pathname + '"]').addClass('highlight');
```

موفق باشید.

پ.ن.: البته میتونیم از is هم استفاده کنیم که مقایسه رشته‌ها بطور خودکار case-insensitive نیز باشه.

نویسنده: Mehdi
تاریخ: ۱۳۹۲/۰۳/۱۸ ۰:۳۳

منظور از ناحیه headermenu چیست؟

نویسنده: وحید نصیری
تاریخ: ۱۳۹۲/۰۳/۱۹ ۱۷:۱۴

یک مثال فرضی بود. مثلا یک div با id مساوی headermenu.

عنوان: **توابع تعمیم یافته در #C**
نویسنده: سیدمجتبی حسینی
تاریخ: ۱۳۹۱/۰۷/۰۲ ۱:۳
آدرس: www.dotnettips.info
برچسب‌ها: #C

از توابع تعمیم یافته می‌توان برای توسعه توابع هر کلاس یا اینترفیسی استفاده کرد. یعنی می‌توان یک تابع را به هر کلاسی اضافه کرد.

قبل از C# 3.0 فقط می‌شد یک کلاس را از طریق ارث‌بری از آن توسعه داد و به کلاس مهروموم شده یا Sealed نیز نمی‌شد تابعی افزود که البته ممکن است بگویید که این کار قوانین شیءگرایی را نقض می‌کند اما در پاسخ باید گفت که توابع تعمیم یافته به اعضاء خصوصی کلاسی که تعمیم می‌یابد، دسترسی ندارند.

تعمیم یک کلاس در خارج از بدنه کلاس انجام می‌شود و این کار می‌تواند در فضای نام همان کلاس یا خارج از آن انجام شود و شکل کلی آن به صورت زیر است :

```
public static class  ExtendingClassName
 {
      public static ReturnType MethodName(this   ExtendedMethod  arg)
      {
           //دستورات درون متد
          Return ReturnType;
      }
 }
```

توجه کنید که:
کلاس توسعه‌دهنده و تابع توسه‌دهنده باید استاتیک باشند.
در داخل آرگومان تابع، کلمه کلیدی this استفاده می‌شود.
بعد از this عنوان کلاسی که قصد توسعه آن را داریم، ذکر می‌کنیم.
در هرجا که خواستیم از قابلیت تعمیم داده شده استفاده کنیم باید فضای نام مربوط به آن را ذکر کنیم.
با کلمه کلیدی static نمی‌توان کلاسی با متدهای abstract ، virtual و override را توسعه داد.

مثلا اگر قصد داریم به کلاس String تابع AddPrefix را اضافه کنیم به این شکل عمل می‌کنیم :

```
public static class  ExtendingString
{
  public static  string  AddPrefix(this   string  arg)
  {
     return String.Format("prefix{0}",arg);
  }
}
```

که نحوه استفاده از آن به شکل زیر است:

```
string  s = "Student";
Console.WriteLine(s.AddPrefix());
```

و خروجی آن نمایش prefixStudent است.

اگر بخواهیم عبارت پیشوند را از طریق آرگومان ارسال کنیم به این شکل عمل می‌کنیم:

```
public static class ExtendingString
{
```

```
    public static string AddPrefix(this   string arg, string prefix)
    {
         return String.Format("{0}{1}", prefix, arg);
     }
}
```

که نحوه استفاده از آن به شکل زیر است:

```
var  s = "Student";
Console.WriteLine(s.AddPrefix("tbl"));
```

و خروجی آن نمایش tblStudent است.

به عنوان مثال دوم کلاس زیر را در نظر بگیرید:

```
public class Test
{
     public int AddOne(int val)
     {
         return val + 1;
     }
  }
```

اگر بخواهیم کلاس فوق را توسعه داده و متد دیگری مثلا با عنوان AddTwo اضافه کنیم، کلاس توسعه دهنده را به این شکل می‌نویسیم:

```
public static class ExtendingTest
{
     public static int AddTwo(this   Test  arg, int val)
     {
         return val + 2;
     }
}
```

و روش استفاده از آن بصورت زیر است:

```
static void Main(string[] args)
{
       var x = new Test();
       Console.WriteLine(x.AddOne(10));
       Console.WriteLine(x.AddTwo(10));
       Console.Read();
}
```

نظرات خوانندگان

نویسنده: محسن
تاریخ: ۱۳۹۱/۰۷/۰۲ ۹:۲۸

بسیار عالی بیان کردید.ممنون از شما.
موفق باشید.

نویسنده: سعید
تاریخ: ۱۳۹۱/۰۷/۰۲ ۱۶:۲۰

من ترجمه extension methods رو «متدهای الحاقی» هم دیدم.

نویسنده: محمد صافدل
تاریخ: ۱۳۹۱/۰۷/۰۳ ۱۳:۳۳

ممنون از مقاله خوبتون.
به نظر من هم متدهای الحاقی ترجمه صحیح Extension Method است. اول اینکه متد با تابع فرق دارد دوم اینکه تعمیم به معنای گسترش چیزی است و توابع تعمیم یافته یعنی اینکه توابع موجود را گسترش میدید در صورتی که در Extension Methodها چیزی که تعمیم می‌یابد در اصل کلاسها هستند نه متدها که این تعمیم کلاسها با الحاق متدهای جدید انجام میشود

نویسنده: علی
تاریخ: ۱۳۹۱/۰۷/۰۳ ۲۰:۲۹

البته توسعه تعداد توابع نیز به نوعی توسعه توابع است و الزامی ندارد فقط یک تابع توسعه یابد تا تعمیم و توسعه صدق کند

البته الحاق هم صحیح است .

نکته دیگر آنکه برای متد و فانکشن و مانند آن در فارسی معادلی غیر تابع ندیده‌ام که متداول باشد. بعلاوه آنکه تابع مفهوم همه اینها را شامل می‌شود خواه نوع بازگشتی داشته باشد یا void باشد، لذا تعبیر غیر صحیحی نیست

باتشکر

نویسنده: محمد صافدل
تاریخ: ۱۳۹۱/۰۷/۰۴ ۹:۴۳

بله با شما موافقم که تابع مفهوم همه اینها را شامل می‌شود اما در اینجا دقیقا "متد" مد نظر است. مفهوم "تابع" همه رویه هایی که متعلق به کلاس باشند یا نباشند را در بر میگیرد در صورتی که "متد" قطعا متعلق به یک کلاس است. با توجه به ذکر صریح کلمه Method در Extension Method و مفهوم آن به نظر من استفاده از متد صحیح است.
در فارسی مدتی کلمه "روش" به جای "متد" در ترجمه‌ها رایج شد که زیاد جا نیفتاد و بیشتر مترجمین جدیدا از همون کلمه "متد" استفاده می‌کنن

عنوان: **تفاوت انواع var و dynamic**
نویسنده: سیدمجتبی حسینی
تاریخ: ۱۳۹۱/۰۷/۱۱ ۲۲:۳۰
آدرس: www.dotnettips.info
برچسب‌ها: #C

در ابتدا به توضیحاتی درباره کنترل نوع به صورت ایستا و کنترل نوع در زمان اجرا، توجه کنید:
کنترل نوع ایستا (Static Type Checking) کامپایلر را قادر به بررسی درستی برنامه می‌کند، بدون آنکه آن را اجرا کند. مثلاً کد زیر با خطا مواجه می‌شود:

```
int x = "5";
```

کنترل نوع در زمان اجرا (RunTime Type Checking)، هنگامی که برنامه اجرا می‌شود این کنترل توسط CLR صورت می‌گیرد و موقع تایپ کد، خطایی گرفته نمی‌شود مثلاً:

```
object y = "5";
int z = (int) y; //downcast خطای زمان اجرا و شکست
```

downcast به عمل تبدیل نوع کلاس پایه به یکی از کلاس‌های مشتق شده، گفته می‌شود.
پس از ذکر مقدمه بالا به این سؤال می‌پردازیم که تفاوت انواع var و dynamic چیست؟
کلمه کلیدی var و کلمه کلیدی dynamic علی‌الظاهر کاربرد یکسانی دارند اما تفاوت اساسی آن‌ها عبارت است از این‌که نوع واقعی متغیرهایی از نوع var ، توسط کامپایلر تعیین می‌شود. یعنی متغیرهایی از نوع var کنترل نوعشان به صورت ایستاست اما نوع واقعی متغیرهایی از نوع dynamic ، در زمان اجرا مشخص می‌شود. یعنی متغیرهایی از نوع dynamic کنترل نوعشان به صورت کنترل نوع در زمان اجراست. به کد زیر توجه کنید:

```
dynamic x = "hello"; // این متغیر در زمان کامپایل از نوع دینامیک است و در زمان اجرا از نوع رشته‌ای است
var y = "hello"; //این متغیر هم در زمان کامپایل و هم در زمان اجرا از نوع رشته‌ای است
int i = x; // خطای زمان اجرا
int j = y; //خطای زمان کامپایل
```

همچنین متغیری که نوعش var است می‌تواند مقداری از نوع dynamic را شامل شود.

```
dynamic x = "hello";
var y = x; // این متغیر در زمان کامپایل دینامیک است
int z = y; // خطای زمان اجرا
```

نظرات خوانندگان

نویسنده: محسن
تاریخ: ۱۳۹۱/۰۷/۱۲ ۸:۳۱

تفاوت مهم دیگه‌ای که بین این دو وجود دارد این است که می‌توان نوع تعریف شده توسط dynamic را در حین کدنویسی تغییر داد اما اینکار برای var امکان پذیر نیست(با انتساب اول نوع برای متغیر درنظر گرفته می‌شود.برای مثال:

```
int k=100;
var data1=k;
dynamic data2=k;
data1="Hello World!"// خطای حین کامپایل به دلیل تغییر نوع
data2="Hello World!"//بدون خطا در زمان کامپایل و یا اجرا
```

نویسنده: علیرضا صالحی
تاریخ: ۱۳۹۱/۰۷/۱۲ ۱۰:۳۲

کارکرد کلمه کلیدی var تحت عنوان Type Inference (استنتاج نوع) شناخته می‌شود. این اصطلاح به این معنی است که در زمان کامپایل نوع متغییر قابل تشخیص است و صرفا یک راهکار برای سادتر شدن کار برنامه نویس است، در عمل و از دید کامپایلر این همان تعریف متغییر معمولی (statically typed) است.

Type inference

از آنجایی که DLR بر روی CLR پیاده شده است dynamic در واقع خودش استاتیک است! اما قابلیت‌های دینامیک بودن را ارائه می‌کند.

statically typed dynamic

نویسنده: وحید نصیری
تاریخ: ۱۳۹۱/۰۷/۱۲ ۱۲:۲۷

DLR از CLR که مساوی (GC/JIT/sandboxed security model/etc) هست استفاده می‌کنه اما فقط از سرویس‌های آن و نه اینکه ذاتا و نهایتا استاتیک کار کند.

DLR یک opcode جدید را به این مجموعه اضافه می‌کند به نام InvokeDynamic که پیشتر در CLR موجود نبوده .

نویسنده: علیرضا صالحی
تاریخ: ۱۳۹۱/۰۷/۱۲ ۱۶:۵۲

بله اینها صحیح، ولی statically typed dynamic عبارتی است که Anders Hejlsberg هنگام صحبت در مورد آینده C# در PDC09 به کاربرد.

#The Future of C

The dynamic keyword acts as a static type declaration in the C# type system. This way C# got the dynamic features and at the same time remained a statically typed language.

http://msdn.microsoft.com/en-us/magazine/gg598922.aspx

در واقع کلمه کلیدی dynamic به کامپایلر می‌فهماند که compile-time checking را غیر فعال کن! تا در زمان اجرا به نوع متغییر رسیدگی شود.

شاید اگر بگوییم dynamic نوعی static است که مزایای انواع dynamic را در بر می‌گیرد بهتر باشد.

خواندن این مقاله هم خالی از لطف نیست:

Static Typing Where Possible, Dynamic Typing When Needed: The End of the Cold War Between Programming Languages

عنوان: **Serialization #1**
نویسنده: سیدمجتبی حسینی
تاریخ: ۱۳۹۱/۰۷/۱۳ ۲:۴۲
آدرس: www.dotnettips.info
برچسب‌ها: C#, Serialization

در این نوشتار به Serialization و Deserialization یعنی مکانیزمی که توسط آن اشیاء می‌توانند به صورت متنی مسطح و یا به شکل باینری درآیند، پرداخته می‌شود.

مفهوم Serialization

سریالی کردن، عملیاتی است که یک شیء و یا مجموعه اشیائی که به یکدیگر ارجاع می‌دهند را به شکل گروهی از بایت‌ها یا فرمت XML درآورده که می‌توان آن‌ها را ذخیره کرد و یا انتقال داد.
Deserialization معکوس عملیات بالاست که گروهی از داده‌ها را دریافت کرده و بصورت یک شیء و یا مجموعه‌ای از اشیائی که به یکدیگر ارجاع می‌دهند، تبدیل می‌کند.
Serialization و Deserialization نوعاً برای موارد زیر بکار می‌روند:
انتقال اشیاء از طریق یک شبکه یا مرز یک نرم افزار .

ذخیره اشیاء در یک فایل یا بانک اطلاعاتی.

موتورهای سریالی‌کننده

چهار شیوه برای سریالی کردن در دات نت فریم ورک وجود دارد:
سریالی‌کننده قرارداد داده (Data Contract Serializer)

سریالی‌کننده باینری

سریالی‌کننده XML مبتنی بر صفت

سریالی‌کننده اینترفیس IXmlSerializer

سه مورد اول، موتورهای سریالی‌کننده‌ای هستند که بیشترین یا تقریباً همه کارهای سریالی‌کردن را انجام می‌دهند. مورد آخر برای انجام مواردی است که خودتان قصد سریالی‌سازی دارید که این موتور با استفاده از XmlReader و XmlWriter این کار را انجام می‌دهد. IXmlSerializer می‌تواند به همراه سریالی‌کننده قرارداد داده و یا سریالی‌کننده XML در موارد پیچیده سریالی‌کردن استفاده شود.

سریالی‌کننده Data Contract

برای انجام این کار دو نوع سریالی‌کننده وجود دارد :
DataContractSerializer، که اصطلاحاً loosely Coupled شده است به نوع سریالی‌کننده Data Contract.

NetDataContractSerializer که اصطلاحاً tightly Coupled شده است به نوع سریالی‌کننده Data Contract.

مدل زیر را در نظر بگیرید:

```
public class News
    {
        public int Id;
```

```
        public string Body;
        public DateTime NewsDate;
    }
```

برای سریالی‌کردن نوع News به شیوه Data Contract، باید:

فضای نام System.Runtime.Serialization را به کد برنامه اضافه کنیم.

صفت [DataContract] را به نوعی که تعریف کرده‌ایم، اضافه کنیم.

صفت [DataMember] را به اعضایی که می‌خواهیم در سریالی شدن شرکت‌کنند، اضافه کنیم.

و به این ترتیب کلاس News به شکل زیر درمی‌آید:

```
using System.Runtime.Serialization;
using System.Xml;
using System;
using System.IO;
namespace ConsoleApplication1
{
    [DataContract]
    public class News
    {
        [DataMember] public int Id;
        [DataMember] public string Body;
        [DataMember] public DateTime NewsDate;
    }
}
```

سپس به شکل زیر از مدل خود نمونه‌ای ساخته و با ایجاد یک فایل، نتیجه سریالی شده مدل را در آن ذخیره میکنیم .

```
var news = new News
                            {
                                Id = 1,
                                Body = "NewsBody",
                                NewsDate = new DateTime(2012, 10, 4)
                            };
 var ds = new DataContractSerializer(typeof (News));
 using (Stream s=File.Create("News.Xml"))
 {
        ds.WriteObject(s, news);//سریالی‌کردن
 }
```

که محتویات فایل News.Xml به صورت زیر است:

```
<News xmlns="http://schemas.datacontract.org/2004/07/ConsoleApplication7"
xmlns:i="http://www.w3.org/2001/XMLSchema-instance"><Body>NewsBody</Body><Id>1</Id><NewsDate>2012-10-
04T00:00:00</NewsDate></News>
```

و برای Deserialize کردن این فایل داریم:

```
News deserializednews;
using (Stream s = File.OpenRead("News.Xml"))
{
       deserializednews = (News)ds.ReadObject(s);//Deserializing
}
Console.WriteLine(deserializednews.Body);
```

همان‌طور که ملاحظه می‌کنید فایل ایجاد شده از خوانایی خوبی برخوردار نیست که برای دستیابی به فایلی با خوانایی بالاتر از

XmlWriter استفاده میکنیم:

```
XmlWriterSettings settings = new XmlWriterSettings {Indent = true};
using (XmlWriter writer = XmlWriter.Create("News.Xml", settings))
{
          ds.WriteObject(writer, news);
}
System.Diagnostics.Process.Start("News.Xml");
```

به این ترتیب موفق به ایجاد فایلی با خوانایی بالاتر می‌شویم:

```
<?xml version="1.0" encoding="utf-8"?>
<News xmlns:i="http://www.w3.org/2001/XMLSchema-instance"
xmlns="http://schemas.datacontract.org/2004/07/ConsoleApplication7">
  <Body>NewsBody</Body>
  <Id>1</Id>
  <NewsDate>2012-10-04T00:00:00</NewsDate>
</News>
```

نظرات خوانندگان

نویسنده: محمد
تاریخ: ۱۳۹۱/۰۷/۱۳ ۱۱:۴۲

دست شما درد نکند خیلی اینا رو میدیدم ولی نمیفهمیدم چین.
ممنون که روشنمون کردی

عنوان: **Serialization #2**
نویسنده: سیدمجتبی حسینی
تاریخ: ۱۳۹۱/۰۷/۱۵ ۲۳:۱۷
آدرس: www.dotnettips.info
برچسب‌ها: C#, Serialization

مطابق آنچه در قسمت قبل گفته شد برای آن‌که بتوان از مدل News برای سریالی‌کردن استفاده کرد، باید آن را به شکل ذیل پیاده‌سازی کرد:

```
[DataContract]
public class News
{
     [DataMember] public int Id;
     [DataMember] public string Body;
     [DataMember] public DateTime NewsDate;
}
```

با Override کردن [DataContract] به صورت [DataContract(Name="MyCustomNews")] می‌توان نام ریشه XML فایل را به MyCustomNews تغییر داد. همچنین با Override کردن [DataMember] بصورت [DataMember(Name="MyCustomFieldName")] می‌شود به هر فیلدی عنوان دلخواهی داد و همچنین با تعیین عبارت NameSpace به صورت [DataContract(Name = "MyCustomNews", Namespace = "http://www.my.com") می‌شود فضای نام را تغییر داد که با این تغییرات، خروجی زیر حاصل می‌شود:

```
<?xml version="1.0" encoding="utf-8"?>
<MyCustomNews xmlns:i="http://www.w3.org/2001/XMLSchema-instance" xmlns="http://www.my.com">
  <Body>NewsBody</Body>
  <MyCustomFieldName>111</MyCustomFieldName>
  <NewsDate>2012-10-04T00:00:00</NewsDate>
</MyCustomNews>
```

ویژگی [DataMember] هم ازفیلد ها و هم از property ها، پشتیبانی می‌کند، خواه عمومی باشند یا خصوصی و نوع فیلد یا Property می‌تواند به یکی از اشکال زیر باشد:

انواع اولیه .

انواع Uri ، Guid ، TimeSpan ، DateTime و انواع Enum

انواع پوچ پذیر هر کدام از موارد بالا

نوع byte[]

انواع تعریف شده توسط کاربر که توسط صفت [DataContract] محصور شده‌اند.

هر نوع IEnumerable

هر نوعی که با صفت [Serializable] محصور شود و یا اینترفیس ISerializable را پیاده سازی کند.

هر نوعی که اینترفیس IXmlSerializble را پیاده سازی نماید.

تعیین فرمت باینری برای سریالی‌کردن:

برای سریالی کننده‌های DataContractSerializer و NetDataContractSerializer می‌توان به روش زیر فرمت خروجی را به شکل فرمت باینری درآورد که خروجی آن تاحد زیادی کوچک‌تر و کم حجم‌تر می‌شود:

```
var s = new MemoryStream();
using (XmlDictionaryWriter w=XmlDictionaryWriter.CreateBinaryWriter(s))
{
        ds.WriteObject(w,news);
}
```

و برای Deserialize کردن آن به شیوه زیر عمل می‌کنیم:

```
var s2 = new MemoryStream(s.ToArray());
News deserializednews;
using (XmlDictionaryReader r=XmlDictionaryReader.CreateBinaryReader(s2,XmlDictionaryReaderQuotas.Max))
{
     deserializednews = (News)ds.ReadObject(r);
}
```

که در آن از ویژگی Max کلاس XmlDictionaryReaderQuotas برای به دست آوردن حداکثر سهمیه فضای دیسک مربوط به XmlDictionaryReaders استفاده می‌شود.

نظرات خوانندگان

نویسنده: وحید نصیری
تاریخ: ۱۳۹۱/۰۷/۱۶ ۱:۸

با تشکر. یک کتابخانه XML Serialization هم توسط آقای سینا ایروانیان تهیه شده که یک سری از محدودیت‌های کتابخانه‌های توکار دات نت را برطرف کرده.

عنوان: **اجرای متد از طریق Reflection**
نویسنده: سیدمجتبی حسینی
تاریخ: ۱۳۹۱/۰۷/۱۷ ۱۴:۰
آدرس: www.dotnettips.info
برچسب‌ها: C#, Reflection

برای اجرای متد درون یک کلاس از طریق Reflection ابتدا نوع آن کلاس را به دست می‌آوریم و سپس از طریق کلاس Activator.CreateInstance یک نمونه از آن کلاس را ساخته و در متغیری از نوع object ذخیره کرده و با استفاده از GetMethod اطلاعات متد مورد نظر خود را در متغیری ذخیره کرده و سپس از طریق دستور Invoke آن متد را اجرا می‌کنیم. دستور Invoke دو سربارگذاری دارد که در یک نوع از آن، متغیر حاوی نمونه کلاس و پارامترهای متد مورد نظر، در قالب یک آرایه از نوع object، به عنوان آرگومان پذیرفته می‌شود. با امضای زیر

```
public Object Invoke(Object obj, Object[] parameters)
```

به مثال زیر که چگونگی این عملیات را شرح می‌دهد، توجه کنید:

```
public class TestMath
    {
        public int Squar(int i)
        {
            return i*i;
        }
    }

static void Main(string[] args)
        {
            Type type = typeof (TestMath);//به دست آوردن نوع کلاس
            object obj = Activator.CreateInstance(type);//ساختن نمونه‌ای از نوع مورد نظر
            MethodInfo methodInfo = type.GetMethod("Squar");//یافتن اطلاعات متد مورد نظر
            Console.WriteLine(methodInfo.Invoke(obj, new object[] { 100 }));// و نمایش نتیجه object[]
ارسال عدد 100 به صورت
            Console.Read();
        }
```

توجه کنید که دو متد GetMethod و Invoke در فضای نام System.Reflection قرار دارند.

روش دیگر

در شیوه دیگر برای انجام این کار، نیازی به استفاده از GetMethod و Invoke نیست و فراخوانی متد مورد نظر بسیار شبیه فراخوانی عادی متدهاست و نیازی به ساخت متغیر ویژه‌ای از نوع object[] برای ارسال پارامترها نیست. برای انجام این کار فقط کافیست نوع متغیری که نوع نمونه‌سازی شده را نگه‌می‌دارد (در اینجا نمونه ای از کلاس را نگه‌می‌دارد) به صورت dynamic باشد:

```
static void Main(string[] args)
        {
            Type type = typeof (TestMath);
            dynamic obj = Activator.CreateInstance(type);
            Console.WriteLine(obj.Square(100));
            Console.Read();
        }
```

توجه کنید که بعد از تعریف obj، با درج نقطه در کنار آن، منوی Code Insight متد Square را شامل نمی‌شود اما کامپایلر آن را می‌پذیرد.

نظرات خوانندگان

نویسنده: KishIsland
تاریخ: ۱۳۹۱/۱۲/۱۵ ۱۹:۴۷

برای تست کد بالا من یک کلاس بشکل زیر تعریف کردم:

```
class Test
    {
        public void func1()
        {
            Console.WriteLine("Hello World!");
        }
    }
```

و در قسمت main برنامه فراخوانی رو بشکل زیر نوشتم مطابق روش دوم ولی برنامه Exception داد:

```
Type type = typeof(Test);
            dynamic obj = Activator.CreateInstance(type);
            Console.WriteLine(obj.func1());
```

علت بروز استثناء void بودن متد هست.می خواستم بدونم برای مواقعی که متد بصورت void تعریف شده چکار باید کرد؟
سپاس

نویسنده: وحید نصیری
تاریخ: ۱۳۹۱/۱۲/۱۵ ۲۱:۲۰

در این حالت بجای

```
Console.WriteLine(obj.func1());
```

فقط کافی هست بنویسید

```
obj.func1();
```

عنوان: **به دست آوردن اطلاعات کد اجراکننده یک متد**
نویسنده: سیدمجتبی حسینی
تاریخ: ۱۳۹۱/۰۷/۱۸ ۰:۷
آدرس: www.dotnettips.info
برچسب‌ها: C#, Attributes

در C# 5 به بعد می‌توان به پارامترهای یک متد، پارامترهای دلخواهی را افزود تا به واسطه آن‌ها مشخصات کدی که این متد را فراخوانده، به دست آورد. روش انجام این کار، افزودن صفات زیر به پارامترهای متد مورد نظر است:
[CallerFilePath]:مسیر کد فراخواننده را نگه می‌دارد.

[CallerLineNumber]: شماره خط کد فراخواننده را نگه می‌دارد.

[CallerMemberName] : نام کد فراخوان را نگه می‌دارد .

این صفات کامپایلر را قادر می‌سازد که اطلاعاتی درباره فراخواننده متد مورد نظر، جمع‌آوری کند
مثال زیر را در نظر بگیرید:

```
using System;
using System.Runtime.CompilerServices;
namespace ConsoleApplication8
{
  class Program
    {
        static void Main(string[] args)
        {
            Test();
            Console.Read();
        }
        static void Test(
                [CallerMemberName] string memberName = null,
                [CallerFilePath] string filePath = null,
                [CallerLineNumber] int lineNumber = 0)
        {
            Console.WriteLine(memberName);
            Console.WriteLine(filePath);
            Console.WriteLine(lineNumber);
        }
    }
}
```

که نتیجه اجرای کد فوق به صورت زیر است:

```
Main
c:\Pojects\ConsoleApplication8\Program.cs
9
```

که عبارت Main عنوان متدی است که محل فراخوانی متد مورد نظر ماست و خط دوم حاوی مسیری است که کد فراخواننده متد مورد نظر ما درآن‌جا ذخیره شده است و عدد 9 نشانگر شماره خط محل فراخوانی متد Test است.

عنوان: **محاسبه ی اختلاف زمان رخدادی در گذشته با زمان فعلی به فارسی**
نویسنده: شاهین کیاست
تاریخ: ۱۳۹۱/۰۷/۲۵ ۱۴:۴۹
آدرس: www.dotnettips.info
گروه‌ها: #C

حتما در سایت جاری مشاهده کرده اید در اطلاعات مربوط به پست‌ها زمان تقریبی انتشار پست درج شده است.
12 ساعت قبل
دیروز
لحظاتی پیش
...

نشان دادن همچین اطلاعاتی در برنامه‌های مختلف می‌تواند سودمند باشد ، مثلا در این سایت اگر مطلبی مربوط به گذشته باشد خواننده با دیدن عبارت 4 سال قبل از پرسیدن یک سری سوالات خودداری می‌کند.
آقای Jeff Awtood یکی از خالقین سایت Stackoverflow زمانی سوالی درباره‌ی نحوه‌ی پیاده سازی این ویژگی پرسیده بودند که در نهایت یکی از پاسخ ها پذیرفته شد.
یک مثال از نحوه‌ی پیاده سازی این ویژگی برای زبان فارسی مانند زیر است :

```
public class RelativeTimeCalculator
{
    const int SECOND = 1;
    const int MINUTE = 60 * SECOND;
    const int HOUR = 60 * MINUTE;
    const int DAY = 24 * HOUR;
    const int MONTH = 30 * DAY;

    public static string Calculate(DateTime dateTime)
    {
        var ts = new TimeSpan(DateTime.Now.Ticks - dateTime.Ticks);
        double delta = Math.Abs(ts.TotalSeconds);
        if (delta < 1 * MINUTE)
        {
            return ts.Seconds == 1 ? "لحظه ای قبل" : ts.Seconds + " ثانیه قبل";
        }
        if (delta < 2 * MINUTE)
        {
            return "یک دقیقه قبل";
        }
        if (delta < 45 * MINUTE)
        {
            return ts.Minutes + " دقیقه قبل";
        }
        if (delta < 90 * MINUTE)
        {
            return "یک ساعت قبل";
        }
        if (delta < 24 * HOUR)
        {
            return ts.Hours + " ساعت قبل";
        }
        if (delta < 48 * HOUR)
        {
            return "دیروز";
        }
        if (delta < 30 * DAY)
        {
            return ts.Days + " روز قبل";
        }
        if (delta < 12 * MONTH)
```

```
        {
            int months = Convert.ToInt32(Math.Floor((double)ts.Days / 30));
            return months <= 1 ? "یک ماه قبل" : months + " ماه قبل";
        }
        int years = Convert.ToInt32(Math.Floor((double)ts.Days / 365));
        return years <= 1 ? "یک سال قبل" : years + " سال قبل";
    }

}
```

نحوه‌ی کارکرد کد اینگونه است که دلتای زمان داده شده به متد Calculate با زمان فعلی بر حسب ثانیه محاسبه می‌گردد و با یک سری شرط مقایسه می‌شود ، مثلا اگر دلتا کمتر از 120 ثانیه بود رشته‌ی یک دقیقه قبل باز می‌گردد.

یک مثال از نحوه‌ی استفاده از این کلاس اینگونه است :

```
var relativeTime=RelativeTimeCalculator.Calculate(DateTime.Now.AddMinutes(-10));
```

نظرات خوانندگان

نویسنده: محسن.د
تاریخ: ۱۳۹۱/۰۷/۲۵ ۱۵:۳۲

سلام
این نحوه نمایش تاریخ کار جالبیه . من هم این ویژگی رو (البته به یک روش دیگه) پ پاده سازی کردم . البته فکر کنم نسبت به روش ذکر شده در اینجا سنگین‌تر باشه و سربار بیشتری داره .

نویسنده: شهروز
تاریخ: ۱۳۹۱/۰۷/۲۵ ۱۶:۴

با سلام و خسته نباشید
می خواستم بدونم کاربرد اون constها بالای برنامه چیه ؟

نویسنده: وحید نصیری
تاریخ: ۱۳۹۱/۰۷/۲۵ ۱۶:۱۴

مراجعه کنید به مطلب « زیباتر کد بنویسیم » شماره 9

نویسنده: حسین مرادی نیا
تاریخ: ۱۳۹۱/۰۷/۲۵ ۲۰:۳۰

سلام
مطلب بسیار خوبی بود

const هایی که در بالا تعریف شدن درون کد مورد استفاده قرار نگرفتن. فکر میکنم کد یه مقداری باید اصلاح بشه.

نویسنده: محمد باقر سیف اللهی
تاریخ: ۱۳۹۱/۰۷/۲۵ ۲۲:۴۵

به صورت Client-Side هم می‌شود این موارد را پیاده کرد (که در فروم به آن اشاره شد) : http://momentjs.com
http://timeago.yarp.com
(فارسی)

نویسنده: شاهین کیاست
تاریخ: ۱۳۹۱/۰۷/۲۵ ۲۳:۷

ممنون ،
کد را اصلاح کردم.

نویسنده: reza
تاریخ: ۱۳۹۱/۰۷/۲۶ ۱:۴۰

سلام
من توی پی اچ پی یه همچین تابعی نوشتم

```
/**
* Time Left function
*
* Example
```

```
* $x = 1332140945 ;
* echo time_left($x);
* @param $ts int timestamp's post
* @return string time left like 3mahe ghabl
*/
function time_left($ts = null)
{
if(!$ts)
return '';


$time = time();
$t = $time-$ts;

if(intval($t) < 0)
return 'آینده';

if(floor($t/31536000) >= 1 )
$out = floor($t/31536000).' سال قبل';
elseif(floor($t/2592000) >= 1)
$out = floor($t/2592000).' ماه قبل';
elseif(floor($t/604800) >= 1)
$out = floor($t/604800).' هفته قبل';
elseif(floor($t/86400) >= 1)
$out = floor($t/86400).' روز قبل';
elseif(floor($t/3600) >=1)
$out = floor($t/3600).' ساعت پیش';
elseif(floor($t/60) >= 1)
$out = floor($t/60).' دقیقه پیش';
else
$out = $t.' ثانیه قبل';
return $out;
}
```

فقط واسه بررسی گذاشتم .
موفق باشید

نویسنده: شاهین کیاست
تاریخ: ۱۳۹۱/۰۷/۲۶ ۸:۴۱

سلام ، معادل کد پست جاری به زبان PHP

نویسنده: sorosh
تاریخ: ۱۳۹۲/۱۱/۱۸ ۸:۱۷

با سلام و عرض ادب
بنده نیاز به یک تابع جاوا اسکریپتی ترجیحاً با Jquery دارم که بتونه تاریخ شمسی مثلا 11/01/1392 را بگیره و بگه که تا امروز چند سال و چند ماه گذشته. یعنی یک جور محاسبه سایقه کار برحسب سال و ماه را می‌خواهم روز و ساعتش مهم نیست. با تشکر

نویسنده: وحید نصیری
تاریخ: ۱۳۹۲/۱۱/۱۸ ۱۰:۲۳

در همان لینک آخری که آقای کیاست معرفی کردند، پیاده سازی date_diff به زبان‌های دیگر منجمله جاوا اسکریپت نیز هست.
برای تبدیل شمسی به میلادی هم از کتابخانه moment مثلا کمک بگیرید.

عنوان: **اجرای Stored Procedure با چند نوع مقدار برگشتی توسط EF CodeFirst**
نویسنده: حسین مرادی نیا
تاریخ: ۱۳۹۱/۰۷/۲۵ ۱۶:۲۹
آدرس: www.dotnettips.info
گروه‌ها: C#, Entity framework, EF Code First, SQL Server

فرض کنید Stored Procedure ی با چند مقدار برگشتی را می‌خواهیم در EF CodeFirst مورد استفاده قرار دهیم. برای مثال Stored Procedure زیر را در نظر بگیرید:

```
CREATE PROCEDURE [dbo].[GetAllBlogsAndPosts]
AS
    SELECT * FROM dbo.Blogs
    SELECT * FROM dbo.Posts
```

Stord Procedure ی که توسط این دستور ساخته می‌شود تمام رکوردهای جدول Blogs و تمامی رکوردهای جدول Posts را واکشی کرده و به عنوان خروجی برمیگرداند (دو خروجی متفاوت). روش فراخوانی و استفاده از داده‌های این StoredProcedure در EF CodeFirst به صورت زیر است :

تعریف دو کلاس مدل Blog و Post به ترتیب برای نگهداری اطلاعات وبلاگ‌ها و پست‌ها در زیر آمده است. در ادامه نیز تعریف کلاس BloggingContext را مشاهده می‌کنید.

```
public class Blog
    {
        public int BlogId { get; set; }
        public string Name { get; set; }

        public virtual List<Post> Posts { get; set; }
    }

    public class Post
    {
        public int PostId { get; set; }
        public string Title { get; set; }
        public string Content { get; set; }

        public int BlogId { get; set; }
        public virtual Blog Blog { get; set; }
    }

    public class BloggingContext : DbContext
    {
        public DbSet<Blog> Blogs { get; set; }
        public DbSet<Post> Posts { get; set; }
    }


using System;
using System.Collections.Generic;
using System.Data.Entity;
using System.Data.Entity.Infrastructure;
using System.Data.Objects;

namespace Sproc.Demo
{
    class Program
    {
        static void Main(string[] args)
        {
            using (var db = new BloggingContext())
            {
                 db.Database.Initialize(force: false);

                var cmd = db.Database.Connection.CreateCommand();
                cmd.CommandText = "[dbo].[GetAllBlogsAndPosts]";

                try
                {
                    // اجرای پروسیجر
                    db.Database.Connection.Open();
                    var reader = cmd.ExecuteReader();

                    // خواند رکوردهای blogs
```

```
                    var blogs = ((IObjectContextAdapter)db)
                        .ObjectContext
                        .Translate<Blog>(reader, "Blogs", MergeOption.AppendOnly);

                    foreach (var item in blogs)
                    {
                        Console.WriteLine(item.Name);
                    }

                    // پرش به نتایج بعدی (همان Posts)
                    reader.NextResult();
                    var posts = ((IObjectContextAdapter)db)
                        .ObjectContext
                        .Translate<Post>(reader, "Posts", MergeOption.AppendOnly);

                    foreach (var item in posts)
                    {
                        Console.WriteLine(item.Title);
                    }
                }
                finally
                {
                    db.Database.Connection.Close();
                }
            }
        }
    }
```

در کدهای بالا ابتدا یک Connection به بانک اطلاعاتی باز می‌شود:

```
db.Database.Connection.Open();
```

و پس از آن نوبت به اجرای Stored Procedure می‌رسد:

```
var reader = cmd.ExecuteReader();
```

در کد بالا پس از اجرای Stored Procudure نتایج بدست آمده در یک reader ذخیره می‌شود. شئ reader از نوع DBDataReader می‌باشد. پس از اجرای Stored Procedure و دریافت نتایج و ذخیره سازی در شئی reader ، نوبت به جداسازی رکوردها می‌رسد. همانطور که در تعریف Stored procedure مشخص است این Stored Procedure دارای دو نوع خروجی از نوع‌های Blog و Post می‌باشد و این دو نوع باید از هم جدا شوند.برای انجام این کار از متد Translate شئی Context استفاده می‌شود. این متد قابلیت کپی کردن نتایج موجود از یک شئی DBDataReader به یک شئی از نوع مدل را دارد. برای مثال :

```
var blogs = ((IObjectContextAdapter)db)
           .ObjectContext
           .Translate<Blog>(reader, "Blogs", MergeOption.AppendOnly);
```

در کدهای بالا تمامی رکوردهایی از نوع Blog از شئی reader خوانده شده و پس از تبدیل به نوع Blog درون شئی Blogs ذخیره می‌شود.
پس از آن توسط حلقه foreach محتویات Blogs پیمایش شده و مقدار موجود در فیلد Name نمایش داده می‌شود.

```
foreach (var item in blogs)
  {
            Console.WriteLine(item.Name);
  }
```

با توجه به اینکه حاصل اجرای این Stored Procedure دو خروجی متفاوت بوده است ، پس از پیمایش رکوردهای Blogs باید به سراغ نتایج بعدی که همان رکوردهای Post می‌باشد برویم. برای اینکار از متد NextResult شئی reader استفاده می‌شود:

```
reader.NextResult();
```

در ادامه برای خواندن رکوردهایی از نوع Post نیز به همان روشی که برای Blog انجام شد عمل می‌شود. مطالعه بیشتر

نظرات خوانندگان

نویسنده: ashi mashi
تاریخ: ۱۳۹۲/۰۲/۱۱ ۱۴:۳۶

لطفا نحوه اجرا کردن پروسیجرها با مقادیر مختلف در ورودی هم کمی توضیح می‌دادید ممنون می‌شدم.

با تشکر از توضیحات خوبتون،

نویسنده: محسن خان
تاریخ: ۱۳۹۲/۰۲/۱۱ ۱۴:۵۴

استفاده مستقیم از عبارات SQL در EF Code first و مثال void runSp آن.

نویسنده: بهمن آبادی
تاریخ: ۱۳۹۲/۰۲/۱۱ ۱۶:۱۷

منظور من روش مشابه parameterAttributeها در mapping کردن ورودی‌های پروسیجرها در linq می‌باشد.
نه اینکه صرفا از دستورات sql بصورت command استفاده شود.

مانند استفاده از پروسیجرها با چند ورودی و multiResult بودن آن در linq

نویسنده: محسن خان
تاریخ: ۱۳۹۲/۰۲/۱۱ ۱۶:۲۷

در مورد نحوه اجرای رویه‌های ذخیره شده با ورودی‌های مختلف پرسیدید، روش اجراش تا EF 5.0 به همین صورتی است که ملاحظه می‌کنید .

قرار است در EF 6.0 این مساله با Fluent API به نحو دیگری پوشش داده شود.

نویسنده: بهمن آبادی
تاریخ: ۱۳۹۲/۰۲/۱۱ ۱۶:۴۱

دقیقا منظورم همون لینکه EF 6.0 که گفتین هستش.
Road Map برای انتشار نسخه جدید EF 6.0 وجود داره که قراره نهایتا تا کی انتشار پیدا بکنه ؟

بازم ممنون.

نویسنده: سید مهدی فاطمی
تاریخ: ۱۳۹۲/۰۷/۰۴ ۲۱:۴۷

تشکر دوست عزیز

اما در مدل من که از دیتابیس گرفتم شی رویه ذخیره شده هم به عنوان یک تابع شبیه سازی شده و مثل شما عمل نکردم اما نمی‌تونم مقادیر رو از این شی بگیرم.

```
var sp1=_db.splogin(user,pas,value1,value2);
```

در توضیح این کد هم بگم که این اس پی 4 پارامتر داره و قراره دو مقدار رو برای من برگردونه .

نویسنده: سانای رحیمی
تاریخ: ۱۳۹۳/۰۴/۲۲ ۱۹:۳۹

سلام
اگر SP ما بدون پارامتر out باشد و ما در آخر دستورات آن از دستور SCOPE_IDENTITY() استفاده کرده باشیم چگونه می‌توانیم مقدار آن را به دست بیاوریم؟

نویسنده: محسن خان
تاریخ: ۱۳۹۳/۰۴/۲۳ ۱۱:۰

در مطلب استفاده مستقیم از عبارات SQL در EF Code first بحث شده. از متد جنریک db.Database.SqlQuery باید استفاده کنید.

عنوان: **مثالی از الگوی Delegate Dictionary**
نویسنده: محمد صاحب
تاریخ: ۱۳۹۱/۰۸/۱۹ ۱۵:۲۵
آدرس: www.dotnettips.info
برچسب‌ها: C#, Design patterns, Refactoring

این الگو چیز جدیدی نیست و قبلا تو سری مطالب « مروری بر کاربردهای Action و Func » دربارش مطلب نوشته شده و...
البته با توجه به جدید بودن این الگو اسم واحدی براش مشخص نشده ولی تو این مطلب «الگوی Delegate Dictionary» معرفی شده که بنظرم از بقیه بهتره.
به طور خلاصه این الگو میگه اگه قراره براساس شرایط (ورودی) خاصی کار خاصی انجام بشه بجای استفاده از IF و Switch از FuncوDictionary یا Action استفاده کنیم.

برای مثال فرض کنید مدلی به شکل زیر داریم

```
public class Person
{
    public int Id { get; set; }
    public Gender Gender { get; set; }
    public string FirstName { get; set; }
    public string LastName { get; set; }
}
```

قراره براساس جنسیت(**شرایط**) شخص اعتبارسنجی متفاوتی(**کار خاص**) رو انجام بدیدم.مثلا در اینجا قراره چک کنیم اگه شخص مرد بود اسم زنونه انتخاب نکرده باشه و...
خب روش معمول به این شکل میتونه باشه

```
switch (person.Gender)
{
    case Gender.Male:
        if (IsMale(person.FirstName))
        {
            //Isvalid
        }
        break;
    case Gender.Female:
        if (IsFemale(person.FirstName))
        {
            //Isvalid
        }
        break;
}
```

خب این روش خوب جواب میده ولی باید در حد توان استفاده از IF و Switch رو کم کرد.مثلا تو همین مثال ما اصل Open/Closed رو نقض کردیم فکر کنید قرار باشه اعتبارسنجی دیگه ای از همین دست به این کد(کلاس) اضافه بشه باید تغیرش بدیم پس این کد(کلاس) برای تغییر بسته نیست.در اینجور موارد «الگوی Delegate Dictionary» به کار ما میاد.
ما میایم توابع مورد نظرمون رو داخل یک Dictionary ذخیره میکنیم.

```
var genderFuncs = new Dictionary<Gender, Func<string, bool>>
                {
                    {Gender.Male , (x) => IsMale(x)},
                    {Gender.Female , (x) => IsFemale(x)}
                };
```

فرض کنید پیاده سازی توابع به شکل زیر باشه

```
public static bool IsMale(string name)
{
    //check...
    return true;
}
public static bool IsFemale(string name)
```

```
{
    //check...
    if (name == "Farzad")
    {
        return false;
    }
    return true;
}
```

نحوه استفاده

```
var dummyPerson = new List<Person>
                {
                    new Person
                        {Id = 1, Gender = Gender.Male, FirstName = "Mohammad", LastName = "Saheb"},
                    new Person
                        {Id = 2, Gender = Gender.Female, FirstName = "Farzad", LastName = "Mojidi"}
                };

foreach (var person in dummyPerson)
{
    bool isValid = genderFuncs[person.Gender].Invoke(person.FirstName);
}
```

با همین روش میشه قسمت آخر مقاله ی خوب آقای کیاست رو هم Refactor کرد.

```
var query = context.Students.AsQueryable();
  if (searchByName)
  {
      query= query.FindStudentsByName(name);
  }
  if (orderByAge)
  {
      query = query.OrderByAge();
  }
  if (paging)
  {
     query =  query.SkipAndTake(skip, take);
  }
  return query.ToList();
```

توابع رو داخل یک دیکشنری ذخیره میکنیم

```
var searchTypeFuncs = new Dictionary<SearchType, Func<IQueryable<Student>, string,
IQueryable<Student>>>
                    {
                        {SearchType.FirstName, (x, y) => x.FindStudentsByName(y)},
                        {SearchType.LastName, (x, y) => x.FindStudentsByLastName(y)}
                    };
```

نحوه استفاده

```
public static IList<Student> SearchStudents(IQueryable<Student> students, SearchType type, string
keyword)
{
    var result = searchTypeFuncs[type].Invoke(students, keyword);
    return result.ToList();
}
```

عنوان: **متدی برای بررسی صحت کد ملی وارد شده**
نویسنده: صابر فتح الهی
تاریخ: ۱۳۹۱/۰۸/۱۹ ۱۵:۵۰
آدرس: www.dotnettips.info
گروه‌ها: C#, Validation

در بعضی از سایت‌ها به عنوان داده ورودی کد ملی فرد دریافت می‌شود در این پست می‌خواهیم بررسی کنیم که آیا کد ملی وارد شده از نظر صحت درسا وارد شده است یا خیر.
قبل از نوشتن متد قالب کد ملی را شرح می‌دهیم.

کد ملی شماره ای است 10 رقمی که از سمت چپ سه رقم کد شهرستان ، شش رقم بعدی کد منحصر به فرد برای فرد دارنده و رقم آخر آن هم یک رقم کنترل است که از روی 9 رقم سمت چپ بدست می‌آید. برای بررسی کنترل کد کافی است مجدد از روی 9 رقم سمت چپ رقم کنترل را محاسبه کنیم

از آنجایی که درسیستم کد ملی معمولا قبل از کد تعدادی صفر وجود دارد.(رقم اول و رقم دوم از سمت چپ کد ملی ممکن است صفر باشد) و در بسیاری از موارد ممکن است کاربر این صفرها را وارد نکرده باشد و یا نرم افزار این صفرها را ذخیره نکرده باشد بهتر است قبل از هر کاری در صورتی که طول کد بزرگتر مساوی 8 و کمتر از 10 باشد به تعداد لازم (یک تا دو تا صفر) به سمت چپ عدد اضافه کنید. ساختار کد ملی در زیر نشان داده شده است.

ساختار کد ملی										
ارقام کد	رقم کنترل	9 رقم سمت چپ کد ملی								
موقعیت	1	2	3	4	5	6	7	8	9	10

کدهای ملی که همه ارقام آنها مثل هم باشند معتبر نیستند مثل:

```
۰۰۰۰۰۰۰۰۰۰
۱۱۱۱۱۱۱۱۱۱
۲۲۲۲۲۲۲۲۲۲
۳۳۳۳۳۳۳۳۳۳
۴۴۴۴۴۴۴۴۴۴
۵۵۵۵۵۵۵۵۵۵
۶۶۶۶۶۶۶۶۶۶
۷۷۷۷۷۷۷۷۷۷
۸۸۸۸۸۸۸۸۸۸
۹۹۹۹۹۹۹۹۹۹
```

روش اعتبار سنجی کد ملی :

دهمین رقم شماره ملی را (از سمت چپ) به عنوان TempAدر نظر می‌گیریم.

یک مقدار TempB در نظر می‌گیریم و آن را برابر با =

(اولین رقم * ۱۰) + (دومین رقم * ۹) + (سومین رقم * ۸) + (چهارمین رقم * ۷) + (پنجمین رقم * ۶) + (ششمین رقم * ۵) + (هفتمین رقم * ۴) + (هشتمین رقم * ۳) + (نهمین رقم * ۲)

قرار می‌دهیم.

مقدار TempC را برابر با TempB – (TempB/11)*11 = قرار می‌دهیم.

اگر مقدار TempC برابر با صفر باشد و مقدار TempA برابر TempC باشد کد ملی صحیح است.

اگر مقدار TempC برابر با ۱ باشد و مقدار TempA برابر با ۱ باشد کد ملی صحیح است.

اگر مقدار TempC بزرگتر از ۱ باشد و مقدار TempA برابر با ۱۱ – TempC باشد کد ملی صحیح است.

در ادامه متد نوشته شده به زبان C#.NET را مشاهده می‌کنید.

```
public static class Helpers
    {
        /// <summary>
        /// تعیین معتبر بودن کد ملی
        /// </summary>
        /// <param name="nationalCode">کد ملی وارد شده</param>
        /// <returns>
        /// و در صورتی که کد ملی اشتباه باشد خروجی <c>true</c> در صورتی که کد ملی صحیح باشد خروجی
<c>false</c> خواهد بود
        /// </returns>
        /// <exception cref="System.Exception"></exception>
        public static Boolean IsValidNationalCode(this String nationalCode)
        {
           //در صورتی که کد ملی وارد شده تهی باشد

            if (String.IsNullOrEmpty(nationalCode))
                throw new Exception("لطفا کد ملی را صحیح وارد نمایید");


            //در صورتی که کد ملی وارد شده طولش کمتر از 10 رقم باشد
            if (nationalCode.Length != 10)
                throw new Exception("طول کد ملی باید ده کاراکتر باشد");

            //در صورتی که کد ملی ده رقم عددی نباشد
            var regex = new Regex(@"\d{10}");
            if (!regex.IsMatch(nationalCode))
                throw new Exception("کد ملی تشکیل شده از ده رقم عددی می‌باشد؛ لطفا کد ملی را صحیح وارد
نمایید");

            //در صورتی که رقم‌های کد ملی وارد شده یکسان باشد
            var allDigitEqual =
new[]{"0000000000","1111111111","2222222222","3333333333","4444444444","5555555555","6666666666","77777
77777","8888888888","9999999999"};
            if (allDigitEqual.Contains(nationalCode)) return false;


            //عملیات شرح داده شده در بالا
            var chArray = nationalCode.ToCharArray();
            var num0 = Convert.ToInt32(chArray[0].ToString())*10;
            var num2 = Convert.ToInt32(chArray[1].ToString())*9;
            var num3 = Convert.ToInt32(chArray[2].ToString())*8;
            var num4 = Convert.ToInt32(chArray[3].ToString())*7;
            var num5 = Convert.ToInt32(chArray[4].ToString())*6;
            var num6 = Convert.ToInt32(chArray[5].ToString())*5;
            var num7 = Convert.ToInt32(chArray[6].ToString())*4;
            var num8 = Convert.ToInt32(chArray[7].ToString())*3;
            var num9 = Convert.ToInt32(chArray[8].ToString())*2;
            var a = Convert.ToInt32(chArray[9].ToString());

            var b = (((((((num0 + num2) + num3) + num4) + num5) + num6) + num7) + num8) + num9;
            var c = b%11;

            return (((c < 2) && (a == c)) || ((c >= 2) && ((11 - c) == a)));
        }
    }
```

نحوه استفاده از این متد به این صورت می‌باشد:

```
if(TextBoxNationalCode.Text.IsValidNationalCode ())
     //some code

//OR

if(Helpers.IsValidNationalCode (TextBoxNationalCode.Text))
     //some code
```

موفق ومومید باشید

نظرات خوانندگان

نویسنده: صابر احمدی
تاریخ: ۱۳۹۱/۰۸/۲۱ ۹:۷

نحوه استفاده از این کد بدون using system.linq چگونه است؟

نویسنده: صابر فتح اللهی
تاریخ: ۱۳۹۱/۰۸/۲۱ ۹:۴۸

سلام
من که منظور شمارو نفهمیدم.
اما اگر منظورتون این خطه:

```
var allDigitEqual =
new[]{"0000000000","1111111111","2222222222","3333333333","4444444444","5555555555","6666666666","77777
77777","8888888888","9999999999"};
            if (allDigitEqual.Contains(nationalCode)) return false;
```

که شما می‌تونی به جای اینکار پارامتر allDigitEqual را با مقادیر "1....1" و "2....2" و غیره از نظر تساوی با if بررسی کنی البته باید این شرطهارو OR کنید.
یعنی این قسمت به این شکل بنویسید.

```
if (nationalCode == "1111111111" ||
    nationalCode == "0000000000" ||
    nationalCode == "2222222222" ||
    nationalCode == "3333333333" ||
    nationalCode == "4444444444" ||
    nationalCode == "5555555555" ||
    nationalCode == "6666666666" ||
    nationalCode == "7777777777" ||
    nationalCode == "8888888888" ||
    nationalCode == "9999999999")
         return false;
```

امیدوارم منظور شمارو درست فهمیده باشم.

نویسنده: سعید
تاریخ: ۱۳۹۱/۰۹/۱۶ ۱۹:۴۰

سلام آقای مهندس فتح الهی لازم دونستم بابت ارائه این کد از شما تشکر کنم
موفق باشید

نویسنده: مهدی موسوی
تاریخ: ۱۳۹۲/۰۱/۰۶ ۲۰:۲۸

سلام.
من از این مطلب به مطلب شما رسیدم. اونطور که قبلا خونده بودم، کد 1111111111 کد ملی معتبری هستش بنابراین نمیشه در مورد کدهای مشابه نیز زیاد مطمئن بود چرا که ممکنه اونها به افراد دیگه نیز Assign شده باشه. نکته دوم اینکه این بخش از کد شما واقعا آزاردهنده هستش:

```
var chArray = nationalCode.ToCharArray();
var num0 = Convert.ToInt32(chArray[0].ToString())*10;
```

```
var num2 = Convert.ToInt32(chArray[1].ToString())*9;
var num3 = Convert.ToInt32(chArray[2].ToString())*8;
var num4 = Convert.ToInt32(chArray[3].ToString())*7;
var num5 = Convert.ToInt32(chArray[4].ToString())*6;
var num6 = Convert.ToInt32(chArray[5].ToString())*5;
var num7 = Convert.ToInt32(chArray[6].ToString())*4;
var num8 = Convert.ToInt32(chArray[7].ToString())*3;
var num9 = Convert.ToInt32(chArray[8].ToString())*2;
var a = Convert.ToInt32(chArray[9].ToString());

var b = (((((((num0 + num2) + num3) + num4) + num5) + num6) + num7) + num8) + num9;
var c = b%11;

return (((c < 2) && (a == c)) || ((c >= 2) && ((11 - c) == a)));
```

شما می‌تونید کد فوق رو بدین شکل بازنویسی کنید:

```
int result = 0, controlNr = (int)(input[9] - 48);
for (int i = 0; i < input.Length - 1; i++)
    result += (input[i] - 48) * (10 - i);

int remainder = result % 11;
bool isValid = controlNr == (remainder < 2 ? remainder : 11 - remainder);
```

با فرض اینکه input، رشته حاوی کد ملی باشه. در مورد اون بخش از کد که اعداد 1111111111 و 2222222222 و ... رو مقایسه کرده اید نیز، می‌تونید از Regular Expressionها بهره ببرید تا کدتون بسیار خواناتر بشه.

موفق باشید.

نویسنده: صابر فتح الهی
تاریخ: ۱۳۹۲/۰۱/۰۶ ۲۳:۳

بله این قسمت کد آزار دهنده هست و می‌تواند بهتر شود، در ضمن کدهای یکسان طبق الگوریتم خود سازمان ثبت احوال نامعتبر هست در لینکی که دادین به همین موضوع نیز اشاره شده است.
نکته دوم اینکه ممنون میشم Regular Expression مورد نظر بنویسین، چون من زیاد باهاش کار نکردم.
موفق ومومید باشید

نویسنده: مهدی موسوی
تاریخ: ۱۳۹۲/۰۱/۰۷ ۸:۱۸

سلام.
با این Pattern میشه اون اعداد رو جدا کرد:

```
(\d)\1{9}
```

به بیان دیگه، کافیه تا بدین شکل عمل کنیم:

```
if (Regex.IsMatch(input, @"(\d)\1{9}"))
{
    //Invalid Code...
}
```

موفق باشید.

پ.ن.: برای افرادی که هنوز متوجه موضوع نشدن باید عرض کنم که میشه جای این سه خط،

```
var allDigitEqual = new[] { "0000000000", "1111111111", "2222222222", "3333333333", "4444444444",
"5555555555", "6666666666", "7777777777", "8888888888", "9999999999" };
if (allDigitEqual.Contains(nationalCode))
    return false;
```

اینو نوشت:

```
if (Regex.IsMatch(nationalCode, @"(\d)\1{9}"))
    return false;
```

نویسنده: ابراهیم بیاگوی
تاریخ: ۱۳۹۲/۰۱/۱۳ ۱۷:۳۱

سلام. من کدتون رو بهینه و بازنویسی کردم. خوشحال می‌شم تست کنید ببینید رفتار کد عوض نشده باشه:

```
public static bool IsValidIranianNationalCode(string input)
{
    // input has 10 digits that all of them are not equal
    if (!System.Text.RegularExpressions.Regex.IsMatch(input, @"^(?!(\d)\1{9})\d{10}$"))
        return false;

    var check = Convert.ToInt32(input.Substring(9, 1));
    var sum = Enumerable.Range(0, 9)
        .Select(x => Convert.ToInt32(input.Substring(x, 1)) * (10 - x))
        .Sum() % 11;

    return sum < 2 && check == sum || sum >= 2 && check + sum == 11;
}
```

gist

در regex از negative lookahead استفاده شده که بررسی‌شه هر ده عدد یکی نباشن، از Substring استفاده‌شده که با توجه به پیاده‌سازی کلاس String به نظر من خیلی بهینه هست، پرانتزها هم حذف شدند چون بدون پرانتز با توجه به اولویت عملگرها همان معنی را می‌دهد.

نویسنده: صابر فتح الهی
تاریخ: ۱۳۹۲/۰۱/۱۴ ۸:۲۹

به نظر که درست هست، از همکاری شما متشکرم

نویسنده: سجـــــــاد ف
تاریخ: ۱۳۹۲/۰۶/۲۸ ۸:۴۶

با سلام
اینم واسه **شناسه ملی اشخاص حقوقی** منبع

```
public bool IsValidIranianLegalCode(string input)
        {
            //input has 11 digits that all of them are not equal
            if (!Regex.IsMatch(input, @"^(?!(\d)\1{10})\d{11}$"))
                return false;

            var check = Convert.ToInt32(input.Substring(10, 1));
            int dec = Convert.ToInt32(input.Substring(9, 1)) + 2;
            int[] Coef = new int[10] { 29, 27, 23, 19, 17, 29, 27, 23, 19, 17 };

            var sum = Enumerable.Range(0, 10)
                          .Select(x => (Convert.ToInt32(input.Substring(x, 1)) + dec) * Coef[x])
```

```
                        .Sum() % 11;

        return sum == check;
    }
```

نویسنده: افشین
تاریخ: ۱۳۹۲/۰۸/۰۶ ۱۳:۵۹

عجیبه !
کد ملی 1000000001 که همه میدونیم، اشتباهه رو معتبر می‌دونه !

نویسنده: صابر فتح الهی
تاریخ: ۱۳۹۲/۰۸/۰۷ ۰:۵۷

طبق فرمول باید صحیح باشه منم چن سایت آنلاین بررسی کردم حق با شماست دارم بررسی می‌کنم

نویسنده: مهمان
تاریخ: ۱۳۹۲/۰۸/۱۰ ۱۲:۵۰

عذر میخوام،
چرا کدملی 1000000001 اشتباه است؟ طبق فرمولی که معرفی شده که صحیحه.
(باقیمانده 1*10 تقسیم بر 11 برابر با 10 میشه و چون این عدد بزرگتر از 1 است باید از 11 کسر بشه که نتیجه میشه 1)

آیا فرمول دقیق‌تری برای تشخیص صحت کدملی وجود دارد؟
به چه علت و بر چه اساسی کد ملی مذکور اشتباه است؟
آیا اداره ثبت احوال فرمول دقیق تشخیص صحت کدملی را در ارائه داده است؟

و ضمنا آیا بررسی یکسان بودن تمام ارقام ضروری است؟ بنظر میرسه نیازی اصلا به این بررسی وجود نداشته باشد. مثلا برای کدملی که هر ده رقم صفر است هیچگاه رقم کنترل صحیح نخواهد بود.
و همچنین در این لینک هیچ اشاره ای به این موضوع نشده است.

با تشکر

نویسنده: مهمان
تاریخ: ۱۳۹۲/۰۸/۱۰ ۱۳:۲۰

راستی موضوعی را فراموش کردم مطرح کنم.
واقعیتش من نیاز دارم کدملی را دریافت کنم که واقعی باشه اونم به معنای واقعی کلمه. عرض می‌کنم.

همانطور که میدانید (مطابق با تصویری که در ویکیا قرار دارد و قبلا لینکش را قرار دادم) سه رقم اول کد ملی (از چپ) مربوط به کد شهرستان محل صدور شناسنامه فرد است. حالا باید ما لیست کدهای شهرستان‌ها را داشته باشیم تا بتونیم کدملی را بپذیریم که مربوط باشه به یک شهرستانی که وجود خارجی داره (به عبارتی از بین تمام ترکیبات ممکن، اگر اشتباه نکنم 10 به قوه‌ی 3 که به بیان دیگر 1 تا 999 که میشه 1000 مورد؛ من اطلاعی از تعداد شهرستان‌های ایران ندارم ولی چیزی که در اینجا می‌بینم بعید میرسه به هزار برسه البته از صحت و سقم آن بی اطلاع هستم)

نکته‌ی بعدی مرتبط هست به کدمنحصربفرد (شش رقم از راست بدون در نظر گرفتن رقم کنترل) این هم فکر کنم با یک قاعده و قانونی مقداردهی شده است یا شایدم نه بصورت ترتیبی باشه مثلا از 100001 (یکصد و یک هزار) آغاز شده و یکی یکی افزایش پیدا کرده.

به هر حال اگر اطلاعات راجب این موضوع دارید لطفا دریغ نفرمایید.
منتظر پاسخ دوستان هستم.
با تشکر.

نویسنده: م منفرد
تاریخ: ۱۳۹۲/۰۸/۱۰ ۱۳:۳۹

سلام
دوست عزیز ویکیپدیا منبع معتبری در این زمینه نمی‌باشد چون هر فردی می‌تواند مقاله در آن ثبت کند و افرادی که سابقه بیشتری داشته باشند می‌توانند آنها را اصلاح و... نمایند

نویسنده: صابر فتح الهی
تاریخ: ۱۳۹۲/۰۸/۱۱ ۱۲:۲۵

دوست گلم سلام
طبق فرمول‌های ارائه شده با بررسی کد ملی به طریق بالا که اگر اینترنت هم جستجو بزنین فرمول هاش ارائه شده این اعداد با روش بالا بررسی شده و در صورت عدم اعتبار کد ملی به شما خروجی میده. روشی که شما می‌خوای پیش بگیری اینه که باید برین و لیست کد تمام شهرستان‌های ایران بگیرین و توی برنامه به کار ببندین. که به نظرم روش جالبی نیست

نویسنده: افشین
تاریخ: ۱۳۹۲/۰۸/۲۶ ۱۳:۵۹

با استعلام از سازمان ثبت احوال متوجه شدیم که کد ملی 172942284 کاملا صحیحه ولی کد شما اون رو نادرست می‌دونه.

نویسنده: صابر فتح الهی
تاریخ: ۱۳۹۲/۰۸/۲۶ ۱۹:۱۲

کدها اصلاح شد

```
namespace ConsoleApplicationTest
{
    class Program
    {
        static void Main(string[] args)
        {
            Console.WriteLine("0172942284 => {0}", "0172942284".IsValidNationalCode());
            Console.WriteLine("1000000001 => {0}", "1000000001".IsValidNationalCode());
        }
    }

    public static class Helpers
    {
        public static Boolean IsValidNationalCode(this String nationalCode)
        {
            if (String.IsNullOrEmpty(nationalCode))
                throw new Exception("لطفا کد ملی را صحیح وارد نمایید");

            if (nationalCode.Length != 10)
                throw new Exception("طول کد ملی باید ده کاراکتر باشد");

            var regex = new Regex(@"[^0-9]
```

```
");
            if (!regex.IsMatch(nationalCode))
                throw new Exception("کد ملی تشکیل شده از ده رقم عددی می‌باشد؛ لطفا کد ملی را صحیح وارد
نمایید");

            if (!Regex.IsMatch(nationalCode, @"^(?!(\d)\1{9})\d{10}$"))
                return false;
```

```
            var check = Convert.ToInt32(nationalCode.Substring(9, 1));
            var result = Enumerable.Range(0, 9)
                .Select(x => Convert.ToInt32(nationalCode.Substring(x, 1)) * (10 - x))
                .Sum() % 11;

            int remainder = result % 11;
            return check == (remainder < 2 ? remainder : 11 - remainder);

        }
    }
}
```

پ.ن. لازم به ذکر است از کدهای اقای بیاگوی استفاده شده است.

نویسنده: ابراهیم بیاگوی
تاریخ: ۱۳۹۲/۰۸/۲۶ ۲۱:۴۴

اگر کارت‌های ملی که شمارهٔ شناسنامهٔ ۹ و ۸ رقمی دارند را دیده باشید متوجه می‌شوید که صفرهای اول جز شمارهٔ ملی **است** . به هر حال اگر به هر دلیل صفرهای اول را به گونه‌ای پاک‌کرده‌اید می‌توان کد را اینگونه نوشت که کدهای ۹ یا ۸ رقمی را پس از افزوده‌شدن صفر مانند کدهای ۱۰ رقمی صحت‌سنجی کند:

```
public static bool IsValidIranianNationalCode(string input)
        {
            input = input.PadLeft(10, '0');

            if (!Regex.IsMatch(input, @"^\d{10}$"))
                return false;

            var check = Convert.ToInt32(input.Substring(9, 1));
            var sum = Enumerable.Range(0, 9)
                .Select(x => Convert.ToInt32(input.Substring(x, 1)) * (10 - x))
                .Sum() % 11;

            return sum < 2 && check == sum || sum >= 2 && check + sum == 11;
        }
```

راستی اعداد یکسان نامعتبر **نیست** : http://www.fardanews.com/fa/news/127747
برای زبان‌های دیگر این کد https://gist.github.com/ebraminio/5292017

نویسنده: صابر فتح الهی
تاریخ: ۱۳۹۲/۰۸/۲۷ ۱۰:۱۰

ظاهرا در صورتی که عدد قرینه باشه جواب درست در میاد

نویسنده: ژ محمدی
تاریخ: ۱۳۹۲/۰۹/۰۳ ۷:۵

سلام
اگر بخواهیم صحت کد ملی را از طریق jqueryvalidation چک کنیم کد ان به چه صورت خواهد بود
ممنون میشم این را هم تکه کد را هم اضافه کنید

نویسنده: محسن خان
تاریخ: ۱۳۹۲/۰۹/۰۳ ۹:۲۱

آقای بیاگوی چند نظر بالاتر کدهای جاوا اسکریپتی آن‌را قرار دادند (قسمت برای زبان‌های دیگر این کد ...).

نویسنده: ژ محمدی
تاریخ: ۱۳۹۲/۰۹/۰۸ ۱۸:۳۲

سلام؛ دوتا سوال داشتم. آیا ابتدای شناسه ملی اشخاص حقوقی میتواند با 4 تا صفر شروع شود؟ میشه کد جاوا اسکریپت آن را هم بدهید ؟

نویسنده: صابر فتح الهی
تاریخ: ۱۳۹۲/۰۹/۰۹ ۱۳:۲۹

سلام

می‌تونین چک کنین فکر نکنم معتبر باشه

کد جاوا اسکریپت تشخیص صحت کد ملی، منبعش این سایته

```
function checkCodeMeli(code)
{

  var L=code.length;

  if(L<8 || parseInt(code,10)==0) return false;
  code=('0000'+code).substr(L+4-10);
  if(parseInt(code.substr(3,6),10)==0) return false;
  var c=parseInt(code.substr(9,1),10);
  var s=0;
  for(var i=0;i<9;i++)
    s+=parseInt(code.substr(i,1),10)*(10-i);
  s=s%11;
  return (s<2 && c==s) || (s>=2 && c==(11-s));
return true;
}
```

نویسنده: صابر فتح الهی
تاریخ: ۱۳۹۲/۰۹/۰۹ ۱۳:۵۶

اینم برای تشخیص صحت کد ملی اشخاص حقوقی با استفاده از جاوا اسکریپت

```
function checkCodeMeli(code)
{

  var L=code.length;

  if(L<11 || parseInt(code,10)==0) return false;

  if(parseInt(code.substr(3,6),10)==0) return false;
  var c=parseInt(code.substr(10,1),10);
  var d=parseInt(code.substr(9,1),10)+2;
  var z=new Array(29,27,23,19,17);
  var s=0;
  for(var i=0;i<10;i++)
    s+=(d+parseInt(code.substr(i,1),10))*z[i%5];
  s=s%11;if(s==10) s=0;
  return (c==s);

}
```

نویسنده: ژ محمدی
تاریخ: ۱۳۹۲/۰۹/۱۱ ۲۳:۲۵

در مورد کد اقتصادی چطور ؟ آن را هم میدونید به چه صورت هست ؟ و کد اعتبار سنجی به چه صورت هست

نویسنده: صابر فتح الهی
تاریخ: ۱۳۹۲/۰۹/۱۲ ۱۵:۴۰

فعلا نمیدونم بررسی نکردم

نویسنده: جواد
تاریخ: ۱۳۹۲/۱۲/۲۳ ۱۹:۵۷

لطف کنید کد تشخیص کد ملی را به زبان vb 2008 برام بزارید ممنون

نویسنده: صابر فتح الهی
تاریخ: ۱۳۹۲/۱۲/۲۳ ۲۲:۳۶

سلام
من از یک مبدل آنلاین استفاده کردم به این نتیجه رسیدم.

```
Namespace ConsoleApplicationTest
Class Program
Private Shared Sub Main(args As String())
Console.WriteLine("0172942284 => {0}", "0172942284".IsValidNationalCode())
Console.WriteLine("1000000001 => {0}", "1000000001".IsValidNationalCode())
End Sub
End Class

Public NotInheritable Class Helpers
Private Sub New()
End Sub
<System.Runtime.CompilerServices.Extension> _
Public Shared Function IsValidNationalCode(nationalCode As [String]) As [Boolean]
If [String].IsNullOrEmpty(nationalCode) Then
Throw New Exception("لطفا کد ملی را صحیح وارد نمایید")
End If

If nationalCode.Length <> 10 Then
Throw New Exception("طول کد ملی باید ده کاراکتر باشد")
End If

Dim regex__1 = New Regex("[^0-9]<span> </span>")
If Not regex__1.IsMatch(nationalCode) Then
Throw New Exception("کد ملی تشکیل شده از ده رقم عددی می‌باشد؛ لطفا کد ملی را صحیح وارد نمایید")
End If

If Not Regex.IsMatch(nationalCode, "^(?!(\d)\1{9})\d{10}$") Then
Return False
End If

Dim check = Convert.ToInt32(nationalCode.Substring(9, 1))
Dim result = Enumerable.Range(0, 9).[Select](Function(x) Convert.ToInt32(nationalCode.Substring(x, 1))
* (10 - x)).Sum() Mod 11

Dim remainder As Integer = result Mod 11
Return check = (If(remainder < 2, remainder, 11 - remainder))

End Function
End Class
End Namespace
```

نویسنده: ناشناس
تاریخ: ۱۳۹۳/۰۴/۰۱ ۱۵:۰

با سلام،

تو عبارت "مقدار TempC را برابر با TempB - (TempB/11)*11 = قرار می‌دهیم. " منظور اینه که C باید برابر با باقیمانده تقسیم صحیح TempB بر 11 باشه دیگه؟

چون TempB - (TempB/11)*11 که مقدارش 0 میشه به نظر.

ممنون

نویسنده: مهیار
تاریخ: ۱۳۹۳/۰۵/۰۶ ۱۵:۶

عذرخواهی می‌کنم ولی کد ملی من در این متد نا معتبر است !
و یا مثل این کد ملی "0081037511" که معتبر است ولی حتی متد اصلاح شده شما درون exception زیر می‌افتد

```
throw new Exception("کد ملی تشکیل شده از ده رقم عددی می‌باشد؛ لطفا کد ملی را صحیح وارد نمایید");
```

نویسنده: محسن خان
تاریخ: ۱۳۹۳/۰۵/۰۸ ۱۲:۱۰

خوب کاربر رو اینقدر عذاب ندید. همون قسمت بررسی با Regex کافی هست. بیشتر نیازی نیست.

```
input = input.PadLeft(10, '0');
if (!Regex.IsMatch(input, @"^\d{10}$"))
        return false;
```

عنوان: **کلاس کمکی جهت مشاهده آیتم های موجود در حافظه کش و حذف آنها**
نویسنده: صابر فتح الهی
تاریخ: ۱۳۹۱/۰۸/۱۹ ۱۶:۲۰
آدرس: www.dotnettips.info
برچسب‌ها: #ASP.Net, C

مواقع بسیاری پیش می‌آید که در زمان کار با یک نرم افزار تحت وب زمان اشکال زدایی پیش می‌آید که به دلیل موجود بودن داده در حافظه کش برنامه نویس نمی‌تواند داده‌های واقعی را ببیند و داده‌های موجود در حافظه کش را مشاهده می‌کند (بیشتر مواقعی که از طریق بانک اطلاعاتی مستقیما اقدام به حذف و اضافه داده می‌کنیم) در این بخش یک کلاس آماده کرده ام که همیشه خودم در نرم افزار هایم استفاده می‌کنم.

شما می‌توانید این کلاس را به یک GridView یا کنترل‌های دیگر بایند کرده و کلیدهای موجود در حافظه کش را مشاهده کنید، و در صورتی که خواستید یک کلید خاص را از حافظه کش حذف نمایید (البته این کلاس بیشتر برای مدیر نرم فزار کاربرد دارد).

می‌توانید فایل مورد نظر را از طریق لینک کلاس کمکی جهت مشاهده آیتم‌های موجود در حافظه کش و حذف آنها دانلود نمایید.
در کلاس زیر هر کدام از قسمت‌ها را شرح می‌دهیم.

```
using System;
using System.Collections.Generic;
using System.ComponentModel;
using System.Web;
using System.Web.Caching;

namespace PWS.BLL
{
    /// <summary>
    /// کلاس  آیتم‌های حافظه کش
    /// </summary>
    [DataObject(true)]
    public class CacheItems
    {
#region Constructors (2)

        /// <summary>
        /// سازنده اصلی
        /// </summary>
        /// <param name="cacheItem">عنوان آیتم ذخیره شده در حافظه کش</param>
        public CacheItems(String cacheItem)
        {
            CacheItem = cacheItem;
        }

        /// <summary>
        /// سازنده پیش فرض
        /// </summary>
        public CacheItems(){}

#endregion Constructors

#region Properties (2)

        /// <summary>
        /// کش کانتکست جاری
        /// </summary>
        /// <value>
        /// The cache.
        /// </value>
        private static Cache Cache
        {
            get {return HttpContext.Current.Cache; }
        }

        /// <summary>
        /// عنوان آیتم ذخیره شده در حافظه کش
        /// </summary>
        public String CacheItem{ get; set;}

#endregion Properties

#region Methods (4)
```

```
// Public Methods (3)

        /// <summary>
        /// لیست تمام آیتم‌های ذخیره شده در حافظه کش
        /// </summary>
        /// <returns></returns>
        public List<CacheItems> GetCaches()
        {
            var items = new List<CacheItems>();
            //بازیابی کل کلیدهای موجود در حافظه کش و اضافه کردن آن به لیست مربوطه
            var enumerator = Cache.GetEnumerator();
            while (enumerator.MoveNext())
            {
                 items.Add(new CacheItems(enumerator.Key.ToString()));
            }
            return items;
        }

        /// <summary>
        /// حذف آیتم جاری از حافظه کش
        /// </summary>
        public void RemoveItemFromCache()
        {
            RemoveItemFromCache(CacheItem);
        }

        /// <summary>
        /// حذف کردن یک آیتم از حافظه کش
        /// </summary>
        /// <param name="key">کلید ذخیره شده در حافظه کش</param>
        public static void RemoveItemFromCache(string key)
        {
            PurgeCacheItems(key);
        }
// Private Methods (1)

        /// <summary>
        /// حذف کردن یک ایتم از حافظه کش با پشوند وارد شده
        /// </summary>
        /// <param name="prefix">پیشوندی از کلید موجود در حافظه کش</param>
        private static void PurgeCacheItems(String prefix)
        {
            prefix = prefix.ToLower();
            var itemsToRemove = new List<String>();
           //لیست آیتم‌های موجود در حافظه کش
            var enumerator = Cache.GetEnumerator();
            while (enumerator.MoveNext())
            {


                //در صورتی که کلید مورد نظر با پارامتر وارد شده شروع شده باشد آن را به یک لیست اضافه
می‌کنیم


                if (enumerator.Key.ToString().ToLower().StartsWith(prefix))
                    itemsToRemove.Add(enumerator.Key.ToString());
            }
             //لیست مورد نظر را پیمایش کرده و گزینه‌های آن را از حافظه کش حذف می‌کنیم
            foreach (var itemToRemove in itemsToRemove)
                Cache.Remove(itemToRemove);
        }

#endregion Methods
    }
}
```

موفق ومومید باشید

نظرات خوانندگان

نویسنده: سی شارپ 2012
تاریخ: ۱۳۹۲/۰۸/۲۵ ۱۳:۲۶

چگونه از این کلاس استفاده کنیم ؟
لطفا مثال بزنید

نویسنده: صابر فتح الهی
تاریخ: ۱۳۹۲/۰۸/۲۶ ۱۲:۹

به نظرم کاملا واضحه
شما می‌تونین متد GetCaches را به یک گرید ویو بایند کنید.
با این کار شما لیست ایتم‌های موجود در کش دارید
در مرحله بعد برای حذف هر ایتم از کش می‌توانید ایتم انتخاب شده را به تابع PurgeCacheItems(String prefix) پاس داده و ایتم مورد نظر حذف کنید.
کد نوشته شده چیز چندان پیچیده ای نداره اگرم باز مشکل دارید لطفا مطرح کنید
موفق و موید باشید

عنوان: **سرریز اعداد و checked در #C**
نویسنده: رحمت اله رضایی
تاریخ: ۱۳۹۱/۰۸/۲۴ ۸:۵
آدرس: www.dotnettips.info
برچسب‌ها: C#, checked, Overflow

حلقه زیر را در نظر بگیرید :

```
for (byte i = 0; i <= 255; i++)
                {
                    ...
                }
```

به نظر شما این حلقه چند بار اجرا می‌شود؟

این حلقه را در یک برنامه استفاده کرده بودم. مشکل اینجا بود که برنامه وارد حلقه می‌شد اما از آن خارج نمی‌شد و یک حلقه بی‌نهایت ایجاد شده بود.
اما چرا؟

دلیل آن این است که وقتی i به 255 رسیده و می‌خواهد به اضافه یک شود، حاصل 256 نمی‌شود که شرط i<=255 برقرار نشود و برنامه از حلقه خارج شود. بلکه چون i از نوع byte است، سرریز کرده و نتیجه صفر می‌شود.
این حالت برای int و سایر نوع اعداد هم صادق است.
برای رفع این مشکل چکار باید کرد؟

چاره آن کلمه کلیدی checked است که از سرریز اعداد جلوگیری می‌کند :

```
checked
            {
                for (byte i = 0; i <= 255; i++)
                {
                    ...
                }
            }
```

در این حالت زمانی که i به 255 رسیده و می‌خواهد به اضافه یک شود، OverflowException ایجاد می‌شود.

در مقابل کلمه unchecked هم هست که عکس checked عمل می‌کند.

نظرات خوانندگان

نویسنده: وحید نصیری
تاریخ: ۱۳۹۱/۰۸/۲۴ ۹:۳۹

روش دیگر کامپایل کردن کد با پارامتر زیر است

```
/checked
```

در کل کارآیی این روش حدود سه بار کندتر است از روش معمولی بدون checked و اگر مشکل یافت شده بهتر است از بازه صحیح اعداد استفاده شود.

نویسنده: رحمت اله رضایی
تاریخ: ۱۳۹۱/۰۸/۲۴ ۱۴:۱۳

ممنون. اصلاح کردم به :

```
for (int i = 0; i <= 255; i++)
            {
                ...
            }
```

عنوان: **از متد DateTime.ToString بدون پارامتر استفاده نکنید!**
نویسنده: وحید نصیری
تاریخ: ۱۳۹۱/۰۸/۲۶ ۲۱:۵
آدرس: www.dotnettips.info
برچسب‌ها: C#, DateTime

در حین تهیه کتابخانه Silverlight DatePicker فارسی، گاها استفاده کنندگان گزارش می‌دادند که برنامه روی سیستم‌های مختلف کرش می‌کند یا تبدیل تاریخ درست انجام نمی‌شود. مشکل هم پس از بررسی طولانی به این ترتیب مشخص شد که استفاده از DateTime.ToString بدون ذکر پارامترهایی که در ادامه توضیح داده خواهند شد، اشتباه است.

متد ToString بر اساس تنظیمات محلی عمل می‌کند

خروجی فراخوانی ذیل

```
DateTime.Now.ToString()
```

در یک سیستم می‌تواند

```
01/11/2012 09:49:08 ق.ظ
```

و در سیستمی دیگر

```
11/1/2012 9:49:08 AM
```

باشد.
این مساله خصوصا برای ذخیره سازی و پردازش اطلاعات به صورت رشته بسیار مهم و مساله ساز است.
فرض کنید در یک شبکه با تنظیمات محلی متفاوت، کاربران اطلاعات تاریخ متفاوتی را به بانک اطلاعاتی ارسال کنند. پردازش صحیح این تاریخ‌ها تقریبا غیرممکن است. حالت اول روز 11 ماه یک را نمایش می‌دهد و حالت دوم روز 1 ماه 11. در حالیکه هر دو تاریخ در یک روز ثبت شده‌اند اما تنظیمات محلی کاربران متناظر یکسان نبوده است.
برای رفع این مشکل نیاز است ToString را مستقل از تنظیمات محلی کاربران کرد:

```
DateTime.Now.ToString(CultureInfo.InvariantCulture)
```

این مورد نکته‌ای است که اگر از FxCop برای آنالیز اسمبلی‌های برنامه خود استفاده کنید، حتما گوشزد خواهد شد. همچنین ReSharper نیز رعایت آن‌را در نگارش‌های اخیر خود گنجانده است.

مشکل دیگر مشابه در حین کار با Silverlight و WPF، استفاده و پردازش e.NewValue تغییرات خواص است. این مقدار به صورت object ارسال می‌شود و برای پردازش آن نباید e.NewValue.ToString فراخوانی شود. روش صحیح دریافت تاریخ از آن باید به صورت زیر باشد:

```
public static DateTime? DateTimeTryParse(object data)
        {
            if (data == null)
                return null;

            if (data.GetType().Equals(typeof(DateTime)))
                return (DateTime)data;

            DateTime result;
            if (DateTime.TryParse((string)data, CultureInfo.InvariantCulture, DateTimeStyles.None, out
result))
                return result;

            return null;
        }
```

دو نکته در اینجا قابل توجه است:

- در متد DateTime.TryParse بجای data.ToString، به string تبدیل شده است. متد DateTime.TryParse نیز حالت ویژه‌ای پیدا کرده و CultureInfo.InvariantCulture در آن قید شده است.

- همچنین چون نوع e.NewValue تغییرات دریافتی از نوع object می‌باشد، بهتر است ابتدا بررسی شود که آیا DateTime است یا خیر. سپس سایر بررسی‌ها صورت گیرد.

از متد DateTime.ToString بدون پارامتر استفاده نکنید!

نظرات خوانندگان

نویسنده: رضا نباتی
تاریخ: ۱۳۹۱/۰۸/۲۶ ۲۱:۲۳

به نظر بنده بهترین راه برای گرفتن خروجی دلخواه ارسال فرمت استرینگ DateTime برای نمایش آن در خروجی می‌باشد.

```
DateTime thisDate1 = new DateTime(2011, 6, 10);
Console.WriteLine("Today is " + thisDate1.ToString("MMMM dd, yyyy") + ".");
```

نویسنده: بهروز راد
تاریخ: ۱۳۹۱/۰۸/۲۷ ۸:۲۷

البته ابزار Code Analysis موجود در Visual Studio Ultimate هم از FxCop در پشت صحنه استفاده می‌کنه و این موارد رو گوشزد می‌کنه.

نویسنده: میثم جوادی
تاریخ: ۱۳۹۱/۰۸/۲۷ ۱۲:۷

من هم یه بار به همچین مشکلی ، شبیه به این برخورده بودم

توضیحات

نویسنده: صابر فتح اللهی
تاریخ: ۱۳۹۱/۰۸/۲۷ ۱۴:۱۴

همونطوری که مهندس توی پست شون ذکر کردن نسخه جدید reSharper هم این عمل گوشزد می‌کنه

عنوان: **تبدیل عدد صحیح به هگزا دسیمال و بلعکس**
نویسنده: امیر هاشم زاده
تاریخ: ۱۳۹۱/۰۹/۰۳ ۲:۱۵
آدرس: www.dotnettips.info
برچسب‌ها: C#, hex

برای تبدیل یک عدد صحیح به هگزا دسیمال معادل و بلعکس از کد زیر استفاده می‌کنیم.

```
int intValue = 182;

string hexValue = intValue.ToString("X");

int intAgain = int.Parse(hexValue, System.Globalization.NumberStyles.HexNumber);
```

کدهای بالا بقدر کافی روشن و واضح هستند که نیازی به توضیح اضافی نداشته باشند ولی فرض کنید قصد دارید عدد هگزا دسیمال 20 رقمی زیر را به معادل عدد صحیح آن تبدیل کنید، این عدد نیاز به 80 بیت دارد که امکان ذخیره سازی آن در int و یا long وجود ندارد چون طول آن‌ها بترتیب 32بیت و 64بیت است.

```
var hexValue = "0x00010471000001BF001F";
```

برای حل این مشکل ابتدا ارجاعی از System.Numerics رابه پروژه اضافه کنید سپس به یکی از 2 روش زیر آن را به عدد صحیح تبدیل کنید.

روش اول:

```
byte[] bigNumber = new byte[]
            {
                0, 0, 0, 1, 0, 4, 7, 1, 0, 0, 0, 0, 0, 1,
                byte.Parse("B", System.Globalization.NumberStyles.HexNumber),
                byte.Parse("F", System.Globalization.NumberStyles.HexNumber),
                0, 0, 1,
                byte.Parse("F", System.Globalization.NumberStyles.HexNumber)
            };

            Array.Reverse(bigNumber);
            var bigInt = new System.Numerics.BigInteger(bigNumber);
```

در این روش ابتدا ارقام عدد هگزا دسیمال را در یک آرایه از نوع داده بایت ذخیره کنید سپس آرایه را معکوس کرده و آن را به متد BigInteger پاس دهید. علت معکوس کردن آرایه این است که متد سازنده BigInteger به آرایه little-endian نیاز دارد.

روش دوم:

```
var bigint =
                System.Numerics.BigInteger.Parse("00010471000001BF001F",
                System.Globalization.NumberStyles.HexNumber,
                System.Globalization.CultureInfo.InvariantCulture);
```

در این روش باید "0x" را از ابتدای عدد هگزا دسیمال جدا کنید و سپس آن را به BigInteger.Parse پاس دهید.

منابع:

C# convert integer to hex and back again

How do I convert this Hex to an Integer?

عنوان: **تغییر اندازه تصاویر #1**
نویسنده: صابر فتح الهی
تاریخ: ۱۳۹۱/۰۹/۱۴ ۱۱:۳۰
آدرس: www.dotnettips.info
گروه‌ها: #ASP.Net, C

برای برنامه نویسان همیشه این امکان هست که تصاویری را که از کاربر دریافت می‌کنند تغییر اندازه دهند، مثلا در همین سایت تصاویری از کاربران جهت نمایش در پروفایل آنها دریافت می‌شود، در همین سایت نیز این اتفاق می‌افتد مثلا تصاویر پروفایل کاربران با اندازه‌های متفاوتی نشان داده می‌شود.

برای انجام این کارها میتوان به دو طریق عمل کرد:

تغییر اندازه تصویر در زمان ذخیره‌سازی
در زمانی که می‌خواهیم تصویر را به بازدید کننده نشان دهیم

در حالت 1 زمانی که تصویری را از کاربر دریافت می‌کنیم با توجه به اینکه تصویر را با چه اندازه‌هایی در نرم افزار نیاز داریم تغییر اندازه داده و تک تک ذخیره می‌کنیم، این روش کل عملیات در زمان ثبت و تنها یکبار اتفاق می‌افتد، این روش جای بیشتری از منابع (مانند هارد دیسک یا دیتابیس) سرور را اشغال می‌کند اما در عوض می‌توان گفت سرعت بالاتری دارد، در روش دوم زمانی که بازدید کننده از سایت (نرم افزار) بازدید می‌کند تصویر اصلی با توجه به نیاز تغییر اندازه داده شده و برای کاربر ارسال می‌شود (در واقع کاربر آن را مشاهده می‌کند)، این روش فضای کمتری از منابع را اشغال می‌کند اما در زمان اجرا عملیات اضافی برای هر کاربری (البته با کش کردن این عملیات کم می‌شود) انجام می‌شود.
در هر دو روش گفته شده در هر صورت ما باید متد (توابعی) برای تغییر اندازه تصویر داشته باشیم که در زیر نحوه نوشتن آن را شرح خواهیم داد.

```
using System;
using System.Drawing;
using System.Drawing.Drawing2D;
using System.Drawing.Imaging;
using System.IO;

namespace PWS
{
    public static class Helpers
    {
        /// <summary>
        /// تغییر اندازه تصویر
        /// </summary>
        /// <param name="imageFile">آرایه بایتی از تصویر مورد نظر</param>
        /// <param name="targetSize">اندازه تصویر خروجی</param>
        /// <param name="format">فرمت تصویر خروجی</param>
```

```
        /// <returns></returns>
        public static byte[] ResizeImageFile(this byte[] imageFile, Int32 targetSize, ImageFormat
format)
        {
            if (imageFile == null)
                throw new Exception("لطفا تصویر اصلی را مشخص نمایید");
            //باز کردن تصویر اصلی به عنوان یک جریان
            using (var oldImage = Image.FromStream(new MemoryStream(imageFile)))
            {
                //محاسبه اندازه تصویر خروجی با توجه به اندازه داده شده
                var newSize = CalculateDimensions(oldImage.Size, targetSize);
                //ایجاد تصویر جدید
                using (var newImage = new Bitmap(newSize.Width, newSize.Height,
PixelFormat.Format24bppRgb))
                {
                    using (var canvas = Graphics.FromImage(newImage))
                    {
                        canvas.SmoothingMode = SmoothingMode.AntiAlias;
                        canvas.InterpolationMode = InterpolationMode.HighQualityBicubic;
                        canvas.PixelOffsetMode = PixelOffsetMode.HighQuality;
                        //تغییر اندازه تصویر اصلی و قرار دادن آن در تصویر جدید
                        canvas.DrawImage(oldImage, new Rectangle(new Point(0, 0), newSize));
                        var m = new MemoryStream();
                        //ذخیره تصویر جدید با فرمت وارد شده
                        newImage.Save(m, format);
                        return m.GetBuffer();
                    }
                }
            }
        }

        /// <summary>
        /// محاسبه ابعاد تصویر خروجی
        /// </summary>
        /// <param name="oldSize">اندازه تصویر اصلی</param>
        /// <param name="targetSize">اندازه تصویر خروجی</param>
        /// <returns></returns>
        private static Size CalculateDimensions(Size oldSize, Int32 targetSize)
        {
            var newSize = new Size();
            if (oldSize.Height > oldSize.Width)
            {
                newSize.Width = Convert.ToInt32(oldSize.Width * (targetSize / (float)oldSize.Height));
                newSize.Height = targetSize;
            }
            else
            {
                newSize.Width = targetSize;
                newSize.Height = Convert.ToInt32(oldSize.Height * (targetSize / (float)oldSize.Width));
            }
            return newSize;
        }
    }
}
```

در متد ResizeImageFileتصویر اصلی به عنوان یک جریان باز می‌شود. (سطر 23)

اندازه تصویر خروجی با توجه به اندازه وارد شده توسط متد CalculateDimensions تعیین می‌شود؛ روال کار متد CalculateDimensions اینگونه است که اندازه عرض و ارتفاع تصویر اصلی بررسی می‌شود و با توجه به اینکه کدام یک از اینها بزرگتر است تغییر اندازه در آن صورت می‌گیرد، مثلا در صورتی که عکس ارتفاع بیشتری نسبت به عرض تصویر داشته باشد تصویر تغییر اندازه داده شده نیز با توجه به تناسب ارتفاع تغییر داده می‌شود و بالعکس. (سطر 26)

پس از تغییر اندازه تصویر جدیدی در حافظه ایجاد می‌شود. (سطر 28)

سپس تنظیمات گرافیکی لازم بروی تصویر جدید اعمال می‌شود. (سطر 30 تا 34)

تصویر اصلی با توجه به اندازه جدید تغییر کرده و در تصویر جدید قرار می‌گیرد. (سطر 36)

در نهایت تصویر جدید با فرمت وارد شده در متد ذخیره شده و به عنوان خروجی متد برگشت داده می‌شود. (سطر 37 تا 40)

خروجی این متد نیز آرایه بایتی می‌باشد که به سادگی می‌توانید از آن برای ذخیره تصاویر در دیتابیس استفاده نمایید.

نحوه استفاده از این تابع در ASP.NET می‌تواند به صورت زیر باشد :

```
byte[] oldImage = FileUploadImage.FileBytes;
byte[] target = oldImage.ResizeImageFile(100, ImageFormat.Jpeg);
```

در واقع فراخوانی مذکور تصویر ورودی را به اندازه 100 پیکسل تغییر داده و در ارایه بایتی به نام target ذخیره می‌کند.
در بخش بعد در زمان نمایش تصویر به کاربر آن را تغییر اندازه خواهیم داد.
لازم به ذکر است که کدهای تغییر اندازه از StarterKitهای میکروسافت کپی‌برداری شده است.

نظرات خوانندگان

نویسنده: محسن.د
تاریخ: ۱۳۹۱/۰۹/۱۴ ۱۳:۲۴

بسیار عالی

در asp.net MVC کلاس webImage در فضانام System.Web.Helpers این امکانات رو در اختیار توسعه دهنده قرار میده .

نویسنده: صابر فتح الهی
تاریخ: ۱۳۹۱/۰۹/۱۴ ۱۵:۰

بله دیدم جالب بود واسم، اما تغییر اندازه ندیدم کدوم متده؟

نویسنده: محسن.د
تاریخ: ۱۳۹۱/۰۹/۱۴ ۱۸:۴

متد WebImage.Resize

نویسنده: پ.م
تاریخ: ۱۳۹۳/۰۲/۲۷ ۱۴:۳۶

لطفا در مورد اینکه در زمان واکشی تصاویر از دیتابیس (کش کنیم) توضیح دهید
چون خیلی سرعت کار با سایت پایین میاد
مثلا هر بار کاربر که بخواد از صفحه ای به صفحه ای دیگه بره تصاویر از نو بارگزاری میشن و این سرعت سایت رو کند میکنه

نویسنده: وحید نصیری
تاریخ: ۱۳۹۳/۰۲/۲۷ ۱۴:۳۹

به مطلب و مآخذ « مدیریت درخواست‌های شرطی در ASP.NET MVC » مراجعه کنید. نکات کلی در بحث عنوان شده. در مآخذ آن پیاده سازی‌های مخصوص وب فرم‌ها هم هست.

نویسنده: رضایی
تاریخ: ۱۳۹۳/۰۵/۳۱ ۱۹:۱۷

سلام؛ موقع تغییر سایز یک تصویر png به png پس زمینه رو مشکی میکنه. آیا راهی هست در صورتی که تصویر اولیه png باشه تصویر نهایی هم بدون پس زمینه باشه.

نویسنده: وحید نصیری
تاریخ: ۱۳۹۳/۰۶/۲۰ ۱۳:۵

Resize PNG Image and Keep Transparent Background

نویسنده: میثم شریفی
تاریخ: ۱۳۹۳/۱۲/۰۳ ۱۹:۲۵

با تشکر مقاله بسیار خوبی بود.
ImageResizer یک کتابخانه متن باز هست که امکانات کاملی ارائه کرده و helper خوبی هم برای mvc داره

عنوان: ارسال انواع بی نام (Anonymous) بازگشتی توسط Entity framework به توابع خارجی
نویسنده: سید مهران موسوی
تاریخ: ۱۳۹۱/۰۹/۲۱ ۱۶:۵۱
آدرس: www.dotnettips.info
برچسب‌ها: C#, Entity framework, ASP.Net MVC, Anonymous Type

فرض کنید ساختار زیر را در مدل ساخته شده به وسیله‌ی Entity framework در پروژه‌ی خود داریم .

جدول Post با جدول GroupsDetail ارتباط یک به چند و در مقابل ان جدول GroupsDetail با جدول PostGroup ارتباط چند به یک دارد . به زبان ساده ما تعدادی گروه بندی برای مطالب داریم (در جدول PostGroup) و میتوانیم برای هر مطلب تعدادی از گروه‌ها را در جدول GroupsDetail مشخص کنیم ...

فکر میکنم همین مقدار توضیح به اندازه‌ی کافی برای درک روابط مشخص شده در مدل رابطه ای مفروض کفایت کند.

حالا فرض کنید ما میخواهیم لیستی از عنوان مطالب موجود به همراه [نام] گروه‌های هر مطلب را داشته باشیم .

توجه شما رو به قطعه کد ساده‌ی زیر جلب میکنم:

```
var context = new Models.EntitiesConnection();
var query = context.Posts.Select(pst => new {
            id = pst.id,
            Title = pst.Title,
            GNames = pst.GroupsDetails.Select(grd => new { Name = grd.PostGroup.name })
            })
            .OrderByDescending(c => c.id)
            .ToList();
```

همانطور که شاهد هستید در قطعه کد بالا توسط خواص راهبری (Navigation Properties) به صورت مستقیم نام گروه‌های ثبت شده برای هر مطلب در جدول GroupsDetail را از جدول PostGroup استخراج کردیم . خب تا اینجا مسئله ای وجود نداشت (البته با ORMها این کار بسیار سهل و اسان شده است تا جایی که در حالت عادی برای همین کار باید کلی join نویسی کرد و .... در کل خدارو شکر که از دست کارهای تکراری و خسته کننده توسط این موجودات مهربان (ORM's) نجات یافتیم)...

خب میرسیم به ادامه‌ی مطلبمون . نتیجه‌ی این query چه خواهد بود ؟ کاملا واضح است که ما تعداد دلخواهی از فیلدها رو برای واکشی مشخص کردیم پس در نتیجه نوع داده ای که توسط این query بازگشت داده خواهد شد یک لیست از یک نوع بی نام میباشد ... چگونه از نتیجه‌ی بازگشتی این query در صورت ارسال اون به عنوان یک پارامتر به یک تابع استفاده کنیم ؟ اگر ما تمامی فیلدهای جدول Post را واکشی میکریم مقدار بازگشتی یک لیست از نوع Post بود که به راحتی قابل استفاده بود مثل مثال زیر

```
public bool myfunc(List<Post> query)
        {
            foreach (var item in query)
            {
            .....  string title =  item.Title;
            }
...return true;
        }
.
.
.

var context = new Models.EntitiesConnection();
var queryx = context.Posts.ToList();
.... myfunc(queryx );
```

اما حالا با یک لیست از یک نوع بی نام رو به رو هستیم ... چند راه مختلف برای دسترسی به این گونه از مقادیر بازگشتی داریم .

1 : استفاده از Reflection برای دسترسی به فیلدهای مشخص شده .

2 : تعریف یک مدل کامل بر اساس فیلدهای مشخص شده‌ی بازگشتی و ارسال یک لیست از نوع تعریف شده به تابع . (به نظرم این روش خسته کنندست و زیاد شکیل نیست چون برای هر query خاص مجبوریم یک مدل کلی تعریف کنیم و این باعث میشه تعداد زیادی مدل در نهایت داشته باشیم که استفاده‌ی زیادی از اونها نمیشه عملا)

3 : استفاده از یکی از روش‌های خلاقانه‌ی تبدیل نوع Anonymousها .

روش سوم روش مورد علاقه‌ی من هست و انعطاف بالایی داره و خیلی هم شیکو مجلسیه ! در ضمن این روش که قراره ذکر بشه کاربردهای فراوانی داره که این مورد فقط یکی از اونهاست

خب بریم سر اصل مطلب . به کد زیر توجه کنید :

```
public static List<T> CreateGenericListFromAnonymous<T>(object obj, T example)
        {
            return (List<T>)obj;
        }
        public static IEnumerable<T> CreateEmptyAnonymousIEnumerable<T>(T example)
        {
            return new List<T>(); ;
        }

////

        public bool myfunc(object query)
        {
            var cquery = CreateGenericListFromAnonymous(query,
                                new {
                                        id = 0,
                                        Title = string.Empty,
                                        GNames = CreateEmptyAnonymousIEnumerable(new { Name =
string.Empty })
                                });
            foreach (var item in cquery)
            {
```

```
                string title = item.Title;
                foreach (var gname in item.GNames)
                {
                    string gn = gname.Name;
                }
.
.
.
            }
            return false;
        }

.
.
.
var context = new Models.EntitiesConnection();
var query = context.Posts.Select(pst => new
                                {
                                     id = pst.id,
                                     Title = pst.Title,
                                     GNames = pst.GroupsDetails.Select(grd => new { Name =
grd.PostGroup.name })
                                }).OrderByDescending(c => c.id);
            myfunc(query.ToList());
```

در کد بالا به صورت تو در تو از انواع بی نام استفاده شده تا مطلب براتون خوب جا بیوفته . یعنی یک نوع بی نام که یکی از فیلدهای اون یک لیست از یک نوع بی نام دیگست ... خب میبینید که خیلی راحت با دو تابع ما تبدیل انواع بی نام رو به صورت inline انجام دادیم و ازش استفاده کردیم . بدون اینکه نیاز باشه ما یک مدل مجزا ایجاد کنیم ...

تابع CreateGenericListFromAnonymous : یک object رو میگیره و اون رو به یک لیست تبدیل میکنه بر اساس نوعی که به صورت inline براش مشخص میکنیم .

تابع CreateEmptyAnonymousIEnumerable یک لیست از نوع IEnumerable رو بر اساس نوعی که به صورت inline براش مشخص کردیم بر میگردونه . دلیل اینکه من در اینجا این تابع رو نوشتم این بود که ما در query یک فیلد با نام GNames داشتیم که مجموعه ای از نام گروه‌های هر مطلب بود که از نوع IEnumerable هستش . در واقع ما در اینجا نوع بی نامی داریم که یکی از فیلدهای اون یک لیست از یک نوع بی نام دیگست . امیدوارم مطلب براتون جا افتاده باشه.

دوستان عزیز سوالی بود در قسمت نظرات مطرح کنید.

نظرات خوانندگان

نویسنده: سعید
تاریخ: ۱۳۹۱/۰۹/۲۱ ۱۷:۱۱

با تشکر. روش‌های دیگری هم برای بازگشت انواع بی‌نام و نشان وجود دارند:

- خروجی متد را object تعریف کنیم

- خروجی متد را یک لیست از نوع dynamic (سی شارپ 4) تعریف کنیم

- خروجی متد را فقط ienumerable تعریف کنیم (نیازی به ذکر t ندارد الزاما)

نویسنده: سید مهران موسوی
تاریخ: ۱۳۹۱/۰۹/۲۱ ۲۰:۴۷

ممنون از نکاتی که ذکر کردید . از نکات ذکر شده برجسته ترینش استفاده از انواع dynamic هستش که کار رو ساده میکنه ولی خب مشکلاتی رو هم به وجود میاره مثلا اشکال زدایی رو سخت میکنه. هر چند که این نوع کار خودش رو با Reflection در زمان اجرا انجام میده و استفاده از اون رو ساده میکنه ولی خب استفاده مستقیم از روش‌های سطح پایین‌تر مزایایی رو هم داره مثلا مثال زیر رو در نظر بگیرید

```
public bool sample(object query)
        {
            System.Reflection.BindingFlags flags = System.Reflection.BindingFlags.Public |
                System.Reflection.BindingFlags.NonPublic |
                System.Reflection.BindingFlags.Static |
                System.Reflection.BindingFlags.Instance |
                System.Reflection.BindingFlags.DeclaredOnly;
            foreach (var item in query.GetType().GetProperties(flags))
            {

                string test = item.GetValue(query,null).ToString();
                // Do Something ...
            }
            return false;
        }
```

خب حلا فرض کنید ما میخوایم یک Table Generator بسازیم که بر اساس لیست نتایج بازگشتی توسط query بازگشت داده شده توسط Entity framework یک جدول رو برامون ایجاد میکنه . طبیعی هست که هر بار ما یک نوع داده ای بی نام رو براش ارسال میکنیم اگه ما بخوایم نوع dynamic رو به عنوان پارامتر تابع تعریف کنیم نمیتونیم به فیلدهای نوع بی نام دسترسی داشته باشیم چرا که هر بار فیلدها فرق میکنه و این تابع قراره در زمان اجرا فیلدها رو تشخیص بده ولی در انواع dynamic ما نام فیلد رو در زمان طراحی مشخص میکنیم و در زمان اجرا توسط نوع dynamic بسط داده میشه که این موضوع واسه همچین مواردی کارایی نداره ... و باید در نظر داشته که انواع dynamic فقط میتونن به فیلدهای public دسترسی داشته باشن ولی ما با reflection میتونیم محدوده‌ی دسترسی رو مشخص کنیم...

و اما در رابطه با مطلب بالا : بر فرض ما مجموعه ای از داده‌ها رو توسط Entity framework واکشی کردیم و یک نوع بی نام داریم و حالا میخوایم مثلا با Table Generator فرضی دو جدول رو از همین یک بار واکشی ایجاد کنیم که هر کدوم شامل یک سری فیلدها هستن و یک سری فیلدها رو از نوع بی نام واکشی شده شامل نمیشن . ما میتونیم یک تابع داشته باشیم که لیست نوع بی نام مرجع رو براش ارسال کنیم و یک نوع بی نام هم به عنوان یک پارامتر به صورت inline براش بفرستیم که فیلدهای مورد نظرمون رو از نوع بی نام مرجع شامل میشه . حالا با کمی توسعه CreateGenericListFromAnonymous و مپ کردن نوع بی نام مرجع با نوع بی نام ارسال شده توسط پارامتر و استفاده از Reflection میتونیم فقط فیلدهایی رو که به صورت inline مشخص کردیم داشته باشیم و با یک بار واکشی اطلاعات چندین بار اون رو با شکل‌های مختلف پردازش کنیم و .... این فقط یک مثال بود و مطلب بالا صرفا ایده ای بود که دوستان بتونن کارهای خلاقانه ای رو از طریق اون انجام بدن ....

نویسنده: ایلیا
تاریخ: ۱۳۹۱/۰۹/۲۱ ۲۰:۵۹

عالی بود. من همیشه یک مدل کامل ایجاد می‌کردم. ولی حالا خیلی راحت شد. تشکر فراوان.

نویسنده: سجاد
تاریخ: ۱۳۹۱/۰۹/۲۲ ۱۷:۴۴

بسیار عالی ! اگه در مورد " **حالا با کمی توسعه CreateGenericListFromAnonymous و مپ کردن نوع بی نام مرجع با نوع بی نام ارسال شده توسط پارامتر و استفاده از Reflection میتونیم فقط فیلدهایی رو که به صورت inline مشخص کردیم داشته باشیم** " بیشتر توضیح بدین ممنون میشم

نویسنده: سید مهران موسوی
تاریخ: ۱۳۹۱/۰۹/۲۲ ۲۲:۱۹

خواهش میکنم قابل نداشت . دوست عزیز این مطلب واقعا پیچیده و تخصصی هستش من یک نمونه واستون نوشتم که کد رو براتون میزارم ولی برای فهم کاملش نیاز به اشنایی عمقی با ساختار دات نت و کار با رفلکشن داره که توضیحش در چند خط نمیگنجه بحثه یک کتاب کامله. این نمونه کد که نوشتم دقیقا همون چیزی هست که درخواست توضیحش رو دادید .

```
public static List<T> CreateGenericListFromAnonymous<T>(object obj, T example)
        {
            var newquery = new List<T>();
            var constructor = typeof(T).GetConstructors(
            System.Reflection.BindingFlags.Public | System.Reflection.BindingFlags.NonPublic |
System.Reflection.BindingFlags.Instance
            ).OrderBy(c => c.GetParameters().Length).First();
            foreach (var item in ((IEnumerable<object>)obj))
            {
                var mapobj = new object[example.GetType().GetProperties().Count()];
                int counter = 0;
                foreach (var itemmap in example.GetType().GetProperties())
                {

                    object value = item.GetType().GetProperty(itemmap.Name).GetValue(item, null);
                    Type t = itemmap.PropertyType;
                    mapobj[counter] = Convert.ChangeType(value, t);
                    counter++;
                }
                newquery.Add((T)constructor.Invoke(mapobj));
            }

            return newquery;
        }


var context = new Models.EntitiesConnection();
var query = context.Posts.Select(pst => new
                                {
                                     id = pst.id,
                                     Title = pst.Title,
                                      Likes = pst.Likes,
                                      Unlikes = pst.Unlikes
                                }).OrderByDescending(c => c.id);

object test_custom_casting = CreateGenericListFromAnonymous(query.ToList(), new { id = 0, Title =
string.Empty });
```

همینطور که میبینید نتیجه‌ی بازگشتی از یک query مرجع یک list شامل **فقط و فقط** فیلدهایی هست که به صورت inline توسط انواع بی نام مشخص شده ! امیدوارم مفید واقع شده باشه . یا حق

نویسنده: ناصر فرجی
تاریخ: ۱۳۹۲/۰۱/۱۸ ۱۲:۲۳

من همیشه از viewmodel استفاده میکنم و فک میکنم روش استانداردتری باشه. ممنون بابت اشتراک گذاری این مطلب.

نویسنده: pjimax
تاریخ: ۱۳۹۲/۰۲/۲۱ ۲۱:۹

با سلام و تشکر من مقاله شما رو در پروژه خودم پیاده سازی کردم.فقط جای دوتا چهارتا جدول join کردم اما return(List<T>)obj; خطا میده
لطفا راهنمایی کنید

نویسنده: محسن خان
تاریخ: ۱۳۹۲/۰۲/۲۱ ۲۱:۲۱

خطا رو اگر نوشته بودی الان جواب گرفته بودی!

نویسنده: pjimax
تاریخ: ۱۳۹۲/۰۲/۲۲ ۱۰:۵۳

من میخام جداول زیر رو به روش بالا با هم join و استفاده کنم

```
var query = pro.types.Select(ty => new
       {
           typID = ty.type_id,
           typ_nam = ty.type_nam,
           cat_names = ty.categories.Select(cat => new
           {
               catgID = cat.cat_id,
               catg_name = cat.cat_nam,
               items = pro.items.Select(itm => new
               {
                   item_ID = itm.item_id,
                   item_nam = itm.item_nam,
                   item_path = itm.url,
                   coments = pro.comments.Select(cmm => new
                   {
                       comm_title = cmm.comment_nam,
                       comm_mail = cmm.mail,
                       comm_text = cmm.text,
                   })

               })


           })

       }).ToList();

 bool l = myfunc(query);

  protected bool myfunc(object q)
```

```
    {
        var cquery = CreateGenericListFromAnonymous(q,
            new
            {
                typID = 0,
                typ_nam = string.Empty,
                cat_names = CreateEmptyAnonymousIEnumerable(new
                {
                    catgID = 0,
                    catg_name = string.Empty,
                    items = CreateEmptyAnonymousIEnumerable(new
                    {
                        item_ID = 0,
                        item_name = string.Empty,
                        item_path = string.Empty,
                        coments = CreateEmptyAnonymousIEnumerable(new
                        {
                            comm_title = string.Empty,
                            comm_mail = string.Empty,
                            comm_text = string.Empty,

                        })
                    })
                })
            });
        foreach (var item in cquery)
        {
            int ID = item.typID;
            string type_title = item.typ_nam;
            foreach (var Cname in item.cat_names)
            {
                int categoryID = Cname.catgID;
                string category_name = string.Empty;

                foreach (var Iname in Cname.items)
                {
                    int ItmID = Iname.item_ID;
                    string ItmName = Iname.item_name;
                    string ItmPath = Iname.item_path;
                    foreach (var cm in Iname.coments)
                    {
                        string com_title = cm.comm_title;
                        string com_mail = cm.comm_mail;
                        string com_text = cm.comm_text;
                    }
                }
            }
        }
        return true;
    }
```

```
public class DBupload
{
    projeEntities pro = new projeEntities();
    uploadEntity up = new uploadEntity();
    public static List<T> CreateGenericListFromAnonymous<T>(object obj, T example)
    {
        return (List<T>)obj;
    }
    public static IEnumerable
    {
        return new List<T>();
    }
    public uploadEntity m(){
        uploadEntity tup = new
        //var query = from c
        //            join b
        //            join d
        //            join e
```

InvalidCastException was unhandled by user code

Unable to cast object of type 'System.Collections.Generic.List`1
[<>f__AnonymousType3`3
[System.Int32,System.String,System.Collections.Generic.IEnumerable`1
[<>f__AnonymousType2`3
[System.Int32,System.String,System.Linq.IQueryable`1
[<>f__AnonymousType1`4

Troubleshooting tips:

When casting from a number, the value must be a number less than infinity.
Make sure the source type is convertible to the destination type.
Get general help for this exception.

Search for more Help Online...

Actions:

View Detail...
Copy exception detail to the clipboard

Errors | 10 Warnings | 0 Messa

Description

نویسنده: محسن خان
تاریخ: ۱۳۹۲/۰۲/۲۲ ۱۱:۳۷

```
IList<Types> typesList = context.Types.Include(x=>x.Categories)
                                      .Include(x=>x.Items).Include(x=>x.Comments).ToList();
```

اگر از EF استفاده می‌کنید، استفاده از متد Include کار جوین رو برای شما انجام می‌ده. بعدش نیازی به استفاده از متد CreateGenericListFromAnonymous که فقط یک سطح رو بررسی می‌کنه نیست. برای بررسی بیشتر از یک سطح باید متد بازگشتی نوشته بشه ولی در حالت شما واقعا نیازی نیست. ضمنا متد تصویر شما با متد نوشته شده یکی نیست.

عنوان: **چندین Submit در یک Html Form و انتساب Action های مجزا به هر یک از Submit ها در MVC**
نویسنده: سید مهران موسوی
تاریخ: ۱۳۹۱/۰۹/۲۲ ۱:۰
آدرس: www.dotnettips.info
برچسب‌ها: C#, ASP.Net MVC

تا به حال به این نکته برخورد کردید که برای یک فرم Html نیاز به چندین Submit داشته باشید که هر کدوم یک Action مجزا داشته باشن و یک کار متفاوت انجام بدن ؟
برای مثال فرمی داریم که داده‌های وارد شده در ان باید به دو صورت برای یک کاربر ارسال بشن یا از طریق پیامک یا از طریق ایمیل (این فقط یک مثال پیش فرض هست) و .... در حالت عادی ما در یک فرم نمیتونیم دو عدد Submit داشته باشیم که هر کدوم به یک Action جدا بسط داده بشه خب راه حل چیه ؟ شاید با خودتون بگید خب دو input از نوع radio قرار میدیم و در یک اکشن کنترل میکنیم که کدوم یکی انتخاب شده و عملیات رو با اون معیار انجام میدیم ... به نظرتون زیباتر نیست برای هر عملیات که ممکن باشه هر کدوم کاملا روال کاری متفاوتی داشته باشه یک Action وجود داشته باشه ؟ در این صورت خوانایی کد خیلی بالاتر میره و Unit Test هر Action کاملا مشخص هست که قراره چه فرایندی رو مورد تست قرار بده و مجبور نیستیم چندین حالت رو با عبارات شرطی از هم جدا کنیم و همه چی قاطی بشه با هم ... من در کل با امکاناتی که C# و MVC در اختیارم قرار میده حاظر نیستم تن به کد نویسی به صورت کلاسیک و قاطی پاتی بدم سعی میکنم با مطالعه‌ی سورس MVC بهترین حالت رو انتخاب کنم شما چطور ؟ معلومه که همه همینو میخوان پس بریم سر اصل مطلب .
قطعه کد Html و Razor ساده‌ی زیر رو در نظر بگیرید برای View :

```
@model Models.MyModel
@{
    Layout = null;
}
<!DOCTYPE html>
<html>
<head>
    <title>ViewPage1</title>
</head>
<body>
    <div>
        @using (Html.BeginForm("SendMessage", "Home", FormMethod.Post))
        {

            @Html.LabelFor(x => x.Name);
            @Html.TextBoxFor(x => x.Name);

            <input type="submit" value="ارسال توسط پیامک" name="Send_sms" />
            <input type="submit" value="ارسال توسط ایمیل" name="Send_email" />
        }
    </div>
</body>
</html>
```

خب ما دو تا Submit داریم . یکم اگه شیطنت کنید و مقادیر ارسال شده بعد از submit این فرم رو توسط ابزارهای مانیتورینگ بررسی کنید میبینید که روی هر کدوم از Submitها که کلیک میشه داده ای با نام اون که در خاصیت name اون و مقدار موجود در value اون همراه اون فرم به سرور ارسال میشه و اون یکی Submit از این اتفاق بی نصیب میمونه ... خب ما هم استفاده‌ی لازم رو از این موضوع شیرین میبریم و با یک تکنیک تهاجمی از این موضوع برای رسیدن به هدفمون استفاده میکنیم .

این هم کلاس Model ماست :

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Web;
using System.ComponentModel;

namespace Models
{
    public class MyModel
    {
        [DisplayName("نام خود را وارد کنید :")]
```

```
        public string Name { get; set; }
    }
}
```

و اما یک نکته‌ی دیگه . توجه داشته باشید که ما در قسمت View نام Action رو در فرم, SendMessage مشخص کردیم . ولی ... اصلا در واقع همچین اکشنی وجود نداره ! پس چرا ما همچین نامی رو واسه اکشن فرم گذاشتیم !؟
دلیل اینه که ما قصد داریم با یک ActionNameSelectorAttribute درخواست کاربر رو شکار کنیم و اون رو به اکشن دلخواه ارجاع بدیم ... جالبه نه ؟ ولی چه جوری ... کلاس زیر رو بهش دقت مضاعف کنید و در پروژتون ایجادش کنید :

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Web;
using System.Web.Mvc;
using System.Reflection;

namespace ActionHandlers
{
    public class SendMessageHandlerAttribute : ActionNameSelectorAttribute
    {
        public override bool IsValidName(ControllerContext controllerContext, string actionName,
MethodInfo methodInfo)
        {
            if (actionName.Equals(methodInfo.Name, StringComparison.InvariantCultureIgnoreCase))
                return true;

            if (!actionName.Equals("SendMessage", StringComparison.InvariantCultureIgnoreCase))
                return false;

            var request = controllerContext.RequestContext.HttpContext.Request;
            return request[methodInfo.Name] != null;
        }
    }
}
```

خب حالا بخش Controller رو بهش دقت کنید که ما در اون دو اکشن رو با نام هایی که برای هر Submit مشخص کردیم مینویسیم و ActionNameSelectorAttribute نوشته شده رو به اونها بسط میدیم.

```
[SendMessageHandler ]
        [HttpPost]
        public ContentResult Send_sms(MyModel mdl)
        {
            /// Do something ...
           return string.Empty ;
        }

        [SendMessageHandler ]
        [HttpPost]
        public ContentResult Send_email(MyModel mdl)
        {
           /// Do something ...
          return  string.Empty;
        }
```

خب حالا بعد از کلیک بر روی هر Submit اکشن متناظر با اون اجرا میشه . بعد از ارسال درخواست به سرور MVC در بین اکشن‌های موجود در Controller مشخص شده به دنبال اکشن معین شده میگرده و وقتی به اکشن‌های ما میرسه میبینه عجب ! اون دوتا ActionNameSelectorAttribute سفارشی دارن پس میره ببینه چه خبره اونجا که ما با یک حرکت تهاجمی بررسی میکنیم که اگه نام اکشن مشخص شده در فرم با نام اکشن در حال بررسی مساوی بود که همینو اجرا کن (یعنی ما میتونی اکشنی با نام SendMessage هم داشته باشیم) . اگه نام اکشن مشخص شده در فرم اون نامی نبود که ما میخوایم که کلا بیخیال هندل کردن اکشن میشیم میزاریم خود MVC تصمیم بگیره . و در اخر بررسی میکنیم که ایا در درخواست جاری مقداری با نام اکشن در حال بررسی وجود داره !؟ اگه داشت یعنی همون Submit که ما میخوایم وصل بشه به این اکشن کلیک شده پس اکشن در حال بررسی رو بسط میدیم به درخواست ارسال شده ... به همین سادگی ...

پیروز و موفق باشید .

نظرات خوانندگان

نویسنده: سعید
تاریخ: ۱۳۹۱/۰۹/۲۲ ۱۷:۳۱

راه جالبی است. نمی‌دونستم که ActionNameSelectorAttribute وجود خارجی دارد!
چند راه حل دیگر:
الف) استفاده از قابلیت‌های binding . مثلا اگر نام پارامترها را به نام همان دکمه‌های موجود تنظیم کنیم، این نام موقع دریافت از دکمه کلیک شده، نال نیست:

```
[HttpPost]
public ActionResult MyAction([other params here], string buttonName1, string buttonName2, etc)
{
     if(!string.IsNullOrEmtpy(buttonName1)) { //button1 was clicked}
}
```

ب) میشه از ویژگی‌های جدید HTML5 مثل data-form-action استفاده کرد:

```
<input type="submit" value="Standard action">
<input data-form-action="@Url.Action("mysecondaction")" type="submit" value="Second action">
```

بعد موقع ارسال کمی از jQuery استفاده کرد

```
$(document).on('click', '[type="submit"][data-form-action]', function(event) {
    var $this = $(this),
        formAction = $this.attr('data-form-action'),
        $form = $($this.closest('form'));
        $form.attr('action', formAction);
});
```

نویسنده: سید مهران موسوی
تاریخ: ۱۳۹۱/۰۹/۲۲ ۱۸:۱

ممنون دوست عزیز که توجه میکنی به مطالب بنده . به جز روشی که در مقاله بالا ذکر شد استفاده از data-form-action موجود در نگارش 5 از HTML اصولیترین روش یا بهتره بگیم تمیزترین روش در بین مواردی هست که ذکر کردید ولی خب برنامه نویسای وب همیشه با مرورگرهای مختلف در جنگ هستن و فعلا زیاد استفاده از این روش جایز نیست چون بعضی از مرورگرهای کمی قدیمیتر پشتیبانی نمیکنند پس بهتره از تکنیکهای مورد اطمینانتر استفاده کنیم که نمونش در مقاله‌ی بالا ذکر شد ...

این نمونه روش‌های خلاقانه و جالب از جمله ActionNameSelectorAttribute به وفور در نسخه‌های جدید mvc وجود داره خوشبختانه . در مقالات بعدیم سعی میکنم مطالب تخصصی و تکنیکهای جالب دیگرو هم با دوستان به اشتراک بزارم . انصافا mvc یکی از شیرین‌ترین تکنلوژی هایی هست که هر سری سورسش رو مطالعه میکنم راه کارهای جدیدی برای پیاده سازی نرم افزارها توسط اون تکنیک‌ها دستگیرم میشه ...
پاینده باشید و موفق

عنوان: **پیاده سازی پروژه نقاشی (Paint) به صورت شی گرا #2**
نویسنده: صابر فتح الهی
تاریخ: ۱۳۹۱/۱۱/۱۸ ۰:۳۵
آدرس: www.dotnettips.info
برچسب‌ها: OOP, C#, Project, Paint

در ادامه مطلب پیاده سازی پروژه نقاشی (Paint) به صورت شی گرا #1 به تشریح مابقی کلاس‌های برنامه می‌پردازیم.

با توجه به تجزیه و تحلیل انجام شده تمامی اشیا از کلاس پایه به نام Shape ارث بری دارند حال به توضیح کدهای این کلاس می‌پردازیم. (به دلیل اینکه توضیحات این کلاس در دو پست نوشته خواهد شد برای این کلاس‌ها از partial class استفاده شده است)

```
using System;
using System.Drawing;
using System.Drawing.Drawing2D;
using System.Net;

namespace PWS.ObjectOrientedPaint.Models
{
    /// <summary>
    /// Shape (Base Class)
    /// </summary>
    public abstract partial class Shape
    {
        #region Fields (1)

        private Brush _backgroundBrush;

        #endregion Fields

        #region Properties (16)

        /// <summary>
        /// Gets or sets the brush.
        /// </summary>
        /// <value>
        /// The brush.
        /// </value>
        public Brush BackgroundBrush
        {
            get { return _backgroundBrush ?? (_backgroundBrush = new SolidBrush(BackgroundColor)); }
            private set
            {
                _backgroundBrush = value ?? new SolidBrush(BackgroundColor);
            }
        }

        /// <summary>
        /// Gets or sets the color of the background.
        /// </summary>
        /// <value>
        /// The color of the background.
        /// </value>
        public Color BackgroundColor { get; set; }

        /// <summary>
        /// Gets or sets the end point.
        /// </summary>
        /// <value>
        /// The end point.
        /// </value>
        public PointF EndPoint { get; set; }

        /// <summary>
        /// Gets or sets the color of the fore.
        /// </summary>
        /// <value>
        /// The color of the fore.
        /// </value>
        public Color ForeColor { get; set; }

        /// <summary>
        /// Gets or sets the height.
        /// </summary>
        /// <value>
```

```
        /// The height.
        /// </value>
        public float Height
        {
            get
            {
                return Math.Abs(StartPoint.Y - EndPoint.Y);
            }
            set
            {
                if (value > 0)
                    EndPoint = new PointF(EndPoint.X, StartPoint.Y + value);
            }
        }

        /// <summary>
        /// Gets or sets a value indicating whether this instance is fill.
        /// </summary>
        /// <value>
        ///   <c>true</c> if this instance is fill; otherwise, <c>false</c>.
        /// </value>
        public bool IsFill { get; set; }

        /// <summary>
        /// Gets or sets a value indicating whether this instance is selected.
        /// </summary>
        /// <value>
        /// <c>true</c> if this instance is selected; otherwise, <c>false</c>.
        /// </value>
        public bool IsSelected { get; set; }

        /// <summary>
        /// Gets or sets my pen.
        /// </summary>
        /// <value>
        /// My pen.
        /// </value>
        public Pen Pen
        {
            get
            {
                return new Pen(ForeColor, Thickness);
            }
        }

        /// <summary>
        /// Gets or sets the type of the shape.
        /// </summary>
        /// <value>
        /// The type of the shape.
        /// </value>
        public ShapeType ShapeType { get; protected set; }

        /// <summary>
        /// Gets the size.
        /// </summary>
        /// <value>
        /// The size.
        /// </value>
        public SizeF Size
        {
            get
            {
                return new SizeF(Width, Height);
            }
        }

        /// <summary>
        /// Gets or sets the start point.
        /// </summary>
        /// <value>
        /// The start point.
        /// </value>
        public PointF StartPoint { get; set; }

        /// <summary>
        /// Gets or sets the thickness.
        /// </summary>
        /// <value>
        /// The thickness.
        /// </value>
```

```
        public byte Thickness { get; set; }

        /// <summary>
        /// Gets or sets the width.
        /// </summary>
        /// <value>
        /// The width.
        /// </value>
        public float Width
        {
            get
            {
                return Math.Abs(StartPoint.X - EndPoint.X);
            }
            set
            {
                if (value > 0)
                    EndPoint = new PointF(StartPoint.X + value, EndPoint.Y);
            }
        }

        /// <summary>
        /// Gets or sets the X.
        /// </summary>
        /// <value>
        /// The X.
        /// </value>
        public float X
        {
            get
            {
                return StartPoint.X;
            }
            set
            {
                if (value > 0)
                    StartPoint = new PointF(value, StartPoint.Y);
            }
        }

        /// <summary>
        /// Gets or sets the Y.
        /// </summary>
        /// <value>
        /// The Y.
        /// </value>
        public float Y
        {
            get
            {
                return StartPoint.Y;
            }
            set
            {
                if (value > 0)
                    StartPoint = new PointF(StartPoint.X, value);
            }
        }

        /// <summary>
        /// Gets or sets the index of the Z.
        /// </summary>
        /// <value>
        /// The index of the Z.
        /// </value>
        public int Zindex { get; set; }

        #endregion Properties
        }
}
```

ابتدا به تشریح خصوصیات کلاس می‌پردازیم:

خصوصیات:

BackgroundColor : در صورتی که شی مورد نظر به صورت توپررسم شود، این خاصیت رنگ پس زمینه شی را مشخص می‌کند.
BackgroundBrush : خاصیتی است که با توجه به خاصیت BackgroundColor یک الگوی پر کردن زمینه شی می‌سازد.
StartPoint : نقطه شروع شی را در خود نگهداری می‌کند.
EndPoint : نقطه انتهای شی را در خود نگهداری می‌کند. (قبلا گفته شد که هر شی را در صورتی که در یک مستطیل فرض کنیم یک نقطه شروع و یک نقطه پایان دارد)
ForeColor : رنگ قلم ترسیم شی مورد نظر را تعیین می‌کند.
Height : ارتفاع شی مورد نظر را تعیین می‌کند (این خصوصیت اختلاف عمودی StartPoint.Y و EndPoint.Y را محاسبه می‌کند و در زمان مقدار دهی EndPoint جدیدی ایجاد می‌کند).
Width : عرض شی مورد نظر را تعیین می‌کند (این خصوصیت اختلاف افقیStartPoint.X و EndPoint.X را محاسبه می‌کند و در زمان مقدار دهی EndPoint جدیدی ایجاد می‌کند).
IsFill : این خصوصیت تعیین کننده توپر و یا توخالی بودن شی است.
IsSelected : این خاصیت تعیین می‌کند که آیا شی انتخاب شده است یا خیر (در زمان انتخاب شی چهار مربع کوچک روی شی رسم می‌شود).
Pen : قلم خط ترسیم شی را مشخص می‌کند. (قلم با ضخامت دلخواه)
ShapeType : این خصوصیت نوع شی را مشخص می‌کند (این خاصیت بیشتر برای زمان پیش نمایش ترسیم شی در زمان اجراست البته به نظر خودم اضافه هست اما راه بهتری به ذهنم نرسید)
Size : با استفاده از خصوصیات Height و Width ایجاد شده و تعیین کننده Size شی است.
Thickness : ضخامت خط ترسیمی شی را مشخص می‌کند، این خاصیت در خصوصیت Pen استفاده شده است.
X : مقدار افقی نقطه شروع شی را تعیین می‌کند در واقع StartPoint.X را برمی‌گرداند (این خاصیت اضافی بوده و جهت راحتی کار استفاده شده می‌توان آن را ننوشت).
Y : مقدار عمودی نقطه شروع شی را تعیین می‌کند در واقع StartPoint.Y را برمی‌گرداند (این خاصیت اضافی بوده و جهت راحتی کار استفاده شده می‌توان آن را ننوشت).
Zindex : در زمان ترسیم اشیا ممکن است اشیا روی هم ترسیم شوند، در واقع Zindex تعیین کننده عمق شی روی بوم گرافیکی است.

در پست بعدی به توضیح متدهای این کلاس می‌پردازیم.

نظرات خوانندگان

نویسنده: بتیسا
تاریخ: ۱۳۹۱/۱۱/۱۸ ۱۰:۲۶

با سلام

از مطلب مفیدی که تهیه کردید ممنون.
می‌شود از طریق خاصیت Brush که فعلا فقط خواندنی هست، طرح‌های مختلفی برای پس زمینه اشیاع ایجاد کرد. مانند Paint.net و یا MS Paint.

اگر به صورت زیر تعریف کنیم فکر می‌کنم کمی کامل‌تر باشه!

```
private Brush _backgroundBrush;

        /// <summary>
        /// Gets or sets the brush.
        /// </summary>
        /// <value>
        /// The brush.
        /// </value>
        public Brush BackgroundBrush
        {
            get
            {
                return _backgroundBrush;
            }
            private set
            {
                _backgroundBrush = (value != null) ? value : new SolidBrush(BackgroundColor);
            }
        }
        //-------------------------------------[Methode for set brush]--------------------------------
-----
        public virtual void SetBackgroundBrushAsHatch(HatchStyle hatchStyle)
        {
            HatchBrush brush = new HatchBrush(hatchStyle, BackgroundColor);
            BackgroundBrush = brush;
        }

        public virtual void SetBackgroundBrushAsSolid()
        {
            SolidBrush brush = new SolidBrush(BackgroundColor);
            BackgroundBrush = brush;
        }

        public virtual void SetBackgroundBrushAsLinearGradient()
        {
            LinearGradientBrush brush = new LinearGradientBrush(StartPoint, EndPoint, ForeColor,
BackgroundColor);
            BackgroundBrush = brush;
        }
```

که اگر بخواهیم میتونیم باز بیشتر Customize بکنیمشون.

نویسنده: صابر فتح الهی
تاریخ: ۱۳۹۱/۱۱/۱۸ ۱۰:۴۰

بله کاملا حق با شماست خیلی کارها می‌شه روش انجام داد (قصد آموزش یک مبحث به زبان ساده بود)==> نظر شما اعمال شد.
سعی می‌کنم در زمان ارائه پروژه نهایی همه اینها اعمال بشه

عنوان: **پیاده سازی پروژه نقاشی (Paint) به صورت شی گرا #3**
نویسنده: صابر فتح الهی
تاریخ: ۱۳۹۱/۱۱/۱۸ ۲:۱۵
آدرس: www.dotnettips.info
برچسب‌ها: OOP, C#, Project, Paint

در ادامه مطالب قبل

پیاده سازی پروژه نقاشی (Paint) به صورت شی گرا #1
پیاده سازی پروژه نقاشی (Paint) به صورت شی گرا #2

قبل از شروع توضیحات متدهای کلاس Shape در ادامه پست‌های قبل در ^ و ^ ابتدا به تشریح یک تصویر می‌پردازیم.

خوب همانگونه که در تصویر بالا مشاده می‌نمایید، برای رسم یک شی چهار حالت متفاوت ممکن است پیش بیاید. (دقت کنید که ربع اول محور مختصات روی بوم گرافیکی قرار گرفته است، در واقع گوشه بالا و سمت چپ بوم گرافیکی نقطه (0 و 0) محور مختصات است و عرض بوم گرافیکی محور Xها و ارتفاع بوم گرافیکی محور Yها را نشان می‌دهد)

در این حالت StartPoint.X < EndPoint.X و StartPoint.Y < EndPoint.Y خواهد بود. (StartPoint نقطه ای است که ابتدا ماوس شروع به ترسیم می‌کند، و EndPoint زمانی است که ماوس رها شده و پایان ترسیم را مشخص می‌کند.)
در این حالت StartPoint.X > EndPoint.X و StartPoint.Y > EndPoint.Y خواهد بود.
در این حالت StartPoint.X > EndPoint.X و StartPoint.Y > EndPoint.Y خواهد بود.
در این حالت StartPoint.X < EndPoint.X و StartPoint.Y > EndPoint.Y خواهد بود.

ابتدا یک کلاس کمکی به صورت استاتیک تعریف می‌کنیم که متدی جهت پیش نمایش رسم شی در حالت جابجایی ، رسم، و تغییر اندازه دارد.

```
using System;
using System.Drawing;

namespace PWS.ObjectOrientedPaint.Models
{
    /// <summary>
    /// Helpers
    /// </summary>
    public static class Helpers
    {
        /// <summary>
        /// Draws the preview.
        /// </summary>
        /// <param name="g">The g.</param>
        /// <param name="startPoint">The start point.</param>
        /// <param name="endPoint">The end point.</param>
        /// <param name="foreColor">Color of the fore.</param>
        /// <param name="thickness">The thickness.</param>
        /// <param name="isFill">if set to <c>true</c> [is fill].</param>
        /// <param name="backgroundBrush">The background brush.</param>
        /// <param name="shapeType">Type of the shape.</param>
        public static void DrawPreview(Graphics g, PointF startPoint, PointF endPoint, Color foreColor,
byte thickness, bool isFill, Brush backgroundBrush, ShapeType shapeType)
        {
            float x = 0, y = 0;
            float width = Math.Abs(endPoint.X - startPoint.X);
            float height = Math.Abs(endPoint.Y - startPoint.Y);
            if (startPoint.X <= endPoint.X && startPoint.Y <= endPoint.Y)
            {
                x = startPoint.X;
                y = startPoint.Y;
            }
            else if (startPoint.X >= endPoint.X && startPoint.Y >= endPoint.Y)
            {
                x = endPoint.X;
                y = endPoint.Y;
            }
            else if (startPoint.X >= endPoint.X && startPoint.Y <= endPoint.Y)
            {
                x = endPoint.X;
                y = startPoint.Y;
            }
            else if (startPoint.X <= endPoint.X && startPoint.Y >= endPoint.Y)
            {
                x = startPoint.X;
                y = endPoint.Y;
            }

            switch (shapeType)
            {
                case ShapeType.Ellipse:
                    if (isFill)
                        g.FillEllipse(backgroundBrush, x, y, width, height);
                    //else
                    g.DrawEllipse(new Pen(foreColor, thickness), x, y, width, height);
                    break;
                case ShapeType.Rectangle:
                    if (isFill)
                        g.FillRectangle(backgroundBrush, x, y, width, height);
                    //else
                    g.DrawRectangle(new Pen(foreColor, thickness), x, y, width, height);
                    break;
                case ShapeType.Circle:
                    float raduis = Math.Max(width, height);

                    if (isFill)
                        g.FillEllipse(backgroundBrush, x, y, raduis, raduis);
                    //else
                    g.DrawEllipse(new Pen(foreColor, thickness), x, y, raduis, raduis);
                    break;
                case ShapeType.Square:
                    float side = Math.Max(width, height);

                    if (isFill)
                        g.FillRectangle(backgroundBrush, x, y, side, side);
                    //else
                    g.DrawRectangle(new Pen(foreColor, thickness), x, y, side, side);
```

```
                    break;
                case ShapeType.Line:
                    g.DrawLine(new Pen(foreColor, thickness), startPoint, endPoint);
                    break;
                case ShapeType.Diamond:
                    var points = new PointF[4];
                    points[0] = new PointF(x + width / 2, y);
                    points[1] = new PointF(x + width, y + height / 2);
                    points[2] = new PointF(x + width / 2, y + height);
                    points[3] = new PointF(x, y + height / 2);
                    if (isFill)
                        g.FillPolygon(backgroundBrush, points);
                    //else
                    g.DrawPolygon(new Pen(foreColor, thickness), points);
                    break;
                case ShapeType.Triangle:
                    var tPoints = new PointF[3];
                    tPoints[0] = new PointF(x + width / 2, y);
                    tPoints[1] = new PointF(x + width, y + height);
                    tPoints[2] = new PointF(x, y + height);
                    if (isFill)
                        g.FillPolygon(backgroundBrush, tPoints);
                    //else
                    g.DrawPolygon(new Pen(foreColor, thickness), tPoints);
                    break;
            }
            if (shapeType != ShapeType.Line)
            {
                g.DrawString(String.Format("({0},{1})", x, y), new Font(new FontFamily("Tahoma"), 10),
new SolidBrush(foreColor), x - 20, y - 25);
                g.DrawString(String.Format("({0},{1})", x + width, y + height), new Font(new
FontFamily("Tahoma"), 10), new SolidBrush(foreColor), x + width - 20, y + height + 5);
            }
            else
            {
                g.DrawString(String.Format("({0},{1})", startPoint.X, startPoint.Y), new Font(new
FontFamily("Tahoma"), 10), new SolidBrush(foreColor), startPoint.X - 20, startPoint.Y - 25);
                g.DrawString(String.Format("({0},{1})", endPoint.X, endPoint.Y), new Font(new
FontFamily("Tahoma"), 10), new SolidBrush(foreColor), endPoint.X - 20, endPoint.Y + 5);
            }

        }
    }
}
```

متد های این کلاس:

DrawPreview : این متد پیش نمایشی برای شی در زمان ترسیم، جابجایی و تغییر اندازه آماده می‌کند، پارامترهای آن عبارتند از : **بوم گرافیکی** ، **نقطه شروع** ، **نقطه پایان** و **رنگ قلم ترسیم** پیش نمایش شی، **ضخامت خط** ، **آیا شی توپر باشد** ؟، **الگوی پر کردن پس زمینه شی** ، و **نوع شی ترسیمی** می‌باشد.

در ادامه پست‌های قبل ادامه کد کلاس Shape را تشریح می‌کنیم.

```
using System;
using System.Drawing;
using System.Drawing.Drawing2D;
using System.Net;

namespace PWS.ObjectOrientedPaint.Models
{
    /// <summary>
    /// Shape (Base Class)
    /// </summary>
    public abstract partial class Shape
    {
#region Constructors (2)

        /// <summary>
        /// Initializes a new instance of the <see cref="Shape" /> class.
        /// </summary>
        /// <param name="startPoint">The start point.</param>
        /// <param name="endPoint">The end point.</param>
        /// <param name="zIndex">Index of the z.</param>
        /// <param name="foreColor">Color of the fore.</param>
        /// <param name="thickness">The thickness.</param>
```

```
        /// <param name="isFill">if set to <c>true</c> [is fill].</param>
        /// <param name="backgroundColor">Color of the background.</param>
        protected Shape(PointF startPoint, PointF endPoint, int zIndex, Color foreColor, byte
thickness, bool isFill, Color backgroundColor)
        {
            CalulateLocationAndSize(startPoint, endPoint);
            Zindex = zIndex;
            ForeColor = foreColor;
            Thickness = thickness;
            IsFill = isFill;
            BackgroundColor = backgroundColor;
        }

        /// <summary>
        /// Initializes a new instance of the <see cref="Shape" /> class.
        /// </summary>
        protected Shape() { }

#endregion Constructors

#region Methods (10)

// Public Methods (9)

        /// <summary>
        /// Draws the specified g.
        /// </summary>
        /// <param name="g">The g.</param>
        public virtual void Draw(Graphics g)
        {
            if (!IsSelected) return;
            float diff = Thickness + 4;
            Color myColor = Color.DarkSeaGreen;
            g.DrawString(String.Format("({0},{1})", StartPoint.X, StartPoint.Y), new Font(new
FontFamily("Tahoma"), 10), new SolidBrush(myColor), StartPoint.X - 20, StartPoint.Y - 25);
            g.DrawString(String.Format("({0},{1})", EndPoint.X, EndPoint.Y), new Font(new
FontFamily("Tahoma"), 10), new SolidBrush(myColor), EndPoint.X - 20, EndPoint.Y + 5);
            if (ShapeType != ShapeType.Line)
            {
                g.DrawRectangle(new Pen(myColor), X, Y, Width, Height);

                //  1 2 3
                //  8   4
                //  7 6 5
                var point1 = new PointF(StartPoint.X - diff / 2, StartPoint.Y - diff / 2);
                var point2 = new PointF((StartPoint.X - diff / 2 + EndPoint.X) / 2, StartPoint.Y - diff
/ 2);
                var point3 = new PointF(EndPoint.X - diff / 2, StartPoint.Y - diff / 2);
                var point4 = new PointF(EndPoint.X - diff / 2, (EndPoint.Y + StartPoint.Y) / 2 - diff /
2);
                var point5 = new PointF(EndPoint.X - diff / 2, EndPoint.Y - diff / 2);
                var point6 = new PointF((StartPoint.X - diff / 2 + EndPoint.X) / 2, EndPoint.Y - diff /
2);
                var point7 = new PointF(StartPoint.X - diff / 2, EndPoint.Y - diff / 2);
                var point8 = new PointF(StartPoint.X - diff / 2, (EndPoint.Y + StartPoint.Y) / 2 - diff
/ 2);


                g.FillRectangle(new SolidBrush(myColor), point1.X, point1.Y, diff, diff);
                g.FillRectangle(new SolidBrush(myColor), point2.X, point2.Y, diff, diff);
                g.FillRectangle(new SolidBrush(myColor), point3.X, point3.Y, diff, diff);
                g.FillRectangle(new SolidBrush(myColor), point4.X, point4.Y, diff, diff);
                g.FillRectangle(new SolidBrush(myColor), point5.X, point5.Y, diff, diff);
                g.FillRectangle(new SolidBrush(myColor), point6.X, point6.Y, diff, diff);
                g.FillRectangle(new SolidBrush(myColor), point7.X, point7.Y, diff, diff);
                g.FillRectangle(new SolidBrush(myColor), point8.X, point8.Y, diff, diff);
            }
            else
            {
                var point1 = new PointF(StartPoint.X - diff / 2, StartPoint.Y - diff / 2);
                var point2 = new PointF(EndPoint.X - diff / 2, EndPoint.Y - diff / 2);
                g.FillRectangle(new SolidBrush(myColor), point1.X, point1.Y, diff, diff);
                g.FillRectangle(new SolidBrush(myColor), point2.X, point2.Y, diff, diff);
            }
        }

        /// <summary>
        /// Points the in sahpe.
        /// </summary>
        /// <param name="point">The point.</param>
        /// <param name="tolerance">The tolerance.</param>
```

```
        /// <returns>
        ///   <c>true</c> if [has point in sahpe] [the specified point]; otherwise, <c>false</c>.
        /// </returns>
        public virtual bool HasPointInSahpe(PointF point, byte tolerance = 5)
        {
            return point.X > (StartPoint.X - tolerance) && point.X < (EndPoint.X + tolerance) &&
point.Y > (StartPoint.Y - tolerance) && point.Y < (EndPoint.Y + tolerance);
        }

        /// <summary>
        /// Moves the specified location.
        /// </summary>
        /// <param name="location">The location.</param>
        /// <returns></returns>
        public virtual PointF Move(Point location)
        {
            StartPoint = new PointF(location.X, location.Y);
            EndPoint = new PointF(location.X + Width, location.Y + Height);
            return StartPoint;
        }

        /// <summary>
        /// Moves the specified dx.
        /// </summary>
        /// <param name="dx">The dx.</param>
        /// <param name="dy">The dy.</param>
        /// <returns></returns>
        public virtual PointF Move(int dx, int dy)
        {
            StartPoint = new PointF(StartPoint.X + dx, StartPoint.Y + dy);
            EndPoint = new PointF(EndPoint.X + dx, EndPoint.Y + dy);
            return StartPoint;
        }

        /// <summary>
        /// Resizes the specified dx.
        /// </summary>
        /// <param name="dx">The dx.</param>
        /// <param name="dy">The dy.</param>
        /// <returns></returns>
        public virtual SizeF Resize(int dx, int dy)
        {
            EndPoint = new PointF(EndPoint.X + dx, EndPoint.Y + dy);
            return new SizeF(Width, Height);
        }

        /// <summary>
        /// Resizes the specified start point.
        /// </summary>
        /// <param name="startPoint">The start point.</param>
        /// <param name="currentPoint">The current point.</param>
        public virtual void Resize(PointF startPoint, PointF currentPoint)
        {
            var dx = (int)(currentPoint.X - startPoint.X);
            var dy = (int)(currentPoint.Y - startPoint.Y);
            if (startPoint.X >= X - 5 && startPoint.X <= X + 5)
            {
                StartPoint = new PointF(currentPoint.X, StartPoint.Y);
                if (ShapeType == ShapeType.Circle || ShapeType == ShapeType.Square)
                {
                    Height = Width;
                }
            }
            else if (startPoint.X >= EndPoint.X - 5 && startPoint.X <= EndPoint.X + 5)
            {
                Width += dx;
                if (ShapeType == ShapeType.Circle || ShapeType == ShapeType.Square)
                {
                    Height = Width;
                }
            }
            else if (startPoint.Y >= Y - 5 && startPoint.Y <= Y + 5)
            {
                Y = currentPoint.Y;
                if (ShapeType == ShapeType.Circle || ShapeType == ShapeType.Square)
                {
                    Width = Height;
                }
            }
            else if (startPoint.Y >= EndPoint.Y - 5 && startPoint.Y <= EndPoint.Y + 5)
            {
```

```
                Height += dy;
                if (ShapeType == ShapeType.Circle || ShapeType == ShapeType.Square)
                {
                    Width = Height;
                }
            }
        }

        /// <summary>
        /// Sets the background brush as hatch.
        /// </summary>
        /// <param name="hatchStyle">The hatch style.</param>
        public virtual void SetBackgroundBrushAsHatch(HatchStyle hatchStyle)
        {
            var brush = new HatchBrush(hatchStyle, BackgroundColor);
            BackgroundBrush = brush;
        }

        /// <summary>
        /// Sets the background brush as linear gradient.
        /// </summary>
        public virtual void SetBackgroundBrushAsLinearGradient()
        {
            var brush = new LinearGradientBrush(StartPoint, EndPoint, ForeColor, BackgroundColor);
            BackgroundBrush = brush;
        }

        /// <summary>
        /// Sets the background brush as solid.
        /// </summary>
        public virtual void SetBackgroundBrushAsSolid()
        {
            var brush = new SolidBrush(BackgroundColor);
            BackgroundBrush = brush;
        }
// Private Methods (1)

        /// <summary>
        /// Calulates the size of the location and.
        /// </summary>
        /// <param name="startPoint">The start point.</param>
        /// <param name="endPoint">The end point.</param>
        private void CalulateLocationAndSize(PointF startPoint, PointF endPoint)
        {
            float x = 0, y = 0;
            float width = Math.Abs(endPoint.X - startPoint.X);
            float height = Math.Abs(endPoint.Y - startPoint.Y);
            if (startPoint.X <= endPoint.X && startPoint.Y <= endPoint.Y)
            {
                x = startPoint.X;
                y = startPoint.Y;
            }
            else if (startPoint.X >= endPoint.X && startPoint.Y >= endPoint.Y)
            {
                x = endPoint.X;
                y = endPoint.Y;
            }
            else if (startPoint.X >= endPoint.X && startPoint.Y <= endPoint.Y)
            {
                x = endPoint.X;
                y = startPoint.Y;
            }
            else if (startPoint.X <= endPoint.X && startPoint.Y >= endPoint.Y)
            {
                x = startPoint.X;
                y = endPoint.Y;
            }
            StartPoint = new PointF(x, y);
            EndPoint = new PointF(X + width, Y + height);
        }

#endregion Methods
    }
}
```

حال به تشریح **سازنده** کلاس می‌پردازیم:

Shape : پارامترهای این سازنده به ترتیب عبارتند از **نقطه شروع** ، **نقطه پایان** ، **عمق شی** ، **رنگ قلم** ، **ضخامت خط** ، **آیا شی توپر باشد** ؟، و **رنگ پر کردن شی** ، در این سازنده ابتدا توسط متدی به نام CalulateLocationAndSize(startPoint, endPoint); b نقاط ابتدا و انتهای شی مورد نظر تنظیم می‌شود، در متد مذکور بررسی می‌شود در صورتی که نقاط شروع و پایان یکی از حالت‌های 1 ، 2، 3، 4 از تصویر ابتدا پست باشد همگی تبدیل به حالت 1 خواهد شد.

سپس به تشریح **متدهای** کلاس Shape می‌پردازیم:
Draw : این متد دارای یک پارامتر ورودی است که **بوم گرافیکی** مورد نظر می‌باشد، در واقع شی مورد نظر خود را بروی این بوم گرافیکی ترسیم می‌کند. در کلاس پایه کار این متد زیاد پیچیده نیست، در صورتی که شی در حالت انتخاب باشد (IsSelected = true) بروی شی مورد نظر 8 مربع کوچک ترسیم می‌شود و اگر شی مورد نظر خط باشد دو مربع کوچک در طرفین خط رسم می‌شود که نشان دهنده انتخاب شدن شی مورد نظر است. این متد به صورت virtual تعریف شده است یعنی کلاس هایی که از Shape ارث میبرند می‌توانند این متد را برای خود از نو بازنویسی کرده (override کنند) و تغییر رفتار دهند.
HasPointInSahpe : این متد نیز به صورت virtual تعریف شده است دارای خروجی بولین می‌باشد. پارامترهای این متد عبارتند از **یک نقطه** و **یک عدد** که نشان دهنده تلرانش نقطه بر حسب پیکسل می‌باشد. کار این متد این است که یک نقطه را گرفته و بررسی می‌کند که آیا نقطه مورد نظر با تلرانس وارد شده آیا در داخل شی واقع شده است یا خیر (مثلا وجود نقطه در مستطیل یا وجود نقطه در دایره فرمول‌های متفاوتی دارند که در اینجا پیش فرض برای تمامی اشیا حالت مستطیل در نظر گرفته شده که می‌توانید آنها را بازنویسی (override) کنید).

Move : این متد به عنوان پارامتر **یک نقطه** را گرفته و شی مورد نظر را به آن نقطه منتقل می‌کند در واقع نقطه شروع و پایان ترسیم شی را تغییر می‌دهد.

Move : این متد نیز برای جابجایی شی به کار می‌رود، این متد دارای پارامترهای **جابجایی در راستای محور Xها** , **جابجایی در راستای محور Yها** ؛ و شی مورد نظر را به آن نقطه منتقل می‌کند در واقع نقطه شروع و پایان ترسیم شی را با توجه به پارامترهای ورودی تغییر می‌دهد.
Resize : این متد نیز برای تغییر اندازه شی به کار می‌رود، این متد دارای پارامترهای **تغییر اندازه در راستای محور Xها** , **تغییر اندازه در راستای محور Yها** می‌باشد و نقطه پایان شی مورد نظر را تغییر می‌دهد اما نقطه شروع تغییری نمی‌کند.

Resize : این متد نیز برای تغییر اندازه شی به کار می‌رود، در زمان تغییر اندازه شی با ماوس ابتدا یک نقطه شروع وجود دارد که ماوس در آن نقطه کلیک شده و شروع به درگ کردن شی جهت تغییر اندازه می‌کند (پارامتر اول این متد نقطه شروع درگ کردن جهت تغییر اندازه را مشخص می‌کند startPoint)، سپس در یک نقطه ای درگ کردن تمام می‌شود در این نقطه باید شی تغییر اندازه پیدا کرده و ترسیم شود (پارامتر دوم این متد نقطه مذکور می‌باشد currentLocation). سپس با توجه با این دو نقطه بررسی می‌شود که تغییر اندازه در کدام جهت صورت گرفته است و اعداد جهت تغییرات نقاط شروع و پایان شی مورد نظر محاسبه می‌شوند. (مثلا تغییر اندازه در مستطیل از ضلع بالا به طرفین، یا از ضلع سمت راست به طرفین و ....). البته برای مربع و دایره باید کاری کنیم که طول و عرض تغییر اندازه یکسان باشد.

CalulateLocationAndSize : این متد که در سازنده کلاس استفاده شده در واقع دو نقطه شروع و پایان را گرفته و با توجه به تصویر ابتدای پست حالت‌های 1 و 2 و3 و 4 را به حالت 1 تبدیل کرده و StartPoint و EndPoint را اصلاح می‌کند.
SetBackgroundBrushAsHatch : این متد یک الگوی Brush گرفته و با توجه به رنگ پس زمینه شی خصوصیت BackgroundBrush را مقداردهی می‌کند.

SetBackgroundBrushAsLinearGradient : این متد با توجه به خصوصیت ForeColor و BackgroundColor یک Gradiant Brush ساخته و آن را به خصوصیت
BackgroundBrush نسبت می‌کند.
SetBackgroundBrushAsSolid : یک الگوی پر کردن توپر برای شی مورد نظر با توجه به خصوصیت BackgroundColor شی ایجاد کرده و آن را به خصوصیت BackgroundBrush شی نسبت می‌دهد.

تذکر : متدهای Move، Resize و HasPointInShape به صورت virtual تعریف شده تا کلاس‌های مشتق شده در صورت نیاز خود کد رفتار مورد نظر خود را override کرده یا از همین رفتار استفاده نمایند.

خوشحال می‌شم در صورتی که در Refactoring کد نوشته شده با من همکاری کنید.

در پست‌های آینده به بررسی و پیاده سازی دیگر کلاس‌ها خواهیم پرداخت.

نظرات خوانندگان

نویسنده: بتیسا
تاریخ: ۱۳۹۱/۱۱/۱۸ ۱۰:۴۱

با سلام

از مطلب مفیدتون ممنونم
در متد **DrawPreview** اصلی که نوشته شده در بخش هایی که اشیاء توپر رسم می‌شوند مانند خط 143، 149 و... بجای استفاده از خصوصیت Brush که در بخش قبل برای پس زمینه در نظر گرفته شده بود هر بار یک براش ایجاد شده که می‌توانیم به صورت زیر اصلاح کنیم.

```
case ShapeType.Ellipse:
                    if (isFill)
                        g.FillEllipse(new SolidBrush(backgroundColor), x, y, width, height);
                    //else
                    g.DrawEllipse(new Pen(foreColor, thickness), x, y, width, height);
                    break;

//--------------------------------[Change to]-------=>

               case ShapeType.Ellipse:
                    if (isFill)
                        g.FillEllipse(Brush, x, y, width, height);
                    //else
                    g.DrawEllipse(new Pen(foreColor, thickness), x, y, width, height);
                    break;
```

نویسنده: صابر فتح الهی
تاریخ: ۱۳۹۱/۱۱/۱۸ ۱۰:۴۶

بله دوست گلم میشد اینکارو انجام داد
اما با توجه به اینکه متد **DrawPreview** به صورت static تعریف شده نمی‌توان از خصوصیات غیر استاتیک کلاس در آن استفاده کرد.
درسته؟

نویسنده: بتیسا
تاریخ: ۱۳۹۱/۱۱/۱۸ ۱۰:۵۸

بله به static بودن متد توجه نکرده بودم

نویسنده: بتیسا
تاریخ: ۱۳۹۱/۱۱/۱۸ ۱۱:۱۹

برای برطرف کردن این مسئله هم می‌توانیم همانطور که در ورودی foreColor را دریافت کردیم brush را نیز دریافت کنیم.

```
public static void DrawPreview(Graphics g, PointF startPoint, PointF endPoint, Color foreColor, byte
thickness, bool isFill, Color backgroundColor, ShapeType shapeType)
//--------------------------------[Change to]-------=>
public static void DrawPreview(Graphics g, PointF startPoint, PointF endPoint, Color foreColor, byte
thickness, bool isFill, Brush backgroundBrush , ShapeType shapeType)


//--------------------------------------------------
    case ShapeType.Ellipse:
                    if (isFill)
                        g.FillEllipse(new SolidBrush(backgroundColor), x, y, width, height);
                    //else
```

```
                    g.DrawEllipse(new Pen(foreColor, thickness), x, y, width, height);
                    break;

//--------------------------------[Change to]-------=>

               case ShapeType.Ellipse:
                    if (isFill)
                        g.FillEllipse(backgroundBrush, x, y, width, height);
                    //else
                    g.DrawEllipse(new Pen(foreColor, thickness), x, y, width, height);
                    break;
```

نویسنده: صابر فتح الهی
تاریخ: ۱۳۹۱/۱۱/۱۸ ۱۲:۷

درسته اما در قسمت اینترفیس کاربر باید Brushهای مورد نظر ساخته شده و به این متد پاس داده شود، در مراحل پایانی فکر می‌کنم بهتر منظورم بتونم برسونم، به دلایلی (که در پست‌های آینده گفته میشه) از این روش‌ها استفاده نکردم.

نویسنده: سعید
تاریخ: ۱۳۹۱/۱۱/۱۸ ۱۴:۱۷

چرا drawpreview به صورت استاتیک تعریف شده؟ چرا دوبار تعریف شده؟ و چرا این کلاس پایه اطلاعات زیادی در مورد رسم زیر مجموعه‌های خودش داره؟ آیا بهتر نیست جزئیات ترسیم هر شیء با override شدن به زیر کلاس‌های مشتق شده واگذار بشن؟ چرا این متدها از خاصیت‌های کلاس تعریف شده استفاده نمی‌کنن و دوباره این خاصیت‌ها رو به صورت پارامتر دریافت کردن؟ (همون بحث اعلام استقلال متد تعریف شده به صورت استاتیک و اینکه چرا؟) و اگر این کلاس پایه تا این اندازه لازم هست در مورد رسم دایره و سایر اشکال اطلاعات داشته باشد، چه ضرورتی به abstarct بودن آن هست؟ اصلا چه ضرورتی به تعریف اشیاء مشتق شده از آن هست؟

نویسنده: صابر فتح الهی
تاریخ: ۱۳۹۱/۱۱/۱۸ ۱۷:۱۱

جز متد **DrawPreview** هیچکدام از متدها پارامتری دریافت نمی‌کنند، اون هم به دلیل اینه که می‌خوام پیش نمایشی از یک شی ترسیم کنم البته می‌تونستیم اون توی یک کلاس جدا بنویسیم که این کلاس زیاد شلوغ نشه فکر می‌کنم با نوشتن کلاس‌های مشتق شده بهتر بشه توی این زمینه‌ها بحث کرد.

پ.ن : متد **DrawPreview** به یک کلاس ثالث منتقل شد.

نویسنده: صابر فتح الهی
تاریخ: ۱۳۹۱/۱۱/۱۸ ۱۷:۵۹

نظر شما اعمال شد

نویسنده: مسعود بهرامی
تاریخ: ۱۳۹۲/۰۹/۰۱ ۱۶:۵۸

با اجازه دوست عزیزم مهندس فتح الهی
من به نظرم Helpers رو اگه به شکل زیر Re factor کنیم بهتر باشه :)
اول یه کلاس تعریف می‌کنیم و اطلاعات لازم برای ترسیم پیش نمایش رو تو اون کلاس میزاریم

```
public class ShapeSpecification
    {
```

```
        public PointF StartPoint{get;set;}
        public PointF EndPoint{get;set;}
        public Color ForeColor{get;set;}
        public byte Thickness{get;set;}
        public bool IsFill{get;set;}
        public Brush BackgroundBrush{get;set;}
    }
```

یه کلاس دیگه هم نقاط ابتدا و انتها و طول و عرض رو تو خودش داره

```
public class StartPoints
    {
        public float XPoint { get; set; }
        public float YPoint { get; set; }
        public float Width { get; set; }
        public float Height { get; set; }
    }
```

حالا یه اینترفیس تعریف می‌کنیم که فقط یه متد داره به نام Draw

```
public interface IPeiview
    {
        void Draw(ShapeSpecification shapeScepification);
    }
```

حالا می‌رسیم به کلاس Helpers اصلیمون که میتونه هم استاتیک باشه و هم معمولی به دو شکل زیر

```
public class Helpers
    {
        private readonly IPeiview peiview;

        public Helpers(IPeiview peiview)
        {
            this.peiview = peiview;
        }

        public void Draw(ShapeSpecification shapeSpecification)
        {
            peiview.Draw(shapeSpecification);
        }
    }
```

```
public static  class Helpers
    {

        public static void Draw(ShapeSpecification shapeSpecification, IPeiview peiview)
        {
            peiview.Draw(shapeSpecification);
        }
    }
```

که فقط یه متد ساده Draw داره و اونم تابع Draw اینترفیسی که بش دادیم رو صدا میزینه

یه کلاس دیگه هم تعریف میکنیم که مسئولیتش تشخیص بوم‌های چهارگانه است برای شروع نقطه‌ی ترسیم

```
public static class AreaParser
    {
        public static StartPoints Parse(PointF startPoint, PointF endPoint)
        {
            var startPoints = new StartPoints();

            startPoints.Width = Math.Abs(endPoint.X - startPoint.X);
            startPoints.Height = Math.Abs(endPoint.Y - startPoint.Y);

            if (startPoint.X <= endPoint.X && startPoint.Y <= endPoint.Y)
            {
                startPoints.XPoint = startPoint.X;
                startPoints.YPoint = startPoint.Y;
            }
```

```
            else if (startPoint.X >= endPoint.X && startPoint.Y >= endPoint.Y)
            {
                startPoints.XPoint = endPoint.X;
                startPoints.YPoint = endPoint.Y;
            }

            else if (startPoint.X >= endPoint.X && startPoint.Y <= endPoint.Y)
            {
                startPoints.XPoint = endPoint.X;
                startPoints.YPoint = startPoint.Y;
            }

            else if (startPoint.X <= endPoint.X && startPoint.Y >= endPoint.Y)
            {
                startPoints.XPoint = startPoint.X;
                startPoints.YPoint = endPoint.Y;
            }
            return startPoints;
        }
    }
```

نکته: این کلاس رو اگه با Func ایجاد کنیم خیلی بهتر و تمیزتر وقشنکتر هم میشد که من می‌گزرم فعلا ازش
حالا ما هر شکل جدید که اضافه کنیم به پروژه Paint خودمون و قصد پیش نمایش اونو داشته باشیم فقط کافیه یه کلاس برا پیش نمایشش ایجاد کنیم که کلاس Ipreview رو Implement کنه و متد Draw مخصوص به خودش را داشته باشه و از شر Swithهای طولانی خلاص میشیم مثلا من برای دایره اینکارو کردم

```
public class CirclePreview:IPeiview
    {
        private readonly Graphics graphics;

        public CirclePreview(Graphics graphics)
        {
            this.graphics = graphics;
        }
        public void Draw(ShapeSpecification shapeScepification)
        {
            var startPoints = AreaParser.Parse(shapeScepification.StartPoint,
shapeScepification.EndPoint);

            float raduis = Math.Max(startPoints.Width, startPoints.Height);

            if (shapeScepification.IsFill)
                this.graphics.FillEllipse(shapeScepification.BackgroundBrush, startPoints.XPoint,
startPoints.YPoint, raduis, raduis);
            else
                this.graphics.DrawEllipse(new Pen(shapeScepification.ForeColor,
shapeScepification.Thickness), startPoints.XPoint, startPoints.YPoint, raduis, raduis);
        }
    }
```

نویسنده: صابر فتح الهی
تاریخ: ۱۳۹۲/۰۹/۰۱ ۲۳:۲۷

ظاهرا همه چیز مرتبه و درست هست.
من بررسی می‌کنم و نتیجش به شما اطلاع میدم
در هر صورت از وقتی که گذاشتین متشکرم

عنوان: **پیاده سازی پروژه نقاشی (Paint) به صورت شی گرا #4**
نویسنده: صابر فتح الهی
تاریخ: ۱۳۹۱/۱۱/۲۲ ۲۳:۳۰
آدرس: www.dotnettips.info
برچسب‌ها: C#, OOP, Project, Paint

در ادامه پست قبل ، در این پست به بررسی کلاس Triangle جهت رسم مثلث و کلاس Diamond جهت رسم لوزی می‌پردازیم.

```
using System.Drawing;

namespace PWS.ObjectOrientedPaint.Models
{
    /// <summary>
    /// Triangle
    /// </summary>
    public class Triangle : Shape
    {
        #region Constructors (2)

        /// <summary>
        /// Initializes a new instance of the <see cref="Triangle" /> class.
        /// </summary>
        /// <param name="startPoint">The start point.</param>
        /// <param name="endPoint">The end point.</param>
        /// <param name="zIndex">Index of the z.</param>
        /// <param name="foreColor">Color of the fore.</param>
        /// <param name="thickness">The thickness.</param>
        /// <param name="isFill">if set to <c>true</c> [is fill].</param>
        /// <param name="backgroundColor">Color of the background.</param>
        public Triangle(PointF startPoint, PointF endPoint, int zIndex, Color foreColor, byte
thickness, bool isFill, Color backgroundColor)
            : base(startPoint, endPoint, zIndex, foreColor, thickness, isFill, backgroundColor)
        {
            ShapeType = ShapeType.Triangle;
        }


        /// <summary>
        /// Initializes a new instance of the <see cref="Triangle" /> class.
        /// </summary>
        public Triangle()
        {
            ShapeType = ShapeType.Triangle;
        }

        #endregion Constructors

        #region Methods (1)

        // Public Methods (1)

        /// <summary>
        /// Draws the specified g.
        /// </summary>
        /// <param name="g">The g.</param>
        public override void Draw(Graphics g)
        {
            var points = new PointF[3];
            points[0] = new PointF(X + Width / 2, Y);
            points[1] = new PointF(X + Width, Y + Height);
            points[2] = new PointF(X, Y + Height);
            if (IsFill)
                g.FillPolygon(BackgroundBrush, points);
            g.DrawPolygon(new Pen(ForeColor, Thickness), points);
            base.Draw(g);
        }

        #endregion Methods
    }
}
```

همانگونه که مشاهده می‌کنید کلاس مثلث از کلاس Shape ارث برده و تشکیل شده از یک سازنده و بازنویسی (override) متد Draw می‌باشد، البته متد HasPointInSahpe در کلاس پایه قاعدتا باید بازنویسی شود، برای تشخیص وجود نقطه در شکل مثلث،

(اگر دوستان فرمولش می‌دونن ممنون می‌شم در اختیار بذارن). در متد Draw سه نقطه مثلث در نظر گرفته شده که بر طبق آن با استفاده از متدهای رسم منحنی اقدام به رسم مثلث توپر یا تو خالی نموده‌ایم.

کلاس لوزی نیز دقیقا مانند کلاس مثلث عمل می‌کند.

```
using System.Drawing;

namespace PWS.ObjectOrientedPaint.Models
{
    /// <summary>
    /// Diamond
    /// </summary>
    public class Diamond : Shape
    {
        #region Constructors (2)

        /// <summary>
        /// Initializes a new instance of the <see cref="Diamond" /> class.
        /// </summary>
        /// <param name="startPoint">The start point.</param>
        /// <param name="endPoint">The end point.</param>
        /// <param name="zIndex">Index of the z.</param>
        /// <param name="foreColor">Color of the fore.</param>
        /// <param name="thickness">The thickness.</param>
        /// <param name="isFill">if set to <c>true</c> [is fill].</param>
        /// <param name="backgroundColor">Color of the background.</param>
        public Diamond(PointF startPoint, PointF endPoint, int zIndex, Color foreColor, byte thickness,
bool isFill, Color backgroundColor)
            : base(startPoint, endPoint, zIndex, foreColor, thickness, isFill, backgroundColor)
        {
            ShapeType = ShapeType.Diamond;
        }

        /// <summary>
        /// Initializes a new instance of the <see cref="Diamond" /> class.
        /// </summary>
        public Diamond()
        {
            ShapeType = ShapeType.Diamond;
        }

        #endregion Constructors

        #region Methods (1)

        // Public Methods (1)

        /// <summary>
        /// Draws the specified g.
        /// </summary>
        /// <param name="g">The g.</param>
        public override void Draw(Graphics g)
        {
            var points = new PointF[4];
            points[0] = new PointF(X + Width / 2, Y);
            points[1] = new PointF(X + Width, Y + Height / 2);
            points[2] = new PointF(X + Width / 2, Y + Height);
            points[3] = new PointF(X, Y + Height / 2);
            if (IsFill)
                g.FillPolygon(BackgroundBrush, points);
            g.DrawPolygon(new Pen(ForeColor, Thickness), points);
            base.Draw(g);
        }

        #endregion Methods
    }
}
```

این کلاس نیز از کلاس Shape ارث برده و دارای یک سازنده بوده و متد Draw را ازنو بازنویسی می‌کند، این متد نیز با استفاده از چهار نقطه و استفاده از متد رسم منحنی در دات نت اقدام به طراحی لوزی توپر یا تو خالی می‌کند، متد HasPointInSahpe در کلاس پایه قاعدتا باید بازنویسی شود، برای تشخیص وجود نقطه در شکل لوزی، برای رسم لوزی توپر نیز خصوصیت BackgroundBrush استفاده کرده و شی توپر را رسم می‌کند.

مباحث رسم مستطیل و مربع، دایره و بیضی در پست‌های بعد بررسی خواهند شد.

پیاده سازی پروژه نقاشی (Paint) به صورت شی گرا #1
پیاده سازی پروژه نقاشی (Paint) به صورت شی گرا #2
پیاده سازی پروژه نقاشی (Paint) به صورت شی گرا #3

موفق ومومید باشید.

عنوان: **پیاده سازی پروژه نقاشی (Paint) به صورت شی گرا #5**
نویسنده: صابر فتح الهی
تاریخ: ۱۳۹۱/۱۲/۰۳ ۰:۵۰
آدرس: www.dotnettips.info
برچسب‌ها: C#, OOP, Project, Paint

در ادامه مطلب پیاده سازی پروژه نقاشی (Paint) به صورت شی گرا #4 به تشریح مابقی کلاس‌های برنامه می‌پردازیم.

در این پست به شرح کلاس Rectangle جهت رسم مستطیل و Square جهت رسم مربع می‌پردازیم

```
using System.Drawing;

namespace PWS.ObjectOrientedPaint.Models
{
    /// <summary>
    /// Rectangle
    /// </summary>
    public class Rectangle : Shape
    {
        #region Constructors (2)

        /// <summary>
        /// Initializes a new instance of the <see cref="Rectangle" /> class.
        /// </summary>
        /// <param name="startPoint">The start point.</param>
        /// <param name="endPoint">The end point.</param>
        /// <param name="zIndex">Index of the z.</param>
        /// <param name="foreColor">Color of the fore.</param>
        /// <param name="thickness">The thickness.</param>
        /// <param name="isFill">if set to <c>true</c> [is fill].</param>
        /// <param name="backgroundColor">Color of the background.</param>
        public Rectangle(PointF startPoint, PointF endPoint, int zIndex, Color foreColor, byte
thickness, bool isFill, Color backgroundColor)
            : base(startPoint, endPoint, zIndex, foreColor, thickness, isFill, backgroundColor)
        {
            ShapeType = ShapeType.Rectangle;
        }

        /// <summary>
        /// Initializes a new instance of the <see cref="Rectangle" /> class.
        /// </summary>
        public Rectangle()
        {
            ShapeType = ShapeType.Rectangle;
        }

        #endregion Constructors

        #region Methods (1)

        // Public Methods (1)

        /// <summary>
        /// Draws the specified g.
        /// </summary>
        /// <param name="g">The g.</param>
        public override void Draw(Graphics g)
        {
            if (IsFill)
                g.FillRectangle(BackgroundBrush, StartPoint.X, StartPoint.Y, Width, Height);
            g.DrawRectangle(Pen, StartPoint.X, StartPoint.Y, Width, Height);
            base.Draw(g);
        }

        #endregion Methods
    }
}
```

کلاس Rectangle از کلاس پایه طراحی شده در ^ ارث بری دارد. این کلاس ساده بوده و تنها شامل یک سازنده و متد ترسیم شی مستطیل می‌باشد.

کلاس بعدی کلاس Square می‌باشد، که از کلاس بالا (Rectangle) ارث بری داشته است، کدهای این کلاس را در زیر مشاهده می‌کنید.

```
using System;
using System.Drawing;

namespace PWS.ObjectOrientedPaint.Models
{
    /// <summary>
    /// Square
    /// </summary>
    public class Square : Rectangle
    {
#region Constructors (2)

        /// <summary>
        /// Initializes a new instance of the <see cref="Square" /> class.
        /// </summary>
        /// <param name="startPoint">The start point.</param>
        /// <param name="endPoint">The end point.</param>
        /// <param name="zIndex">Index of the z.</param>
        /// <param name="foreColor">Color of the fore.</param>
        /// <param name="thickness">The thickness.</param>
        /// <param name="isFill">if set to <c>true</c> [is fill].</param>
        /// <param name="backgroundColor">Color of the background.</param>
        public Square(PointF startPoint, PointF endPoint, int zIndex, Color foreColor, byte thickness,
bool isFill, Color backgroundColor)
        {
            float x = 0, y = 0;
            float width = Math.Abs(endPoint.X - startPoint.X);
            float height = Math.Abs(endPoint.Y - startPoint.Y);
            if (startPoint.X <= endPoint.X && startPoint.Y <= endPoint.Y)
            {
                x = startPoint.X;
                y = startPoint.Y;
            }
            else if (startPoint.X >= endPoint.X && startPoint.Y >= endPoint.Y)
            {
                x = endPoint.X;
                y = endPoint.Y;
            }
            else if (startPoint.X >= endPoint.X && startPoint.Y <= endPoint.Y)
            {
                x = endPoint.X;
                y = startPoint.Y;
            }
            else if (startPoint.X <= endPoint.X && startPoint.Y >= endPoint.Y)
            {
                x = startPoint.X;
                y = endPoint.Y;
            }
            StartPoint = new PointF(x, y);
            var side = Math.Max(width, height);
            EndPoint = new PointF(x+side, y+side);
            ShapeType = ShapeType.Square;
            Zindex = zIndex;
            ForeColor = foreColor;
            Thickness = thickness;
            BackgroundColor = backgroundColor;
            IsFill = isFill;
        }

        /// <summary>
        /// Initializes a new instance of the <see cref="Square" /> class.
        /// </summary>
        public Square()
        {
            ShapeType = ShapeType.Square;
        }

#endregion Constructors
    }
}
```

این کلاس شامل دو سازنده می‌باشد که سازنده دوم فقط نوع شی را تعیین می‌کند و بقیه کارهای آن مانند مستطیل است، در واقع می‌توان از یک دیدگاه گفت که مربع یک مستطیل است که اندازه طول و عرض آن یکسان است. در سازنده اول (نحوه ترسیم

شکل) ابتدا نقاط ابتدا و انتهای رسم شکل تعیین شده و سپس با توجه به پارامترهای محاسبه شده نوع شی جهت ترسیم و دیگر خصوصیات کلاس مقدار دهی می‌شود، با این تفاوت که در نقطه EndPoint طول و عرض مربع برابر با بزرگترین مقدار طول و عرض وارد شده در سازنده کلاس تعیین شده و مربع شکل می‌گیرد. مابقی متدهای ترسیم و ... طبق کلاس پایه مستطیل و Shape تعیین می‌شود.

مطالب قبل:

پیاده سازی پروژه نقاشی (Paint) به صورت شی گرا #1
پیاده سازی پروژه نقاشی (Paint) به صورت شی گرا #2
پیاده سازی پروژه نقاشی (Paint) به صورت شی گرا #3
پیاده سازی پروژه نقاشی (Paint) به صورت شی گرا #4

نظرات خوانندگان

نویسنده: کاوه احمدی
تاریخ: ۱۳۹۱/۱۲/۰۳ ۱۰:۵۸

امروز فرصتی دست داد نگاهی اجمالی به این پروژه بیندازم. به نظرم کد نوشته شده تا به اینجا شی گرا محسوب نمی‌شود. یعنی برخی اهدافی که به واسطه آن پارادایم شی گرایی شکل گرفته در آن رعایت نشده است.
به طور مشخص منظورم متد DrawPreview است که در بخش سوم در کلاس Helpers نوشته شده. تکرار کد شدیدی که در دستور switch این متد دیده می‌شود به سادگی قابل حذف است. کد فوق 2 مشکل اساسی دارد: اول آنکه با زیاد شدن تعداد اشیای قابل رسم، این دستور switch بسیار طولانی شده (با تکرار کد) و کد ناخوانا می‌شود و دوم آنکه با اضافه شدن هر شی قابل رسم جدید به پروژه یک case باید به این دستور اضافه شود. یعنی تغییر در یک بخش از نرم‌افزار منجر به تغییر در سایر بخش‌ها (کلاس Helpers) می‌شود. بدیهی است پارادایم شی‌گرا برای جلوگیری از چنین مسائلی شکل گرفته. در غیر این صورت این کد همان کدهای ساخت‌یافته است که در قالب کلاس نوشته شده. به نظر می‌آید بهتر باشد یک اینترفیس drawable در نظر گرفته می‌شد، در این متد از آن استفاده می‌شد و اشیای قابل رسم آنرا پیاده‌سازی می‌کردند. یک راه بسیار ساده و کارامد

نویسنده: محسن
تاریخ: ۱۳۹۱/۱۲/۰۳ ۱۱:۲۵

البته میشه این قسمت کلاس پایه رو که if و else و switch زیاد داره، با توجه به مطلب « کمپین ضد IF ! » بهبود بخشید.

نویسنده: صابر فتح الهی
تاریخ: ۱۳۹۱/۱۲/۰۳ ۲۳:۴

پاسخ شما کاملا صحیح است، توی همون پست گفتم که خیلی خوشحال میشم دوستان ایده ای به من بدهند که این قسمت با استفاده از Action و Func طراحی کنم، خودم راهی به ذهنم نرسید. بله کاملا شما درست می‌فرمایید، راه حل چیست؟

پ. ن: البته این متد کاملا قابل حذف است و می‌تواند در سیستم استفاده نشود. فقط جهت پیش نمایش رسم اشیا بکار می‌رود.

نویسنده: صابر فتح الهی
تاریخ: ۱۳۹۱/۱۲/۰۵ ۱۲:۵

@کاوه احمدی
من خیلی سعی کردم طبق الگوی « کمپین ضد IF ! » عمل کردم، و پیش رفتم درست شد، اما به دلیل اینکه در زمان رسم شی در برنامه کاربری (اینترفیس) در زمان MouseMove پیش نمایش شی رسم می‌شود و در زمان MouseUp خود شی رسم می‌شود این امکان نداشتم تا از شی نمونه سازی کنم و طبق اون الگو پیش برم ، لطفا در صورتی که روشم اشتباست اصلاح بفرمایین.
موفق وموید باشید.

نویسنده: محسن
تاریخ: ۱۳۹۱/۱۲/۰۶ ۱۶:۵۵

سلام

می‌تونید از ابزارهای تزریق وابستگی برای تامین وهله مورد نیاز استفاده کنید.

عنوان: **پیاده سازی پروژه نقاشی (Paint) به صورت شی گرا #6**
نویسنده: صابر فتح الهی
تاریخ: ۱۳۹۱/۱۲/۰۵ ۰:۵
آدرس: www.dotnettips.info
برچسب‌ها: C#, OOP, Project, Paint

در ادامه پست پیاده سازی پروژه نقاشی (Paint) به صورت شی گرا #5 ، در این پست به تشریح کلاس دایره و بیضی می‌پردازیم.

ابتدا به تشریح کلاس ترسیم بیضی (Ellipse) می‌پردازیم.

```
using System.Drawing;

namespace PWS.ObjectOrientedPaint.Models
{
    /// <summary>
    /// Ellipse Draw
    /// </summary>
    public class Ellipse : Shape
    {
        #region Constructors (2)

        /// <summary>
        /// Initializes a new instance of the <see cref="Ellipse" /> class.
        /// </summary>
        /// <param name="startPoint">The start point.</param>
        /// <param name="endPoint">The end point.</param>
        /// <param name="zIndex">Index of the z.</param>
        /// <param name="foreColor">Color of the fore.</param>
        /// <param name="thickness">The thickness.</param>
        /// <param name="isFill">if set to <c>true</c> [is fill].</param>
        /// <param name="backgroundColor">Color of the background.</param>
        public Ellipse(PointF startPoint, PointF endPoint, int zIndex, Color foreColor, byte thickness,
bool isFill, Color backgroundColor)
            : base(startPoint, endPoint, zIndex, foreColor, thickness, isFill, backgroundColor)
        {
            ShapeType = ShapeType.Ellipse;
        }

        /// <summary>
        /// Initializes a new instance of the <see cref="Ellipse" /> class.
        /// </summary>
        public Ellipse()
        {
            ShapeType = ShapeType.Ellipse;
        }

        #endregion Constructors

        #region Methods (1)

        // Public Methods (1)

        /// <summary>
        /// Draws the specified g.
        /// </summary>
        /// <param name="g">The g.</param>
        public override void Draw(Graphics g)
        {
            if (IsFill)
                g.FillEllipse(BackgroundBrush, StartPoint.X, StartPoint.Y, Width, Height);
            g.DrawEllipse(Pen, StartPoint.X, StartPoint.Y, Width, Height);
            base.Draw(g);
        }

        #endregion Methods
    }
}
```

این کلاس از شی Shape ارث برده و دارای دو سازنده ساده می‌باشد که نوع شی ترسیمی را مشخص می‌کنند، در متد Draw نیز با توجه به توپر یا توخالی بودن شی ترسیم آن انجام میشود، در این کلاس باید متد **HasPointInShape** بازنویسی (override) شود، در این متد باید تعیین شود که یک نقطه در داخل بیضی قرار گرفته است یا خیر که متاسفانه فرمول بیضی خاطرم نبود. البته به

صورت پیش فرض نقطه با توجه به چهارگوشی که بیضی را احاطه می‌کند سنجیده می‌شود.

کلاس دایره (Circle) از کلاس بالا (Ellipse) ارث بری دارد که کد آن را در زیر مشاهده می‌نمایید.

```
using System;
using System.Drawing;

namespace PWS.ObjectOrientedPaint.Models
{
    /// <summary>
    /// Circle
    /// </summary>
    public class Circle : Ellipse
    {
#region Constructors (2)

        /// <summary>
        /// Initializes a new instance of the <see cref="Circle" /> class.
        /// </summary>
        /// <param name="startPoint">The start point.</param>
        /// <param name="endPoint">The end point.</param>
        /// <param name="zIndex">Index of the z.</param>
        /// <param name="foreColor">Color of the fore.</param>
        /// <param name="thickness">The thickness.</param>
        /// <param name="isFill">if set to <c>true</c> [is fill].</param>
        /// <param name="backgroundColor">Color of the background.</param>
        public Circle(PointF startPoint, PointF endPoint, int zIndex, Color foreColor, byte thickness,
bool isFill, Color backgroundColor)
        {
            float x = 0, y = 0;
            float width = Math.Abs(endPoint.X - startPoint.X);
            float height = Math.Abs(endPoint.Y - startPoint.Y);
            if (startPoint.X <= endPoint.X && startPoint.Y <= endPoint.Y)
            {
                x = startPoint.X;
                y = startPoint.Y;
            }
            else if (startPoint.X >= endPoint.X && startPoint.Y >= endPoint.Y)
            {
                x = endPoint.X;
                y = endPoint.Y;
            }
            else if (startPoint.X >= endPoint.X && startPoint.Y <= endPoint.Y)
            {
                x = endPoint.X;
                y = startPoint.Y;
            }
            else if (startPoint.X <= endPoint.X && startPoint.Y >= endPoint.Y)
            {
                x = startPoint.X;
                y = endPoint.Y;
            }
            StartPoint = new PointF(x, y);
            var side = Math.Max(width, height);
            EndPoint = new PointF(x + side, y + side);
            ShapeType = ShapeType.Circle;
            Zindex = zIndex;
            ForeColor = foreColor;
            Thickness = thickness;
            BackgroundColor = backgroundColor;
            IsFill = isFill;
        }

        /// <summary>
        /// Initializes a new instance of the <see cref="Circle" /> class.
        /// </summary>
        public Circle()
        {
            ShapeType = ShapeType.Circle;
        }

#endregion Constructors

#region Methods (1)

// Public Methods (1)

        /// <summary>
        /// Points the in sahpe.
```

```
        /// </summary>
        /// <param name="point">The point.</param>
        /// <param name="tolerance">The tolerance.</param>
        /// <returns>
        ///   <c>true</c> if [has point in sahpe] [the specified point]; otherwise, <c>false</c>.
        /// </returns>
        public override bool HasPointInSahpe(PointF point, byte tolerance = 5)
        {
            float width = Math.Abs(EndPoint.X+tolerance - StartPoint.X-tolerance);
            float height = Math.Abs(EndPoint.Y+tolerance - StartPoint.Y-tolerance);
            float diagonal = Math.Max(height, width);
            float raduis = diagonal / 2;
            float dx = Math.Abs(point.X - (X + Width / 2));
            float dy = Math.Abs(point.Y - (Y + height / 2));
            return (dx + dy <= raduis);
        }

#endregion Methods
    }
}
```

این کلاس شامل دو سازنده می‌باشد، که در سازنده اول با توجه به نقاط ایتدا و انتهای ترسیم شکل مقدار طول و عرض مستطیل احاطه کننده دایره محاسبه شده و باتوجه به آنها بزرگترین ضلع به عنوان قطر دایره در نظر گرفته می‌شود و EndPoint شکل مورد نظر تعیین می‌شود.

در متد **HasPointInShape** با استفاده از فرمول دایره تعیین می‌شود که آیا نقطه پارامتر ورودی متد در داخل دایره واقع شده است یا خیر (جهت انتخاب شکل برای جابجایی یا تغییر اندازه).
در پست‌های بعد به پیاده سازی اینترفیس نرم افزار خواهیم پرداخت.

موفق و موید باشید

در ادامه مطالب قبل:
پیاده سازی پروژه نقاشی (Paint) به صورت شی گرا #1
پیاده سازی پروژه نقاشی (Paint) به صورت شی گرا #2
پیاده سازی پروژه نقاشی (Paint) به صورت شی گرا #3
پیاده سازی پروژه نقاشی (Paint) به صورت شی گرا #4
پیاده سازی پروژه نقاشی (Paint) به صورت شی گرا #5

نظرات خوانندگان

نویسنده: بتیسا
تاریخ: ۱۳۹۱/۱۲/۰۷ ۹:۸

با سلام برای پیدا کردن نقطه در بیضی من چند لینک پیدا کردم امیدوارم که به کارتون بیاد
لینک اول از ویکی پدیا
لینک دوم از stackoverflow
لینک سوم mathforum
لینک چهارم mathopenref

عنوان: **تغییر اندازه تصاویر #2**
نویسنده: صابر فتح الهی
تاریخ: ۱۳۹۱/۱۲/۱۱ ۱۲:۳۵
آدرس: www.dotnettips.info
گروه‌ها: #ASP.Net, C

در ادامه مطلب تغییر اندازه تصاویر #1 ، در این پست می‌خواهیم نحوه تغییر اندازه تصاویر را در زمان درخواست کاربر بررسی کنیم.

در پست قبلی بررسی کردیم که کاربر می‌تواند در دوحالت تصاویر دریافتی از کاربران سایت را تغییر اندازه دهد، یکی در زمان ذخیره سازی تصاویر بود و دیگری در زمانی که کاربر درخواست نمایش یک تصویر را دارد.

خوب ابتدا فرض می‌کنیم برای نمایش تصاویر چند حالت داریم مثلا کوچک، متوسط، بزرگ و حالت واقعی (اندازه اصلی).
البته دقت نمایید که این طبقه بندی فرضی بوده و ممکن است برای پروژه‌های مختلف این طبقه بندی متفاوت باشد. (در این پست قصد فقط اشنایی با تغییر اندازه تصاویر است و شاید کد به درستی refactor نشده باشد).
برای تغییر اندازه تصاویر در زمان اجرا یکی از روش ها، می‌تواند استفاده از Handler باشد. خوب برای ایجاد Handler ابتدا در پروژه وب خود بروی پروژه راست کلیک کرده، و گزینه New Item را برگزینید، و در پنجره ظاهر شده مانند تصویر زیر گزینه Generic Handler را انتخاب نمایید.

پس از ایجاد هندلر، فایل کد آن مانند زیر خواهد بود، ما باید کدهای خود را در متد **ProcessRequest** بنویسیم.

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Web;
```

```
namespace PWS.UI.Handler
{
    /// <summary>
    /// Summary description for PhotoHandler
    /// </summary>
    public class PhotoHandler : IHttpHandler
    {

        public void ProcessRequest(HttpContext context)
        {
            context.Response.ContentType = "text/plain";
            context.Response.Write("Hello World");
        }

        public bool IsReusable
        {
            get
            {
                return false;
            }
        }
    }
}
```

خوب برای نوشتن کد در این مرحله ما باید چند کار انجام دهیم.

1- گرفتن پارامترهای ورودی کاربر جهت تغییر سایز از طریق روش‌های انتقال مقادیر بین صفحات (در اینجا استفاده از Query String).

2-بازیابی تصویر از دیتابیس یا از دیسک به صورت یک آرایه بایتی.

3- تغییر اندازه تصویر مرحله 2 و ارسال تصویر به خروجی.

```
using System;
using System.Data.SqlClient;
using System.Diagnostics;
using System.Drawing;
using System.Drawing.Drawing2D;
using System.Drawing.Imaging;
using System.Globalization;
using System.IO;
using System.Web;

namespace PWS.UI.Handler
{
    /// <summary>
    /// Summary description for PhotoHandler
    /// </summary>
    public class PhotoHandler : IHttpHandler
    {

        /// <summary>
        /// بازیابی تصویر اصلی از بانک اطلاعاتی
        /// </summary>
        /// <param name="photoId">کد تصویر</param>
        /// <returns></returns>
        private byte[] GetImageFromDatabase(int photoId)
        {
            using (var connection = new SqlConnection("ConnectionString"))
            {
                using (var command = new SqlCommand("Select Photo From tblPhotos Where Id = @PhotoId",
connection))
                {
                    command.Parameters.Add(new SqlParameter("@PhotoId", photoId));
                    connection.Open();
                    var result = command.ExecuteScalar();
                    return ((byte[])result);
                }
            }
        }

        /// <summary>
        /// بازیابی فایل از دیسک
        /// </summary>
        /// <param name="photoId">با فرض اینکه نام فایل این است</param>
        /// <returns></returns>
```

```
        private byte[] GetImageFromDisk(string photoId /* or somting */)
        {
                using (var sourceStream = new FileStream("Original File Path + id", FileMode.Open,
FileAccess.Read))
                {
                    return StreamToByteArray(sourceStream);
                }
        }

        /// <summary>
        /// Streams to byte array.
        /// </summary>
        /// <param name="inputStream">The input stream.</param>
        /// <returns></returns>
        /// <exception cref="System.ArgumentException"></exception>
        static byte[] StreamToByteArray(Stream inputStream)
        {
            if (!inputStream.CanRead)
            {
                throw new ArgumentException();
            }

            // This is optional
            if (inputStream.CanSeek)
            {
                inputStream.Seek(0, SeekOrigin.Begin);
            }

            var output = new byte[inputStream.Length];
            int bytesRead = inputStream.Read(output, 0, output.Length);
            Debug.Assert(bytesRead == output.Length, "Bytes read from stream matches stream length");
            return output;
        }

        /// <summary>
        /// Enables processing of HTTP Web requests by a custom HttpHandler that implements the <see
cref="T:System.Web.IHttpHandler" /> interface.
        /// </summary>
        /// <param name="context">An <see cref="T:System.Web.HttpContext" /> object that provides
references to the intrinsic server objects (for example, Request, Response, Session, and Server) used
to service HTTP requests.</param>
        public void ProcessRequest(HttpContext context)
        {
            // Set up the response settings
            context.Response.ContentType = "image/jpeg";
            context.Response.Cache.SetCacheability(HttpCacheability.Public);
            context.Response.BufferOutput = false;

            // مرحله اول
            int size = 0;
            switch (context.Request.QueryString["Size"])
            {
                case "S":
                    size = 100; //100px
                    break;
                case "M":
                    size = 198; //198px
                    break;
                case "L":
                    size = 500; //500px
                    break;
            }
            byte[] changedImage;
            var id = Convert.ToInt32(context.Request.QueryString["PhotoId"]);
            byte[] sourceImage = GetImageFromDatabase(id);
            // یا
            //byte[] sourceImage = GetImageFromDisk(id.ToString(CultureInfo.InvariantCulture));

            //2 مرحله
            if (size != 0)  //غیر از حالت واقعی تصویر
            {
                changedImage = Helpers.ResizeImageFile(sourceImage, size, ImageFormat.Jpeg);
            }
            else
            {
                changedImage = (byte[])sourceImage.Clone();
            }

            // 3 مرحله
            if (changedImage == null) return;
            context.Response.AddHeader("Content-Length",
```

```
changedImage.Length.ToString(CultureInfo.InvariantCulture));
            context.Response.BinaryWrite(changedImage);
        }

        public bool IsReusable
        {
            get
            {
                return false;
            }
        }
    }
}
```

در این هندلر ما چند متد اضافه کردیم.

1- متد **GetImageFromDatabase** : این متد یک کد تصویر را گرفته و آن را از بانک اطلاعاتی بازیابی میکند. (در صورتی که تصویر در بانک ذخیره شده باشد)
2- متد **GetImageFromDisk** : این متد نام تصویر (**با فرض اینکه یک عدد می‌باشد**) را به عنوان پارامتر گرفته و آنرا بازیابی می‌کند (در صورتی که تصویر در دیسک ذخیره شده باشد.)
3- متد **StreamToByteArray** : زمانی که تصویر از فایل خوانده می‌شود به صورت Stream است این متد یک Stream را گرفته و تبدیل به یک آرایه بایتی می‌کند.

در نهایت در متد **ProcessRequest** تصویر خوانده شده با توجه به پارامترهای ورودی تغییر اندازه داده شده و در نهایت به خروجی نوشته می‌شود.

برای استفاده این هندلر، کافی است در توصیر خود به عنوان مسیر رشته ای شبیه زیر وارد نمایید:

```
PhotoHandler.ashx?PhotoId=10&Size=S
```

مانند

```
<img src='PhotoHandler.ashx?PhotoId=10&Size=S' alt='تصویر ازمایشی' />
```

پ.ن : هرچند می‌توانستیم کد هارا بهبود داده و خیلی بهینه‌تر بنویسیم اما هدف فقط اشنایی با عمل تغییر اندازه تصویر در زمان اجرا بود، امیدوارم اساتید من ببخشن.

نظرات اقای موسوی تا حدودی اعمال شد و در پست تغییراتی انجام شد.
موفق ومومید باشید

نظرات خوانندگان

نویسنده: مهدی موسوی
تاریخ: ۱۳۹۱/۱۲/۱۱ ۱۳:۳۴

سلام.
چند نکته جهت بهبود کیفیت کد نوشته شده:

شما changedImage رو که byte[] هستش دارید، به چه دلیل اونو به Stream تبدیلش می‌کنید؟ چرا با استفاده از Context.Response.BinaryWrite همون آرایه changedImage رو بهش نمی‌دید؟ اینطوری دیگه به اون حلقه و همچنین تبدیل byte[] به Stream و مجددا خوندن از Stream نیازی نخواهد بود.
Request.QueryString["photoId"] رو بهتر نیست یک بار یک جا تعریف کنید و از اون در طول Function استفاده کنید؟ یا حتی Property ی جداگانه ای برای اینکار تعریف کنید و هر جا لازم بود اونو Call کنید؟
Dispose شدن Stream هایی که ایجاد کرده اید چی میشه؟
Catch کردن Exception کار صحیحی نیست وقتی قرار نیست کاری با اون Exception انجام بشه. از اون بدتر، در .NETهای 4.5 به قبل، مشکلات متعدد دیگه ای رو میتونه در پی داشته باشه.
در پیاده سازی StreamToByteArray، برخی از کدهایی که نوشته اید باید حقیقتا ASSERT باشن، نه اینکه در Runtime اونها رو چک کنید و ...

موفق باشید.

نویسنده: صابر فتح الهی
تاریخ: ۱۳۹۱/۱۲/۱۱ ۱۷:۵۴

بله حق با شماست.
-1بله درسته، حواسم به چنین متدی نبود، در کد پست اعمال کردم.
-2 بله می‌توان اینکار را کرد و حتی بهبودهای بیشتری روی آن داد. اما قصد اموزش یک مبحث بود والا می‌توانستیم کل ifها و switchها را حذف کنیم.
-3 بله درسته حذفش می‌کنم.
-4 این متد نوشته من نیست و من ارجاع دادم به منبع اصلی.

موفق باشید

نویسنده: دانشجو
تاریخ: ۱۳۹۲/۰۱/۱۹ ۹:۱

دلیل استفاده از Handler برای تغیر اندازه تصویر چیه ؟ در روش قبلی شما با یک تابع این کار رو انجام دادین ، آیا تفاوتی بین این دو روش وجود داره ؟ کدوم بهتره استفاده بشه ؟ اگه ممکنه در مورد Handler توضیحی بدین . با تشکر

نویسنده: صابر فتح الهی
تاریخ: ۱۳۹۲/۰۱/۱۹ ۱۳:۱۴

سلام
دوست گلم توی پستی که شما ارجاع دادین گفته شده که برای تغییر اندازه تصاویر دو راه میشه در نظر گرفت:
-1 در زمان ثبت، تصویر تغییر اندازه داده شود.
-2 در زمان نیاز تغییر اندازه داده شود.

که در مورد هر کدوم توضیحاتی داده شده است، توی این پست با استفاده از هندلر در زمان اجرا (و استفاده از تابع تغییر اندازه تصویر) تصویر مورد نظر تغییر اندازه داده شده است.
در مورد هندلر هم لینک داده بودم توی پست.
موفق و موید باشید.

نویسنده: شاهین
تاریخ: ۱۳۹۲/۰۲/۰۵ ۲:۵۵

دوست عزیز یه مثال میذاری که چطور ازش استفاده کنم؟
هرکاری می‌کنم ارور میده میکه آدرس فایل اشتباهه

نویسنده: صابر فتح الهی
تاریخ: ۱۳۹۲/۰۲/۰۵ ۱۰:۱۵

سلام
دوست من مشکلتون کجاست، لطفا کدهاتون بذارین که بهتر بتونم شمارو راهنمایی کنم.
کاری که باید بکنین
1- نوشتن متد ResizeImage
2-ایجاد یک هندلر مانند بالا
3-استفاده در نمایش تصاویر که مثالش آخر پست هست.

نویسنده: شاهین
تاریخ: ۱۳۹۲/۰۲/۰۵ ۱۵:۳۷

مرسی بابت جواب دادنتون این خط کد رو چطوری باید بنویسم؟
Original File Path + id

نویسنده: صابر فتح الهی
تاریخ: ۱۳۹۲/۰۲/۰۵ ۱۶:۵۵

شما می‌تونین در پارامتر ورودی متد نام فایل و مسیر اون پاس بدین.

در خط 103 از برنامه بالا شما می‌توانید به جای کد زیر

```
byte[] sourceImage = GetImageFromDatabase(id);
```

از کدی شبیه به این استفاده نمایید

```
byte[] sourceImage = GetImageFromDisk(
     Path.Combine(context.Server.MapPath("~/uploads"),Path.GetFileName(id.ToString())));
```

که مسیر uploads یک مسیر دلخواه است که فایل‌های شما در ان قرار گرفته است. نحوه فراخوانی در این حالت به شکل زیر خواهد بود

```
PhotoHandler.ashx?PhotoId=test.jpg&Size=S
```

مانند

```
<img src='PhotoHandler.ashx?PhotoId=test.jpg&Size=S' alt='تصویر ازمایشی' />
```

البته باید تبدیل نوع ورودی در خط 102 تغییر کند(نام فایل از نوع رشته ای است)، هر چند می‌توان کدها را بهتر کرد.

پ.ن: نام فایل می‌تواند test.jpg یا هر نامی باشد که در فولدر مورد نظر وجود دارد. البته برای این کار باید تمیز کاری روی ورودی انجام شود که می‌توانید از این پست بهره بگیرید.
موفق و موید باشید

نویسنده: رضا گرمارودی
تاریخ: ۱۳۹۲/۱۲/۱۲ ۱۳:۳۳

سلام؛ هر دو مطلب شما را مطالعه کردم. همچنین لینک هندلر شما که به مایکروسافت و MSDN ارجا شده بود و در webform مشکلی ندارم اما در Mvc هندلر را نمی‌شناسد. قسمت زیر را هم به وبکانفیگ اضافه کردم ولی موثر واقع نشد. لطفا راهنمایی بفرمایید برای استفاده از هندلری همانند هندلر فوق در mvc چکار باید انجام داد.

```
<system.webServer>
      <handlers>
         <add name="ImageHandler" path="ImageHandler.ashx" verb="*" type="ImageHandler" />
      </handlers>
  </system.webServer>
```

نسخه مورد استفاده : mvc5 , visulstudio2013

نویسنده: وحید نصیری
تاریخ: ۱۳۹۲/۱۲/۱۲ ۱۳:۳۹

هندلر را می‌توان تبدیل به یک فیلتر و یا حتی یک ActionResult کرد. برای مثال در مورد تصاویر « اضافه کردن Watermark به تصاویر یک برنامه ASP.NET MVC در صورت لینک شدن در سایتی دیگر »

عنوان: **معماری لایه بندی نرم افزار #1**
نویسنده: میثم خوشبخت
تاریخ: ۱۳۹۱/۱۲/۲۸ ۱۸:۵
آدرس: www.dotnettips.info
گروه‌ها: Design patterns, SoC, Separation of Concerns, ASP.Net, Domain Driven Design, DDD, SOLID Principals, C#, MVC, WPF, N-Layer Architecture

طراحی یک معماری خوب و مناسب یکی از عوامل مهم تولید یک برنامه کاربردی موفق می‌باشد. بنابراین انتخاب یک ساختار مناسب به منظور تولید برنامه کاربردی بسیار مهم و تا حدودی نیز سخت است. در اینجا یاد خواهیم گرفت که چگونه یک طراحی مناسب را انتخاب نماییم. همچنین روش‌های مختلف تولید برنامه‌های کاربردی را که مطمئنا شما هم از برخی از این روشها استفاده نمودید را بررسی می‌نماییم و مزایا و معایب آن را نیز به چالش می‌کشیم.

ضد الگو (Antipattern) – رابط کاربری هوشمند (Smart UI)

با استفاده از Visual Studio یا به اختصار VS ، می‌توانید برنامه‌های کاربردی را به راحتی تولید نمایید. طراحی رابط کاربری به آسانی عمل کشیدن و رها کردن (Drag & Drop) کنترل‌ها بر روی رابط کاربری قابل انجام است. همچنین در پشت رابط کاربری (Code Behind) تمامی عملیات مربوط به مدیریت رویدادها، دسترسی به داده ها، منطق تجاری و سایر نیازهای برنامه کاربردی، کد نویسی خواهند شد. مشکل این نوع کدنویسی بدین شرح است که تمامی نیازهای برنامه در پشت رابط کاربری قرار می‌گیرند و موجب تولید کدهای تکراری، غیر قابل تست، پیچیدگی کدنویسی و کاهش قابلیت استفاده مجدد از کد می‌گردد.

به این روش کد نویسی Smart UI می‌گویند که موجب تسهیل تولید برنامه‌های کاربردی می‌گردد. اما یکی از مشکلات عمده‌ی این روش، کاهش قابلیت نگهداری و پشتیبانی و عمر کوتاه برنامه‌های کاربردی می‌باشد که در برنامه‌های بزرگ به خوبی این مشکلات را حس خواهید کرد.

از آنجایی که تمامی برنامه نویسان مبتدی و تازه کار، از جمله من و شما در روزهای اول برنامه نویسی، به همین روش کدنویسی می‌کردیم، لزومی به ارائه مثال در رابطه با این نوع کدنویسی نمی‌بینم.

تفکیک و جدا سازی اجزای برنامه کاربردی (Separating Your Concern)

راه حل رفع مشکل Smart UI ، لایه بندی یا تفکیک اجزای برنامه از یکدیگر می‌باشد. لایه بندی برنامه می‌تواند به شکل‌های مختلفی صورت بگیرد. این کار می‌تواند توسط تفکیک کدها از طریق فضای نام (Namespace) ، پوشه بندی فایلهای حاوی کد و یا جداسازی کدها در پروژه‌های متفاوت انجام شود. در شکل زیر نمونه ای از معماری لایه بندی را برای یک برنامه کاربردی بزرگ می‌بینید.

به منظور پیاده سازی یک برنامه کاربردی لایه بندی شده و تفکیک اجزای برنامه از یکدیگر، مثالی را پیاده سازی خواهیم کرد. ممکن است در این مثال با مسائل جدید و شیوه‌های پیاده سازی جدیدی مواجه شوید که این نوع پیاده سازی برای شما قابل درک نباشد. اگر کمی صبر پیشه نمایید و این مجموعه‌ی آموزشی را پیگیری کنید، تمامی مسائل نامانوس با جزئیات بیان خواهند شد و درک آن برای شما ساده خواهد گشت. قبل از شروع این موضوع را هم به عرض برسانم که علت اصلی این نوع پیاده سازی، انعطاف پذیری بالای برنامه کاربردی، پشتیبانی و نگهداری آسان، قابلیت تست پذیری با استفاده از ابزارهای تست، پیاده سازی پروژه بصورت تیمی و تقسیم بخشهای مختلف برنامه بین اعضای تیم و سایر مزایای فوق العاده آن می‌باشد.

1- Visual Studio را باز کنید و یک Solution خالی با نام SoCPatterns.Layered ایجاد نمایید.

- جهت ایجاد Solution خالی، پس از انتخاب New Project ، از سمت چپ گزینه Other Project Types و سپس Visual Studio Solutions را انتخاب نمایید. از سمت راست گزینه Blank Solution را انتخاب کنید.

2- بر روی Solution کلیک راست نموده و از گزینه Add > New Project یک پروژه Class Library با نام SoCPatterns.Layered.Repository ایجاد کنید.

-3 با استفاده از روش فوق سه پروژه Class Library دیگر با نامهای زیر را به Solution اضافه کنید:

SoCPatterns.Layered.Model

SoCPatterns.Layered.Service

SoCPatterns.Layered.Presentation

-4 با توجه به نیاز خود یک پروژه دیگر را باید به Solution اضافه نمایید. نوع و نام پروژه در زیر لیست شده است که شما باید با توجه به نیاز خود یکی از پروژه‌های موجود در لیست را به Solution اضافه کنید.

(Windows Forms Application (SoCPatterns.Layered.WinUI

(WPF Application (SoCPatterns.Layered.WpfUI

(ASP.NET Empty Web Application (SoCPatterns.Layered.WebUI

(ASP.NET MVC 4 Web Application (SoCPatterns.Layered.MvcUI

-5 بر روی پروژه SoCPatterns.Layered.Repository کلیک راست نمایید و با انتخاب گزینه Add Reference به پروژه‌ی SoCPatterns.Layered.Model ارجاع دهید.

-6 بر روی پروژه SoCPatterns.Layered.Service کلیک راست نمایید و با انتخاب گزینه Add Reference به پروژه‌های SoCPatterns.Layered.Model و SoCPatterns.Layered.Repository ارجاع دهید.

-7 بر روی پروژه SoCPatterns.Layered.Presentation کلیک راست نمایید و با انتخاب گزینه Add Reference به پروژه‌های SoCPatterns.Layered.Model و SoCPatterns.Layered.Service ارجاع دهید.

-8 بر روی پروژه‌ی UI خود به عنوان مثال SoCPatterns.Layered.WebUI کلیک راست نمایید و با انتخاب گزینه Add Reference به پروژه‌های SoCPatterns.Layered.Service ، SoCPatterns.Layered.Repository ، SoCPatterns.Layered.Model و SoCPatterns.Layered.Presentation ارجاع دهید.

-9 بر روی پروژه‌ی UI خود به عنوان مثال SoCPatterns.Layered.WebUI کلیک راست نمایید و با انتخاب گزینه Set as StartUp Project پروژه‌ی اجرایی را مشخص کنید.

-10 بر روی Solution کلیک راست نمایید و با انتخاب گزینه Add > New Solution Folder پوشه‌های زیر را اضافه نموده و پروژه‌های مرتبط را با عمل Drag & Drop در داخل پوشه‌ی مورد نظر قرار دهید.

1. UI
§ SoCPatterns.Layered.WebUI

2. Presentation Layer
§ SoCPatterns.Layered.Presentation

3. Service Layer
§ SoCPatterns.Layered.Service

4. Domain Layer
§ SoCPatterns.Layered.Model

5. Data Layer
§ SoCPatterns.Layered.Repository

توجه داشته باشید که پوشه بندی برای مرتب سازی لایه‌ها و دسترسی راحت‌تر به آنها می‌باشد.

پیاده سازی ساختار لایه بندی برنامه به صورت کامل انجام شد. حال به پیاده سازی کدهای مربوط به هر یک از لایه‌ها و بخش‌ها می‌پردازیم و از لایه Domain شروع خواهیم کرد.

نظرات خوانندگان

نویسنده: آرمان فرقانی
تاریخ: ۱۳۹۱/۱۲/۲۸ ۲۰:۲

مباحثی از این دست بسیار مفید و ضروری است و به شدت استقبال می‌کنم از شروع این سری مقالات. البته پیش‌تر هم مطالبی از این دست در سایت ارائه شده است که امیدوارم این سری مقالات بتونه تا حدی پراکندگی مطالب مربوطه را از بین ببرد. فقط لطف بفرمایید در این سری مقالات مرز بندی مشخصی برای برخی مفاهیم در نظر داشته باشید. به عنوان مثال گاهی در یک مقاله مفهوم Repository معادل مفهوم لایه سرویس در مقاله دیگر است. یا Domain Model مرز مشخصی با View Model داشته باشد. همچنین بحث‌های خوبی مهندس نصیری عزیز در مورد عدم نیاز به ایجاد Repository در مفهوم متداول در هنگام استفاده از EF داشتند که در رفرنس‌های معتبر دیگری هم مشاهده می‌شود. لطفاً در این مورد نیز بحث بیشتری با مرز بندی مشخص داشته باشید.

نویسنده: حسن
تاریخ: ۱۳۹۱/۱۲/۲۸ ۲۲:۵

آیا صرفا تعریف چند ماژول مختلف برنامه را لایه بندی می‌کند و ضمانتی است بر لایه بندی صحیح، یا اینکه استفاده از الگوهای MVC و MVVM می‌توانند ضمانتی باشند بر جدا سازی حداقل لایه نمایشی برنامه از لایه‌های دیگر، حتی اگر تمام اجزای یک برنامه داخل یک اسمبلی اصلی قرار گرفته باشند؟

نویسنده: میثم خوشبخت
تاریخ: ۱۳۹۱/۱۲/۲۹ ۰:۵۳

این سری مقالات جمع بندی کامل معماری لایه بندی نرم افزار است. پس از پایان مقالات یک پروژه کامل رو با معماری منتخب پیاده سازی میکنم تا تمامی شک و شبهات برطرف بشه. در مورد مرزبندی لایه‌ها هم صحبت می‌کنم و مفهوم هر کدام را دقیقا توضیح میدم.

نویسنده: میثم خوشبخت
تاریخ: ۱۳۹۱/۱۲/۲۹ ۰:۵۹

اگر مقاله فوق رو با دقت بخونید متوجه میشید که MVC و MVVM در لایه UI پیاده سازی میشن. البته در MVC لایه Model رو به Domain Model و Repository در برخی مواقع لایه Controller رو در لایه Presentation قرار میدن. در MVVM نیز لایه Model در Domain Model و Repository و لایه View Model نیز در لایه Presentation قرار میگیره. همچنین View Modelها نیز در لایه Service قرار میگیرن.
در مورد ماژول بندی هم اگر در مقاله خونده باشید میتونید لایه‌ها رو از طریق پوشه ها، فضای نام و یا پروژه‌ها از هم جدا کنید

نویسنده: حسن
تاریخ: ۱۳۹۱/۱۲/۲۹ ۱۰:۱۴

شما در مطلبتون با ضدالگو شروع کردید و عنوان کردید که روش code behind یک سری مشکلاتی رو داره. سؤال من هم این بود که آیا صرفا تعریف چند ماژول جدید می‌تواند ضمانتی باشد بر رفع مشکل code behind یا اینکه با این ماژول‌ها هم نهایتا همان مشکل قبل پابرجا است یا می‌تواند پابرجا باشد.

ضمن اینکه تعریف شما از لایه دقیقا چی هست؟ به نظر فقط تعریف یک اسمبلی در اینجا لایه نام گرفته.

نویسنده: آرمان فرقانی
تاریخ: ۱۳۹۱/۱۲/۲۹ ۱۱:۵۸

صحبت شما کاملاً صحیح است و صرفاً با ماژولار کردن به معماری چند لایه نمی‌رسیم. اما نویسنده مقاله نیز چنین نگفته و در پایان مقاله بحث پایان **ساختار** چند لایه است و نه پایان پروژه. این قسمت اول این سری مقالات است و قطعاً در هنگام پیاده سازی کدهای هر لایه مباحثی مطرح خواهد شد تا تضمین مفهوم مورد نظر شما باشد.

نویسنده: میثم خوشبخت
تاریخ: ۱۳۹۱/۱۲/۲۹ ۱۲:۳۸

با تشکر از دوست عزیزم جناب آقای آرمان فرقانی با توضیحی که دادند.
یکی از دلایل این شیوه کد نویسی امکان تست نویسی برای هر یک از لایه‌ها و همچنین استقلال لایه‌ها از هم دیگه هست که هر لایه بتونه بدون وجود لایه‌ی دیگه تست بشه. ماژولار کردنه ممکنه مشکل Smart UI رو حل کنه و ممکنه حل نکنه. بستگی به شیوه کد نویسی داره.

نویسنده: بهروز
تاریخ: ۱۳۹۱/۱۲/۲۹ ۱۳:۱۱

وقتی نظرات زیر مطلب شما رو میخونم میفهمم که نیاز به این سری آموزشی که دارید ارائه میدید چقدر زیاد احساس میشه فقط می‌خواستم بگم بر سر این مبحثی که دارید ارائه می‌دید اختلاف بین علما زیاد است!(حتی در عمل و در شرکت‌های نرم افزاری که تا به حال دیدم چه برسد در سطح آموزش...)
امیدوارم این حساسیت رو در نظر بگیرید و همه ما پس از مطالعه این سری آموزشی به فهم مشترک و یکسانی در مورد مفاهیم موجود برسیم
فکر می‌کنم وجود یک پروژه برای دست یافتن به این هدف هم ضروری باشد
باز هم تشکر

نویسنده: میثم خوشبخت
تاریخ: ۱۳۹۱/۱۲/۲۹ ۱۳:۵۹

من هم وقتی کار بر روی این معماری رو شروع کردم با مشکلات زیادی روبرو بودم و خیلی از مسائل برای من هم نامانوس و غیر قابل هضم بود. ولی بعد از اینکه چند پروژه نرم افزاری رو با این معماری پیاده سازی کردم فهم بیشتری نسبت به اون پیدا کردم و خیلی از مشکلات موجود رو با دقت بالا و با در نظر گرفتن تمامی الگوها رفع کردم. امیدوارم این حس مشترک بوجود بیاد. ولی دلیل اصلی ایجاد تکنولوژی‌ها و معماری‌های جدید اختلاف نظر بین علماست. این اختلاف نظر در اکثر مواقع میتونه مفید باشه.
ممنون دوست عزیز

نویسنده: مسعود مشهدی
تاریخ: ۱۳۹۲/۰۱/۰۴ ۱۸:۳۳

با سلام
بابت مطالبتون سپاسگذارم
همون طور که خودتون گفتید نظرات و شیوه‌های متفاوتی در نوع لایه بندی‌ها وجود داره.
در مقام مقایسه لایه بندی زیر چه وجه اشتراک و تفاوتی با لایه بندی شما داره.

```
Application.Web
Application.Manager

Application.DAL

Application.DTO

Application.Core
```

Application.Common

با تشکر

Business Layer یا Domain Model

پیاده سازی را از منطق تجاری یا Business Logic آغاز می‌کنیم. در روش کد نویسی Smart UI ، منطق تجاری در Code Behind قرار می‌گرفت اما در روش لایه بندی، منطق تجاری و روابط بین داده‌ها در Domain Model طراحی و پیاده سازی می‌شوند. در مطالب بعدی راجع به Domain Model و الگوهای پیاده سازی آن بیشتر صحبت خواهم کرد اما بصورت خلاصه این لایه یک مدل مفهومی از سیستم می‌باشد که شامل تمامی موجودیت‌ها و روابط بین آنهاست.

الگوی Domain Model جهت سازماندهی پیچیدگی‌های موجود در منطق تجاری و روابط بین موجودیت‌ها طراحی شده است.

شکل زیر مدلی را نشان می‌دهد که می‌خواهیم آن را پیاده سازی نماییم. کلاس Product موجودیتی برای ارائه محصولات یک فروشگاه می‌باشد. کلاس Price جهت تشخیص قیمت محصول، میزان سود و تخفیف محصول و همچنین استراتژی‌های تخفیف با توجه به منطق تجاری سیستم می‌باشد. در این استراتژی همکاران تجاری از مشتریان عادی تفکیک شده اند.

Domain Model را در پروژه SoCPatterns.Layered.Model پیاده سازی می‌کنیم. بنابراین به این پروژه یک Interface به نام IDiscountStrategy را با کد زیر اضافه نمایید:

```
public interface IDiscountStrategy
{
    decimal ApplyExtraDiscountsTo(decimal originalSalePrice);
}
```

علت این نوع نامگذاری Interface فوق، انطباق آن با الگوی Strategy Design Pattern می‌باشد که در مطالب بعدی در مورد این الگو بیشتر صحبت خواهم کرد. استفاده از این الگو نیز به این دلیل بود که این الگو مختص الگوریتم هایی است که در زمان اجرا قابل انتخاب و تغییر خواهند بود.

توجه داشته باشید که معمولا نام Design Pattern انتخاب شده برای پیاده سازی کلاس را بصورت پسوند در انتهای نام کلاس ذکر می‌کنند تا با یک نگاه، برنامه نویس بتواند الگوی مورد نظر را تشخیص دهد و مجبور به بررسی کد نباشد. البته به دلیل تشابه برخی از الگوها، امکان تشخیص الگو، در پاره ای از موارد وجود ندارد و یا به سختی امکان پذیر است.

الگوی Strategy یک الگوریتم را قادر می‌سازد تا در داخل یک کلاس کپسوله شود و در زمان اجرا به منظور تغییر رفتار شی، بین رفتارهای مختلف سوئیچ شود.

حال باید دو کلاس به منظور پیاده سازی روال تخفیف ایجاد کنیم. ابتدا کلاسی با نام TradeDiscountStrategy را با کد زیر به پروژه SoCPatterns.Layered.Model اضافه کنید:

```
public class TradeDiscountStrategy : IDiscountStrategy
{
    public decimal ApplyExtraDiscountsTo(decimal originalSalePrice)
    {
        return originalSalePrice * 0.95M;
    }
}
```

سپس با توجه به الگوی Null Object کلاسی با نام NullDiscountStrategy را با کد زیر به پروژه SoCPatterns.Layered.Model اضافه کنید:

```
public class NullDiscountStrategy : IDiscountStrategy
{
    public decimal ApplyExtraDiscountsTo(decimal originalSalePrice)
    {
        return originalSalePrice;
    }
}
```

از الگوی Null Object زمانی استفاده می‌شود که نمی‌خواهید و یا در برخی مواقع نمی‌توانید یک نمونه (Instance) معتبر را برای یک کلاس ایجاد نمایید و همچنین مایل نیستید که مقدار Null را برای یک نمونه از کلاس برگردانید. در مباحث بعدی با جزئیات بیشتری در مورد الگوها صحبت خواهم کرد.

با توجه به استراتژی‌های تخفیف کلاس Price را ایجاد کنید. کلاسی با نام Price را با کد زیر به پروژه SoCPatterns.Layered.Model اضافه کنید:

```
public class Price
{
    private IDiscountStrategy _discountStrategy = new NullDiscountStrategy();
    private decimal _rrp;
    private decimal _sellingPrice;
    public Price(decimal rrp, decimal sellingPrice)
    {
        _rrp = rrp;
        _sellingPrice = sellingPrice;
    }
    public void SetDiscountStrategyTo(IDiscountStrategy discountStrategy)
    {
        _discountStrategy = discountStrategy;
    }
    public decimal SellingPrice
    {
        get { return _discountStrategy.ApplyExtraDiscountsTo(_sellingPrice); }
```

```
    }
    public decimal Rrp
    {
        get { return _rrp; }
    }
    public decimal Discount
    {
        get {
            if (Rrp > SellingPrice)
                return (Rrp - SellingPrice);
            else
                return 0;
        }
    }
    public decimal Savings
    {
        get{
            if (Rrp > SellingPrice)
                return 1 - (SellingPrice / Rrp);
            else
                return 0;
        }
    }
}
```

کلاس Price از نوعی Dependency Injection به نام Setter Injection در متد SetDiscountStrategyTo استفاده نموده است که استراتژی تخفیف را برای کالا مشخص می‌نماید. نوع دیگری از Dependency Injection با نام Constructor Injection وجود دارد که در مباحث بعدی در مورد آن بیشتر صحبت خواهم کرد.

جهت تکمیل لایه Model ، کلاس Product را با کد زیر به پروژه SoCPatterns.Layered.Model اضافه کنید:

```
public class Product
{
    public int Id {get; set;}
    public string Name { get; set; }
    public Price Price { get; set; }
}
```

موجودیت‌های تجاری ایجاد شدند اما باید روشی اتخاذ نمایید تا لایه Model نسبت به منبع داده ای بصورت مستقل عمل نماید. به سرویسی نیاز دارید که به کلاینت‌ها اجازه بدهد تا با لایه مدل در اتباط باشند و محصولات مورد نظر خود را با توجه به تخفیف اعمال شده برای رابط کاربری برگردانند. برای اینکه کلاینت‌ها قادر باشند تا نوع تخفیف را مشخص نمایند، باید یک نوع شمارشی ایجاد کنید که به عنوان پارامتر ورودی متد سرویس استفاده شود. بنابراین نوع شمارشی CustomerType را با کد زیر به پروژه SoCPatterns.Layered.Model اضافه کنید:

```
public enum CustomerType
{
    Standard = 0,
    Trade = 1
}
```

برای اینکه تشخیص دهیم کدام یک از استراتژی‌های تخفیف باید بر روی قیمت محصول اعمال گردد، نیاز داریم کلاسی را ایجاد کنیم تا با توجه به CustomerType تخفیف مورد نظر را اعمال نماید. کلاسی با نام DiscountFactory را با کد زیر ایجاد نمایید:

```
public static class DiscountFactory
{
```

```
    public static IDiscountStrategy GetDiscountStrategyFor
        (CustomerType customerType)
    {
        switch (customerType)
        {
            case CustomerType.Trade:
                return new TradeDiscountStrategy();
            default:
                return new NullDiscountStrategy();
        }
    }
}
```

در طراحی کلاس فوق از الگوی Factory استفاده شده است. این الگو یک کلاس را قادر می‌سازد تا با توجه به شرایط، یک شی معتبر را از یک کلاس ایجاد نماید. همانند الگوهای قبلی، در مورد این الگو نیز در مباحث بعدی بیشتر صحبت خواهم کرد.

لایه‌ی سرویس با برقراری ارتباط با منبع داده ای، داده‌های مورد نیاز خود را بر می‌گرداند. برای این منظور از الگوی Repository استفاده می‌کنیم. از آنجایی که لایه Model باید مستقل از منبع داده ای عمل کند و نیازی به شناسایی نوع منبع داده ای ندارد، جهت پیاده سازی الگوی Repository از Interface استفاده می‌شود. یک Interface به نام IProductRepository را با کد زیر به پروژه SoCPatterns.Layered.Model اضافه کنید:

```
public interface IProductRepository
{
    IList<Product> FindAll();
}
```

الگوی Repository به عنوان یک مجموعه‌ی در حافظه (In-Memory Collection) یا انباره ای از موجودیت‌های تجاری عمل می‌کند که نسبت به زیر بنای ساختاری منبع داده ای کاملا مستقل می‌باشد.

کلاس سرویس باید بتواند استراتژی تخفیف را بر روی مجموعه ای از محصولات اعمال نماید. برای این منظور باید یک Collection سفارشی ایجاد نماییم. اما من ترجیح می‌دهم از Extension Methods برای اعمال تخفیف بر روی محصولات استفاده کنم. بنابراین کلاسی به نام ProductListExtensionMethods را با کد زیر به پروژه SoCPatterns.Layered.Model اضافه کنید:

```
public static class ProductListExtensionMethods
{
    public static void Apply(this IList<Product> products,
                                     IDiscountStrategy discountStrategy)
    {
        foreach (Product p in products)
        {
            p.Price.SetDiscountStrategyTo(discountStrategy);
        }
    }
}
```

الگوی Separated Interface تضمین می‌کند که کلاینت از پیاده سازی واقعی کاملا نامطلع می‌باشد و می‌تواند برنامه نویس را به سمت Abstraction و Dependency Inversion به جای پیاده سازی واقعی سوق دهد.

حال باید کلاس Service را ایجاد کنیم تا از طریق این کلاس، کلاینت با لایه Model در ارتباط باشد. کلاسی به نام ProductService را با کد زیر به پروژه SoCPatterns.Layered.Model اضافه کنید:

```
public class ProductService
{
    private IProductRepository _productRepository;
    public ProductService(IProductRepository productRepository)
    {
        _productRepository = productRepository;
    }
    public IList<Product> GetAllProductsFor(CustomerType customerType)
    {
        IDiscountStrategy discountStrategy =
                                DiscountFactory.GetDiscountStrategyFor(customerType);
        IList<Product> products = _productRepository.FindAll();
        products.Apply(discountStrategy);
        return products;
    }
}
```

در اینجا کدنویسی منطق تجاری در Domain Model به پایان رسیده است. همانطور که گفته شد، لایه‌ی Business یا همان Domain Model به هیچ منبع داده ای خاصی وابسته نیست و به جای پیاده سازی کدهای منبع داده ای، از Interface ها به منظور برقراری ارتباط با پایگاه داده استفاده شده است. پیاده سازی کدهای منبع داده ای را به لایه‌ی Repository واگذار نمودیم که در بخش‌های بعدی نحوه پیاده سازی آن را مشاهده خواهید کرد. این امر موجب می‌شود تا لایه Model درگیر پیچیدگی‌ها و کد نویسی‌های منبع داده ای نشود و بتواند به صورت مستقل و فارغ از بخشهای مختلف برنامه تست شود. لایه بعدی که می‌خواهیم کد نویسی آن را آغاز کنیم، لایه‌ی Service می‌باشد.

در کد نویسی‌های فوق از الگوهای طراحی (Design Patterns) متعددی استفاده شده است که به صورت مختصر در مورد آنها صحبت کردم. اصلا جای نگرانی نیست، چون در مباحث بعدی به صورت مفصل در مورد آنها صحبت خواهم کرد. در ضمن، ممکن است روال یادگیری و آموزش بسیار نامفهوم باشد که برای فهم بیشتر موضوع، باید کدها را بصورت کامل تست نموده و مثالهایی را پیاده سازی نمایید.

نظرات خوانندگان

نویسنده: سینا کردی
تاریخ: ۱۳۹۱/۱۲/۳۰ ۴:۱۰

سلام
ممنون از شما این بخش هم کامل و زیبا بود
ولی کمی فشرده بود
لطفا اگر ممکن هست در مورد معماری‌ها و الگوها و بهترین‌های آنها کمی توضیح دهید یا منبعی معرفی کنید تا این الگوها و معماری برای ما بیشتر مفهوم بشه
من در این زمینه تازه کارم و از شما میخواهم که من رو راهنمایی کنید که چه مقدماتی در این زمینه‌ها نیاز دارم
باز هم ممنون.

نویسنده: علي
تاریخ: ۱۳۹۱/۱۲/۳۰ ۹:۱۲

در همین سایت مباحث الگوهای طراحی و Refactoring مفید هستند.

و یا الگوهای طراحی Agile رو هم در اینجا می‌تونید پیگیری کنید.

نویسنده: میثم خوشبخت
تاریخ: ۱۳۹۱/۱۲/۳۰ ۱۱:۳۸

فشردگی این مباحث بخاطر این بود که میخواستم فعلا یک نمونه پروژه رو آموزش بودم تا یک شمای کلی از کاری که می‌خواهیم انجام بدیم رو ببینید. در مباحث بعدی این مباحث رو بازتر می‌کنم. خود من برای مطالعه و جمع بندی این مباحث منابع زیادی رو مطالعه کردم. واقعا برای بعضی مباحث نمیشه به یک منبع اکتفا کرد.

نویسنده: محسن.د
تاریخ: ۱۳۹۱/۱۲/۳۰ ۱۷:۱

بسیار عالی

آیا فراخوانی مستقیم تابع SetDiscountStrategyTo کلاس Price در تابع الحاقی Apply از نظر کپسوله سازی مورد اشکال نیست ؟ بهتر نیست که برای خود کلاس Product یک تابع پیاده سازی کنیم که در درون خودش تابع Price.SetDiscountStrategyTo را فراخوانی کند و به این شکل کلاس‌های بیرونی رو از تغییرات درونی کلاس Product مستقل کنیم ؟

نویسنده: میثم خوشبخت
تاریخ: ۱۳۹۱/۱۲/۳۰ ۱۸:۱

دوست عزیزم. متد Apply یک Extension Method برای IList<Product> است. اگر این متد تعریف نمی‌شد شما باید در کلاس سرویس حلقه foreach رو قرار می‌دادید. البته با این حال در قسمت هایی از طراحی کلاسها که الگوهای طراحی را زیر سوال نمی‌برد و تست پذیری را دچار مشکل نمی‌کند، طراحی سلیقه ای است. مقاله من هم آیه‌ی نازل شده نیست که دستخوش تغییرات نشود. شما می‌توانید با سلیقه و دید فنی خود تغییرات مورد نظر رو اعمال کنید. ولی اگر نظر من را بخواهید این طراحی مناسب‌تر است.

نویسنده: رضا عرب
تاریخ: ۱۳۹۲/۰۱/۰۹ ۱۴:۴۵

خسته نباشید، واقعا ممنونم آقای خوشبخت، لطفا به نگارش این دست مطالب مرتبط با طراحی ادامه دهید، زمینه بکریه که کمتر عملی به آن پرداخته شده و این نوع نگارش شما فراتر از یک معرفیه که واقعا جای تشکر داره.

نویسنده: f.tahan36
تاریخ: ۱۳۹۲/۰۲/۲۹ ۱۷:۱۰

با سلام

تفاوت factory با design factory در چیست؟ (با مثال کد)

و virtual کردن یک تابع معمولی با virtual کردن تابع سازنده چه تفاوتی دارد؟

با تشکر

نویسنده: محسن خان
تاریخ: ۱۳۹۲/۰۲/۳۰ ۰:۴۰

از همون رندهایی هستی که تمرین کلاسیت رو آوردی اینجا؟! :)

Service Layer

نقش لایه‌ی سرویس این است که به عنوان یک مدخل ورودی به برنامه کاربردی عمل کند. در برخی مواقع این لایه را به عنوان لایه‌ی Facade نیز می‌شناسند. این لایه، داده‌ها را در قالب یک نوع داده ای قوی (Strongly Typed) به نام View Model ، برای لایه‌ی Presentation فراهم می‌کند. کلاس View Model یک Strongly Typed محسوب می‌شود که نماهای خاصی از داده‌ها را که متفاوت از دید یا نمای تجاری آن است، بصورت بهینه ارائه می‌نماید. در مورد الگوی View Model در مباحث بعدی بیشتر صحبت خواهم کرد.

الگوی Facade یک Interface ساده را به منظور کنترل دسترسی به مجموعه ای از Interface ها و زیر سیستم‌های پیچیده ارائه می‌کند. در مباحث بعدی در مورد آن بیشتر صحبت خواهم کرد.

کلاسی با نام ProductViewModel را با کد زیر به پروژه SoCPatterns.Layered.Service اضافه کنید:

```
public class ProductViewModel
{
    Public int ProductId {get; set;}
    public string Name { get; set; }
    public string Rrp { get; set; }
    public string SellingPrice { get; set; }
    public string Discount { get; set; }
    public string Savings { get; set; }
}
```

برای اینکه کلاینت با لایه‌ی سرویس در تعامل باشد باید از الگوی Request/Response Message استفاده کنیم. بخش Request توسط کلاینت تغذیه می‌شود و پارامترهای مورد نیاز را فراهم می‌کند. کلاسی با نام ProductListRequest را با کد زیر به پروژه SoCPatterns.Layered.Service اضافه کنید:

```
using SoCPatterns.Layered.Model;

namespace SoCPatterns.Layered.Service
{
    public class ProductListRequest
    {
        public CustomerType CustomerType { get; set; }
    }
}
```

در شی Response نیز بررسی می‌کنیم که درخواست به درستی انجام شده باشد، داده‌های مورد نیاز را برای کلاینت فراهم می‌کنیم و همچنین در صورت عدم اجرای صحیح درخواست، پیام مناسب را به کلاینت ارسال می‌نماییم. کلاسی با نام ProductListResponse را با کد زیر به پروژه SoCPatterns.Layered.Service اضافه کنید:

```
public class ProductListResponse
{
    public bool Success { get; set; }
```

```
    public string Message { get; set; }
    public IList<ProductViewModel> Products { get; set; }
}
```

به منظور تبدیل موجودیت Product به ProductViewModel ، به دو متد نیاز داریم، یکی برای تبدیل یک Product و دیگری برای تبدیل لیستی از Product . شما می‌توانید این دو متد را به کلاس Product موجود در Domain Model اضافه نمایید، اما این متدها نیاز واقعی منطق تجاری نمی‌باشند. بنابراین بهترین انتخاب، استفاده از Extension Method ها می‌باشد که باید برای کلاس Product و در لایه‌ی سرویس ایجاد نمایید. کلاسی با نام ProductMapperExtensionMethods را با کد زیر به پروژه SoCPatterns.Layered.Service اضافه کنید:

```
public static class ProductMapperExtensionMethods
{
    public static ProductViewModel ConvertToProductViewModel(this Model.Product product)
    {
        ProductViewModel productViewModel = new ProductViewModel();
        productViewModel.ProductId = product.Id;
        productViewModel.Name = product.Name;
        productViewModel.RRP = String.Format("{0:C}", product.Price.RRP);
        productViewModel.SellingPrice = String.Format("{0:C}", product.Price.SellingPrice);
        if (product.Price.Discount > 0)
            productViewModel.Discount = String.Format("{0:C}", product.Price.Discount);
        if (product.Price.Savings < 1 && product.Price.Savings > 0)
            productViewModel.Savings = product.Price.Savings.ToString("#%");
        return productViewModel;
    }
    public static IList<ProductViewModel> ConvertToProductListViewModel(
        this IList<Model.Product> products)
    {
        IList<ProductViewModel> productViewModels = new List<ProductViewModel>();
        foreach(Model.Product p in products)
        {
            productViewModels.Add(p.ConvertToProductViewModel());
        }
        return productViewModels;
    }
}
```

حال کلاس ProductService را جهت تعامل با کلاس سرویس موجود در Domain Model و به منظور برگرداندن لیستی از محصولات و تبدیل آن به لیستی از ProductViewModel ، ایجاد می‌نماییم. کلاسی با نام ProductService را با کد زیر به پروژه SoCPatterns.Layered.Service اضافه کنید:

```
public class ProductService
{
    private Model.ProductService _productService;
    public ProductService(Model.ProductService ProductService)
    {
        _productService = ProductService;
    }
    public ProductListResponse GetAllProductsFor(
        ProductListRequest productListRequest)
    {
        ProductListResponse productListResponse = new ProductListResponse();
        try
        {
            IList<Model.Product> productEntities =
                _productService.GetAllProductsFor(productListRequest.CustomerType);
            productListResponse.Products = productEntities.ConvertToProductListViewModel();
            productListResponse.Success = true;
        }
        catch (Exception ex)
        {
            // Log the exception…
            productListResponse.Success = false;
            // Return a friendly error message
```

```
            productListResponse.Message = ex.Message;
        }
        return productListResponse;
    }
}
```

کلاس Service تمامی خطاها را دریافت نموده و پس از مدیریت خطا، پیغامی مناسب را به کلاینت ارسال می‌کند. همچنین این لایه محل مناسبی برای Log کردن خطاها می‌باشد. در اینجا کد نویسی لایه سرویس به پایان رسید و در ادامه به کدنویسی Data Layer می‌پردازیم.

Data Layer

برای ذخیره سازی محصولات، یک بانک اطلاعاتی با نام Shop01 ایجاد کنید که شامل جدولی به نام Product با ساختار زیر باشد:

Product

Column Name	Data Type	Allow Nulls
ProductId	int	☐
ProductName	nvarchar(50)	☐
Rrp	smallmoney	☐
SellingPrice	smallmoney	☐
		☐

برای اینکه کدهای بانک اطلاعاتی را سریعتر تولید کنیم از روش Linq to SQL در Data Layer استفاده می‌کنم. برای این منظور یک Data Context برای Linq to SQL به این لایه اضافه می‌کنیم. بر روی پروژه SoCPatterns.Layered.Repository کلیک راست نمایید و گزینه Add > New Item را انتخاب کنید. در پنجره ظاهر شده و از سمت چپ گزینه Data و سپس از سمت راست گزینه Linq to SQL Classes را انتخاب نموده و نام آن را Shop.dbml تعیین نمایید.

از طریق پنجره Server Explorer به پایگاه داده مورد نظر متصل شوید و با عمل Drag & Drop جدول Product را به بخش Design کشیده و رها نمایید.

اگر به یاد داشته باشید، در لایه Model برای برقراری ارتباط با پایگاه داده از یک Interface به نام IProductRepository استفاده نمودیم. حال باید این Interface را پیاده سازی نماییم. کلاسی با نام ProductRepository را با کد زیر به پروژه SoCPatterns.Layered.Repository اضافه کنید:

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;
using SoCPatterns.Layered.Model;

namespace SoCPatterns.Layered.Repository
{
    public class ProductRepository : IProductRepository
    {
        public IList<Model.Product> FindAll()
        {
            var products = from p in new ShopDataContext().Products
                           select new Model.Product
                           {
                               Id = p.ProductId,
                               Name = p.ProductName,
                               Price = new Model.Price(p.Rrp, p.SellingPrice)
                           };
            return products.ToList();
        }
    }
}
```

در متد FindAll ، با استفاده از دستورات Linq to SQL ، لیست تمامی محصولات را برگرداندیم. کدنویسی لایه‌ی Data هم به پایان رسید و در ادامه به کدنویسی لایه‌ی Presentation و UI می‌پردازیم.

Presentation Layer

به منظور جداسازی منطق نمایش (Presentation) از رابط کاربری (User Interface) ، از الگوی Model View Presenter یا همان MVP استفاده می‌کنیم که در مباحث بعدی با جزئیات بیشتری در مورد آن صحبت خواهم کرد. یک Interface با نام IProductListView را با کد زیر به پروژه SoCPatterns.Layered.Presentation اضافه کنید:

```
using SoCPatterns.Layered.Service;

public interface IProductListView
{
    void Display(IList<ProductViewModel> Products);
    Model.CustomerType CustomerType { get; }
    string ErrorMessage { set; }
}
```

این Interface توسط Web Form های ASP.NET و یا Win Form ها باید پیاده سازی شوند. کار با Interface ها موجب می‌شود تا تست View ها به راحتی انجام شوند. کلاسی با نام ProductListPresenter را با کد زیر به پروژه SoCPatterns.Layered.Presentation اضافه کنید:

```
using SoCPatterns.Layered.Service;

namespace SoCPatterns.Layered.Presentation
{
    public class ProductListPresenter
    {
        private IProductListView _productListView;
        private Service.ProductService _productService;
        public ProductListPresenter(IProductListView ProductListView,
            Service.ProductService ProductService)
        {
            _productService = ProductService;
            _productListView = ProductListView;
        }
        public void Display()
        {
            ProductListRequest productListRequest = new ProductListRequest();
            productListRequest.CustomerType = _productListView.CustomerType;
            ProductListResponse productResponse =
                _productService.GetAllProductsFor(productListRequest);
            if (productResponse.Success)
            {
                _productListView.Display(productResponse.Products);
            }
            else
            {
                _productListView.ErrorMessage = productResponse.Message;
            }
        }
    }
}
```

کلاس Presenter وظیفه‌ی واکشی داده ها، مدیریت رویدادها و بروزرسانی UI را دارد. در اینجا کدنویسی لایه‌ی Presentation به پایان رسیده است. از مزایای وجود لایه‌ی Presentation این است که تست نویسی مربوط به نمایش داده‌ها و تعامل بین کاربر و سیستم به سهولت انجام می‌شود بدون آنکه نگران دشواری Unit Test نویسی Web Form ها باشید. حال می‌توانید کد نویسی مربوط به UI را انجام دهید که در ادامه به کد نویسی در Win Forms و Web Forms خواهیم پرداخت.

نظرات خوانندگان

نویسنده: محسن
تاریخ: ۱۳۹۲/۰۱/۰۲ ۱۸:۲۹

ممنون از زحمات شما.
چند سؤال و نظر:
- با تعریف الگوی مخزن به چه مزیتی دست پیدا کردید؟ برای مثال آیا هدف این است که کدهای پیاده سازی آن، با توجه به وجود اینترفیس تعریف شده، شاید روزی با مثلا NHibernate تعویض شود؟ در عمل متاسفانه حتی پیاده سازی LINQ این‌ها هم متفاوت است و من تابحال در عمل ندیدم که ORM یک پروژه بزرگ رو عوض کنند. یعنی تا آخر و تا روزی که پروژه زنده است با همان انتخاب اول سر می‌کنند. یعنی شاید بهتر باشه قسمت مخزن و همچنین سرویس یکی بشن.
- چرا لایه سرویس تعریف شده از یک یا چند اینترفیس مشتق نمی‌شود؟ اینطوری تهیه تست برای اون ساده‌تر میشه. همچنین پیاده سازی‌ها هم وابسته به یک کلاس خاص نمی‌شن چون از اینترفیس دارن استفاده می‌کنند.
- این اشیاء Request و Response هم در عمل به نظر نوعی ViewModel هستند. درسته؟ اگر اینطوره بهتر یک مفهوم کلی دنبال بشه تا سردرگمی‌ها رو کمتر کنه.

یک سری نکته جانبی هم هست که می‌تونه برای تکمیل بحث جالب باشه:
- مثلا الگوی Context per request بجای نوشتن new ShopDataContext بهتر استفاده بشه تا برنامه در طی یک درخواست در یک تراکنش و اتصال کار کنه.
- در مورد try/catch و استفاده از اون بحث زیاد هست. خیلی‌ها توصیه می‌کنن که یا اصلا استفاده نکنید یا استفاده از اون‌ها رو به بالاترین لایه برنامه موکول کنید تا این وسط کرش یک قسمت و بروز استثناء در اون، از ادامه انتشار صدمه به قسمت‌های بعدی جلوگیری کنه.

نویسنده: میثم خوشبخت
تاریخ: ۱۳۹۲/۰۱/۰۲ ۲۳:۳۵

محسن عزیز. از شما ممنونم که به نکته‌های ظریفی اشاره کردید.

در سری مقالات اولیه فقط دارم یک دید کلی به کسایی میدم که تازه دارن با این مفاهیم آشنا میشن. این پروژه اولیه دستخوش تغییرات زیادی میشه. در واقع محصول نهایی این مجموعه مقالات بر پایه همین نوع لایه بندی ولی بادید و طراحی مناسب‌تر خواهد بود.
در مورد ORM هم من با چند Application سروکار داشتم که در روال توسعه بخش‌های جدید رو بنا به دلایلی با ORM یا DB متفاوتی توسعه داده اند. غیر از این موضوع، حتی بخشهایی از مدل، سرویس و یا مخزن رو در پروژه‌های دیگری استفاده کرده اند. همچنین برخی از نکات مربوط به تفکیک لایه‌ها به منظور تست پذیری راحت‌تر رو هم در نظر بگیرید.
در مورد اشیا Request و Response هم باید خدمتتان عرض کنم که برای درخواست و پاسخ به درخواست استفاده می‌شوند که چون پروژه ای که مثال زدم کوچک بوده ممکنه کاملا درکش نکرده باشید. ما کلاسهای Request و Response متعددی در پروژه داریم که ممکنه خیلی از اونها فقط از یک View Model استفاده کنن ولی پارامترهای ارسالی یا دریافتی آنها متفاوت باشد.
در مورد try...catch هم من با شما کاملا موافقم. به دلیل هزینه ای که دارد باید در آخرین سطح قرار بگیرد. در این مورد ما میتونیم اونو به Presentation و یا در MVC به Controller منتقل کنیم.
در مورد DbContext هم هنوز الگویی رو معرفی نکردم. در واقع هنوز وارد جزئیات لایه‌ی Data نشدم. در مورد اون اگه اجازه بدی بعدا صحبت میکنم.

نویسنده: ایلیا
تاریخ: ۱۳۹۲/۰۱/۰۳ ۰:۴۳

آقای خوشبخت خداقوت.

مرسی از مطالب خوبتون.

نویسنده: وحید نصیری
تاریخ: ۱۳۹۲/۰۱/۰۳ ۰:۴۸

لطفا برای اینکه نظرات حالت فنی‌تر و غنای بیشتری پیدا کنند، از ارسال پیام‌های تشکر خودداری کنید. برای ابراز احساسات و همچنین تشکر، لطفا از گزینه رای دادن به هر مطلب که ذیل آن قرار دارد استفاده کنید.
این مطلب تا این لحظه 76 بار دیده شده، اما فقط 4 رای دارد. لطفا برای ابراز تشکر، امتیاز بدهید. ممنون.

نویسنده: محسن
تاریخ: ۱۳۹۲/۰۱/۰۳ ۱:۰

- من در عمل تفاوتی بین لایه مخزن و سرویس شما مشاهده نمی‌کنم. یعنی لایه مخزن داره GetAll می‌کنه، بعد لایه سرویس هم داره همون رو به یک شکل دیگری بر می‌گردونه. این تکرار کد نیست؟ این دو یکی نیستند؟

عموما در منابع لایه مخزن رو به صورت روکشی برای دستورات مثلا EF یا LINQ to SQL معرفی می‌کنند. فرضشون هم این است که این روش ما رو از تماس مستقیم با ORM برحذر می‌داره (شاید فکر می‌کنند ایدز می‌گیرند اگر مستقیم کار کنند!). ولی عرض کردم این روکش در واقعیت فقط شاید با EF یا L2S قابل تعویض باشه نه با ORMهای دیگر با روش‌های مختلف و بیشتر یک تصور واهی هست که جنبه عملی نداره. بیشتر تئوری هست بدون پایه تجربه دنیای واقعی. ضمن اینکه این روکش باعث میشه نتونید از خیلی از امکانات ORM مورد استفاده درست استفاده کنید. مثلا ترکیب کوئری‌ها یا روش‌های به تاخیر افتاده و امثال این.

- پس در عمل شما Request ViewModel و Response ViewModel تعریف کردید.

نویسنده: شاهین کیاست
تاریخ: ۱۳۹۲/۰۱/۰۳ ۱۲:۲۷

سپاس از سری مطالبی که منتشر می‌کنید.
-پیشنهادی که من دارم اینه که لایه‌ی Repository حذف شود ، همانطور که در مطالب قبلی ذکر شده DbSet در Entity Framework همان پیاده سازی الگوی مخزن هست و ایجاد Repository جدید روی آن یک Abstraction اضافه هست. در نتیجه اگر Repository حذف شود همه‌ی منطق‌ها مانند GetBlaBla به Service منتقل می‌شود.
-یک پیشنهاد دیگر اینکه استفاده از کلمه‌ی New در Presentation Layer را به حداقل رساند و همه جا نیاز مندی‌ها را به صورت وابستگی به کلاس‌های استفاده کننده تزریق شود تا در زمان نوشتن تست‌ها همه‌ی اجزاء قابل تعویض با Mock objects باشند.

نویسنده: افشین
تاریخ: ۱۳۹۲/۰۱/۰۶ ۱۱:۱۵

لطفا دمو یا سورس برنامه رو هم قرار بدید که یادگیری و آموزش سریعتر انجام بشه.

ممنون

نویسنده: صابر فتح الهی
تاریخ: ۱۳۹۲/۰۱/۱۰ ۰:۱۱

با سلام از کار بزرگی که دارین می‌کنین سپاس
یک سوال؟
جای الگوی Unit Of Work در این پروژه کجا میشه؟

در این پست جناب آقای مهندس نصیری در لایه سرویس الگوی واحد کار را پیاده کرده اند، با توجه به وجود الگوی Repository

در پروژه شما ممنون میشم شرح بیشتری بدین که جایگاه پیاده سازی الگو واحد کار با توجه به مزایایی که دارد در کدام لایه است؟

نویسنده: رام
تاریخ: ۱۳۹۲/۰۱/۱۶ ۵:۲۹

محسن جان، چیزی که من از این الگو در مورد واکشی و نمایش داده‌ها برداشت میکنم اینه:

کلاس‌های لایه مخزن با دریافت دستور از لایه سرویس آبجکت مدل مربوطه را پر میکنند و به بالا (لایه سرویس) پاس میدند.

بعد

در لایه سرویس نمونه‌ی مدل مربوطه به ویـــــومدل متناظر باهاش تبدیل میشه و به لایه بالاتر فرستاده میشه

بنابراین
کار در **"لایه مخزن"** روی **"مدل ها"** انجام میگیره
و
کار در **"لایه سرویس"** روی **"ویـــــومدل ها"** انجام میشه

نتیجه: لایه سرویس هدف دیگری را نسبت به لایه مخزن دنبال میکند و این هدف آنقدر بزرگ و مهم هست که برایش یه لایه مجزا در نظر گرفته بشه

نویسنده: رام
تاریخ: ۱۳۹۲/۰۱/۱۶ ۵:۴۹

شاهین جان، من با حذف لایه مخزن مخالف هستم. زیرا:

ما لایه ای به نام "لایه مخزن" را میسازیم تا در نهایت کلیه متدهایی که برای حرف زدن با داده هامون را نیاز داریم داشته باشیم. حالا این اطلاعات ممکنه از پایگاه داده یا جاهای دیگه جمع آوری بشوند (و الزاما توسط EF قابل دسترسی و ارائه نباشند)

همچنین گاهی نیاز هست که بر مبنای چند متد که EF به ما میرسونه (مثلا چند SP) یک متد کلی‌تر را تعریف کنیم (چند فراخوانی را در یک متد مثلا متد X در لایه مخزن انجام دهیم) و در لایه بالاتر آن متد را صدا بزنیم (بجای نوشتن و تکرار پاپی همه کدهای نوشت شده در متد X)

علاوه بر این در لایه مخزن میشه چند ORM را هم کنار هم دید (نه فقط EF) که همونطور که آقای خوشبخت در کامنت‌ها نوشتند گاهی نیاز میشه.

بنابراین:
من وجود لایه مخزن را ضروری میدونم.

(فراموش نکنیم که هدف از این آموزش تعریف یک الگوی معماری مناسب برای پروژه‌های بزرگ هست و الا بدون خیلی از اینها هم میشه برنامه ساخت. همونطور که اکثرا بدون این ساختارها و خیلی ساده‌تر میسازند)

نویسنده: محسن
تاریخ: ۱۳۹۲/۰۱/۱۶ ۹:۳

- بحث آقای شاهین و من در مورد مثال عینی بود که زده شد. در مورد کار با ORM که کدهاش دقیقا ارائه شده. این روش قابل نقد و رد است.

شما الان اومدی یک بحث انتزاعی کلی رو شروع کردید. بله. اگر ORM رو کنار بگذارید مثلا می‌رسید به ADO.NET (یک نمونه که خیلی‌ها در این سایت حداقل یکبار باهاش کار کردن). این افراد پیش از اینکه این مباحث مطرح باشن برای خودشون لایه DAL داشتند و تمام جزئیات ADO.NET رو کپسوله کرده بودن در اون. حالا با اومدن ORMها این لایه DAL کنار رفته چون خود ORM هست که کپسوله کننده ADO.NET است. همین‌ها هم یک لایه دیگر داشتند به نام BLL که از لایه DAL استفاده می‌کرد برای پیاده سازی منطق تجاری برنامه. این لایه الان اسمش شده لایه سرویس.

یعنی تمام مواردی رو که عنوان کردید در مورد ADO.NET صدق می‌کنه. یکی اسمش رو می‌ذاره DAL شما اسمش رو گذشتید Repository. ولی این مباحث ربطی به یک ORM تمام عیار که کپسوله کننده ADO.NET است ندارد.

- ترکیب چند SP در لایه مخزن انجام نمیشه. چیزی رو که عنوان کردید یعنی پیاده سازی منطق تجاری و این مورد باید در لایه سرویس باشه. اگر از ADO.NET استفاده میشه، می‌تونیم با استفاده از DAL جزئیات دسترسی به SP رو مخفی و ساده‌تر کنیم با کدی یک دست‌تر در تمام برنامه. اگر از EF استفاده می‌کنیم، باز همین ساده سازی در طی فراخوانی فقط یک متد انجام شده. بنابراین بهتر است وضعیت و سطح لایه‌ای رو که داریم باهاش کار می‌کنیم خوب بررسی و درک کنیم.

- می‌تونید در عمل در بین پروژه‌های سورس باز و معتبر موجود فقط یک نمونه رو به من ارائه بدید که در اون از 2 مورد ORM مختلف همزمان استفاده شده باشه؟ این مورد یعنی سؤ مدیریت. یعنی پراکندگی و انجام کاری بسیار مشکل مثلا یک نمونه: ORMها لایه‌ای دارند به نام سطح اول کش که مثلا در EF اسمش هست Trackig API. این لایه فقط در حین کار با Context همون ORM کار می‌کنه. اگر دو مورد رو با هم مخلوط کنید، قابل استفاده نیست، ترکیب پذیر نیستند. از این دست باز هم هست مثلا در مورد نحوه تولید پروکسی‌هایی که برای lazy loading تولید می‌کنند و خیلی از مسایل دیگری از این دست. ضمن اینکه مدیریت چند Context فقط در یک لایه خودش یعنی نقض اصل تک مسئولیتی کلاس‌ها.

نویسنده: محسن
تاریخ: ۱۳۹۲/۰۱/۱۶ ۹:۱۵

سعی نکنید انتزاعی بحث کنید. چون در این حالت این حرف می‌تونه درست باشه یا حتی نباشه. اگر از ADO.NET استفاده می‌کنید، درسته. اگر از EF استفاده می‌کنید غلط هست. لازم هست منطق کار با ADO.NET رو یک سطح کپسوله کنیم. چون از تکرار کد جلوگیری می‌کنه و نهایتا به یک کد یک دست خواهیم رسید. لازم نیست اعمال یک ORM رو در لایه‌ای به نام مخزن کپسوله کنیم، چون خودش کپسوله سازی ADO.NET رو به بهترین نحوی انجام داده. برای نمونه در همین مثال عینی بالا به هیچ مزیتی نرسیدیم. فقط یک تکرار کد است. فقط بازی با کدها است.

نویسنده: رام
تاریخ: ۱۳۹۲/۰۱/۱۶ ۱۶:۴۶

من منظور شما را خوب متوجه میشم ولی حرفام یه بحث انتزاعی نیست چون پروژه عملی زیر دستم دارم که توی اون هم با پر کردن View Model کار میکنم.

مشکل از اینجا شروع میشه که شما فکر میکنید همیشه مدل ای که در EF ساختید را باید بدون تغییر در ساختارش به پوسته برنامه برسونید و از پوسته هم دقیقا نمونه ای از همون را بگیرید و به لایه‌های پایین بفرستید ولی یکی از مهمترین کارهای View Model اینه که این قانون را از این سفتی و سختی در بیاره چون خیلی مواقع هست که شما در پوسته برنامه به شکل دیگه ای از داده‌ها (متفاوت با اونچه در Model تعریف کردید و EF باهاش کار میکنه) نیاز دارید. مثلا فیلد تاریخ از نوع DateTime در Model و نوع String در پوسته و یا حتی اضافه و کم کردن فیلدهای یک Model و ایجاد ساختارهای متفاوتی از اون برای عملیات‌های Select, Update و Delete. لذا لایه سرویس قرار نیست فقط همون کار لایه مخزن را تکرار کنه (به قول شما GetAll). بلکه در زمان لزوم تغییرات لازم که نام بردم را هم رووش اعمال میکنه (که به نظر من آقای خوشبخت هم به خوبی از کلمه Convert در لایه سرویس استفاده کردند.)

اما بحث اینکه ما در لایه مخزن روی EF یک سطح کپسوله میسازیم جای گفتگو داره هرچند من در اون مورد هم با وجد لایه مخزن بیشتر موافقم تا گفتگوی مستقیم لایه سرویس با چیزی مثل EF

نتیجه: فرقی نمیکنه شما از Asp.Net استفاده میکنید یا هر ORM مورد نظرتون. کلاس‌های مدل باید در ارتباط با لایه بالاتر خودشون به ویـــو مدل تبدیل بشند و در این الگو این کار در لایه سرویس انجام میشه.

نویسنده: محسن
تاریخ: ۱۳۹۲/۰۱/۱۶ ۱۷:۱۰

- پیاده سازی الگوی مخزن در عمل (بر اساس بحث فعلی که در مورد کار با ORMها است) به صورت کپسوله سازی ORM در همه جا مطرح میشه و اینکار اساسا اشتباه هست. چون هم شما رو محروم می‌کنه از قابلیت‌های پیشرفته ORM و هم ارزش افزوده‌ای رو به همراه نداره. دست آخر می‌بینید در لایه مخزن GetAll دارید در لایه سرویس هم GetAll دارید. این مساله هیچ مزیتی نداره. یک زمانی در ADO.NET برای GetAll کردن باید کلی کد شبیه به کدهای یک ORM نوشته می‌شد. خود ORM الان اومده این‌ها رو کپسوله کرده و لایه‌ای هست روی اون. اینکه ما مجددا یک پوسته روی این بکشیم حاصلی نداره بجز تکرار کد. عده‌ای عنوان می‌کنند که حاصل اینکار امکان تعویض ORM رو ممکن می‌کنه ولی این‌ها هم بعد از یک مدت تجربه با ORMهای مختلف به این نتیجه می‌رسند که ای بابا! حتی پیاده سازی LINQ این ORMها یکی نیست چه برسه به قابلیت‌های پیشرفته‌ای که در یکی هست در دوتای دیگر نیست (واقع بینی، بجای بحث تئوری محض).

- اینکه این تبدیلات (پر کردن ViewModel از روی مدل) هم می‌تونه و بهتره که (نه الزاما) در لایه سرویس انجام بشه، نتیجه مناسبی هست.

عنوان: **اجرای یک Script حاوی دستورات Go در سی شارپ**
نویسنده: شاهین کیاست
تاریخ: ۱۳۹۲/۰۱/۰۳ ۱۲:۱۰
آدرس: www.dotnettips.info
برچسب‌ها: C#, SQL Server

سلام ؛

سال نو مبارک ! امیدوارم سال بسیار خوبی در پیش داشته باشید :)

از زمانی که استفاده از ORMهای Code First رایج شده ، اجرای اسکریپت‌های طولانی جهت ایجاد دیتابیس خیلی استفاده ندارد، اما حالت خاص همیشه پیش می‌آید.
مثلا قصد داریم پیش از آغاز برنامه پس از ایجاد دیتابیس توسط Entity Framework به یک سری جداول فیلدی با نوع داده‌ی Geometry اضافه کنیم. یا باید به دیتابیس یک سری Stored Procedure و View اضافه کرد.
Scriptها ی Generate شده توسط SQL Server حاوی دستور Go هستند. ADO.NET اجرای Script که حاوی Go باشد را پشتیبانی نمی‌کند.

اما روش‌ها و ترفندهای زیادی برای اجرای یک فایل Script طولانی حاوی دستور Go روی دیتابیس وجود دارد :

```
private static string GetScript()
        {
            string path = AppDomain.CurrentDomain.BaseDirectory +
                          @"Scripts\script.sql";
            var file = new FileInfo(path);

            string script = file.OpenText().ReadToEnd();
            return script;
        }

        private static void ExecuteScript()
        {
            string script = GetScript();

            //split the script on "GO" commands
            var splitter = new[] {"\r\nGO\r\n"};
            string[] commandTexts = script.Split(splitter,
                                                 StringSplitOptions.RemoveEmptyEntries);
            foreach (string commandText in commandTexts)
            {
                using (var ctx = new DbContext())
                {
                    if (!string.IsNullOrEmpty(commandText))
                    {
                        ctx.Database.ExecuteSqlCommand(commandText);
                    }
                }
            }
        }
```

در اینجا به جای آنکه تلاش کنیم یک فایل را روی دیتابیس یک جا اجرا کنیم دستورات را جدا کرده و تک به تک اجرا می‌کنیم.
بروزرسانی: روش بهتر

نظرات خوانندگان

نویسنده: محسن
تاریخ: ۱۳۹۲/۰۱/۰۳ ۱۷:۱۷

ممنون. میشه برای بهبود سرعت insert های زیاد، از SQL Bulk Copy هم استفاده کرد.

نویسنده: شاهین کیاست
تاریخ: ۱۳۹۲/۰۱/۰۳ ۱۹:۲۲

بله درسته،
در سناریویی که برای خودم پیش آمده بود کوئری‌ها فقط Insert نبودند و همچنین فقط قرار بود یک بار در ابتدای برنامه اجرا شوند برای همین روی سرعت حساسیت نداشتم.

نویسنده: مهدی موسوی
تاریخ: ۱۳۹۲/۰۱/۰۳ ۱۹:۵۳

سلام؛ سال نو مبارک :)
آقای کیاست، بهترین روش برای اجرای Script هایی که حاوی کلمه کلیدی GO هستند، استفاده از SMO است. به کد زیر دقت کنید:

```
Server server = new Server(new ServerConnection() { ConnectionString = cnnStr });
server.ConnectionContext.ExecuteNonQuery(sql);
```

اینجا همه چیز توسط SMO کنترل میشه و دیگه نیازی به آنالیز Script اصلی بر اساس عبارت GO نیست.

موفق باشید.

نویسنده: شاهین کیاست
تاریخ: ۱۳۹۲/۰۱/۰۴ ۰:۱۴

سلام قربان ؛
خیلی ممنون.
در استفاده از SMO در .NET 4 مشکل داشتم در نتیجه از این روش که نیازمند به Assemblyهای SMO نیست استفاده کردم.
کامنت شما را در انتهای مطلب قرار دادم.

نویسنده: میثم خوشبخت
تاریخ: ۱۳۹۲/۰۱/۰۵ ۱۷:۱۸

شاهین جان
ممنون از مطلب مفیدت
من این مورد رو قبلا نوشتم. به همین روشی که شما نوشتید. منتها تو یه Query با مشکل مواجه شد و اجرا نشد. واسه همین از روش زیر استفاده کردم که همه جا جواب میده:

```
var commandTexts=Regex.Split(script,@"\s*GO\s+");
foreach(var commandText in commandTexts)
{
    // Execution Code
}
```

نویسنده: مهدی موسوی

تاریخ: ۱۳۹۲/۰۱/۰۶ ۱۵:۳۶

سلام.
خیر، همه جا جواب نمیده! عرض کردم، نیازی به جداسازی Script بر اساس GOهای موجود در اون ندارید. Script زیر رو در نظر بگیرید (بعنوان مثالی نقض) تا متوجه بشید چرا کد شما نیز ایراد داره:

```
USE [MyDB]
GO

SET ANSI_NULLS ON
GO

SET QUOTED_IDENTIFIER ON
GO

CREATE TABLE [dbo].[GO AND EAT](
[Id] [int] IDENTITY(1,1) NOT NULL,
[MyVal] [decimal](18, 0) NOT NULL,
) ON [PRIMARY]

GO
```

لطفا اینو در نظر بگیرید که منظورم این نیست که با تصحیح Regex Pattern می‌تونید این مشکل رو نیز برطرف کنید، منظورم اینه که با استفاده از SMO می‌تونید از نوشتن کدهای Error-Prone دوری کنید. فقط همین. :)

موفق باشید.

نویسنده: وحید نصیری
تاریخ: ۱۳۹۲/۰۱/۰۶ ۱۷:۱۸

پشت صحنه SMO برای علاقمندان در مورد نحوه جدا سازی batchها از یکدیگر در عمل.

نویسنده: علی
تاریخ: ۱۳۹۲/۰۲/۲۴ ۱۵:۴۷

برای اجرای چنین دستوراتی نیازی به تکه تکه کردن نداریم فقط کافیه go رو از اسکریپت حدف کنیم خودش همرو اجرا می‌کنه قبلا خودم چنین کاری رو انجام دادم

نویسنده: محسن خان
تاریخ: ۱۳۹۲/۰۲/۲۴ ۱۶:۵۸

کل این بحث در مورد نحوه یافتن صحیح این Go بدون تخریب اطلاعات جانبی بود. حالا می‌خوای حذفش کن یا تکه تکه‌اش کن!

UI

در نهایت نوبت به طراحی و کدنویسی UI می‌رسد تا بتوانیم محصولات را به کاربر نمایش دهیم. اما قبل از شروع باید موضوعی را یادآوری کنم. اگر به یاد داشته باشید، در کلاس ProductService موجود در لایه‌ی Domain ، از طریق یکی از روشهای الگوی Dependency Injection به نام Constructor Injection ، فیلدی از نوع IProductRepository را مقداردهی نمودیم. حال زمانی که بخواهیم نمونه ای را از ProductService ایجاد نماییم، باید به عنوان پارامتر ورودی سازنده، شی ایجاد شده از جنس کلاس ProductRepository موجود در لایه Repository را به آن ارسال نماییم. اما از آنجایی که می‌خواهیم تفکیک پذیری لایه‌ها از بین نرود و UI بسته به نیاز خود، نمونه مورد نیاز را ایجاد نموده و به این کلاس ارسال کند، از ابزارهایی برای این منظور استفاده می‌کنیم. یکی از این ابزارها StructureMap می‌باشد که یک Inversion of Control Container یا به اختصار IoC Container نامیده می‌شود. با Inversion of Control در مباحث بعدی بیشتر آشنا خواهید شد. StructureMap ابزاری است که در زمان اجرا، پارامترهای ورودی سازنده‌ی کلاسهایی را که از الگوی Dependency Injection استفاده نموده اند و قطعا پارامتر ورودی آنها از جنس یک Interface می‌باشد را، با ایجاد شی مناسب مقداردهی می‌نماید.

به منظور استفاده از StructureMap در Visual Studio 2012 باید بر روی پروژه UI خود کلیک راست نموده و گزینه‌ی Manage NuGet Packages را انتخاب نمایید. در پنجره ظاهر شده و از سمت چپ گزینه‌ی Online و سپس در کادر جستجوی سمت راست و بالای پنجره واژه‌ی structuremap را جستجو کنید. توجه داشته باشید که باید به اینترنت متصل باشید تا بتوانید Package مورد نظر را نصب نمایید. پس از پایان عمل جستجو، در کادر میانی structuremap ظاهر می‌شود که می‌توانید با انتخاب آن و فشردن کلید Install آن را بر روی پروژه نصب نمایید.

جهت آشنایی بیشتر با NuGet و نصب آن در سایر نسخه‌های Visual Studio می‌توانید به لینک‌های زیر رجوع کنید:

1. آشنایی با NuGet قسمت اول

2. آشنایی با NuGet قسمت دوم

3. Installing NuGet

کلاسی با نام BootStrapper را با کد زیر به پروژه UI خود اضافه کنید:

```
using StructureMap;
using StructureMap.Configuration.DSL;
using SoCPatterns.Layered.Repository;
using SoCPatterns.Layered.Model;

namespace SoCPatterns.Layered.WebUI
{
    public class BootStrapper
    {
        public static void ConfigureStructureMap()
        {
            ObjectFactory.Initialize(x => x.AddRegistry<ProductRegistry>());
        }
    }
```

```
    public class ProductRegistry : Registry
    {
        public ProductRegistry()
        {
            For<IProductRepository>().Use<ProductRepository>();
        }
    }
}
```

ممکن است یک WinUI ایجاد کرده باشید که در این صورت به جای فضای نام SoCPatterns.Layered.WebUI از SoCPatterns.Layered.WinUI استفاده نمایید.

هدف کلاس BootStrapper این است که تمامی وابستگی‌ها را توسط StructureMap در سیستم Register نماید. زمانی که کدهای کلاینت می‌خواهند به یک کلاس از طریق StructureMap دسترسی داشته باشند، Structuremap وابستگی‌های آن کلاس را تشخیص داده و بصورت خودکار پیاده سازی واقعی (Concrete Implementation) آن کلاس را، براساس همان چیزی که ما برایش تعیین کردیم، به کلاس تزریق می‌نماید. متد ConfigureStructureMap باید در همان لحظه ای که Application آغاز به کار می‌کند فراخوانی و اجرا شود. با توجه به نوع UI خود یکی از روالهای زیر را انجام دهید:

در WebUI :

فایل Global.asax را به پروژه اضافه کنید و کد آن را بصورت زیر تغییر دهید:

```
namespace SoCPatterns.Layered.WebUI
{
    public class Global : System.Web.HttpApplication
    {
        protected void Application_Start(object sender, EventArgs e)
        {
            BootStrapper.ConfigureStructureMap();
        }
    }
}
```

در WinUI :

در فایل Program.cs کد زیر را اضافه کنید:

```
namespace SoCPatterns.Layered.WinUI
{
    static class Program
    {
        [STAThread]
        static void Main()
        {
            Application.EnableVisualStyles();
            Application.SetCompatibleTextRenderingDefault(false);
```

BootStrapper.ConfigureStructureMap();

```
Application.Run(new Form1());
        }
    }
}
```

سپس برای طراحی رابط کاربری، با توجه به نوع UI خود یکی از روالهای زیر را انجام دهید:

در WebUI :

صفحه Default.aspx را باز نموده و کد زیر را به آن اضافه کنید:

```
<asp:DropDownList AutoPostBack="true" ID="ddlCustomerType" runat="server">
    <asp:ListItem Value="0">Standard</asp:ListItem>
    <asp:ListItem Value="1">Trade</asp:ListItem>
</asp:DropDownList>
<asp:Label ID="lblErrorMessage" runat="server" ></asp:Label>
<asp:Repeater ID="rptProducts" runat="server" >
    <HeaderTemplate>
        <table>
            <tr>
                <td>Name</td>
                <td>RRP</td>
                <td>Selling Price</td>
                <td>Discount</td>
                <td>Savings</td>
            </tr>
            <tr>
                <td colspan="5"><hr /></td>
            </tr>
    </HeaderTemplate>
    <ItemTemplate>
            <tr>
                <td><%# Eval("Name") %></td>
                <td><%# Eval("RRP")%></td>
                <td><%# Eval("SellingPrice") %></td>
                <td><%# Eval("Discount") %></td>
                <td><%# Eval("Savings") %></td>
            </tr>
    </ItemTemplate>
    <FooterTemplate>
        </table>
    </FooterTemplate>
</asp:Repeater>
```

در WinUI :

فایل Form1.Designer.cs را باز نموده و کد آن را بصورت زیر تغییر دهید:

```
#region Windows Form Designer generated code
/// <summary>
/// Required method for Designer support - do not modify
/// the contents of this method with the code editor.
/// </summary>
private void InitializeComponent()
{
    this.cmbCustomerType = new System.Windows.Forms.ComboBox();
    this.dgvProducts = new System.Windows.Forms.DataGridView();
    this.colName = new System.Windows.Forms.DataGridViewTextBoxColumn();
    this.colRrp = new System.Windows.Forms.DataGridViewTextBoxColumn();
    this.colSellingPrice = new System.Windows.Forms.DataGridViewTextBoxColumn();
    this.colDiscount = new System.Windows.Forms.DataGridViewTextBoxColumn();
    this.colSavings = new System.Windows.Forms.DataGridViewTextBoxColumn();
    ((System.ComponentModel.ISupportInitialize)(this.dgvProducts)).BeginInit();
    this.SuspendLayout();
    //
    // cmbCustomerType
    //
    this.cmbCustomerType.DropDownStyle =
System.Windows.Forms.ComboBoxStyle.DropDownList;
    this.cmbCustomerType.FormattingEnabled = true;
    this.cmbCustomerType.Items.AddRange(new object[] {
        "Standard",
        "Trade"});
    this.cmbCustomerType.Location = new System.Drawing.Point(12, 90);
```

```
    this.cmbCustomerType.Name = "cmbCustomerType";
    this.cmbCustomerType.Size = new System.Drawing.Size(121, 21);
    this.cmbCustomerType.TabIndex = 3;
    //
    // dgvProducts
    //
    this.dgvProducts.ColumnHeadersHeightSizeMode =
System.Windows.Forms.DataGridViewColumnHeadersHeightSizeMode.AutoSize;
    this.dgvProducts.Columns.AddRange(new System.Windows.Forms.DataGridViewColumn[] {
    this.colName,
    this.colRrp,
    this.colSellingPrice,
    this.colDiscount,
    this.colSavings});
    this.dgvProducts.Location = new System.Drawing.Point(12, 117);
    this.dgvProducts.Name = "dgvProducts";
    this.dgvProducts.Size = new System.Drawing.Size(561, 206);
    this.dgvProducts.TabIndex = 2;
    //
    // colName
    //
    this.colName.DataPropertyName = "Name";
    this.colName.HeaderText = "Product Name";
    this.colName.Name = "colName";
    this.colName.ReadOnly = true;
    //
    // colRrp
    //
    this.colRrp.DataPropertyName = "Rrp";
    this.colRrp.HeaderText = "RRP";
    this.colRrp.Name = "colRrp";
    this.colRrp.ReadOnly = true;
    //
    // colSellingPrice
    //
    this.colSellingPrice.DataPropertyName = "SellingPrice";
    this.colSellingPrice.HeaderText = "Selling Price";
    this.colSellingPrice.Name = "colSellingPrice";
    this.colSellingPrice.ReadOnly = true;
    //
    // colDiscount
    //
    this.colDiscount.DataPropertyName = "Discount";
    this.colDiscount.HeaderText = "Discount";
    this.colDiscount.Name = "colDiscount";
    //
    // colSavings
    //
    this.colSavings.DataPropertyName = "Savings";
    this.colSavings.HeaderText = "Savings";
    this.colSavings.Name = "colSavings";
    this.colSavings.ReadOnly = true;
    //
    // Form1
    //
    this.AutoScaleDimensions = new System.Drawing.SizeF(6F, 13F);
    this.AutoScaleMode = System.Windows.Forms.AutoScaleMode.Font;
    this.ClientSize = new System.Drawing.Size(589, 338);
    this.Controls.Add(this.cmbCustomerType);
    this.Controls.Add(this.dgvProducts);
    this.Name = "Form1";
    this.Text = "Form1";
    ((System.ComponentModel.ISupportInitialize)(this.dgvProducts)).EndInit();
    this.ResumeLayout(false);
}
#endregion
private System.Windows.Forms.ComboBox cmbCustomerType;
private System.Windows.Forms.DataGridView dgvProducts;
private System.Windows.Forms.DataGridViewTextBoxColumn colName;
private System.Windows.Forms.DataGridViewTextBoxColumn colRrp;
private System.Windows.Forms.DataGridViewTextBoxColumn colSellingPrice;
private System.Windows.Forms.DataGridViewTextBoxColumn colDiscount;
private System.Windows.Forms.DataGridViewTextBoxColumn colSavings;
```

سپس در Code Behind ، با توجه به نوع UI خود یکی از روالهای زیر را انجام دهید:

در WebUI :

وارد کد نویسی صفحه Default.aspx شده و کد آن را بصورت زیر تغییر دهید:

```
using System;
using System.Collections.Generic;
using SoCPatterns.Layered.Model;
using SoCPatterns.Layered.Presentation;
using SoCPatterns.Layered.Service;
using StructureMap;

namespace SoCPatterns.Layered.WebUI
{
    public partial class Default : System.Web.UI.Page, IProductListView
    {
        private ProductListPresenter _productListPresenter;
        protected void Page_Init(object sender, EventArgs e)
        {
            _productListPresenter = new
ProductListPresenter(this,ObjectFactory.GetInstance<Service.ProductService>());
            this.ddlCustomerType.SelectedIndexChanged +=
                delegate { _productListPresenter.Display(); };
        }
        protected void Page_Load(object sender, EventArgs e)
        {
            if(!Page.IsPostBack)
                _productListPresenter.Display();
        }
        public void Display(IList<ProductViewModel> products)
        {
            rptProducts.DataSource = products;
            rptProducts.DataBind();
        }
        public CustomerType CustomerType
        {
            get { return (CustomerType) int.Parse(ddlCustomerType.SelectedValue); }
        }
        public string ErrorMessage
        {
            set
            {
                lblErrorMessage.Text =
                    String.Format("<p><strong>Error:</strong><br/>{0}</p>", value);
            }
        }
    }
}
```

در WinUI :

وارد کدنویسی Form1 شوید و کد آن را بصورت زیر تغییر دهید:

```
using System;
using System.Collections.Generic;
using System.Windows.Forms;
using SoCPatterns.Layered.Model;
using SoCPatterns.Layered.Presentation;
using SoCPatterns.Layered.Service;
using StructureMap;

namespace SoCPatterns.Layered.WinUI
{
    public partial class Form1 : Form, IProductListView
    {
        private ProductListPresenter _productListPresenter;
        public Form1()
        {
            InitializeComponent();
            _productListPresenter =
                new ProductListPresenter(this, ObjectFactory.GetInstance<Service.ProductService>());
            this.cmbCustomerType.SelectedIndexChanged +=
                delegate { _productListPresenter.Display(); };
            dgvProducts.AutoGenerateColumns = false;
```

```
            cmbCustomerType.SelectedIndex = 0;
        }
        public void Display(IList<ProductViewModel> products)
        {
            dgvProducts.DataSource = products;
        }
        public CustomerType CustomerType
        {
            get { return (CustomerType)cmbCustomerType.SelectedIndex; }
        }
        public string ErrorMessage
        {
            set
            {
                MessageBox.Show(
                    String.Format("Error:{0}{1}", Environment.NewLine, value));
            }
        }
    }
}
```

با توجه به کد فوق، نمونه ای را از کلاس ProductListPresenter ، در لحظه‌ی نمونه سازی اولیه‌ی کلاس UI ، ایجاد نمودیم. با استفاده از متد ObjectFactory.GetInstance مربوط به StructureMap ، نمونه ای از کلاس ProductService ایجاد شده است و به سازنده‌ی کلاس ProductListPresenter ارسال گردیده است. در مورد Structuremap در مباحث بعدی با جزئیات بیشتری صحبت خواهم کرد. پیاده سازی معماری لایه بندی در اینجا به پایان رسید.

اما اصلا نگران نباشید، شما فقط پرواز کوتاه و مختصری را بر فراز کدهای معماری لایه بندی داشته اید که این فقط یک دید کلی را به شما در مورد این معماری داده است. این معماری هنوز جای زیادی برای کار دارد، اما در حال حاضر شما یک Applicaion با پیوند ضعیف (Loosely Coupled) بین لایه‌ها دارید که دارای قابلیت تست پذیری قوی، نگهداری و پشتیبانی آسان و تفکیک پذیری قدرتمند بین اجزای آن می‌باشد. شکل زیر تعامل بین لایه‌ها و وظایف هر یک از آنها را نمایش می‌دهد.

نظرات خوانندگان

نویسنده: حامد
تاریخ: ۱۳۹۲/۰۱/۰۳ ۱۴:۲۹

ممنون از مقاله خوبتون
به نظر شما امکانش هست برای این معماری یک generator بسازیم به طوری که فقط ما تمام جداول دیتابیس و رابطه‌ی آنها را بسازیم و بعد این generator از روی اون تمام لایه‌ها را بر اساس آن بسازه و بعد ما صرفا جاهایی که نیاز به جزییات داره را کامل کنیم
آیا نمونه ای از این برنامه‌ها هست که این معماری یا معماری‌های مشابه را بسازه؟

نویسنده: شاهین کیاست
تاریخ: ۱۳۹۲/۰۱/۰۳ ۱۵:۳۱

اگر با T4 آشنایی داشته باشید بر اساس هر قالبی می‌توانید کد تولید کنید.

نویسنده: حامد
تاریخ: ۱۳۹۲/۰۱/۰۳ ۱۶:۳۶

متاسفانه آشنایی ندارم میشه یه توضیح مختصر بدین و یا منبع معرفی کنید

نویسنده: محسن
تاریخ: ۱۳۹۲/۰۱/۰۳ ۲۳:۵۹

چون اینجا بحث طراحی مطرح شده یک اصل رو در برنامه‌های وب باید رعایت کرد:

هیچ وقت متن خطای حاصل رو به کاربر نمایش ندید (از لحاظ امنیتی). فقط به ذکر عبارت خطایی رخ داده بسنده کنید. خطا رو مثلا توسط ELMAH لاگ کنید برای بررسی بعدی برنامه نویس.

نویسنده: شاهین کیاست
تاریخ: ۱۳۹۲/۰۱/۰۴ ۱:۲۰

http://codepanic.blogspot.com/2012/03/t4-enum.html

نویسنده: M.Q
تاریخ: ۱۳۹۲/۰۱/۰۴ ۲۲:۱۵

دوست عزیز غیر از ELMAH ابزار دیگری برای لاگ گیری از خطاها وجود دارد که قابل اعتماد باشد؟

همچنین اگر ابزاری جهت لاگ گیری از عملیات کاربران (CRUD => حالا R خیلی مهم نیست) می‌شناسید معرفی نمائید.

با سپاس

نویسنده: محسن
تاریخ: ۱۳۹۲/۰۱/۰۵ ۰:۳۳

متد auditFields مطرح شده در مطلب ردیابی اطلاعات این سایت برای مقصود شما مناسب است.

نویسنده: صابر فتح الهی
تاریخ: ۱۳۹۲/۰۱/۱۲ ۱۴:۲۳

سلام با تشکر از شما
من نفهمیدم که توی ASP.NET MVC شما چگونه از الگوی MVP استفاده کردین؟
ظاهرا مثال این قسمت هم توی پست وجود نداره، اگر اشتباه می‌کنم لطفا تصحیح بفرمایید.

نویسنده: علي
تاریخ: ۱۳۹۲/۰۱/۱۲ ۱۶:۳

مثال وب فرم هست. post back و page load داره.

نویسنده: شاهین کیاست
تاریخ: ۱۳۹۲/۰۱/۱۲ ۱۶:۴

اگر توجه کنید از الگوی MVP در Web Forms استفاده شده و نه در MVC.

نویسنده: صابر فتح الهی
تاریخ: ۱۳۹۲/۰۱/۱۲ ۱۸:۳۰

آقای کیاست و علی آقا
می‌دونم که پروژه چی هست، یکی از UIهای ما قرار بود MVC باشه خواستم بدونم چطور می‌خوان استفاده کنن، اینجا (در این پست) که می‌دونم ASP.NET Web form هست و در MVC می‌دونم که Page_Load و.. وجود نداره سوال من چیز دیگه بود دوستان

نویسنده: شاهین کیاست
تاریخ: ۱۳۹۲/۰۱/۱۲ ۱۸:۴۴

شما گفتید:

سلام با تشکر از شما
من نفهمیدم که توی ASP.NET MVC شما چگونه از الگوی MVP استفاده کردین؟
ظاهرا مثال این قسمت هم توی پست وجود نداره، اگر اشتباه می‌کنم لطفا تصحیح بفرمایید.
با خواندن کامنت شما برداشت کردم شما تصور کردید کدهای پست جاری مربوط به تکنولوژی ASP.NET MVC هست.

به نظر نویسنده هنوز برای MVC و WPF مثال‌ها را ایجاد نکرده و توضیح نداده اند.
اما برای استفاده از این نوع معماری در MVC کار خاصی لازم نیست انجام شود. همانطور که قبلا در مثال‌های آقای نصیری دیده ایم کافی است Service Layer در Controller مدل مناسب را تغذیه کند و برای View فراهم کند.

نویسنده: صابر فتح الهی
تاریخ: ۱۳۹۲/۰۱/۱۲ ۱۹:۲۶

من هم با توجه به مثال آقای نصیری و استفاده از الگوی کار گیج شدم، این معماری یک لایه Repository دارد، من الگوی کار توی این لایه پیاده کردم، با پیاده سازی در این لایه به نظر میاد لایه سرویس کاربردش از دست میده توی پست‌های قبل هم از آقای خوشبخت سوال کردم اما طاهرا هنوز وقت نکردن پاسخ بدن.

مورد دوم اینکه در این پست الگوی کار شرح داده شده و پیاده سازی شده، و در این پست گفته شده " *حین استفاده از EF code first، الگوی واحد کار، همان DbContext است و الگوی مخزن، همان DbSetها. ضرورتی به ایجاد یک لایه محافظ اضافی بر روی*

این‌ها وجود ندارد. " با توجه به این مسائل کلا مسائل قاطی کردم متاسفانه آقای نصیری هم سرشون شلوغ و درگیر دوره ها است، که بحثی بر سر این معماری بشه.

نویسنده: شاهین کیاست
تاریخ: ۱۳۹۲/۰۱/۱۲ ۲۰:۴۶

روشی که در مثال آقای نصیری گفته شده با روش این سری مقالات کمی متفاوت هست.
در آنجا از روکش اضافه برای Repository استفاده نشده همچنین از الگوی واحد کار استفاده شده.
به علاوه این سری مقالات ممکن است هنوز تکمیل نشده باشند.
به نظر من هر کس با توجه به میزان اطلاعاتی که دارد و درکی که از الگوها دارد با مقایسه‌ی روش‌ها و مقالات می‌تواند تصمیم بگیرد چه معماری به کار بگیرد.

نویسنده: صابر فتح الهی
تاریخ: ۱۳۹۲/۰۱/۱۲ ۲۱:۳

حرف شما کاملا متین هست

من قبلا معماری سه لایه کار می‌کردم، که نمونه اون توی همین سایت بخش پروژه ها گذاشتم، اما الان با EF , MVC کمی به مشکل بر خوردم و درست نتونستم تا حالا لایه‌های مورد نظر برای خودم در پروژه‌ها تفکیک کنم، این معماری به نظرم جالب اومد، خواستم که الگوی کار هم توی اون به کار ببرم که به مشکل بر خوردم (چون درک درستی از الگوی کار پیدا نکردم یا شایدم کلا دارم اشتباه می‌کنم). البته به قول شما شاید این معماری هنوز تکمیل نشده پروژه اش، در هر صورت از پاسخ‌های شما متشکرم.

نویسنده: شاهین کیاست
تاریخ: ۱۳۹۲/۰۱/۱۲ ۲۱:۷

خواهش می‌کنم.
فقط جهت یادآوری مثال روش آقای نصیری با پوشش MVC و EF قابل دریافت است.

نویسنده: ابوالفضل روشن ضمیر
تاریخ: ۱۳۹۲/۰۱/۱۷ ۱:۴۵

سلام
با تشکر فروان از شما ...
اگر امکان داره این مثال که در قالی یک پروژه نوشته شده برای دانلود قرار دهید ...تا بهتر بتوانیم برنامه را تجزیه و تحلیل کنیم ....ممنون

نویسنده: فرشید علی اکبری
تاریخ: ۱۳۹۲/۰۱/۱۹ ۱۵:۵۳

با سلام و تشکر از زحمات کلیه دوستان
با زحمتی که آقای خوشبخت تا اینجا کشیدن فکر کنم در صورتیکه خودمون مقاله مربوطه به این پروژه رو قدم به قدم بخونیم وطراحی کینم خیلی بهتر متوجه میشیم تا اینکه اونو آماده دانلود کنیم. من با این روش پیش رفتم و برای ایجاد اون با step by step کردن مراحلش حدود 45 دقیقه وقت گذاشتم ولی درصد یادگیریش خیلی بالاتر بود تا گرفتن فایل آماده...
درضمن لازمه بگم که بخاطر رفع شک وشبهه در سرعت پردازش وبالا اومدن اطلاعات، من تست این روش رو با تعداد 155 هزار رکورد انجام دادم که کمتر از سه ثانیه برام لود شد... باوجودیکه کامپوننت‌های دات نت بار مختلفی رو هم روی فرمم قرار دادم که بیشتر به اهمیت لود اطلاعاتم در پروژه و فرمهای واقعی پی ببرم.
سئوال اینکه :

به نظر شما ما می‌تونیم روی این لایه‌ها الگوی واحد کار رو هم ایجاد کنیم یا نه؟ اصلا ضرورتی داره ؟

نویسنده: علیرضا کیانی مقدم
تاریخ: ۱۳۹۲/۰۱/۲۸ ۱۲:۸

با تشکر از نویسنده مقاله و اهتمام ایشان به بررسی دقیق مفاهیم ،
از آنجا که flexible و reusable بودن برنامه‌ها را نمی‌توان نادیده گرفت تا آنجا که این تفکیک پذیری خود به مسئله ای بغرنج تبدیل نشده و تکرر داده‌ها و پاس دادن غیر ضرور آنها را موجب نشود تلاش در این باره مفید خواهد بود .
امروزه توسعه دهنده گان به سمت کم کردن لایه‌های فرسایشی و حذف پیچیدگی‌های غیر ضرور قدم بر می‌دارند. خلق عبارات لامبادا در دات نت و delegate ها نمونه هایی از تلاش بشر برنامه نویس در این باره است .

نویسنده: مسعود2
تاریخ: ۱۳۹۲/۰۲/۰۹ ۹:۱۲

سلام
Business Rule-1ها و validation-2ها در کجای این معماری اعمال میشوند؟

منظور از DomainService چیست؟

ممکنه منابع بیشتری معرفی نمایید؟
ممنون.

نویسنده: محسن خان
تاریخ: ۱۳۹۲/۰۲/۰۹ ۱۶:۲۶

منبع برای مطالعه بیشتر

عنوان: متدهای الحاقی - Extension Methods
نویسنده: علیرضا اسم‌رام
تاریخ: ۱۳۹۲/۰۱/۰۵ ۰:۳۵
آدرس: www.dotnettips.info
برچسب‌ها: C#, Extension Method

چقدر خوب می‌شد اگر،
نوع داده String دارای متدی جهت حذف تگ‌های HTML داشت:

```
string htmlStr = "<h1>.Net Tips</h1>";
htmlStr.ClearHtmlTags();
```

کلاس Image دارای متدی جهت تغییر اندازه (Resize) داشت:

```
image1.Resize(50, 80);
```

کنترل DropDownList متدی جهت انقیاد داده‌ها داشت:

```
dropDownList1.Bind((List<Category>)categories, "Name", "Id");
```

متدهای الحاقی به همین منظور متولد شده اند. در واقع هر زمان بدنه کلاسی (نوع داده، کنترل و تمام اشیاء دات نتی) در اختیار ما نباشد امکان اضافه کردن متدهای الحاقی به آنها وجود دارد. برای این منظور کافیست چند نکته را رعایت کنید:

کلاس دربرگیرنده متد یا متدهای الحاقی باید Public و Static باشد.
متد الحاقی باید Public و Static باشد.
اولین پارامتر متد الحاقی باید با کلمه کلیدی this همراه باشد و این پارامتر اشاره به کلاسی دارد که متد جاری به آن الحاق (یا ضمیمه) خواهد شد.

یک مثال:
در این مثال متدالحاقی برای بهبود نوع داده String را خواهیم دید. وظیفه‌ی این متد شمارش تعداد کلمات موجود در رشته است.

```
public static class StringExtensions
{

        /// <summary>
        /// Count all words in a given string
        /// </summary>
        /// <param name="input">string to begin with</param>
        /// <returns>int</returns>
        public static int WordCount(this string input)
        {
            var count = 0;
            try
            {
                // Exclude whitespaces, Tabs and line breaks
                var re = new Regex(@"[^\s]+");
                var matches = re.Matches(input);
                count = matches.Count;
            }
            catch (Exception)
            {
                return -1;
            }
            return count;
        }
}
```

نحوه استفاده:

```
var s = "i Love Dot Net Tips.";
var wordCount = s.WordCount();
```

در ضمن وب سایتی جهت به اشتراک گذاری این متدها به عنوان یکی از بهترین مراجع در دسترس است:

http://extensionmethod.net

با توجه به این مطلب توسعه پروژه ای در همین سایت با عنوان " متدهای الحاقی " آغاز شده است. در این پروژه ضمن پوشش متدهای الحاقی پرکاربرد سعی به توسعه متدهای الحاقی داریم که بیشتر در برنامه‌های فارسی کاربرد دارند.

نظرات خوانندگان

نویسنده: محسن
تاریخ: ۱۳۹۲/۰۱/۰۵ ۰:۵۵

ممنون. در صورت امکان لطفا آدرس مخزن کد را به صورت http://projectName.codeplex.com ارائه کنید.

نویسنده: علیرضا اسم‌رام
تاریخ: ۱۳۹۲/۰۱/۰۵ ۱:۵

سلام. برداشت من از آدرس مخزن کد، آدرسی بود که دوستان بتوانند از طریق VisualStudio.NET روی توسعه این پروژه بنده را همراهی کنند.
آدرس مورد نظر شما http://dntextensions.codeplex.com است. اگر اشتباه برداشت کردم لطفاً بفرمایید تا آدرس را در پروژه اصلاح کنم.
همچنین سوالات مربوط به پروژه را در بخش مربوط به آن + مطرح کنید.
با سپاس از لطف شما

نویسنده: وحید نصیری
تاریخ: ۱۳۹۲/۰۱/۰۵ ۱:۱۰

بله. لطفا آدرس عمومی پروژه را لحاظ بفرمائید. با تشکر.

نویسنده: علیرضا اسم‌رام
تاریخ: ۱۳۹۲/۰۱/۰۵ ۱:۱۹

با تشکر، اصلاح شد.

نویسنده: M.Q
تاریخ: ۱۳۹۲/۰۱/۰۵ ۱۵:۵۷

من در زمان کانکت شدن به مخزن کد با خطای زیر مواجه می‌شوم. همچنین فقط کاربران عضو سایت می‌توانند همکاری کنند یا بازدیدکنندگان هم اجازه همکاری دارند؟

```
Microsoft Visual Studio
TF31002: Unable to connect to this Team Foundation Server: http://dntextensions.codeplex.com/.
Team Foundation Server Url: http://dntextensions.codeplex.com/.


Possible reasons for failure include:
- The name, port number, or protocol for the Team Foundation Server is incorrect.
- The Team Foundation Server is offline.
- The password has expired or is incorrect.

Technical information (for administrator):
The remote server returned an error: (404) Not Found.
```

نویسنده: وحید نصیری
تاریخ: ۱۳۹۲/۰۱/۰۵ ۱۶:۴۲

- این مورد مختص تمام سورس کنترلرها است. فقط اعضای تیم و اعضای تعریف شده دسترسی commit دارند.
البته می‌تونید یک کپی فقط خواندنی با checkout مسیر https://dntextensions.svn.codeplex.com/svn تهیه کنید. روش کار در اینجا توضیح داده شده.
و یا در همان سایت کدپلکس در قسمت سورس‌ها، آخرین سورس‌ها را دانلود کنید.
+ می‌تونید برای همکاری حداقل دو کار را انجام دهید:
الف) وصله‌های خودتون رو در اینجا ارسال کنید.

ب) و یا مواردی را که مدنظر دارید در سایت جاری د ر قسمت بازخوردهای پروژه مطرح کنید.

نویسنده: علیرضا اسم‌رام
تاریخ: ۱۳۹۲/۰۱/۰۵ ۲۳:۱۹

سلام، توضیحات آقای نصیری کامل بود. همچنین اضافه کنم که دسترسی به سورس برنامه از طریق این لینک + نیز میسر است.

نویسنده: M.Q
تاریخ: ۱۳۹۲/۰۱/۰۶ ۱۲:۲۰

ممنون

من سعی می‌کنم از روش هاش هایی که آقای نصیری گفتند همکاری کنم.

با سپاس

نویسنده: م.خ
تاریخ: ۱۳۹۲/۰۱/۰۶ ۱۳:۵۳

سلام

مشابه Class helper در دلفی است...

عنوان: **مفاهیم برنامه نویسی ـ مروری بر کلاس و شیء**
نویسنده: آرمان فرقانی
تاریخ: ۱۳۹۲/۰۱/۱۹ ۲۱:۳۵
آدرس: www.dotnettips.info
برچسب‌ها: C#, OOP, Programming, آموزش برنامه نویسی

من قصد دارم در قالب چند مطلب برخی از مفاهیم پایه و مهم برنامه نویسی را که پیش نیازی برای درک اکثر مطالب موجود در وب سایت است به زبان ساده بیان کنم تا دایره افرادی که می‌توانند از مطالب ارزشمند این وب سایت استفاده کنند وسعت بیشتری پیدا کند. لازم به توضیح است از آنجا که علاقه ندارم اینجا تبدیل به نسخه فارسی MSDN یا کتاب آنلاین آموزش برنامه نویسی شود این سری آموزش‌ها بیشتر شامل مفاهیم کلیدی خواهند بود.
این مطلب به عنوان اولین بخش از این سری مطالب منتشر می‌شود.

هدف این نوشته بررسی جزییات برنامه نویسی در رابطه با کلاس و شیء نیست. بلکه دریافتن چگونگی شکل گرفتن ایده شیء گرایی و علت مفید بودن آن است.

مشاهده مفاهیم شیء گرایی در پیرامون خود حتماً در دنیای برنامه نویسی شیء گرا بارها با کلمات کلاس و شیء روبرو شده اید. درک صحیح از این مفاهیم بسیار مهم و البته بسیار ساده است. کار را با یک مثال شروع می‌کنیم. به تصویر زیر نگاه کنید.

در سمت راست بخشی از نقشه یک ساختمان و در سمت چپ ساختمان ساخته شده بر اساس این نقشه را می‌بینید. ساختمان همان شیء است. و نقشه ساختمان کلاس آن است چراکه امکان ایجاد اشیائی که تحت عنوان ساختمان طبقه بندی (کلاس بندی) می‌شوند را فراهم می‌کند. به همین سادگی. کلاس‌ها طرح اولیه، نقشه یا قالبی هستند که جزییات یک شی را توصیف می‌کنند.
حتماً با من موافق هستید اگر بگویم:
در نقشه ساختمان نمی‌توانید زندگی کنید اما در خود ساختمان می‌توانید.
از روی یک نقشه می‌توان به تعداد دلخواه ساختمان ساخت.
هنگامی که در یک ساختمان زندگی می‌کنید نیازی نیست تا دقیقاً بدانید چگونه ساخته شده و مثلاً سیم کشی یا لوله کشی‌های آن

چگونه است! تنها کافیست بدانید برای روشن شدن لامپ باید کلید آن را بزنید.
ساختمان دارای ویژگی هایی مانند متراژ، ضخامت دیوار، تعداد پنجره و ابعاد هر یک و ... است که در هنگام ساخت و بر اساس اطلاعات موجود در نقشه تعیین شده اند.
ساختمان دارای کارکرد هایی است. مانند بالا و پایین رفتن آسانسور و یا باز و بسته شدن درب پارکینگ. هر یک از این کارکردها نیز بر اساس اطلاعات موجود در نقشه پیاده سازی و ساخته شده اند.
ساختمان تمام اجزای لازم برای اینکه از آن بتوانیم استفاده کنیم و به عبارتی در آن بتوانیم زندگی کنیم را در خود دارد.
در محیط پیرامون ما تقریباً هر چیزی را می‌توان در یک دیدگاه شیء تصور کرد. به عبارتی هر چیزی که بتوانید به صورت مستقل در ذهن بیاورید و سپس برخی ویژگی‌ها و رفتارها یا کارکردهای آن‌را برشمارید تا آن چیز را قابل شناسایی کند شیء است. مثلاً من به شما می‌گویم موجودی چهار پا دارد، مو... مو... می‌کند و شیر می‌دهد و ... . شما خواهید گفت گاو! و نمی‌گویید گربه. چرا؟ چون توانستید در ذهن خود موجودیتی را به صورت مستقل تصور کنید و از روی ویژگی‌ها و رفتارش آن‌را دقیقاً شناسایی کنید.
سوال: کلاس یا نقشه ایجاد گاو چیست؟ اگر از من بپرسید خواهم گفت طرح اولیه گاو هم ممکن است وجود داشته باشد البته در اختیار خداوند و با سطح دسترسی ملکوت!
اتومبیل، تلویزیون و ... همگی مثال هایی از اشیاء پیرامون ما در دنیای واقعی هستند که حتماً می‌توانید کلاس یا نقشه ایجاد آن‌ها را نیز بدست آورید و یا ویژگی‌ها و کارکردهای آن‌ها را برشمارید.

مفاهیم شیء گرایی در مهندسی نرم افزار مفاهیمی که تاکنون در مورد دنیای واقعی مرور کردیم همان چیزی است که در دنیای برنامه نویسی ـ به عقیده من دنیای واقعی‌تر از دنیای واقعی ـ با آن سر و کار داریم. علت این امر آن است که اصولاً ایده روش برنامه نویسی شیء گرا با مشاهده محیط پیرامون ما به وجود آمده است.
برای نوشتن برنامه جهت حل یک مسئله بزرگ باید بتوان آن مسئله را به بخش‌های کوچکتری تقسیم نمود. در این رابطه مفهوم شیء و کلاس با همان کیفیتی که در محیط پیرامون ما وجود دارد به صورت مناسبی امکان تقسیم یه مسئله بزرگ به بخش‌های کوچکتر را فراهم می‌کند. و سبب می‌شود هماهنگی و تقارن و تناظر خاصی بین اشیاء برنامه و دنیای واقعی بوجود آید که یکی از مزایای اصلی روش شیء گراست.
از آنجا که در یک برنامه اصولاً همه چیز و همه مفاهیم در قالب کدها و دستورات برنامه معنا دارد، کلاس و شیء نیز چیزی بیش از قطعاتی کد نیستند. قطعه کد هایی که بسته بندی شده اند تا تمام کار مربوط به هدفی که برای آن‌ها در نظر گرفته شده است را انجام دهند.
همان طور که در هر زبان برنامه نویسی دستوراتی برای کارهای مختلف مانند تعریف یک متغیر یا ایجاد یک حلقه و ... در نظر گرفته شده است، در زبان‌های برنامه نویسی شیء گرا نیز دستوراتی وجود دارد تا بتوان قطعه کدی را بر اساس مفهوم کلاس بسته بندی کرد.
به طور مثال قطعه کد زیر را در زبان برنامه نویسی سی شارپ در نظر بگیرید.

```
class Player
{
   public string Name;
   public int Age;
   public void Walk()
   {
      // کدهای مربوط به پیاده سازی راه رفتن
   }
   public void Run()
   {
      // کدهای مربوط به پیاده سازی دویدن
   }
}
```

در این قطعه کد با استفاده از کلمه کلیدی class در زبان سی شارپ کلاسی ایجاد شده است که دارای دو ویژگی نام و سن و دو رفتار راه رفتن و دویدن است.
این کلاس به چه دردی می‌خورد؟ کجا می‌توانیم از این کلاس استفاده کنیم؟
پاسخ این است که این کلاس ممکن است برای ما هیچ سودی نداشته باشد و هیچ کجا نتوانیم از آن استفاده کنیم. اما بیایید فرض کنیم برنامه نویسی هستیم که قصد داریم یک بازی فوتبال بنویسیم. به جای آنکه قطعات کد مجزایی برای هر یک از بازیکنان و کنترل رفتار و ویژگی‌های آنان بنویسیم با اندکی تفکر به این نکته پی می‌بریم که همه بازیکنان مشترکات بسیاری دارند و به عبارتی در یک گروه یا کلاس قابل دسته بندی هستند. پس سعی می‌کنیم نقشه یا قالبی برای بازیکن‌ها ایجاد کنیم که دربردارنده ویژگی‌ها و

رفتارهای آن‌ها باشد.
همان طور که در نقشه ساختمان نمی‌توانیم زندگی کنیم این کلاس هم هنوز آماده انجام کارهای واقعی نیست. چراکه برخی مقادیر هنوز برای آن تنظیم نشده است. مانند نام بازیکن و سن و ....
و همان طور که برای سکونت لازم است ابتدا یک ساختمان از روی نقشه ساختمان بسازیم برای استفاده واقعی از کلاس یاد شده نیز باید از روی آن شیء بسازیم. به این فرآیند وهله سازی یا نمونه سازی نیز می‌گویند. یک زبان برنامه نویسی شیء گرا دستوراتی را برای وهله سازی نیز در نظر گرفته است. در C# کلمه کلیدی new این وظیفه را به عهده دارد.

```
Player objPlayer = new Player();
objPlayer.Name = “Ali Karimi”;
objPlayer.Age = 30;
objPlayer.Run();
```

وقتی فرآیند وهله سازی صورت می‌گیرد یک نمونه یا شیء از آن کلاس در حافظه ساخته می‌شود که در حقیقت می‌توانید آنرا همان کدهای کلاس تصور کنید با این تفاوت که مقداردهی‌های لازم صورت گرفته است. به دلیل تعیین مقادیر لازم، حال شیء تولید شده می‌تواند به خوبی اعمال پیش بینی شده را انجام دهد. توجه نمایید در اینجا پیاده سازی داخلی رفتار دویدن و اینکه مثلاً در هنگام فراخوانی آن چه کدی باید اجرا شود تا تصویر یک بازیکن در حال دویدن در بازی نمایش یابد مد نظر و موضوع بحث ما نیست. بحث ما چگونگی سازماندهی کدها توسط مفهوم کلاس و شیء است. همان طور که مشاهده می‌کنید ما تمام جزییات بازیکن‌ها را یکبار در کلاس پیاده سازی کرده ایم اما به تعداد دلخواه می‌توانیم از روی آن بازیکن‌های مختلف را ایجاد کنیم. همچنین به راحتی رفتار دویدن یک بازیکن را فراخوانی میکنیم بدون آنکه پیاده سازی کامل آن در اختیار و جلوی چشم ما باشد.
تمام آنچه که بازیکن برای انجام امور مربوط به خود نیاز دارد در کلاس بازیکن کپسوله می‌شود. بدیهی است در یک برنامه واقعی ویژگی‌ها و رفتارهای بسیار بیشتری باید برای کلاس بازیکن در نظر گرفته شود. مانند پاس دادن، شوت زدن و غیره.
به این ترتیب ما برای هر برنامه می‌توانیم مسئله اصلی را به تعدادی مسئله کوچکتر تقسیم کنیم و وظیفه حل هر یک از مسائل کوچک را به یک شیء واگذار کنیم. و بر اساس اشیاء تشخیص داده شده کلاس‌های مربوطه را بنویسیم. برنامه نویسی شیء گرا سبب می‌شود تا مسئله توسط تعدادی شیء که دارای نمونه‌های متناظری در دنیای واقعی هستند حل شود که این امر زیبایی و خوانایی و قابلیت نگهداری و توسعه برنامه را بهبود می‌دهد.
احتمالاً تاکنون متوجه شده اید که برای نگهداری ویژگی‌های اشیاء از متغیرها و برای پیاده سازی رفتارها یا کارکردهای اشیاء از توابع استفاده میکنیم.
با توجه به این که هدف این مطلب بررسی مفهوم شیء گرائی بود و نه جزییات برنامه نویسی، بنابراین بیان برخی مفاهیم در این رابطه را که بیشتر در مهندسی نرم افزار معنا دارند تا در دنیای واقعی در مطالب بعدی بررسی می‌کنیم.

نظرات خوانندگان

نویسنده: Hesam
تاریخ: ۱۳۹۲/۰۱/۱۹ ۲۲:۴۵

بسیار عالی :)

نویسنده: من
تاریخ: ۱۳۹۲/۰۱/۲۰ ۱۴:۵۶

شروع خوبی بود، آفرین!
من هنوز وقتی جلسه‌ی اول کلاس میپرسم که پراید یک کلاس هست و یا یک آبجکت. همه میگن آبجکتی از کلاس ماشین!
این در حالیست که خیلی از این افراد سالهاست برنامه نویسی میکنند و هنوز نمیدونند heap یا stack چیه

نویسنده: علي
تاریخ: ۱۳۹۲/۰۱/۲۰ ۱۵:۴

حالا با توجه به توضیحات بالا که گفته شد « *در نقشه ساختمان نمی‌توانید زندگی کنید اما در خود ساختمان می‌توانید.* » با یک پراید می‌شود رانندگی کرد اما با ماشین که بیشتر یک مفهوم است خیر. نه؟

نویسنده: آرمان فرقانی
تاریخ: ۱۳۹۲/۰۱/۲۰ ۱۵:۲۲

تشکر از نظر شما. متوجه صحبت شما هستم ولی این توضیح برای دوستان دیگه می‌تونه مفید باشه.
پراید به عنوان رده ای از اتومبیل‌ها کلاس است. اما اگر به کسی یک اتومبیل پراید در حال عبور را نشان دهید که دارای پلاک و ... است، آن شیء ای است از کلاس پراید. اگر در یک پارکینگ تعدادی اتومبیل باشد و از کسی بخواهیم بر اساس نوع گروه بندی یا کلاس بندی کند، بعد از چند دقیقه خواهیم دید اتومبیل‌های هم نوع را جدا کرده. مثلا همه اتومبیل‌های از نوع پراید را کنار هم قرار داده. این همان مفهوم کلاس بندی است. برای تولید اتومبیلی از نوع پراید کارخانه یک نقشه یا طرحی ایجاد می‌کند که بسیاری مشخصات و چگونگی ساخت را در خود دارد. چگونگی انجام برخی کارکردها مانند حرکت را دارد. این طرح کلاس نامیده می‌شود. اما آیا می‌توان برای آن پلاک در نظر گرفت؟ خیر چون فقط نقشه ایجاد اتومبیل است (یا به عبارتی فقط مفهوم است). همچنین کلاس پراید احتمالاً از کلاس اتومبیل یا ماشین برخی خصوصیات و رفتارها را با ارث برده است.

بیاد داشته باشیم هر فردی در هر سطحی از دانش هم مسلماً بسیاری چیزها را هنوز نمی‌داند و دانسته‌های ما در مقابل نادانسته‌ها قطره ای بیش نیست. تا جایی که می‌توان گفت همه ما از نظر نادانسته‌ها برابریم. تفاوت در دانسته هاست. کسانی که سال‌ها برنامه نویسی می‌کنند هم حتما دانسته هایی دارند که می‌توانند این کار را ادامه دهند. پس کافی است آنچه نمی‌دانند را سعی کنیم به اشتراک بگذاریم تا بدانند و از دانسته هایشان استفاده کنیم.

نویسنده: آرمان فرقانی
تاریخ: ۱۳۹۲/۰۱/۲۰ ۱۵:۲۵

جمله با پراید می‌شود رانندگی کرد ابهام دارد. ابهام آن به این صورت رفع می‌شود که من می‌دانم منظور شما از پراید به عنوان یک اسم عام و یک مفهوم و **نام رده یا کلاسی از اتومبیل ها** نیست. بلکه منظور شما با اتومبیل پرایدی است که دارای یک پلاک مشخص است و مثلاً کسی به تازگی گنجی پیدا کرده و رفته یک پراید خریده! آن اتومبیل پراید مشخص یک شیء است از کلاس پراید. بله با آن شی می‌توان رانندگی کرد. اما با مفهوم یا نقشه یا کلاس یا رده یا گروه یا طرح تولید خودروی پراید یا هر ماشین دیگری نمی‌توان رانندگی کرد.

نویسنده: سید ایوب کوکبی
تاریخ: ۱۳۹۲/۰۱/۲۱ ۱۵:۴۸

ممنون، خیلی خوب بررسی کرده بودید و مثالهایی که هم ارائه کردید به دلم نشست، امیدوارم برای سایر مباحث هم به همین صورت ادامه بدید. البته به نظر بنده نیازی هم نیست خیلی روی این مبحث تناظر بین دنیای واقعی و دنیای شی گرا حساس باشیم، چون اگر به همین نحو بخواهیم این دو را با هم مقایسه کنیم شاید بعضی جاها با مشکل مواجه بشیم. این تناظر فقط برای اینه که دید و تجسمی از این مفهوم داشته باشیم تا بهتر بتوانیم آن را بپذیریم.
یادمون نره که هدف ما یادگیری مفهوم شی گرایی است نه یادگیری تناظر آن با دنیای واقعی، این تناظر فقط یک ابزاره برای دسترسی سریعتر به این هدف!

نویسنده: آرمان فرقانی
تاریخ: ۱۳۹۲/۰۱/۲۱ ۱۷:۳۵

سلام و ممنون از نظر شما.
اتفاق جالبی افتاد و آن این بود که هم اکنون داشتم در OneNote بخشی برای مطلب بعدی می‌نوشتم. دقیقاً داشتم پاراگرافی را می‌نوشتم که جلوی این که ذهن خواننده به سمتی برود که گویی "الزاماً هر مورد در مهندسی نرم افزار را باید پس از یافتن مصداق آن در محیط اطراف یاد گرفت" را بگیرم.
دقیقاً صحیح است. تاکید بر این تناظر در این بخش به دلیل یافتن درک عمیق‌تر از شیء گرایی و علت مفید بودن آن و چگونگی شکل گیری ایده آن است. این درک عمیق‌تر امکان استفاده بهتر و صحیح‌تر این مفاهیم در برنامه را فراهم می‌کند. و سبب می‌شود برنامه نویس شیءگرایی را ابزاری برای حل مسئله بیاد نه راه و روشی که همه میگن خوبه پس باید رعایت کرد. حال آنکه چون درک دقیقی از آن ندارد در حقیقت مسئله را با آن روشی که بهتر بلد است حل می‌کنند و فقط تعدادی کلاس و شیء در برنامه وجود دارد.

نویسنده: آریانا
تاریخ: ۱۳۹۲/۰۲/۰۷ ۱۱:۵۶

بسیار بسیار عالی بود مرسی

نویسنده: سحابی
تاریخ: ۱۳۹۲/۰۷/۱۴ ۱۴:۵۹

سلام
برای یادگیری Desing pattern منابعی وجود دارد ؟ لطفا در صورت امکان اگر منابع و یا لینک کارآمدی معرفی کنید .

نویسنده: محسن خان
تاریخ: ۱۳۹۲/۰۷/۱۴ ۱۷:۱۵

برچسب‌های سایت رو مرور کنید به اندازه کافی در این زمینه مطلب هست. مثلا اینجا

عنوان: **مفاهیم برنامه نویسی ـ مروری بر فیلدها، متدها و ساخت اشیاء**
نویسنده: آرمان فرقانی
تاریخ: ۱۳۹۲/۰۱/۲۳ ۲۰:۱۰
آدرس: www.dotnettips.info
برچسب‌ها: C#, OOP, Programming, آموزش برنامه نویسی

شکستن یک مسئله بزرگ به تعدادی مسئله کوچک‌تر راهکار موثری برای حل آن است. این امر در برنامه نویسی نیز که هدف آن چیزی جز حل یک مسئله نیست همواره مورد توجه بوده است. به همین دلیل روش هایی که به کمک آن‌ها بتوان یک برنامه بزرگ را به قطعات کوچکتری تقسیم کرد تا هر قطعه کد مسئول انجام کار خاصی باشد پیشتر به زبان‌های برنامه نویسی اضافه شده اند. یکی از این ساختارها تابع (Function) نام دارد. برنامه ای که از توابع برای تقسیم کدهای برنامه استفاده می‌کند یک برنامه ساخت‌یافته می‌گوییم.

در مطلب پیشین به پیرامون خود نگاه کردیم و اشیاء گوناگونی را مشاهده کردیم که در حقیقت دنیای ما را تشکیل داده اند و فعالیت‌های روزمره ما با استفاده از آن‌ها صورت می‌گیرد. ایده ای به ذهنمان رسید. اشیاء و مفاهیم مرتبط به آن می‌تواند روش بهتر و موثرتری برای تقسیم کدهای برنامه باشد. مثلاً اگر کل کدهای برنامه که مسئول حل یکی از مسئله‌های کوچک یاد شده است را یکجا بسته بندی کنیم و اصولی که از اشیاء واقعی پیرامون خود آموختیم را در مورد آن رعایت کنیم به برنامه بسیار با کیفیت‌تری از نظر خوانایی، راحتی در توسعه، اشکال زدایی ساده‌تر و بسیاری موارد دیگر خواهیم رسید.

توسعه دهندگان زبان‌های برنامه نویسی که با ما در این مورد هم عقیده بوده اند دست به کار شده و دستورات و ساختارهای لازم برای پیاده کردن این ایده را در زبان برنامه نویسی قرار دادند و آن را زبان برنامه نویسی شیء گرا نامیدند. حتی جهت برخورداری از قابلیت استفاده مجدد از کد و موارد دیگر به جای آنکه کدها را در بسته هایی به عنوان یک شیء خاص قرار دهیم آن‌ها را در بسته هایی به عنوان قالب یا نقشه ساخت اشیاء خاصی که در ذهن داریم قرار می‌دهیم. یعنی مفهوم کلاس یا رده که پیشتر اشاره شد. به این ترتیب یک بار می‌نویسیم و بارها استفاده می‌کنیم. مانند همان مثال بازیکن در بخش نخست . هر زمان که لازم باشد با استفاده از دستورات مربوطه از روی کدهای کلاس که نقشه یا قالب ساخت اشیاء هستند شیء مورد نظر را ساخته و در جهت حل مسئله مورد نظر به کار می‌بریم.

حال برای آنکه به طور عملی بتوانیم از ایده شیء گرایی در برنامه هایمان استفاده کنیم و مسائل بزرگ را حل کنیم لازم است ابتدا مقداری با جزییات و دستورات زبان در این مورد آشنا شویم.

تذکر : دقت کنید برای آنکه از ایده شیء گرایی در برنامه‌ها حداکثر استفاده را ببریم مفاهیمی در مهندسی نرم افزار به آن اضافه شده است که ممکن است در دنیای واقعی نیازی به طرح آن‌ها نباشد. پس لطفاً تلاش نکنید با دیدن هر مفهوم تازه بلافاصله سعی در تطبیق آن با محیط اطراف کنید. هر چند بسیاری از آن‌ها به طور ضمنی در اشیاء پیرامون ما نیز وجود دارند.

زبان برنامه نویسی مورد استفاده برای بیان مفاهیم برنامه نویسی در این سری مقالات زبان سی شارپ است. اما درک برنامه‌های نوشته شده برای علاقه مندان به زبان‌های دیگری مانند وی بی دات نت نیز دشوار نیست. چراکه اکثر دستورات مشابه است و تبدیل Syntax نیز به راحتی با اندکی جستجو میسر می‌باشد. لازم به یادآوری است زبان سی شارپ به بزرگی یا کوچکی حروف حساس است.

تشخیص و تعریف کلاس‌های برنامه کار را با یک مثال شروع می‌کنیم. فرض کنید به عنوان بخشی از راه حل یک مسئله بزرگ، لازم است محیط و مساحت یک سری چهارضلعی را محاسبه کنیم و قصد داریم این وظیفه را به طور کامل بر عهده قطعه کدهای مستقلی در برنامه قرار دهیم. به عبارت دیگر قصد داریم متناظر با هر یک از چهارضلعی‌های موجود در مسئله یک شیء در برنامه داشته باشیم که قادر است محیط و مساحت خود را محاسبه و ارائه نماید. کلاس زیر که با زبان سی شارپ نوشته شده امکان ایجاد اشیاء مورد نظر را فراهم می‌کند.

```
public class Rectangle
{
   public double Width;
   public double Height;

   public double Area()
   {
      return Width*Height;
   }

   public double Perimeter()
   {
      return 2*(Width + Height);
   }
```

```
}
```

در این قطعه برنامه نکات زیر قابل توجه است:
کلاس با کلمه کلیدی class تعریف می‌شود.
همان طور که مشاهده می‌کنید تعریف کلاس با کلمه public آغاز شده است. این کلمه محدوده دسترسی به کلاس را تعیین می‌کند. در اینجا از کلمه public استفاده کردیم تا بخش‌های دیگر برنامه امکان استفاده از این کلاس را داشته باشند.
پس از کلمه کلیدی class نوبت به نام کلاس می‌رسد. اگرچه انتخاب نام مورد نظر امری اختیاری است اما در آینده حتماً اصول و قراردادهای نام‌گذاری در دات‌نت را مطالعه نمایید. در حال حاضر حداقل به خاطر داشته باشید تا انتخاب نامی مناسب که گویای کاربرد کلاس باشد بسیار مهم است.
باقیمانده کد، بدنه کلاس را تشکیل می‌دهد. جاییکه ویژگی ها، رفتارها و ... یا به طور کلی اعضای کلاس تعریف می‌شوند.
نکته : کلماتی مانند public که پیش از تعریف کلاس یا اعضای آن قرار می‌گیرند Modifier یا پیراینده نام دارند. که البته به نظر من ترجمه این گونه واژه‌ها از کارهای شیطان است. بنابراین از این پس بهتر است همان Modifier را به خاطر داشته باشید. از آنجا که public مدیفایری است که سطح دسترسی را تعیین می‌کند به آن یک Access Modifier می‌گویند. در یک بخش از این سری مقالات تمامی مدیفایرها بررسی خواهند شد.

ایجاد شیء از یک کلاس و نحوه دسترسی به شیء ایجاد شده شیء و کلاس چیزهای متفاوتی هستند. یک کلاس نوع یک شیء را تعریف می‌کند. اما یک شیء یک موجودیت عینی و واقعی بر اساس یک کلاس است. در اصطلاح از شیء به عنوان یک نمونه (Instance) یا وهله ای از کلاس مربوطه یاد می‌کنیم. همچنین به عمل ساخت شیء نمونه سازی یا وهله سازی گوییم.
برای ایجاد شیء از کلمه کلیدی new و به دنبال آن نام کلاسی که قصد داریم بر اساس آن یک شیء بسازیم استفاده می‌کنیم. همان طور که اشاره شد کلاس یک نوع را تعریف می‌کند. پس از آن می‌توان همانند سایر انواع مانند int, string, ... برای تعریف متغیر استفاده نمود. به مثال زیر توجه کنید.

```
Rectangle rectangle = new Rectangle();
```

در این مثال rectangle که با حرف کوچک شروع شده و می‌توانست هر نام دلخواه دیگری باشد ارجاعی به شیء ساخته شده را به دست می‌دهد. وقتی نمونه ای از یک کلاس ایجاد می‌شود یک ارجاع به شیء تازه ساخته شده برای برنامه نویس برگشت داده می‌شود. در این مثال rectangle یک ارجاع به شیء تازه ساخته شده است یعنی به آن اشاره می‌کند اما خودش شامل داده‌های آن شیء نیست. تصور کنید این ارجاع تنها دستگیره ای برای شیء ساخته شده است که دسترسی به آن را برای برنامه نویس میسر می‌کند. درک این مطلب از این جهت دارای اهمیت است که بدانید می‌شود یک دستگیره یا ارجاع دیگر بسازید بدون آنکه شیء جدیدی تولید کنید.

```
Rectangle rectangle;
```

البته توصیه نمی‌کنم چنین ارجاعی را تعریف کنید چرا که به هیچ شیء خاصی اشاره نمی‌کند. و تلاش برای استفاده از آن منجر به بروز خطای معروفی در برنامه خواهد شد. به هر حال یک ارجاع می‌توان ساخت چه با ایجاد یک شیء جدید و یا با نسبت دادن یک شیء موجود به آن.

```
Rectangle rectangle1 = new Rectangle();
Rectangle rectangle2 = rectangle1;
```

در این کد دو ارجاع یا دستگیره ایجاد شده است که هر دو به یک شیء اشاره می‌کنند. بنابراین ما با استفاده از هر دو ارجاع می‌توانیم به همان شیء واحد دسترسی پیدا کنیم و اگر مثلاً با rectangle1 در شیء مورد نظر تغییری بدهیم و سپس با rectangle2 شیء را مورد بررسی قرار دهیم تغییرات داده شده قابل مشاهده خواهد بود چون هر دو ارجاع به یک شیء اشاره می‌کنند. به همین دلیل کلاس‌ها را به عنوان نوع ارجاعی می‌شناسیم در مقایسه با انواع داده دیگری که اصطلاحاً نوع مقداری هستند.
حالا می‌توان شیء ساخته شده را با استفاده از ارجاعی که به آن داریم به کار برد.

```
Rectangle rectangle = new Rectangle();
rectangle.Width = 10.5;
rectangle.Height = 10;
double a = rectangle.Area();
```

ابتدا عرض و ارتفاع شیء چهارضلعی را مقدار دهی کرده و سپس مساحت را دریافت کرده ایم. از نقطه برای دسترسی به اعضای یک شیء استفاده می‌شود.

فیلدها اگر به تعریف کلاس دقت کنید مشخص است که دو متغییر Width و Height را با سطح دسترسی عمومی تعریف کرده ایم. به متغیرهایی از هر نوع که مستقیماً درون کلاس تعریف شوند (و نه مثلاً داخل یک تابع درون کلاس) فیلد می‌گوییم. فیلدها از اعضای کلاس دربردارنده آن‌ها محسوب می‌شوند.
تعریف فیلدها مستقیماً در بدنه کلاس با یک Access Modifier شروع می‌شود و به دنبال آن نوع فیلد و سپس نام دلخواه برای فیلد می‌آید.
تذکر : نامگذاری مناسب یکی از مهمترین اصولی است که یک برنامه نویس باید همواره به آن توجه کافی داشته باشد و به شدت در بالا رفتن کیفیت برنامه موثر است. به خاطر داشته باشید تنها اجرا شدن و کار کردن یک برنامه کافی نیست. رعایت بسیاری از اصول مهندسی نرم افزار که ممکن است نقش مستقیمی در کارکرد برنامه نداشته باشند موجب سهولت در نگهداری و توسعه برنامه شده و به همان اندازه کارکرد صحیح برنامه مهم هستند. بنابراین مجدداً شما را دعوت به خواندن مقاله یاد شده بالا در مورد اصول نامگذاری صحیح می‌کنم. هر مفهوم تازه ای که می‌آموزید می‌توانید به اصول نامگذاری همان مورد در مقاله پیش گفته مراجعه نمایید. همچنین افزونه هایی برای Visual Studio وجود دارد که شما را در زمینه نامگذاری صحیح و بسیاری موارد دیگر هدایت می‌کنند که یکی از مهمترین آن‌ها Resharper نام دارد.
مثال:

```
// public field (Generally not recommended.)
public double Width;
```

همان طور که در این قطعه کد به عنوان توضیح درج شده است استفاده از فیلدهایی با دسترسی عمومی توصیه نمی‌شود. علت آن واضح است. چون هیچ کنترلی برای مقداری که برای آن در نظر گرفته می‌شود نداریم. به عنوان مثال امکان دارد یک مقدار منفی برای عرض یا ارتفاع شیء درج شود حال آنکه می‌دانیم عرض یا ارتفاع منفی معنا ندارد. در قسمت بعدی این سری مقالات این مشکل را بررسی و حل خواهیم نمود.
فیلدها معمولاً با سطح دسترسی خصوصی و برای نگهداری از داده‌هایی که مورد نیاز بیش از یک متد (یا تابع) درون کلاس است و آن داده‌ها باید پس از خاتمه کار یک متد همچنان باقی بمانند استفاده می‌شود. بدیهی است در غیر اینصورت به جای تعریف فیلد می‌توان از متغیرهای محلی (متغیری که درون خود تابع تعریف می‌شود) استفاده نمود.
همان طور که پیشتر اشاره شد برای دسترسی به یک فیلد ابتدا یک نقطه پس از نام شیء درج کرده و سپس نام فیلد مورد نظر را می‌نویسیم.

```
Rectangle rectangle = new Rectangle();
rectangle.Width = 10.5;
```

در هنگام تعریف یک فیلد در صورت نیاز می‌توان برای آن یک مقدار اولیه را در نظر گرفت. مانند مثال زیر:

```
public class Rectangle
{
   public double Width = 5;
   // ...
}
```

متدها متدها قطعه کدهایی شامل یک سری دستور هستند. این مجموعه دستورات با فراخوانی متد و تعیین آرگومان‌های مورد نیاز اجرا می‌شوند. در زبان سی شارپ به نوعی تمام دستورات در داخل متدها اجرا می‌شوند. در این زبان تمامی توابع در داخل کلاس‌ها تعریف می‌شوند و بنابراین همه متد هستند.

متدها نیز مانند فیلدها در داخل کلاس تعریف می‌شوند. ابتدا یک Access Modifier سطح دسترسی را تعیین می‌نماید. سپس به ترتیب نوع خروجی، نام متد و لیست پارامترهای آن در صورت وجود درج می‌شود. به مجموعه بخش‌های یاد شده امضای متد می‌گویند.

پارامترهای یک متد داخل یک جفت پرانتز قرار می‌گیرند و با کاما (,) از هم جدا می‌شوند. یک جفت پرانتز خالی نشان دهنده آن است که متد نیاز به هیچ پارامتری ندارد.

بار دیگر به بخش تعریف متدهای کلاسی که ایجاد کردیم توجه نمایید.

```
public class Rectangle
{
   // ...

   public double Area()
   {
      return Width*Height;
   }

   public double Perimeter()
   {
      return 2*(Width + Height);
   }
}
```

در این کلاس دو متد به نام‌های Area و Perimeter به ترتیب برای محاسبه مساحت و محیط چهارضلعی تعریف شده است.

همانطور که پیشتر اشاره شد متدها برای پیاده سازی رفتار اشیاء یا همان کارکردهای آن‌ها استفاده می‌شوند. در این مثال شیء ما قادر است مساحت و محیط خود را محاسبه نماید. چه شیء خوش رفتاری!

همچنین توجه نمایید این شیء برای محاسبه مساحت و محیط خود نگاهی به ویژگی‌های خود یعنی عرض و ارتفاعش که در فیلدهای آن نگهداری می‌کنیم می‌اندازد.

فراخوانی متد یک شیء همانند دسترسی به فیلد آن است. ابتدا نام شیء سپس یک نقطه و به دنبال آن نام متد مورد نظر به همراه پرانترها. آرگومان‌های مورد نیاز در صورت وجود داخل پرانتزها قرار می‌گیرند و با کاما از هم جدا می‌شوند. که البته در این مثال متد ما نیازی به آرگومان ندارد. به همین دلیل برای فراخوانی آن تنها یک جفت پرانتز خالی قرار می‌دهیم.

در این بخش به دو مفهوم پارامتر و آرگومان اشاره شد. تفاوت آن‌ها چیست؟

در هنگام تعریف یک متد نام و نوع پارامترهای مورد نیاز را تعیین و درج می‌نماییم. حال وقتی قصد فراخوانی متد را داریم باید مقادیر واقعی که آرگومان نامیده می‌شود را برای هر یک از پارامترهای تعریف شده فراهم نماییم. نوع آرگومان باید با نوع پارامتر تعریف شده تطبیق داشته باشد. اما اگر یک متغیر را به عنوان آرگومان در هنگام فراخوانی متد استفاده می‌کنیم نیازی به یکسان بودن نام آن متغیر و نام پارامتر تعریف شده نیست.

متدها می‌توانند یک مقدار را به کدی که آن متد را فراخوانی کرده است بازگشت دهند.

```
Rectangle rectangle = new Rectangle();
rectangle.Width = 10.5;
rectangle.Height = 10;
double p = rectangle.Perimeter();
```

در این مثال مشاهده می‌کنید که پس از فراخوانی متد Perimeter مقدار بازگشتی آن در متغیری به نام p قرار گرفته است. اگر نوع خروجی یک متد که در هنگام تعریف آن پیش از نام متد قرار می‌گیرد void یا پوچ نباشد، متد می‌تواند مقدار مورد نظر را با استفاده از کلمه کلیدی return بازگشت دهد. کلمه return و به دنبال آن مقداری که از نظر نوع باید با نوع خروجی تعیین شده تطبیق داشته باشد، مقدار درج شده را به کد فراخوان متد بازگشت می‌دهد.

نکته : کلمه return علاوه بر بازگشت مقدار مورد نظر سبب پایان اجرای متد نیز می‌شود. حتی در صورتی که نوع خروجی یک متد void تعریف شده باشد استفاده از کلمه return بدون اینکه مقداری به دنبال آن بیاید می‌تواند برای پایان اجرای متد، در صورت نیاز و مثلاً برقراری شرطی خاص مفید باشد. بدون کلمه return متد زمانی پایان می‌یابد که به پایان قطعه کد بدنه خود برسد. توجه نمایید که در صورتی که نوع خروجی متد چیزی به جز void است استفاده از کلمه return به همراه مقدار مربوطه الزامی است.

مقدار خروجی یک متد را می‌توان هر جایی که مقداری از همان نوع مناسب است مستقیماً به کار برد. همچنین می‌توان آن را در یک متغیر قرار داد و سپس از آن استفاده نمود.

به عنوان مثال کلاس ساده زیر را در نظر بگیرید که متدی دارد برای جمع دو عدد.

```
public class SimpleMath
{
   public int AddTwoNumbers(int number1, int number2)
   {
      return number1 + number2;
   }
}
```

و حال دو روش استفاده از این متد:

```
SimpleMath obj = new SimpleMath();

Console.WriteLine(obj.AddTwoNumbers(1, 2));

int result = obj.AddTwoNumbers(1, 2);
Console.WriteLine(result);
```

در روش اول مستقیماً خروجی متد مورد استفاده قرار گرفته است و در روش دوم ابتدا مقدار خروجی در یک متغیر قرار گرفته است و سپس از مقدار درون متغیر استفاده شده است. استفاده از متغیر برای نگهداری مقدار خروجی اجباری نبوده و تنها جهت بالا بردن خوانایی برنامه یا حفظ مقدار خروجی تابع برای استفاده‌های بعدی به کار می‌رود.

در بخش‌های بعدی بحث ما در مورد سایر اعضای کلاس و برخی جزییات پیرامون اعضای پیش گفته خواهد بود.

نظرات خوانندگان

نویسنده: سید ایوب کوکبی
تاریخ: ۱۳۹۲/۰۱/۲۴ ۱۱:۵۶

ممنون بابت مطلب آموزشی تون،
تاکیدتان بر استفاده از قرار دادهای نامگذاری، تاکید مثبتی است و واقعا مهم،
کتاب Framework Design Guidelines: Conventions, Idioms, and Patterns for Reusable .NET Libraries در این زمینه اطلاعات کاملتر و دقیقتری در بر داره.
یکی از روش هایی که در رعایت استاندارهای کد نویسی تاثیر مفیدی داره این هستش که در قطعه کد هایی که به عنوان مثال در آموزش‌ها ارائه می‌شه تا حد امکان سعی بشه این اصول رعایت بشه تا به صورت تدریجی این روش کد نویسی جزئی از عادات برنامه نویسی ما بشود.

عنوان: **مفاهیم برنامه نویسی ـ مروری بر پروپرتی‌ها**
نویسنده: آرمان فرقانی
تاریخ: ۱۳۹۲/۰۲/۰۴ ۱۷:۴۰
آدرس: www.dotnettips.info
برچسب‌ها: C#, OOP, Programming, آموزش برنامه نویسی

در مطلب پیشین کلاسی را برای حل بخشی از یک مسئله بزرگ تهیه کردیم. اگر فراموش کردید پیشنهاد می‌کنم یک بار دیگر آن مطلب را مطالعه کنید. بد نیست بار دیگر نگاهی به آن بیاندازیم.

```
public class Rectangle
{
    public double Width;
    public double Height;

    public double Area()
    {
        return Width*Height;
    }

    public double Perimeter()
    {
        return 2*(Width + Height);
    }
}
```

کلاس خوبی است اما همان طور که در بخش قبل مطرح شد این کلاس می‌تواند بهتر هم باشد. در این کلاس برای نگهداری حالت اشیائی که از روی آن ایجاد خواهند شد از متغیرهایی با سطح دسترسی عمومی استفاده شده است. این متغیرهای عمومی فیلد نامیده می‌شوند. مشکل این است که با این تعریف، اشیاء نمی‌توانند هیچ اعتراضی به مقادیر غیر معتبری که ممکن است به آن‌ها اختصاص داده شود، داشته باشند. به عبارت دیگر هیچ کنترلی بر روی مقادیر فیلدها وجود ندارد. مثلاً ممکن است یک مقدار منفی به فیلد طول اختصاص یابد. حال آنکه طول منفی معنایی ندارد.

تذکر : ممکن است این سوال پیش بیاید که خوب ما کلاس را نوشته ایم و خودمان می‌دانیم چه مقادیری برای فیلدهای آن مناسب است. اما مسئله اینجاست که اولاً ممکن است کلاس تهیه شده توسط برنامه نویس دیگری مورد استفاده قرار گیرد. یا حتی پس از مدتی فراموش کنیم چه مقادیری برای کلاسی که مدتی قبل تهیه کردیم مناسب است. و از همه مهمتر این است که کلاس‌ها و اشیاء به عنوان ابزاری برای حل مسائل هستند و ممکن است مقادیری که به فیلدها اختصاص می‌یابد در زمان نوشتن برنامه مشخص نباشد و در زمان اجرای برنامه توسط کاربر یا کدهای بخش‌های دیگر برنامه تعیین گردد.
به طور کلی هر چه کنترل و نظارت بیشتری بر روی مقادیر انتسابی به اشیاء داشته باشیم برنامه بهتر کار می‌کند و کمتر دچار خطاهای مهلک و بدتر از آن خطاهای منطقی می‌گردد. بنابراین باید ساز و کار این نظارت را در کلاس تعریف نماییم.
برای کلاس‌ها یک نوع عضو دیگر هم می‌توان تعریف کرد که دارای این ساز و کار نظارتی است. این عضو **Property** نام دارد و یک مکانیسم انعطاف پذیر برای خواندن، نوشتن یا حتی محاسبه مقدار یک فیلد خصوصی فراهم می‌نماید.
تا اینجا باید به این نتیجه رسیده باشید که تعریف یک متغیر با سطح دسترسی عمومی در کلاس روش پسندیده و قابل توصیه ای نیست. بنابراین متغیرها را در سطح کلاس به صورت خصوصی تعریف می‌کنیم و از طریق تعریف Property امکان استفاده از آن‌ها در بیرون کلاس را ایجاد می‌کنیم.
حال به چگونگی تعریف Propertyها دقت کنید.

```
public class Rectangle
{
    private double _width = 0;
    private double _height = 0;

    public double Width
    {
        get { return _width; }
        set { if (value > 0) { _width = value; } }
    }

    public double Height
    {
        get { return _height; }
        set { if (value > 0) { _height = value; } }
    }
```

```
    public double Area()
    {
        return _width * _height;
    }

    public double Perimeter()
    {
        return 2*(_width + _height);
    }
}
```

به تغییرات ایجاد شده در تعریف کلاس دقت کنید. ابتدا سطح دسترسی دو متغیر خصوصی شده است یعنی فقط اعضای داخل کلاس به آن دسترسی دارند و از بیرون قابل استفاده نیست. نام متغیرهای پیش گفته بر اساس اصول صحیح نامگذاری فیلدهای خصوصی تغییر داده شده است. ببینید اگر اصول نامگذاری را رعایت کنید چقدر زیباست. هر جای برنامه که چشمتان به _width بخورد فوراً متوجه می‌شوید یک فیلد خصوصی است و نیازی نیست به محل تعریف آن مراجعه کنید. از طرفی یک مقدار پیش فرض برای این دو فیلد در نظر گرفته شده است که در صورتی که مقدار مناسبی برای آن‌ها تعیین نشد مورد استفاده قرار خواهند گرفت.

دو قسمت اضافه شده دیگر تعریف دو Property مورد نظر است. یکی عرض و دیگری ارتفاع. خط اول تعریف پروپرتی تفاوتی با تعریف فیلد عمومی ندارد. اما همان طور که می‌بینید هر فیلد دارای یک بدنه است که با {} مشخص می‌شود. در این بدنه ساز و کار نظارتی تعریف می‌شود.

نحوه دسترسی به پروپرتی‌ها مشابه فیلدهای عمومی است. اما پروپرتی‌ها در حقیقت متدهای ویژه ای به نام اکسسور (Accessor) هستند که از طرفی سادگی استفاده از متغیرها را به ارمغان می‌آورند و از طرف دیگر دربردارنده امنیت و انعطاف پذیری متدها هستند. یعنی در عین حال که روشی عمومی برای داد و ستد مقادیر ارایه می‌دهند، کد پیاده سازی یا وارسی اطلاعات را مخفی نموده و استفاده کننده کلاس را با آن درگیر نمی‌کنند. قطعه کد زیر چگونگی استفاده از پروپرتی را نشان می‌دهد.

```
Rectangle rectangle = new Rectangle();
rectangle.Width = 10;
Console.WriteLine(rectangle.Width);
```

به خوبی مشخص است برای کد استفاده کننده از شیء که آن‌را کد مشتری می‌نامیم نحوه دسترسی به پروپرتی یا فیلد تفاوتی ندارد. در اینجا خط دوم که مقداری را به یک پروپرتی منتسب کرده سبب فراخوانی اکسسور set می‌گردد. همچنین مقدار منتسب شده یعنی ۱۰ در داخل بدنه اکسسور از طریق کلمه کلیدی value قابل دسترسی و ارزیابی است. در خط سوم که لازم است مقدار پروپرتی برای چاپ بازیابی یا خوانده شود منجر به فراخوانی اکسسور get می‌گردد.

تذکر : به دو اکسسور get و set مانند دو متد معمولی نگاه کنید از این نظر که می‌توانید در بدنه آن‌ها اعمال دلخواه دیگری بجز ذخیره و بازیابی اطلاعات پروپرتی را نیز انجام دهید.

چند نکته :

اکسسور get هنگام بازگشت یا خواندن مقدار پروپرتی اجرا می‌شود و اکسسور set زمان انتساب یک مقدار جدید به پروپرتی فراخوانی می‌شود. جالب آنکه در صورت لزوم این دو اکسسور می‌توانند دارای سطوح دسترسی متفاوتی باشند.

داخل اکسسور set کلمه کلیدی value مقدار منتسب شده را در اختیار قرار می‌دهد تا در صورت لزوم بتوان بر روی آن پردازش لازم را انجام داد.

یک پروپرتی می‌تواند فاقد اکسسور set باشد که در این صورت یک پروپرتی فقط خواندنی ایجاد می‌گردد. همچنین می‌تواند فقط شامل اکسسور set باشد که در این صورت فقط امکان انتساب مقادر به آن وجود دارد و امکان دریافت یا خواندن مقدار آن میسر نیست. چنین پروپرتی ای فقط نوشتنی خواهد بود.

در بدنه اکسسور set الزامی به انتساب مقدار منتسب توسط کد مشتری نیست. در صورت صلاحدید می‌توانید به جای آن هر مقدار دیگری را در نظر بگیرید یا عملیات مورد نظر خود را انجام دهید.

در بدنه اکسسور get هم هر مقداری را می‌توانید بازگشت دهید. یعنی الزامی وجود ندارد حتماً مقدار فیلد خصوصی متناظر با پروپرتی را بازگشت دهید. حتی الزامی به تعریف فیلد خصوصی برای هر پروپرتی ندارید. به طور مثال ممکن است مقدار بازگشتی اکسسور get حاصل محاسبه و ... باشد.

اکنون مثال دیگری را در نظر بگیرید. فرض کنید در یک پروژه فروشگاهی در حال تهیه کلاسی برای مدیریت محصولات هستید. قصد داریم یک پروپرتی ایجاد کنیم تا نام محصول را نگهداری کند و در حال حاضر هیچ محدودیتی برای نام یک محصول در نظر نداریم. کد زیر را ببینید.

```
public class Product
{
    private string _name;
    public string Name
    {
        get { return _name; }
        set { _name = value; }
    }
}
```

همانطور که می‌بینید در بدنه اکسسورهای پروپرتی Name هیچ عملیات نظارتی ای در نظر گرفته نشده است. آیا بهتر نبود بیهوده پروپرتی تعریف نکنیم و خیلی ساده از یک فیلد عمومی که همین کار را انجام می‌دهد استفاده کنیم؟ خیر. بهتر نبود. مهمترین دلیلی که فعلاً کافی است تا شما را قانع کند تعریف پروپرتی روش پسندیده‌تری از فیلد عمومی است را بررسی می‌کنیم.
فرض کنید پس از مدتی متوجه شدید اگر نام بسیار طولانی ای برای محصول درج شود ظاهر برنامه شما دچار مشکل می‌شود. پس باید بر روی این مورد نظارت داشته باشید. دیدیم که برای رسیدن به این هدف باید فیلد عمومی را فراموش و به جای آن پروپرتی تعریف کنیم. اما مسئله اینست که تبدیل یک فیلد عمومی به پروپرتی میتواند سبب بروز ناسازگاری‌هایی در بخش‌های دیگر برنامه که از این کلاس و آن فیلد استفاده می‌کنند شود. پس بهتر آن است که از ابتدا پروپرتی تعریف کنیم هر چند نیازی به عملیات نظارتی خاصی نداریم. در این حالت اگر نیاز به پردازش بیشتر پیدا شد به راحتی می‌توانیم کد مورد نظر را در اکسسورهای موجود اضافه کنیم بدون آنکه نیازی به تغییر بخش‌های دیگر باشد.
و یک خبر خوب! از سی شارپ ۳ به بعد ویژگی جدیدی در اختیار ما قرار گرفته است که می‌توان پروپرتی‌هایی مانند مثال بالا را که نیازی به عملیات نظارتی ندارند، ساده‌تر و خواناتر تعریف نمود. این ویژگی جدید پروپرتی اتوماتیک یا Auto-Implemented Property نام دارد. مانند نمونه زیر.

```
public class Product
{
    public string Name { get; set; }
}
```

این کد مشابه کد پیشین است با این تفاوت که خود کامپایلر یک متغیر خصوصی و بی نام را ایجاد می‌نماید که فقط داخل اکسسورهای پروپرتی قابل دسترسی است.
البته استفاده از پروپرتی برتری دیگری هم دارد. و آن کنترل سطح دسترسی اکسسورها است. مثال زیر را ببینید.

```
public class Student
{
    public DateTime Birthdate { get; set; }

    public double Age { get; private set; }
}
```

کلاس دانشجو یک پروپرتی به نام تاریخ تولد دارد که قابل خواندن و نوشتن توسط کد مشتری (کد استفاده کننده از کلاس یا اشیاء آن) است. و یک پروپرتی دیگر به نام سن دارد که توسط کد مشتری تنها قابل خواندن است. و تنها توسط سایر اعضای داخل همین کلاس قابل نوشتن است. چون اکسسور set آن به صورت خصوصی تعریف شده است. به این ترتیب بخش دیگری از کلاس سن دانشجو را بر اساس تاریخ تولد او محسابه می‌کند و در پروپرتی Age قرار می‌دهد و کد مشتری می‌تواند آن‌را مورد استفاده قرار دهد اما حق دستکاری آن‌را ندارد. به همین ترتیب در صورت نیاز اکسسور get را می‌توان خصوصی کرد تا پروپرتی از دید کد مشتری فقط نوشتنی باشد. اما حتماً می‌توانید حدس بزنید که نمی‌توان هر دو اکسسور را خصوصی کرد. چرا؟
تذکر: در هنگام تعریف یک فیلد می‌توان از کلمه کلیدی readonly استفاده کرد تا یک فیلد فقط خواندنی ایجاد گردد. اما در اینصورت فیلد تعریف شده حتی داخل کلاس هم فقط خواندنی است و فقط در هنگام تعریف یا در متد سازنده کلاس امکان مقدار دهی به آن وجود دارد. در بخش‌های بعدی مفهوم سازنده کلاس مورد بررسی خواهد گرفت.

نظرات خوانندگان

نویسنده: Kingtak
تاریخ: ۱۳۹۲/۰۲/۰۴ ۱۸:۱۶

مثل همیشه عالی بود....
واقعا با آموزش‌های شما حال میکنم.تا حالاچندین مقاله در مورد پروپرتی‌ها خونده بودم ولی بطور کامل متوجه کاربردش و فوایدش نشده بودم.
منتظر آموزش‌های بعدی هستیم

نویسنده: آرمان فرقانی
تاریخ: ۱۳۹۲/۰۲/۰۴ ۱۹:۱۲

تشکر. شما لطف دارید. امیدوارم مطالب بعدی هم براتون مفید باشه.

نویسنده: سید ایوب کوکبی
تاریخ: ۱۳۹۲/۰۲/۰۴ ۲۱:۲۲

سلام،
خیلی برام لذت بخشه دانسته هام رو به شیوه ای زیباتر مرور کنم و در ضمن با نکاتی ظریفتر آشنا بشم.
چند تا پیشنهاد:
1- خیلی خوب میشه اگه تا حد امکان معادل انگلیسی کلمات تخصصی مربوطه رو هم بزارید، مثلا از اکسسور استفاده میکنید خوبه ولی داخل پرانتز هم اشاره ای به معادل انگلیسی Accessor بکنید تا بدونیم در سینتکسش چه جوریه، و تا حد امکان هم معادل تخصصی واژه مربوطه ذکر بشه مثلا فیلدهای خصوصی داخل کلاس معمولا با عنوان backing field یا فیدلهای پشتی خطاب میشه که اگه به این موارد هم اشاره ای بشه خوبه.
2- جاهایی که به استانداردها و کلا هر چیزی در حوزه مهندسی نرم افزار اشاره می‌کنید لطفا تا حد امکان اشاره ای هم به آن بکنید مثلا فرمودید اصول نامگذاری استاندارد فیلد خصوصی، اینجا به نظر من بهتره که اشاره ای هم بکنید مصلا در استاندارد مایکروسافت فیلدهای خصوصی از قاعده نامگذاری Camel Case استفاده میکنند و یا مثلا در متدها و کلاس‌های از روش Pascal Case استفاده میشه.
3- در مورد اینکه چه مواقعی باید از فلان موضوع، قاعده و مبحث و ... استفاده بشه هم در صورت امکان و تا حد توان اشاره بکنید مثلا فرمودید فیلدهای فقط خواندنی Readonly، مناسبه که اشاره ای هم بکنید معمولا چه زمانی و در چه شرایطی بهتره از این نوع فیلد استفاده کنیم و چرا؟ (البته هنوز که توضیحی ندادید ولی پیشتر اشاره کردم تا انشاء الله در مقاله بعدی تان در صورت صلاحدید لحاظ بفرمایید).

پیشنهاداتی که شد صرفا برای هر چه بهتر شدن مقالات بعدی شماست و اینکه باعث بشه، خواننده وقتی با موضوعاتی در این زمینه در متون تخصصی برخورد داشت احساس بیگانگی نکنه و سریعتر در جریان کار قرار بگیره.
سلسله مباحثی که ارائه کرده اید تا کنون زیبا بود و خواندنی و بنده هم مخاطب همیشگی این سلسله مباحث و البته امیدوارم به کمتر شدن فاصله بین پست‌های ارسالی :).
متشکرم/.

نویسنده: آرمان فرقانی
تاریخ: ۱۳۹۲/۰۲/۰۴ ۲۲:۵۰

ضمن تشکر از پیگیری و پیشنهادهای حضرتعالی و پوزش به جهت طولانی شدن فاصله زمانی ارائه مطالب در مورد پیشنهادهای ارزشمندی که فرمودید باید چند نکته را عرض کنم.
تا حد زیادی معمولاً سعی کردم این موارد محقق بشه. مثلا در مورد همان اکسسور و بیشتر مفاهیم و اصطلاحات مهم، معادل انگلیسی آورده شده است. اصولاً ترجمه برخی مفاهیم را مناسب نمی‌دانم و از طرفی آوردن تعداد زیادی واژه انگلیسی در بین واژگان فارسی سبب کاهش زیبایی متن می‌گردد. بنابراین معمولاً کلمات مهم را یک یا چند بار به صورت انگلیسی بیان می‌کنم و

سپس با حروف فارسی می‌نویسم مانند اکسسور تا به صورت روان‌تری در متن قابل خواندن باشد.
همچنین در امر آموزش ابتدا سعی می‌کنم یک دید کلی و از بالا به دانشجو یا خواننده منتقل کنم. در این مرحله تنها جزییات مهم که برای درک موضوع و شروع کار عملی مانند انجام یک پروژه کاربردی لازم است بیان می‌شود. چراکه اگر از ابتدا ذهن را با تعداد زیادی جزییات درگیر کنیم ممکن است در موقع خواندن هر بخش خواننده مفاهیم را درک کند اما پس از پایان مطالب نمی‌داند از کجا باید شروع کند و قدرت استفاده از آموخته‌ها را ندارد. به همین جهت سعی می‌شود بر روی مفاهیم غیر کلیدی کمتر در مراحل اولیه بحث شود.
از طرفی سعی می‌کنم مطالب دارای حجم مناسب و مفاهیم پیوسته ای باشند تا قابل درک بوده و خسته کننده نباشند. مثلاً از آنجاییکه در بخش‌های پیشین مقاله‌ای که به زحمت یکی از دوستان در سایت قرار گرفته بود برای نامگذاری معرفی شد، از تکرار قوانین یاد شده در این مطالب به جهت جلوگیری از طولانی‌تر شدن خودداری کردم.
با توجه به کارگاه‌های عملی ای که برای تثبیت مطالب در نظر گرفته خواهد شد، تا حد زیادی روش‌های بهینه برای پیاده سازی مفاهیم گوناگون معرفی خواهد شد.

نویسنده: عبداللهی
تاریخ: ۱۳۹۲/۰۲/۰۴ ۲۳:۲۶

سلام.
واقعا عالی بود.مرسی.فقط یه سوالی داشتم. در کلاس student سطح دسترسی بصورت پیش فرض private خواهد بود؟چون اگر درست یادم مونده باشه موقع تعریف متغیر سطح دسترسی بصورت پیشفرض private بود.درسته؟
بازم ممنون.3 تا مقالتون رو خودنم.واقعا مفید بود.باتشکر

نویسنده: آرمان فرقانی
تاریخ: ۱۳۹۲/۰۲/۰۵ ۱۱:۲۲

سلام و تشکر.
سطح دسترسی پیش فرض برای اعضای کلاس (حتی کلاس داخلی‌تر) به صورت پیش فرض private است. اما اکسسورها از سطح دسترسی پروپرتی تبعیت می‌کنند مگر آنکه صراحتاً سطح دسترسی آن‌ها تعیین گردد. بدیهی است در صورتی تعیین صریح سطح دسترسی برای اکسسورها پذیرفته است که نسبت به سطح دسترسی پروپرتی محدودتر باشد. یعنی نمی‌توانید مثلاً پروپرتی ای را private و اکسسور آن را public تعیین کنید.

نویسنده: علی صداقت
تاریخ: ۱۳۹۲/۰۲/۰۷ ۱۰:۲۸

با سپاس فراوان از سلسه مطالب مفید شما. فکر میکنم در کلاس Rectangle و پروپرتی Height در قسمت set، مقدار value باید در شرط مورد ارزیابی قرار گیرد.

نویسنده: محسن خان
تاریخ: ۱۳۹۲/۰۲/۰۷ ۱۱:۲۴

```
set { if (value > 0) { _width = value; } }
```

این نوع طراحی API به نظر من ایراد داره. از این جهت که مصرف کننده نمی‌دونه چرا مقداری که وارد کرده تاثیری نداشته. بهتره در این نوع موارد یک استثناء صادر شود.

نویسنده: آرمان فرقانی
تاریخ: ۱۳۹۲/۰۲/۰۷ ۱۱:۳۱

با تشکر. اصلاح شد.

نویسنده: آرمان فرقانی
تاریخ: ۱۳۹۲/۰۲/۰۷ ۱۱:۴۰

تشکر فراوان از نظر حضرتعالی.
بله صحیح می‌فرمایید. اما کار با این کلاس تمام نشده است و صرفاً مثالی ساده برای بیان مفاهیم پایه ای مورد نظر در مقاله است. در چنین مثالی نباید ذهن خواننده را درگیر مسائلی نمود که مورد هدف بحث نیست. ضمناً هر مقاله دارای یک جامعه هدف است. اگرچه می‌تواند برای افراد دیگر هم مفید واقع شود اما باید دانسته‌های جامعه هدف خود را مد نظر داشته باشد. برای خواننده ای که در حال آشنایی با مفهوم پروپرتی است، صدور یک استثنا و مفاهیم مربوطه نیاز به بحثی جدا دارد.

نویسنده: سید ایوب کوکبی
تاریخ: ۱۳۹۲/۰۲/۱۰ ۲۲:۱۷

البته امیدوارم هستم این موارد را در مقالات بعدی خود شرح دهید.

نویسنده: محمد
تاریخ: ۱۳۹۲/۰۲/۱۷ ۳:۵۲

خیلی ممنون بابت مطلبتون
یه سوالی که مدت‌ها تو ذهنم مونده اینه که فرق این کار (استفاده از property برای مقدار دهی فیلدهای private) با مدل مشابهی که در کتاب‌های جاوا دیده می‌شه (استفاده از دو متد به طور مثال setWidth و getWidth برای مقدار دهی فیلدهای private در اینجا width) در چیه؟

نویسنده: محسن خان
تاریخ: ۱۳۹۲/۰۲/۱۷ ۸:۹

این روش در سی‌شارپ منسوخ شده در نظر گرفته میشه و یکی از مواردی هست که حین تبدیل کدهای جاوا به سی شارپ تبدیل به یک خاصیت خواهد شد بجای دو متد.

نویسنده: صابر فتح الهی
تاریخ: ۱۳۹۲/۰۲/۱۷ ۸:۱۹

با اجازه آقای آرمان فرقانی
دوست گلم دقیقا وقتی برنامه کامپایل میشه خصیصه‌ها به متدهایی مانند جاوا (که با set_ یا get_ شروع شده و به نام خصیصه ختم می‌شود) تبدیل میشوند
مثلا شما توی کلاست اگر خصیصه ای با نام Width داشته باشید و یک متد مانند get_Width تعریف کنی زمان کامپایل خطا زیر دریافت می‌کنی، به منزله اینکه این متد وجود داره:

```
Error 1 Type 'Rectangle' already reserves a member called 'get_Width' with the same parameter types
E:\Test\Rectangle.cs5940
```

نویسنده: آرمان فرقانی
تاریخ: ۱۳۹۲/۰۲/۱۷ ۱۲:۴۱

تشکر از شما و توضیحات ارزشمند دوستان گرامی.
پروپرتی و پروپرتی اتوماتیک امکانی است که در زبان سی شارپ و ... قرار داده شده است. پروپرتی‌ها نیز در حقیقت متدهای مشابهی دارند که همان اکسسورها هستند. تفاوت میزان بیشتر کپسوله سازی و مخفی کردن منطق پیاده سازی، و مهم‌تر سازگاری بیشتر با مفهوم ویژگی است. که البته در هنگام استفاده از پروپرتی سهولت بیشتری را نیز فراهم می‌کند.
همان که دوست عزیزم اشاره فرمودند به دلیل عدم سازگاری ذات زبان‌های مبتنی بر دات فریمورک از اکسسور، به صورت

داخلی به متد تبدیل خواهند شد.
همچنین در مورد جاوا هم پروژه هایی وجود دارند که سعی کرده اند این امکان را به کمک یک سری Annotaion به آن بیافزایند.
در مورد سی شارپ استفاده از پروپرتی روش توصیه شده است.

نویسنده: محسن خان
تاریخ: ۱۳۹۲/۰۲/۱۷ ۱۳:۲۱

دات نت مشکلی با متدهای get و set دار نداره. فقط چون خیلی verbose بوده، جمع و جور شده به auto implemented properties برای زیبایی کار و سهولت تایپ. نکته‌ای هم که آقای فتح الهی عنوان کردند، در مورد ترکیب متد و خاصیت هم نام با هم بود، در یکجا البته اون هم حالتی که بعد از متد get شروع شده با حرف کوچک، یک _ باشد مثلا و نه حالت دیگری.

نویسنده: آرمان فرقانی
تاریخ: ۱۳۹۲/۰۲/۱۷ ۱۴:۲۹

متوجه نکته مورد نظر شما نشدم. بیان شد در زبان سی شارپ و ... ساختار کپسوله‌تر پروپرتی در مقایسه با متدهای صریح تنظیم و بازیابی مقدار فیلدها در جاوا معرفی شده اند ولی پیاده سازی داخلی آن به همان صورت متد است. نکته دوست گرامی آقای فتح الهی هم گمان می‌کنم بیشتر به منظور اشاره به چگونگی پیاده سازی داخلی است و نه اینکه مراقب باشید تداخل نام پیش نیاید.

نویسنده: صابر فتح الهی
تاریخ: ۱۳۹۲/۰۲/۱۷ ۱۷:۱۰

بله من در پاسخ به سوال دوستمون گفتم **پیاده سازی داخلی** خصیصه به این شکل هست که خصیصه به شکل متد پیاده سازی است، و جهت آزمایش گفتم یک متد ... ایجاد کنید.

عنوان: **نحوه کار با ftp - بخش اول**
نویسنده: بهمن آبادی
تاریخ: ۱۳۹۲/۰۲/۱۱ ۱۹:۵۰
آدرس: www.dotnettips.info
گروه‌ها: ASP.Net, C#, system.net, ftp

امروز می‌خوام نحوه کار با FTP بصورت ساده برای کاربران و برنامه نویسان مبتدی رو آموزش بدم.

برای استفاده از FTP نیاز به یک اکانت FTP در سایت مورد نظر بهمراه دسترسی به پوشه ای مشخص می‌باشد.
برای مثال ما یک اکانت FTP در سایت dotnettip.info داریم که به پوشه upload دسترسی داره.

ابتدا در فایل Web.config و در بین تگ های appSettings مقادیر زیر را برای دسترسی به اکانت و نام کاربری و رمز عبور ذخیره می‌کنیم.

```
<add key="FtpAddress" value="ftp://ftp.dotnetips.info" />
<add key="FtpUser" value="uploadcenter" />
<add key="FtpPass" value="123123" />
<add key="FolderPath" value="~/Upload/" />
```

*نکته : برای امنیت بیشتر و دسترسی به اطلاعات اکانت می‌شود از روش‌های دیگری نیز استفاده کرد.

در ادامه یک کلاس در App_code پروژه خود با نام FTPHelper ایجاد می‌کنیم و کد زیر را در آن قرار می‌دهیم:
تکه کد بالا برای ست کردن مقادیر نام کاربری و رمز عبور و آدرس FTP در کلاس مذکور که بصورت پیش‌فرض از web.config پر می‌شود ایجاد و بکار خواهد رفت.

```
using System.Net;
using System.IO;
using System.Configuration;


public class FtpHelper
{
public FtpHelper()
{
       //Default Value Set From Application
         _hostname = ConfigurationManager.AppSettings.GetValues("FtpAddress")[0];
        _username = ConfigurationManager.AppSettings.GetValues("FtpUser")[0];
        _password = ConfigurationManager.AppSettings.GetValues("FtpPass")[0];
}

    #region "Properties"
    private string _hostname;
    /// <summary>
    /// Hostname
    /// </summary>
    /// <value></value>
    /// <remarks>Hostname can be in either the full URL format
    /// ftp://ftp.myhost.com or just ftp.myhost.com
    /// </remarks>
    public string Hostname
    {
        get
        {
            if (_hostname.StartsWith("ftp://"))
            {
                return _hostname;
            }
            else
            {
                return "ftp://" + _hostname;
            }
        }
        set
        {
            _hostname = value;
        }
    }
    private string _username;
```

```
    /// <summary>
    /// Username property
    /// </summary>
    /// <value></value>
    /// <remarks>Can be left blank, in which case 'anonymous' is returned</remarks>
    public string Username
    {
        get
        {
            return (_username == "" ? "anonymous" : _username);
        }
        set
        {
            _username = value;
        }
    }
    private string _password;
    public string Password
    {
        get
        {
            return _password;
        }
        set
        {
            _password = value;
        }
    }


    #endregion
}
```

سپس فضای نام‌های زیر را در کلاس خود قرار می‌دهیم.

```
using System.Net;
using System.IO;
```

حالا برای بارگذاری فایل می‌توانیم از یک تابع بصورت shared استفاده کنیم که بتوان با دادن آدرس فایل بصورت فیزیکی به تابع و مشخص کردن پوشه مورد نظر آنرا در هاست مقصد (FTP) بارگذاری کرد.توجه داشته باشید که تابع فوق نیازی به قرار گرفتن در کلاس بالا (FtpHelper) ندارد.یعنی می‌توان آنرا در هرجای برنامه پیاده سازی نمود.

```
public static bool Upload(string fileUrl)
        {
            if (File.Exists(fileUrl))
            {
                FtpHelper ftpClient = new FtpHelper();
                string ftpUrl = ftpClient.Hostname + System.IO.Path.GetFileName(fileUrl);


                FtpWebRequest ftp = (FtpWebRequest)FtpWebRequest.Create(ftpUrl);
                ftp.Credentials = new NetworkCredential(ftpClient.Username, ftpClient.Password);

                ftp.KeepAlive = true;
                ftp.UseBinary = true;
                ftp.Timeout = 3600000;
                ftp.KeepAlive = true;
                ftp.Method = WebRequestMethods.Ftp.UploadFile;

                const int bufferLength = 102400;
                byte[] buffer = new byte[bufferLength];
                int readBytes = 0;

                //open file for reading
                using (FileStream fs = File.OpenRead(fileUrl))
                {
                    try
                    {
                        //open request to send
                        using (Stream rs = ftp.GetRequestStream())
```

```
                    {
                        do
                        {
                            readBytes = fs.Read(buffer, 0, bufferLength);
                            fs.Write(buffer, 0, readBytes);
                        } while (!(readBytes < bufferLength));
                        rs.Close();
                    }

                }
                catch (Exception)
                {
                 //Optional Alert for Exeption To Application Layer
                   //throw (new ApplicationException("بارگذاری فایل با خطا  رو به رو شد"));

                }
                finally
                {
                    //ensure file closed
                    //fs.Close();
                }

            }

            ftp = null;
            return true;
        }
        return false;



    }
```

تکه کد بالا فایل مورد نظر را در صورت وجود به صورت تکه‌های 100 کیلوبایتی بر روی ftp بارگذاری می‌کند، که می‌توانید مقدار آنرا نیز تغییر دهید.
اینکار باعث افزایش سرعت بارگذاری در فایل‌های با حجم بالا برای بارگذاری می‌شود.

در بخش‌های بعدی نحوه ایجاد پوشه ، حذف فایل ، حذف پوشه و دانلود فایل از روی FTP را بررسی خواهیم کرد.

نظرات خوانندگان

نویسنده: وحید نصیری
تاریخ: ۱۳۹۲/۰۲/۱۱ ۲۲:۳۸

ممنون از مطلب شما.
چند نکته جزئی در مورد کدهای تهیه شده:
- وجود این try/catch در اینجا هیچ هدفی رو برآورده نکرده. از قسمت throw ex هم توصیه می‌شود که استفاده نکنید. از thow خالی استفاده کنید تا stack trace پاک نشه. به علاوه زمانیکه مشغول به طراحی یک کتابخانه هستید تا حد ممکن از ذکر try/catch خودداری کنید. وظیفه بررسی این مسایل مرتبط است به لایه‌های بالاتر استفاده کننده و نه کتابخانه پایه.
- if ابتدای متد هم ضرورتی ندارد. اگر قرار است باشد، باید به استفاده کننده در طی یک استثناء اعلام شود که چرا فایل درخواستی او آپلود نشده. در کل استفاده از متد File.Exists به همراه صدور استثناء در صورت عدم وجود فایل، در اینجا مناسب‌تر است.
- نامگذاری‌هایی مانند obj_ftp مربوط به دوران C است. در سی‌شارپ روش دیگری رو باید استفاده کنید که در مطلب اصول نامگذاری در دات نت به تفصیل بررسی شده.
- بررسی صفر بودن readBytes بهتر است پیش از فراخوانی متد Write انجام شود.
- یک سری از اشیاء در دات نت پیاده سازی کننده IDispoable هستند. به این معنا که بهتر است از using برای استفاده از آن‌ها کمک گرفته شود تا کامپایلر قسمت finally به همراه آزاد سازی منابع را به صورت خودکار اضافه کند. این نکته برای مواردی که در بین کار استثنایی رخ می‌دهد جهت آزاد سازی منابع لازم است. یعنی بهتر بود بجای try/catch از try/finally و یا using در مکان‌های مناسب استفاده می‌شد.
- علت استفاده از شیء Application دراینجا چه چیزی بوده؟ AppSettings خوانده شده از وب کانفیگ برنامه و کل اطلاعات آن در آغاز به کار یک برنامه ASP.NET به صورت خودکار کش می‌شوند. به همین جهت است که اگر حتی یک نقطه در فایل وب کانفیگ تغییر کند برنامه ASP.NET ری استارت می‌شود (تا دوباره تنظیمات را بخواند). بنابراین مستقیما از همان امکانات ConfigurationManager بدون انتساب آن به شیء سراسری Application استفاده کنید. اینکار سربارى آنچنانی هم ندارد؛ چون از حافظه خوانده می‌شود و نه از فایل. هر بار فراخوانی ConfigurationManager.AppSettings به معنای مراجعه به فایل web.config نیست. فقط بار اول اینکار انجام می‌شود؛ تا زمانیکه این فایل تغییر کند یا برنامه ری استارت شود.

نویسنده: بهمن آبادی
تاریخ: ۱۳۹۲/۰۲/۱۲ ۳:۳۰

با تشکر از نظر شما. تمام صحبت‌های شما منطقی است. اولا من این بخش رو می‌خواستم بعدا تکمیل کنم ولی بدلایلی زودتر ارسال کردم.

مواردی هم که اعلام کزدید کاملا صحیح می‌باشد.فقط بخش تابعی که درون آن از try/catch استفاده شده رو میخواستم ابتدا در تیکه کد دیگری از یک صفحه aspx بنویسم که وقتی برای بررسی مجدد تابع پیدا نکردم وگرنه اونجا try/catch بلا استفاده است.

موارد متذکر شده برای سهولت بیشتر کاربران در کدهای فوق لحاظ و ویرایش گردید.
با تشکر.

نویسنده: محسن خان
تاریخ: ۱۳۹۲/۰۲/۱۲ ۱۱:۴

```
.locals init ([0] class [mscorlib]System.IO.TextWriter w)
  IL_0000:  ldstr      "log.txt"
  IL_0005:  call       class [mscorlib]System.IO.StreamWriter
      [mscorlib]System.IO.File::CreateText(string)
  IL_000a:  stloc.0

 .try
  {
    IL_000b:  ldloc.0
```

```
    IL_000c:  ldstr      "This is line one"
    IL_0011:  callvirt   instance void [mscorlib]
      System.IO.TextWriter::WriteLine(string)
    IL_0016:  leave.s    IL_0022
  }  // end .try
  finally
  {
    IL_0018:  ldloc.0
    IL_0019:  brfalse.s  IL_0021
    IL_001b:  ldloc.0
    IL_001c:  callvirt   instance void [mscorlib]
      System.IDisposable::Dispose()
    IL_0021:  endfinally
  }  // end handler
```

ماخذ

زمانیکه از using statement استفاده می‌کنید، خود کامپایلر try/finally رو اضافه می‌کنه. یعنی قسمت close الان بهش نیازی نیست چون استریم مورد استفاده حتما در اینجا dispose میشه.

نویسنده: بهمن آبادی
تاریخ: ۱۳۹۲/۰۲/۱۲ ۱۲:۵۰

از دستم در رفته ...شما ببخشید.

بیشتر هدفم روند بارگذاری و تکنیک بارگذاری اون بود ولی خوب درست کد نوشتن کاملاً صحیحه و من می‌خواستم برای کاربرای آماتور بنویسم ولی مجبورم کردین که کاربرای حرفه ای هم مثل شما به کد ایرادی نباشه.

بازم ممنون از دقت و راهنماییتون .

حالا بهتر شد نسبت به قبلیه ؟

نویسنده: محسن خان
تاریخ: ۱۳۹۲/۰۲/۱۲ ۱۷:۲۸

من فکر می‌کنم اگر ReSharper رو نصب کنید، پیش از ارائه یک مطلب یا پروژه خیلی از ایرادات رو با خط کشیدن زیر اون یا نمایش یک علامت زرد کنار صفحه گوشزد می‌کنه. مثلا می‌گه که این نوع نامگذاری درست نیست یا این شیء رو میشه با using محصور کرد. خلاصه از دستش ندید، حیفه!

نویسنده: سمیرا قادری
تاریخ: ۱۳۹۲/۰۶/۱۳ ۱۲:۳۷

با تشکر از مطلب خوبتان
ببخشید این روش کار کردن با Ftp چه مزایایی دارد و چه کاربردی دارد؟ چون من تا حالا فقط از Cute Ftp استفاده کردم آن هم برای Upload سایت بعد از Publish

نویسنده: محسن خان
تاریخ: ۱۳۹۲/۰۶/۱۳ ۱۲:۵۱

یه خورده وقت بذاری، باهاش می‌تونی یک Cute Ftp بنویسی.

نویسنده: رضا منصوری
تاریخ: ۱۳۹۲/۱۲/۲۰ ۱۵:۵۰

با تشکر از مطلب خوبتون ،
برای اینکه به طور مستقیم از file.SaveAs استفاده کنیم و فایل را مستقیم به Ftp آپلود کنیم (نه اینکه از فضای ذخیره شده ای کپی کنیم) چه پیشنهادی می‌دهید ؟ مثلا ما هاست دانلود داریم و میخواهیم فایل‌های ارسالی را در هاست دانلود ذخیره کنیم.

عنوان: **مفاهیم برنامه نویسی ـ آشنایی با سازنده‌ها**
نویسنده: آرمان فرقانی
تاریخ: ۱۳۹۲/۰۲/۱۴ ۳:۰
آدرس: www.dotnettips.info
گروه‌ها: C#, OOP, Programming, آموزش برنامه نویسی

در مطلب پیشین برای نگهداری حالت شیء یا همان ویژگی‌های آن Propertyها را در کلاس معرفی کردیم و پس از ایجاد شیء مقدار مناسبی را به پروپرتی‌ها اختصاص دادیم.
اگرچه ایجاد شیء و مقداردهی به ویژگی‌های آن ما را به هدفمان می‌رساند، اما بهترین روش نیست چرا که ممکن است مقداردهی به یک ویژگی فراموش شده و سبب شود شیء در وضعیت نادرستی قرار گیرد. این مشکل با استفاده از سازنده‌ها (Constructors) حل می‌شود.
سازنده (Constructor) عضو ویژه ای از کلاس است بسیار شبیه به یک متد که در هنگام وهله سازی (ایجاد یک شیء از کلاس) به صورت خودکار فراخوانی و اجرا می‌شود. وظیفه سازنده مقداردهی به ویژگی‌های عمومی و خصوصی شیء تازه ساخته شده و به طور کلی انجام هر کاری است که برنامه‌نویس در نظر دارد پیش از استفاده از شیء به انجام برساند.
مثالی که در این بخش بررسی می‌کنیم کلاس مثلث است. برای پیاده سازی این کلاس سه ویژگی در نظر گرفته ایم. قاعده، ارتفاع و مساحت. بله مساحت را این بار به جای متد به صورت یک پروپرتی پیاده سازی می‌کنیم. اگرچه در آینده بیشتر راجع به چگونگی انتخاب برای پیاده سازی یک عضو کلاس به صورت پروپرتی یا متد بحث خواهیم کرد اما به عنوان یک قانون کلی در نظر داشته باشید عضوی که به صورت منطقی به عنوان داده مطرح است را به صورت پروپرتی پیاده سازی کنید. مانند نام دانشجو. از طرفی اعضایی که دلالت بر انجام عملی دارند را به صورت متد پیاده سازی می‌کنیم. مانند متد تبدیل به نوع داده دیگر. (مثلاً Object.ToString())

```
public class Triangle
{
    private int _height;
    private int _baseLength;

    public int Height
    {
        get { return _height; }

        set
        {
            if (value < 1 || value > 100)
            {
                // تولید خطا
            }

            _height = value;
        }
    }

    public int BaseLength
    {
        get { return _baseLength; }

        set
        {
            if (value < 1 || value > 100)
            {
                // تولید خطا
            }

            _baseLength = value;
        }
    }

    public double Area
    {
        get { return _height * _baseLength * 0.5; }
    }
}
```

چون در بخشی از یک پروژه نیاز پیدا کردیم با یک سری مثلث کار کنیم، کلاس بالا را طراحی کرده ایم. به نکات زیر توجه نمایید.
• در اکسسور set دو ویژگی قاعده و ارتفاع، محدوده مجاز مقادیر قابل انتساب را بررسی نموده ایم. در صورتی که مقداری خارج از محدوده یاد شده برای این ویژگی‌ها تنظیم شود خطایی را ایجاد خواهیم کرد. شاید برای برنامه نویسانی که تجربه کمتری دارند

زیاد روش مناسبی به نظر نرسد. اما این یک روش قابل توصیه است. مواجه شدن کد مشتری (کد استفاده کننده از کلاس) با یک خطای مهلک که علت رخ دادن خطا را نیز می‌توان به همراه آن ارائه کرد بسیار بهتر از بروز خطاهای منطقی در برنامه است. چون رفع خطاهای منطقی بسیار دشوارتر است. در مطالب آینده راجع به تولید خطا و موارد مرتبط با آن بیشتر صحبت می‌کنیم.

• در مورد ویژگی مساحت، اکسسور set را پیاده سازی نکرده ایم تا این ویژگی را به صورت فقط خواندنی ایجاد کنیم.

وقتی شیء ای از یک کلاس ایجاد می‌شود، بلافاصله سازنده آن فراخوانی می‌گردد. سازنده‌ها هم نام کلاسی هستند که در آن تعریف می‌شوند و معمولاً اعضای داده ای شیء جدید را مقداردهی می‌کند. همانطور که می‌دانید وهله سازی از یک کلاس با عملگر new انجام می‌شود. سازنده کلاس بلافاصله پس از آنکه حافظه برای شیء در حال تولید اختصاص داده شد، توسط عملگر new فراخوانی می‌شود.

سازنده پیش فرض سازنده‌ها مانند متدهای دیگر می‌توانند پارامتر دریافت کنند. سازنده ای که هیچ پارامتری دریافت نمی‌کند سازنده پیش فرض (Default constructor) نامیده می‌شود. سازنده پیش فرض زمانی اجرا می‌شود که با استفاده از عملگر new شیء ای ایجاد می‌کنید اما هیچ آرگومانی را برای این عملگر در نظر نگرفته اید.
اگر برای کلاسی که طراحی می‌کنید سازنده ای تعریف نکرده باشید کامپایلر سی شارپ یک سازنده پیش فرض (بدون پارامتر) خواهد ساخت. این سازنده هنگام ایجاد اشیاء فراخوانی شده و مقدار پیش فرض متغیرها و پروپرتی‌ها را با توجه به نوع آن‌ها تنظیم می‌نماید. مثلاً مقدار صفر برای متغیری از نوع int یا false برای نوع bool و null برای انواع ارجاعی که در آینده در این مورد بیشتر خواهید آموخت.
اگر مقادیر پیش فرض برای متغیرها و پروپرتی‌ها مناسب نباشد، مانند مثال ما، سازنده پیش فرض ساخته شده توسط کامپایلر همواره شیء ای می‌سازد که وضعیت صحیحی ندارد و نمی‌تواند وظیفه خود را انجام دهد. در این گونه موارد باید این سازنده را جایگزین نمود.

جایگزینی سازنده پیش فرض ساخته شده توسط کامپایلر افزودن یک سازنده صریح به کلاس بسیار شبیه به تعریف یک متد در کلاس است. با این تفاوت که:
سازنده هم نام کلاس است.
برای سازنده نوع خروجی در نظر گرفته نمی‌شود.

در مثال ما محدوده مجاز برای قاعده و ارتفاع مثلث بین ۱ تا ۱۰۰ است در حالی که سازنده پیش فرض مقدار صفر را برای آنها تنظیم خواهد نمود. پس برای اینکه مطمئن شویم اشیاء مثلث ساخته شده از این کلاس در همان بدو تولید دارای قاعده و ارتفاع معتبری هستند سازنده زیر را به صورت صریح در کلاس تعریف می‌کنیم تا جایگزین سازنده پیش فرضی شود که کامپایلر خواهد ساخت و به جای آن فراخوانی گردد.

```
public Triangle()
{
    _height = _baseLength = 1;
}
```

در این سازنده مقدار ۱ را برای متغیر خصوصی پشت (backing field یا backing store) هر یک از دو ویژگی قاعده و ارتفاع تنظیم نموده ایم.

اجرای سازنده همانطور که گفته شد سازنده اضافه شده به کلاس جایگزین سازنده پیش فرض کامپایلر شده و در هنگام ایجاد یک شیء جدید از کلاس مثلث توسط عملگر new اجرا می‌شود. برای بررسی اجرا شدن سازنده به سادگی می‌توان کدی مشابه مثال زیر را نوشت.

```
Triangle triangle = new Triangle();

Console.WriteLine(triangle.Height);
Console.WriteLine(triangle.BaseLength);
Console.WriteLine(triangle.Area);
```

کد بالا مقدار ۱ را برای قاعده و ارتفاع و مقدار ۰.۵ را برای مساحت چاپ می‌نماید. بنابراین مشخص است که سازنده اجرا شده و مقادیر مناسب را برای شیء تنظیم نموده به طوری که شیء از بدو تولید در وضعیت مناسبی است.

سازنده‌های پارامتر دار در مثال قبل یک سازنده بدون پارامتر را به کلاس اضافه کردیم. این سازنده تنها مقادیر پیش فرض مناسبی را تنظیم می‌کند. بدیهی است پس از ایجاد شیء در صورت نیاز می‌توان مقادیر مورد نظر دیگر را برای قاعده و ارتفاع تنظیم نمود. اما برای اینکه سازنده بهتر بتواند فرآیند وهله سازی را کنترل نماید می‌توان پارامترهایی را به آن افزود. افزودن پارامتر به سازنده مانند افزودن پارامتر به متدهای دیگر صورت می‌گیرد. در مثال زیر سازنده دیگری تعریف می‌کنیم که دارای دو پارامتر است. یکی قاعده و دیگری ارتفاع. به این ترتیب در حین فرآیند وهله سازی می‌توان مقادیر مورد نظر را منتسب نمود.

```
public Triangle(int height, int baseLength)
{
    Height = height;
    BaseLength = baseLength;
}
```

با توجه به اینکه مقادیر ارسالی به این سازنده توسط کد مشتری در نظر گرفته می‌شود و ممکن است در محدوده مجاز نباشد، به جای انتساب مستقیم این مقادیر به فیلد خصوصی پشت ویژگی قاعده و ارتفاع یعنی _baseLength و _height آنها را به پروپرتی‌ها منتسب کردیم تا قانون اعتبارسنجی موجود در اکسسور set پروپرتی‌ها از انتساب مقادیر غیر مجاز جلوگیری کند.
سازنده اخیر را می‌توان به صورت زیر با استفاده از عملگر new و فراهم کردن آرگومان‌های مورد نظر مورد استفاده قرار داد.

```
Triangle triangle = new Triangle(5, 8);

Console.WriteLine(triangle.Height);
Console.WriteLine(triangle.BaseLength);
Console.WriteLine(triangle.Area);
```

مقادیر چاپ شده برابر ۵ برای ارتفاع، ۸ برای قاعده و ۲۰ برای مساحت خواهد بود که نشان از اجرای صحیح سازنده دارد.
در مطالب بالا چندین بار از سازنده ها صحبت کردیم و گفتیم سازنده دیگری به کلاس اضافه می‌کنیم. این دو نکته را به خاطر داشته باشید:
یک کلاس می‌تواند دارای چندین سازنده باشد که بر اساس آرگومان‌های فراهم شده هنگام وهله سازی، سازنده مورد نظر انتخاب و اجرا می‌شود.
الزامی به تعریف یک سازنده پیش فرض (به معنای بدون پارامتر) نیست. یعنی یک کلاس می‌تواند هیچ سازنده بی پارامتری نداشته باشد.

در بخش‌های بعدی مطالب بیشتری در مورد سازنده‌ها و سایر اعضای کلاس خواهید آموخت.

نظرات خوانندگان

نویسنده: محسن خان
تاریخ: ۱۳۹۲/۰۲/۱۴ ۸:۴۵

ممنون از شما. بنابراین مقدار دهی خواص در سازنده کلاس به معنای نسبت دادن مقدار پیش فرض به آن‌ها است. چون سازنده کلاس پیش از هر کد دیگری در کلاس فراخوانی می‌شود.

نویسنده: آرمان فرقانی
تاریخ: ۱۳۹۲/۰۲/۱۴ ۱۰:۳۱

تشکر. همانطور که گفته شد بلافاصله پس از تخصیص حافظه به شیء سازنده اجرا می‌شود. در صورت استفاده از سازنده پارامتر دار کد مشتری از ابتدا شیء را با مقادیر عملیاتی خود ایجاد می‌کند.

نویسنده: سید ایوب کوکبی
تاریخ: ۱۳۹۲/۰۲/۱۴ ۱۲:۵۹

سلام،
آقای فرقانی بسیار عالی بود، واقعا لذت بردم،
فقط خواهشی از شما دارم این هستش که : دقت و حساسیت مبحث شی گرایی رو احیانا فدای چیزهای دیگر نکنید!، منظورم این هستش که تا حد توان اصول مهندسی نرم افزار رو به صورت کامل رعایت بفرمایید و نگران کاربر مبتدی نباشید، کاربر مبتدی ، ناخودآگاه خودش مطالب غیر قابل فهم رو فیلتر میکنه و در بدترین حالت، در آن مورد، از شما سوال خواهند کرد و شما هم آنها را به موضوع مناسب ارجاع خواهید داد هر چند اگر موضوع ارائه شده بعدا قراره باز بشه، خیلی راحت می‌تونید در متن نوید توضیحات بیشتر رو به خواننده بدهید و در غیر این صورت ارجاع به توضیحات مناسب.
به هر حال باز هم تاکید می‌کنم و سفارش میکنم که دقت بیان و رعایت اصول رو فدای هیچ چیزی نکنید.
ممنونم.

نویسنده: آرمان فرقانی
تاریخ: ۱۳۹۲/۰۲/۱۴ ۱۳:۲۳

تشکر از نظر شما.
اگر مورد خاصی مد نظر دارید بفرمایید تا توضیح بدم یا در صورت لزوم در متن اصلاح کنم. در نظر داشته باشید این سری مطالب با مطالب دیگر موجود در سایت متفاوت است و هدف خاصی را دنبال می‌کند که در بخش اول توضیح داده شد. بنابراین نمی‌توان در این سری مطالب نگران کاربر مبتدی‌تر نبود چراکه جامعه هدف این بحث‌ها هستند. به دلیل همین جامعه هدف مورد نظر، نوشتن این گونه مطالب دشوارتر از بیان مفاهیم پیچیده‌تر برای کاربران حرفه‌ای‌تر است. و همین امر به علاوه وقت محدود بنده سبب تأخیر در ارسال مطالب است. ضمناً صرف بیان انبوهی از اطلاعات تأثیر لازم را در خواننده نمی‌گذارد. در بسیاری مواقع بیان برخی مفاهیم مهندسی نرم افزار یا ویژگی‌های جدید زبان (مانند var) به صورت مقایسه ای با روش پیشین سبب تثبیت بهتر مطالب در ذهن خواننده می‌شود. اما در شیوه کد نویسی تا حد ممکن سعی شده اصول رعایت شود و بیهوده خواننده با روش ناصحیح آشنا نشود. اگر نکته خاصی یافتید بفرمایید اصلاح کنیم.

نویسنده: عبداللهی
تاریخ: ۱۳۹۲/۰۲/۱۶ ۰:۵۱

باسلام.تشکر ازینکه مطالب رو ساده گویا و دقیق بیان میکنید.بسیار عالی بود.سپاسگذارم.
یک کلاس همیشه 1 سازنده پیش فرض بدون پارامتر دارد که هنگام وهله سازی فراخوانی شده و اجرا میشود و در صورت نیاز میتوان 1 سازنده بر اساس نیازهای پروژه تعریف کرد که هنگام وهله سازی از یک کلاس سازنده تعریف شده ما بر سازنده پیش فرض اولویت دارد.درسته؟
بازم متشکرم.مرسی

نویسنده: محسن خان
تاریخ: ۱۳۹۲/۰۲/۱۶ ۱۱:۱۲

محدودیتی برای تعداد متدهای سازنده وجود نداره (مبحث overloading است که نیاز به بحثی جداگانه دارند). در زمان وهله سازی کلاس میشه مشخص کرد کدام متد مورد استفاده قرار بگیره. این متد بر سایرین مقدم خواهد بود. همچنین سازنده استاتیک هم قابل تعریف است که نکته خاص خودش رو داره.

نویسنده: آرمان فرقانی
تاریخ: ۱۳۹۲/۰۲/۱۶ ۱۱:۱۴

سلام و تشکر. به نکات زیر در متن توجه فرمایید.
۱. سازنده پیش فرض منظور همان سازنده بی پارامتر است خواه توسط کامپایلر ایجاد شده باشد خواه توسط برنامه نویس به صورت صریح در کلاس تعریف شده باشد.
۲. اگر توسط برنامه نویس هیچ سازنده ای تعریف نگردد، کامپایلر یک سازنده بدون پارامتر خواهد ساخت که وظیفه تنظیم مقادیر پیش فرض اعضای داده ای کلاس را برعهده بگیرد.
۳. اگر برنامه نویس سازنده ای تعریف کند خواه با پارامتر یا بی پارامتر کامپایلر سازنده ای نخواهد ساخت. پس اگر مثلاً برنامه نویس تنها یک سازنده با پارامتر تعریف کند، کلاس فاقد سازنده بی پارامتر یا پیش فرض خواهد بود.
۴. در یک کلاس می‌توان چندین سازنده تعریف نمود.
موفق باشید.

نویسنده: محسن نجف زاده
تاریخ: ۱۳۹۲/۰۲/۲۸ ۱۴:۴۴

سلام/با تشکر از مطالب پایه ای و مفیدتون
کاربرد دو قطعه کد زیر در چه زمانی بهتر است؟

1.گرفتن مقدار مساحت به صورت یک Property

```
public double Area
{
        get { return _height * _baseLength * 0.5; }
}
```

2.محاسبه مساحت با استفاده از یک Method

```
public double Area()
{
      return _height * _baseLength * 0.5;
}
```

نویسنده: احمد
تاریخ: ۱۳۹۲/۰۲/۲۸ ۱۸:۳۳

Properties vs Methods

نویسنده: محسن نجف زاده
تاریخ: ۱۳۹۲/۰۲/۲۹ ۸:۳۸

احمد عزیز ممنون بابت لینک تون.
سوالم اینه که چرا آقای فرقانی تو این مثال برای Area هم یک Property تعریف کردن (چون صرفا فقط یک مثال | براساس تعریف Method و Property ، متد بهتر نبود؟)

نویسنده: محسن خان
تاریخ: ۱۳۹۲/۰۲/۲۹ ۸:۴۸

خلاصه لینک احمد: اگر محاسبات پیچیده و طولانی است، یا تاثیرات جانبی روی عملکرد سایر قسمت‌های کلاس دارند، بهتره از متد استفاده بشه. اگر کوتاه، سریع و یکی دو سطری است و ترتیب فراخوانی آن اهمیتی ندارد، فرقی نمی‌کنه و بهتره که خاصیت باشه و اگر این شرایط حاصل شد، عموم کاربران تازه کار استفاده از خواص را نسبت به متدها ساده‌تر می‌یابند و به نظر آن‌ها Syntax تمیزتری دارد (هدف این سری مقدماتی).

نویسنده: محسن نجف زاده
تاریخ: ۱۳۹۲/۰۲/۲۹ ۹:۱۳

باز هم به نظر من استفاده از method صحیح‌تر بود
درسته که در اینجا محاسبات پیچیده (computationally complex) نداریم اما چون یک action تلقی می‌شه پس ... .
(البته شاید این حساسیت بیجاست و به قول دوست عزیز 'محسن خان' چون این سری مقدماتی است به این صورت نوشته شده است)

نویسنده: آرمان فرقانی
تاریخ: ۱۳۹۲/۰۲/۳۰ ۱۹:۵

ضمن تشکر از دوستانی که در بحث شرکت کردند و پوزش به دلیل اینکه چند هفته ای در سفر هستم و تهیه مطالب با تأخیر انجام خواهد شد.
برای پاسخ به پرسش دوست گرامی آقای نجف زاده ابتدا بخشی از این مطلب را یادآوری می‌کنم.
"... مساحت را این بار به جای متد به صورت یک پروپرتی پیاده سازی می‌کنیم. اگرچه در آینده بیشتر راجع به چگونگی انتخاب برای پیاده سازی یک عضو کلاس به صورت پروپرتی یا متد بحث خواهیم کرد اما به عنوان یک قانون کلی در نظر داشته باشید عضوی که به صورت منطقی به عنوان داده مطرح است را به صورت پروپرتی پیاده سازی کنید. مانند نام دانشجو. از طرفی اعضایی که دلالت بر انجام عملی دارند را به صورت متد پیاده سازی می‌کنیم. مانند متد تبدیل به نوع داده دیگر. (مثلاً Object.ToString())
..."

بنابراین به نکات زیر توجه فرمایید.
۱. در این مطالب سعی شده است امکان پیاده سازی یک مفهوم به دو صورت متد و پروپرتی نشان داده شود تا در ذهن خواننده زمینه ای برای بررسی بیشتر مفهوم متد و پروپرتی و تفاوت آن‌ها فراهم گردد. این زمینه برای کنجکاوی بیشتر معمولاً با انجام یک جستجوی ساده سبب توسعه و تثبیت علم شخص می‌گردد.
۲. در متن بالاً به صورت کلی اشاره شده است هر یک از دو مفهوم متد و پروپرتی در کجا باید استفاده شوند و نیز خاطرنشان شده است در مطالب بعدی در مورد این موضوع بیشتر صحبت خواهد شد.
۳. نکته مهم در طراحی کلاس، پایگاه داده و ... خرد جهان واقع یا محیط عملیاتی مورد نظر طراح است. به عبارت دیگر گسی نمی‌تواند به یک طراح بگوید به طور مثال مساحت باید متد باشد یا باید پروپرتی باشد. طراح با توجه به مفهوم و کارکردی که برای هر مورد در ذهن دارد بر اساس اصول و قواعد، متد یا پروپرتی را بر می‌گزینند. مثلاً در خرد جهان واقع موجود در ذهن یک طراح مساحت به عنوان یک عمل یا اکشنی که شیء انجام می‌دهد است و بنابراین متد را انتخاب می‌کند. طراح دیگری در خرد جهان واقع دیگری در حال طراحی است و مثلاً متراژ یک شیء خانه را به عنوان یک ویژگی ذاتی و داده ای می‌نگرد و گمان می‌کند خانه نیازی به انجام عملی برای بدست آوردن مساحت خود ندارد بلکه یکی از ویژگی‌های خود را می‌تواند به اطلاع استفاده کننده برساند. پس شما به طراح دیگر نگویید اکشن تلقی میشه پس باید متد استفاده شود. اگر خود در پروژه ای چیزی را اکشن تلقی نمودید بله باید متد به کار ببرید. تلقی‌ها بر اساس خرد جهان واقع معنا دارند.
۴. پروپرتی و متد از نظر شیوه استفاده و ... با هم تفاوت دارند. اما یک تفاوت مهم بین آنها بیان نوع مفاهیم موجود در ذهن طراح به کد مشتری است. فراموش نکنید خود پروپرتی دارای اکسسور است که چیزی مانند متد است. در خیلی از موارد صحیح‌تر بودن پیاده سازی با متد یا با پروپرتی معنا ندارد. انتخاب ما بین متد یا پروپرتی بر اساس نحوه استفاده مطلوب در کد مشتری و نیز اطلاع به مشتری که مثلاً فلان مفهوم از دید ما یک اکشن است و فلان چیز داده صورت می‌گیرد.

نویسنده: محسن نجف زاده
تاریخ: ۱۳۹۲/۰۳/۲۹ ۸:۹

با تشکر از آرمان فرقانی عزیز / مطلب و بحث مفید بود.

عنوان: **ایجاد رشته Alphanumeric تصادفی در سی شارپ**
نویسنده: امیر هاشم زاده
تاریخ: ۱۳۹۲/۰۲/۲۰ ۱۹:۴۵
آدرس: www.dotnettips.info
برچسب‌ها: C#, .NET, Random, alphanumeric

برای ایجاد یک رشته تصادفی Alphanumeric (شامل حرف و عدد) روشهای زیادی وجود دارد ولی در اینجا به تشریح 2 روش آن اکتفا می‌کنیم.

روش کلی: ابتدا بازه رشته تصادفی مورد نظر را تعیین می‌کنیم. سپس به اندازه طول رشته، اندیس تصادفی ایجاد می‌کنیم و بوسیله آنها کاراکتر تصادفی را از بازه بدست می‌آورم و در انتها کاراکترهای تصادفی را با هم ادغام کرده تا رشته نهایی حاصل شود.
روش اول:

ابتدا بازه (char) رشته را مشخص می‌کنیم.

```
var chars = "ABCDEFGHIJKLMNOPQRSTUVWXYZabcdefghijklmnopqrstuvwxyz0123456789";
```

سپس بوسیله LINQ آن را به اندازه طول رشته دلخواه (در این مثال 8کاراکتر) تکرار می‌کنیم و برای انتخاب تصادفی یک کاراکتر در هر بازه (char) تکرار شده از کلاس جهت بدست آوردن اندیس تصادفی بازه استفاده می‌کنیم.

```
var random = new Random();
var result = new string(
Enumerable.Repeat(chars, 8)
          .Select(s => s[random.Next(s.Length)])
          .ToArray());
```

توجه: از این روش برای هیچ کدام از موارد مهم و کلیدی مانند ساخت کلمه عبور و توکن استفاده نکنید.
روش دوم:

همانند روش اول ابتدا بازه رشته را تعیین می‌کنیم.

```
char[] chars = new char[62];
chars="abcdefghijklmnopqrstuvwxyzABCDEFGHIJKLMNOPQRSTUVWXYZ1234567890".ToCharArray();
```

بعد از تعریف بازه، یک سری اعداد تصادفی غیر صفر را بوسیله کلاس RNGCryptoServiceProvider و متد GetNonZeroBytes آن، در متغیری که قرار است بعدا در ایجاد رشته‌ی تصادفی نیاز است پر می‌کنیم.

```
byte[] data = new byte[maxSize];
RNGCryptoServiceProvider crypto = new RNGCryptoServiceProvider();
crypto.GetNonZeroBytes(data);
```

در این مرحله به تعداد طول رشته تصادفی مورد نظر عدد تصادفی بین 0 تا 255 ذخیره شده در متغیر data داریم، برای ایجاد اندیس تصادفی از باقیمانده عدد تصادفی ایجاد شده در مرحله قبل (byte) به طول بازه (chars.Length) استفاده می‌کنیم سپس کاراکترهای تصادفی را کنار یکدیگر قرار می‌دهیم.

```
StringBuilder result = new StringBuilder(maxSize);
foreach (byte b in data)
{
  result.Append(chars[b % (chars.Length)]);
}
```

و در نهایت متد ما جهت ایجاد رشته Alphanumeric در روش دوم به شکل زیر خواهد بود:

```
public static string GetRandomAlphaNumeric (int maxSize)
{
      char[] chars = new char[62];
      chars ="abcdefghijklmnopqrstuvwxyzABCDEFGHIJKLMNOPQRSTUVWXYZ1234567890".ToCharArray();
      RNGCryptoServiceProvider crypto = new RNGCryptoServiceProvider();
      byte[] data = new byte[maxSize];
      crypto.GetNonZeroBytes(data);
      StringBuilder result = new StringBuilder(maxSize);
      foreach (byte b in data)
      {
           result.Append(chars[b % (chars.Length)]);
      }
      return result.ToString();
}
```

لازم به یادآوری است که رشته ایجاد شده در 2روش بیان شده **منحصر بفرد** نیست بلکه **تصادفی** است، درواقع تصادفی بودن با منحصر بودن متفاوت است، برای ایجاد رشته‌های منحصر بفرد روشهایی (البته این روشها 100 درصد نیستند ولی از قابلیت اطمینان بالایی برخوردار هستند) وجود دارد که پست‌های بعدی به آنها اشاره خواهم کرد.

عنوان: **چگونگی رسیدگی به Null property در AutoMapper**
نویسنده: مهراد
تاریخ: ۱۳۹۲/۰۲/۲۲ ۱۷:۴۰
آدرس: www.dotnettips.info
گروه‌ها: C#, MVC, Mapper, AutoMapper

AutoMapper کتابخانه‌ای برای نگاشت اطلاعات یک شیء به شیءایی دیگر به صورت خودکار می‌باشد.

در این مقاله چگونگی رسیدگی به Null property را در AutoMapper بررسی خواهیم کرد. فرض کنید شیء منبع دارای یک خاصیت Null است و می‌خواهید به وسیله Automaper شیء منبع را به مقصد نگاشت نمایید. اما می‌خواهید در صورت Null بودن شیء مبدا، یک مقدار پیش فرض برای شیء مقصد در نظر گرفته شود .

برای نمونه کلاس user را که در آن از کلاس Address یک خاصیت تعریف شده، در نظر بگیرید . اگر مقدار آدرس در شیء منبع خالی بود شاید شما بخواهید مقدار آن را به صورت empty string و یا با یک مقدار پیش فرض در مقصد مقدار دهی کنید.

همانند مثال زیر :

```
public class UserSource
{
  public Address Address{get;set;}
}

public class UserDestination
{
  public string Address{get;set;}
}
```

ابتدا نگاشت‌ها را تعریف می‌کنیم:

```
AutoMapper.Mapper.CreateMap<UserSource, UserDestination>()
          .ForMember(dest => dest.Address
          , opt => opt.NullSubstitute("Address not found")
          );
```

کد بالا نشان دهنده تبدیل Address به Address است ولی دارای متد اختیاری NullSubstitute می‌باشد و بیانگر این است که اگر آدرس شیء منبع Null بود، مقدار پیش فرضی را برای شیء مقصد در نظر بگیرد. در انتها می‌توان نگاشت را در برنامه متناسب با نیاز خود انجام داد:

```
var model = AutoMapper.Mapper.Map<UserSource, UserDestination>(user);
var models = AutoMapper.Mapper.Map<IEnumerable<UserSource>, IEnumerable<UserDestination>>(users);
```

نظرات خوانندگان

نویسنده: daneshjoo
تاریخ: ۱۳۹۲/۰۲/۲۳ ۲۰:۴۱

با سلام

مهندس من یه دیتابیس دارم که حاوی اطلاعات است در جداول اون در تمام ستون‌ها به غیر از ستون کلید اومده تیک alow null رو فعال کرده یعنی این ستون‌ها می‌تونه مقدار null رو بگیره .

حالا اون برنامه که این اطلاعات رو وارد دیتابیس کرده اومده هر ستونی که نوعش رشته بوده مقدار empty وارد کرده نه null و ستون هایی که نوعشون int هست مقدار صفر وارد کرده , مثل همین مطلبی که شما گفتید اما به صورت سنتی .

به نظر شما من باید همین رویه رو با روش شما انجام بدم یا نه همون مقدار null و در دیتابیس ذخیره کنم ؟

نویسنده: MehRad
تاریخ: ۱۳۹۲/۰۲/۲۳ ۲۱:۴۶

سلام
بستگی به کار خودتون داره . مطلب بالا مربوط به نگاشت اطلاعات یک شیء به شیءایی دیگره .اما شما برای در نظر گرفتن مقدار پیش فرض در دیتابیس همون طور که میدونید با تنظیم Default Value or Binding میتونی مقدار پیش فرضی برای ستون‌های Null در نظر بگیری.

نویسنده: ناصر پورعلی
تاریخ: ۱۳۹۳/۰۳/۲۱ ۱۶:۴۴

سلام
من از Automapper واسه مپ کردن مدل و ویو مادل‌ها تو برنامه ام استفاده میکنم
مشکلم اینجاست که در کلاس model یه فیلد از نوع byte[] دارم

```
public class News : BaseEntity
    {
public byte[] SmallImage { get; set; } // SmallImage
     }
```

در viewModel هم دقیقا همین فیلد رو دارم.
منتهی موقع مپ کردن این فیلد null میشه.

```
news = model.ToEntity(news);
```

```
public static News ToEntity(this NewsModel model, News destination)
        {
            Mapper.CreateMap<NewsModel, News>();
            return Mapper.Map(model, destination);
        }
```

علتش رو نمیدونم، کسی هست بدونه؟
باتشکر

نویسنده: علیرضا پایدار
تاریخ: ۱۳۹۳/۰۳/۲۲ ۹:۱۵

```
public class TestModel
    {
        public byte[] MProperty { get; set; }
    }
```

```
public class TestViewModel
    {
       public byte[] VMProperty { get; set; }
    }
```

```
class Program
    {
        static void Main(string[] args)
        {

Mapper.CreateMap<TestModel,TestViewModel>().ForMember(tv=>tv.VMProperty,m=>m.MapFrom(t=>t.MProperty));
            TestModel tm = new TestModel();
            tm.MProperty = new byte[]  { 1, 2, 3 ,4};

            TestViewModel tvm = Mapper.Map<TestModel, TestViewModel>(tm);
            foreach (var item in tvm.VMProperty)
            {
                Console.WriteLine(item.ToString());
            }
            Console.ReadKey();
        }
    }
```

در صورتی که MProperty، مقدار داشته باشه مشکلی پیش نمیاد اگر هم null باشه بازم مشکلی پیش نمیاد و VMProperty برابر با null هست.

عنوان: **intern pool جدول نگهداری رشته‌ها در دات‌نت**
نویسنده: رحمت اله رضایی
تاریخ: ۱۳۹۲/۰۲/۲۹ ۲۲:۳۰
آدرس: www.dotnettips.info
برچسب‌ها: C#, Tips, CLR, string, intern, pool

کد زیر را در نظر بگیرید :

```
object text1 = "test";
object text2 = "test";

object num1 = 1;
object num2 = 1;

Console.WriteLine("text1 == text2 : " + (text1 == text2));
Console.WriteLine("num1 == num2 : " + (num1 == num2));
```

به نظر شما چه چیزی در خروجی نمایش داده میشود؟

هر چهار متغییر text1 و text2 و num1 و num2 از نوع object هستند. با اینکه مقدار text1 و text2 یکی و مقدار num1 و num2 هم یکی است، نتیجه text1==text2 برابر true است اما num1==num2 برابر false.

خطی که text2 تعریف شده است را تغییر میدهیم :

```
object text2 = "test".ToLower();
```

اینبار با این که باز مقدار text1 و text2 یکی و هر دو "test" است، اما نتیجه text1==text2 برابر false است. انتظار ما هم همین است. دو object ایجاد شده است و یکی نیستند. تنها در صورتی باید نتیجه == آنها true باشد که هر دو به یک object اشاره کنند.

اما چرا در کد اولی اینگونه نبود؟

دلیل این کار برمیگردد به رفتار دات‌نت نسبت به رشتههایی که به صورت صریح در برنامه تعریف میشوند. CLR یک جدول برای ذخیره رشتهها به نام **intern pool** برای برنامه میسازد. هر رشتهای تعریف میشود، اگر در intern pool رشتهای با همان مقدار وجود نداشته باشد، یک رشته جدید ایجاد و به جدول اضافه میشود، و اگر موجود باشد متغییر جدید فقط به آن اشاره میکند. در واقع اگر 100 جای برنامه حتی در کلاسهای مختلف، رشتههایی با مقادیر یکسان وجود داشته باشند، برای همه آنها یک نمونه وجود دارد.

بنابراین text1 و text2 در کد اولی واقعا یکی هستند و یک نمونه برای آنها ایجاد شده است.

البته چند نکته در اینجا هست :
اگر text1 و text2 به صورت string تعریف شوند، نتیجه text1==text2 در هر دو حالت فوق برابر true است. چون عملگر == در کلاس string یکبار دیگر overload شده است:

```
public sealed class String : ...
{
    ...
    public static bool operator ==(string a, string b)
    {
```

```
        return string.Equals(a, b);
    }
   ...
}
```

این که کدام یک از overloadها اجرا شوند (کلاس پایه، کلاس اصلی، ...) به نوع دو متغییر اطراف == بستگی دارد. مثلا در کد زیر :

```
string text1 = "test";
string text2 = "test".ToLower();

Console.WriteLine("text1 == text2 (string) : " + (text1 == text2));
Console.WriteLine("text1 == text2 (object) : " + ((object)text1 == (object)text2));
```

اولین نتیجه true و دومی false است. چون در اولی عملگر == تعریف شده در کلاس string مورد استفاده قرار می‌گیرد اما در دومی عملگر == تعریف شده در کلاس object.

اگر دقت نشود این رفتار مشکلزا میشود. مثلا حالتی را در نظر بگیرید که text1 ورودی کاربر است و text2 از بانک اطلاعاتی خوانده شده است و با اینکه مقادیر یکسان دارند نتیجه == آنها false است. اگر تعریف عملگرها در کلاس object به صورت virtual بود و در کلاس‌های دیگر override می‌شد، این تغییر نوع‌ها تاثیری نداشت. اما عملگرها به صورت static تعریف می‌شوند و امکان override شدن ندارند. به همین خاطر کلاس object متدی به اسم Equals در اختیار گذاشته که کلاس‌ها آنرا override می‌کنند و معمولا از این متد برای سنجش برابری دو کلاس استفاده می‌شود :

```
object text1 = "test";
object text2 = "test".ToLower();

Console.WriteLine("text1 Equals text2 : " + text1.Equals(text2));
Console.WriteLine("text1 Equals text2 : " + object.Equals(text1, text2));
```

البته یادآور می‌شوم که **فقط رشته‌هایی که به صورت صریح در برنامه تعریف شده‌اند، در intern pool قرار می‌گیرند و این فهرست شامل رشته‌هایی که از فایل یا بانک اطلاعاتی خوانده می‌شوند یا در برنامه تولید می‌شوند، نیست.** این کار منطقی است وگرنه حافظه زیادی مصرف خواهد شد.

با استفاده از متد string.Intern می‌توان یک رشته را که در intern pool وجود ندارد، به فهرست آن افزود. اگر رشته در intern pool وجود داشته باشد، reference آنرا بر می‌گرداند در غیر اینصورت یک reference به رشته جدید به intern pool اضافه می‌کند و آنرا برمی‌گرداند.

یک مورد استفاده آن هنگام lock روی رشته‌هاست. برای مثال در کد زیر DeviceId یک رشته است که از بانک اطلاعاتی خوانده می‌شود و باعث می‌شود که چند job همزمان به یک دستگاه وصل نشوند :

```
lock (job.DeviceId)
{
    job.Execute();
}
```

اگر یک job با DeviceId برابر COM1 در حال اجرا باشد، این lock جلوی اجرای همزمان job دیگری با همین DeviceId را

نمی‌گیرد. زیرا هر چند مقدار DeviceId دو job یکی است ولی به یک نمونه اشاره نمی‌کنند.

می‌توان lock را اینگونه اصلاح کرد :

```
lock (string.Intern(job.DeviceId))
{
    job.Execute();
}
```

نظرات خوانندگان

نویسنده: محسن خان
تاریخ: ۱۳۹۲/۰۲/۳۰ ۰:۳۵

ممنون. البته شرایط کد خودتون رو کامل اینجا قرار ندادید ولی در حالت کلی توصیه میشه که برای استفاده از lock یک شیء private object در سطح کلاس تعریف بشه و از اون استفاده بشه تا حالت‌های دیگر .

نویسنده: رحمت اله رضایی
تاریخ: ۱۳۹۲/۰۲/۳۱ ۹:۵۶

- البته این فقط یک مثال بود برای درک متد string.Intern .
- چگونگی شی معرفی شده به lock هم بسته به شرایط ممکن است متفاوت باشد. ممکن است یک private object در سطح همان کلاسی که lock در آن استفاده می‌شود، جوابگو باشد. اما در شرایط دیگری ممکن است اینگونه نباشد. مانند مثال فوق.

عنوان: **انتخاب پویای فیلد ها در LINQ**
نویسنده: شاهین کیاست
تاریخ: ۱۳۹۲/۰۳/۰۳ ۸:۳۰
آدرس: www.dotnettips.info
گروه‌ها: C#, LINQ

LINQ یک DLS بر مبنای .NET می باشد که برای پرس و جو در منابع داده ای مانند پایگاه‌های داده ، فایل‌های XML و یا لیستی از اشیاء درون حافظه کاربرد دارد.

یکی از بزرگترین مزیت‌های آن Syntax آسان و خوانا آن می‌باشد.

LINQ از 2 نوع نمادگذاری پشتیبانی می‌کند:

Inline LINQ یا query expressions :

```
var result =
    from product in dbContext.Products
    where product.Category.Name == "Toys"
    where product.Price >= 2.50
    select product.Name;
```

Fluent Syntax :

```
var result = dbContext.Products
    .Where(p => p.Category.Name == "Toys" && p.Price >= 250)
    .Select(p => p.Name);
```

در پرس و چوهای بالا فیلدهای مورد نیاز در قسمت Select در زمان Compile شناخته شده هستند . اما گاهی ممکن است فیلدهای مورد نیاز در زمان اجرا مشخص شوند.

به عنوان مثال یک گزارش ساز پویا که کاربر مشخص می‌کند چه ستون هایی در خروجی نمایش داده شوند یا یک جستجوی پیشرفته که ستون‌های خروجی به اختیار کاربر در زمان اجرا مشخص می‌شوند.

این مدل را در نظر داشته باشید :

```
public class Student
    {
        public int Id { get; set; }
        public string Name { get; set; }
        public string Field1 { get; set; }
```

```
        public string Field2 { get; set; }
        public string Field3 { get; set; }


        public static IEnumerable<Student> GetStudentSource()
        {
            for (int i = 0; i < 10; i++)
            {
                yield return new Student
                                {
                                    Id = i,
                                    Name = "Name " + i,
                                    Field1 = "Field1 " + i,
                                    Field2 = "Field2 " + i,
                                    Field3 = "Field3 " + i
                                };
            }
        }
    }
```

ستون‌های کلاس Student را در رابط کاربری برنامه جهت انتخاب به کاربر نمایش می‌دهیم. سپس کاربر یک یا چند ستون را انتخاب می‌کند که قسمت Select کوئری برنامه باید بر اساس فیلدهای مورد نظر کاربر مشخص شود.

یکی از روش هایی که می‌توان از آن بهره برد استفاده از کتاب خانه Dynamic LINQ معرفی شده در اینجا می باشد.

این کتابخانه جهت سهولت در نصب به کمک NuGet در این آدرس قرار دارد.

فرض بر این است که فیلدهای انتخاب شده توسط کاربر با " , " از یکدیگر جدا شده اند.

```
public class Program
    {
        private static void Main(string[] args)
        {
            System.Console.WriteLine("Specify the desired fields : ");
            string fields = System.Console.ReadLine();
            IEnumerable<Student> students = Student.GetStudentSource();
            IQueryable output = students.AsQueryable().Select(string.Format("new({0})", fields));
            foreach (object item in output)
            {
                System.Console.WriteLine(item);
            }

            System.Console.ReadKey();
        }

    }
```

همانطور که در عکس ذیل مشاهده می‌کنید پس از اجرای برنامه ، فیلدهای انتخاب شده توسط کاربر از منبع داده‌ی دریافت شده و در خروجی نمایش داده شده اند.

```
Specify the desired fields :

Field1,Field2,Id,Name

{Field1=Field1 0, Field2=Field2 0, Id=0, Name=Name 0}
{Field1=Field1 1, Field2=Field2 1, Id=1, Name=Name 1}
{Field1=Field1 2, Field2=Field2 2, Id=2, Name=Name 2}
{Field1=Field1 3, Field2=Field2 3, Id=3, Name=Name 3}
{Field1=Field1 4, Field2=Field2 4, Id=4, Name=Name 4}
{Field1=Field1 5, Field2=Field2 5, Id=5, Name=Name 5}
{Field1=Field1 6, Field2=Field2 6, Id=6, Name=Name 6}
{Field1=Field1 7, Field2=Field2 7, Id=7, Name=Name 7}
{Field1=Field1 8, Field2=Field2 8, Id=8, Name=Name 8}
{Field1=Field1 9, Field2=Field2 9, Id=9, Name=Name 9}
```

این روش مزایا و معایب خودش را دارد ، به عنوان مثال خروجی یک لیست از شیء Student نیست یا این Select فقط برای روی یک شیء IQueryable قابل انجام است.

روش دیگری که می‌توان از آن بهره جست استفاده از یک متد کمکی جهت تولید پویای عبارت Lambda ورودی Select می باشد :

```
public  class SelectBuilder <T>
    {
        public static Func<T, T> CreateNewStatement(string fields)
        {
            // input parameter "o"
            var xParameter = Expression.Parameter(typeof(T), "o");


            // new statement "new T()"
            var xNew = Expression.New(typeof(T));

            // create initializers
            var bindings = fields.Split(',').Select(o => o.Trim())
                .Select(o =>
                {

                    // property "Field1"
                    var property = typeof(T).GetProperty(o);

                    // original value "o.Field1"
                    var xOriginal = Expression.Property(xParameter, property);

                    // set value "Field1 = o.Field1"
                    return Expression.Bind(property, xOriginal);
                }
            ).ToList();

            // initialization "new T { Field1 = o.Field1, Field2 = o.Field2 }"
            var xInit = Expression.MemberInit(xNew, bindings);

            // expression "o => new T { Field1 = o.Field1, Field2 = o.Field2 }"
            var lambda = Expression.Lambda<Func<T, T>>(xInit, xParameter);

            // compile to Func<T, T>
            return lambda.Compile();
        }
    }
```

برای استفاده از متد CreateNewStatement باید اینگونه عمل کرد :

```
IEnumerable<Student> result = students.Select(SelectBuilder<Student>.CreateNewStatement("Field1, Field2")).ToList();

            foreach (Student student in result)
            {
                System.Console.WriteLine(student.Field1);
            }
```

خروجی یک لیست از Student می باشد.

نحوه‌ی کارکرد CreateNewStatement :

ابتدا فیلدهای انتخابی کاربر که با "," جدا شده اند به ورودی پاس داده می‌شود سپس یک statement خالی ایجاد می‌شود :

```
o=>new Student()
```

فیلدهای ورودی از یکدیگر تفکیک می‌شوند و به کمک Reflection پراپرتی معادل فیلد رشته ای در کلاس Student پیدا می‌شود :

```
var property = typeof(T).GetProperty(o);
```

سپس عبارت Select و تولید شیء جدید بر اساس فیلدهای ورودی تولید می‌شود و برای استفاده Compile به Func می‌شود. در نهایت Func تولید شده به Select پاس داده می‌شود و لیستی از Student بر مبنای فیلدهای انتخابی تولید می‌شود.

```
// initialization "new T { Field1 = o.Field1, Field2 = o.Field2 }"
var xInit = Expression.MemberInit(xNew, bindings);

// expression "o => new T { Field1 = o.Field1, Field2 = o.Field2 }"
var lambda = Expression.Lambda<Func<T, T>>(xInit, xParameter);
lambda {o => new Student() {Field1 = o.Field1}}

// compile to Func<T, T>
return lambda.Compile();
```

دریافت مثال : DynamicSelect.zip

نظرات خوانندگان

نویسنده: sorosh
تاریخ: ۱۳۹۲/۱۱/۱۲ ۷:۴۴

با سلام؛ با ایجاد ستون ردیف با Select new در LINQ مشکل دارم. طوریکه بصورت اتوماتیک یک ستون ردیف ایجاد نماییم:

```
var all = (from x in db.tblZones
select new
{
  RowNuber=???????????
  Code = x.xCode,
  Caption = x.xCaption,
  Comment = x.xComments,
  DT_RowId = "tr" + x.xCode.ToString(),

});
```

نویسنده: محسن خان
تاریخ: ۱۳۹۲/۱۱/۱۲ ۱۱:۳۲

یک متغیر count قبل از عبارتی که نوشتی ایجاد کن. اینبار جلوی RowNumber بنویس count++.

نویسنده: شاهین کیاست
تاریخ: ۱۳۹۲/۱۱/۱۲ ۱۱:۴۰

متد Select یک Overload دیگر دارد که Index را فراهم می‌کند :

```
string[] weekDays = { "Monday", "Tuesday", "Wednesday", "Thursday", "Friday" };
weekDays.Select((day, index) => new { Day = day, Index = index })
        .Where(x => x.Day.Contains("s"))
        .Select(x => x.Index)
        .ToArray();
```

البته در این کد از Lambda Syntax استفاده شده که برای کد شما هم ممکن است.

عنوان: **مقایسه بین حلقه های تکرار (Lambda ForEach و for و foreach)**
نویسنده: مسعود پاکدل
تاریخ: ۱۳۹۲/۰۳/۰۵ ۲۳:۴۰
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] );
}
```

#2حلقه 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

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

بررسی نتایج :

سریع‌ترین 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 3برابر سریع‌تر از دسترسی به متد get خاصیت هاست.

مهم‌ترین نکته این است که Lambda ForEach عمکردی 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 می‌کنه.دلیلش هم اینه که کامپایلر هنگام کار با آرایه‌ها و اجرای اون‌ها به صورت حلقه، کد IL خاصی رو تولید می‌کنه که مخصوص کار با آرایه هاست و سرعت اون به مراتب از سرعت کد IL تولید شده برای IEnumeratorها پایین تره.

نظرات خوانندگان

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

خیلی برام جالب بود، ممنون از مطلبت

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

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

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

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: ۱۳۹۲/۰۳/۰۷ ۲۳:۳

جالب بود که این روش از foreach سریعتر عمل میکنه

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

یا استفاده از Microbenchmark برای دریافت نتایج دقیقتر.

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

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 درنتیجه اجرای دستورات بیشتر در هر حلقه کندتر عمل میکند.
اما! اگر هدف تنها بررسی سرعت اجرای حلقه‌های اشاره شده باشه متدهای بالا نتیجه درستی نشان نخواهد داد، چون عملیات انجام شده در حلقه‌های نشان داده شده با هم دقیقا یکسان نیست. بهتره که یه عملیات ثابت و مستقل از متغیرهای درگیر استفاده بشه تا نتایج دقیقتری بدست بیاد. مثلا یه 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 کاملا قابل صرفنظره!
نتایج برای حالات مختلف موجود تفاوتهای زیادی داشت اما درنسخه ریلیز نهایتا نتایج کلی این بود که حلقه for معمولی از همه سریعتر، سپس Array.ForEach و بعد متد ForEach در کلاس List و درنهایت ازهمه 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. متاسفانه تعداد تکرار بیشتر خطای OutOfMemory میداد.
نکته: اجرای تستهای این چنینی 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 (دیباگ یا ریلیز) و محل اجرای تست (منظور اجرا در محیط دیباگ ویژوال استودیو یا اجرای مستقل برنامه) و ... . (همونطور که آقای نصیری هم مطلبی مرتبط رو به اشتراک گذاشتند ^)

نویسنده: مسعود م.پاکدل
تاریخ: ۱۳۹۲/۰۳/۰۸ ۱۴:۲

درابتدا بهتر عنوان کنم که در کل 2 نوع برنامه نویس وجود داره. برنامه نویسی که می‌خواد برنامه درست کار کنه و برنامه نویسی که میخواد برنامه درست نوشته بشه. در این جا هدف اصلی ما نوشتن برنامه به صورت درست هستش. دلیل اینکه foreach کندتر از Lamba 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 می‌بینید از دو دستور enumerator.MoveNext و enumerator.Current در هر تکرار داره استفاده می‌شه 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 آرایه‌های یک بعدی استفاده میشه و کامپایلر هنگام کار با آرایه‌ها کد IEnumerator رو که در بالا توضیح دادم تولید نمی‌کنه در نتیجه در حلقه foreach برای آرایه‌ها هیچ فراخوانی متدی صورت نمی‌گیرد 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 شهروز 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 وقتیکه نتایج دقیق (و نه کلی) نشون داده میشه. مثل همین نتایج دقیق زمانهای اجرای حلقه‌ها.
نکته ای که من درکامنتم اشاره کردم صرفا درباره تست "سرعت اجرای" انواع حلقه‌ها بود که ممکنه با تست کارایی حلقه‌ها در اجرای یک کد خاص فرق داشته باشه.
نکته دیگه هم اینکه نمیدونم که آیا شما از همون متد Console.WriteLine در حلقه‌ها 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 تست‌ها اصلا مناسب نیست و باعث دور شدن زیاد نتایج از واقعیت میشه. مثلا من تست کردم و هر دفعه یه نتیجه‌ای می‌داد که نمیشه بر اساس اون نتیجه‌گیری کرد.

مورد دیگه ای هم که باید اضافه کنم اینه که بهتر بود شما کد کامل تست خودتون رو هم برای دانلود 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.Write(description);
  Console.WriteLine(" Time Elapsed {0} ms", watch.ElapsedMilliseconds);
}
```

قبل از اجرای تست بهتره برنامه رو برای نسخه Release بیلد کنیم. ساده‌ترین 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.

نویسنده: محسن خان

تاریخ: ۱۳۹۲/۰۳/۱۲ ۱۳:۳۸

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: ۱۳۹۲/۰۳/۱۲ ۱۳:۴۷

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: ۱۳۹۲/۰۳/۱۲ ۱۴:۰

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

عنوان: **نکاتی درباره پرس و جو با استفاده از پردازش موازی**
نویسنده: سیدمجتبی حسینی
تاریخ: ۱۳۹۲/۰۳/۱۰ ۱۷:۵
آدرس: www.dotnettips.info
گروه‌ها: C#, Parallel Processing, PLINQ, LINQ

برای انجام عملیات پرس و جوی LINQ با استفاده از روش پردازش موازی به راحتی میتوان الحاقیه AsParallel را به هر داده‌ای از نوع IEnumerable<T> > افزود:

```
var data =
new int[] { 0, 1, 2, 3, 4, 5, 6, 7, 8, 9, 10 };
// پرس و جوی عادی
var q1 = from i in data select i;
// پرس و جو به شیوه موازی
var q2 = from i in data.AsParallel() select i;
```

الحاقیه .AsParallel() در پرس و جوی q2 نسخه موازی LINQ را بر روی متغیر data اجرا میکند و اگر همه چیز به صورت صحیح انجام شود هر دو پرس و جو باید نتایج یکسانی داشته باشند، اما نتایج عبارتند از :

```
//نتیجه اجرای q1
0 1 2 3 4 5 6 7 8 9 10
//نتیجه اجرای q2
0 6 1 7 2 8 3 9 4 10 5
```

همانطور که ملاحظه میکنید ترتیب واقعی نتایج اجرای پرس و جوها با یکدیگر متفاوت‌اند و نکته جالبتر آنکه با هر بار اجرای برنامه نتیجه اجرای پرس و جوی q2 با نتیجه سری قبل خودش متفاوت است که این تفاوت به چگونگی تقسیم بندی انجام کار میان هسته‌های سی پی یو، بستگی دارد. نکته بسیار مهم آن است که عملیات پردازش موازی خود را ملزم به حفظ ترتیب داده‌ها نمی‌داند مگر آنکه مجبورش کنیم و این رفتار پردازش موازی به دلیل بالا بردن راندمان عملیات است در نتیجه انجام پرس و جوهای موازی توسط الحاقیه .AsParallel() خیلی هم ساده نیست و ممکن است منجر به تولید نتایج ناخواسته شود.

حال اگر چگونگی ترتیب داده‌ها، برایمان مهم است به دو روش می‌توانیم آن را انجام دهیم:

1- افزودن عبارت orderby به پرس و جو

2- استفاده از الحاقیه AsOrdered

```
var q3 = from i in data.AsParallel() orderby i  select i;

var q4 = from i in data.AsParallel().AsOrdered() select i;
```

که نتیجه انجام هر دو پرس و جوی بالا یکی خواهد بود. حال مسأله دیگر این است که آیا همیشه استفاده از پردازش موازی مفید خواهد بود یا خیر؟ پاسخ این سؤال وابسته است به نوع مسأله و حجم داده مورد نظر و مشخصات سیستمی که قرار است از آن کد استفاده کند. چگونگی اندازه سرعت و مقدار مصرف حافظه در اجرای چهار پرس و جوی فوق در کامپیوتر من با پردازنده Intel Q9550 به شکل زیر است:

file:///c:/users/m/documents/visual studio 2012/Projects/TestAsParallel/TestAsP...
q1-> Elapsed time: 00:00:00.0068941, Memory Usage: 796.00 KB
q2-> Elapsed time: 00:00:00.0053858, Memory Usage: 632.00 KB
q3-> Elapsed time: 00:00:00.0034320, Memory Usage: 272.00 KB
q4-> Elapsed time: 00:00:00.0023025, Memory Usage: 148.00 KB

نظرات خوانندگان

نویسنده: رضا
تاریخ: ۱۳۹۳/۰۷/۱۶ ۱۰:۲۹

سلام میشه لطفا دستوری که میزان حافظه و زمان پرس و جوهای بالا را برمی گرداند را در سایت قرار دهید

نویسنده: شاهین کیاست
تاریخ: ۱۳۹۳/۰۷/۱۶ ۱۲:۳۶

کلاس مورد نظر در این مقاله قرار دارد

```
public class PerformanceHelper
    {
        public static string RunActionMeasurePerformance(Action action)
        {
            GC.Collect();
            long initMemUsage = Process.GetCurrentProcess().WorkingSet64;

            var stopwatch = new Stopwatch();
            stopwatch.Start();

            action();

            stopwatch.Stop();

            var currentMemUsage = Process.GetCurrentProcess().WorkingSet64;
            var memUsage = currentMemUsage - initMemUsage;
            if (memUsage < 0) memUsage = 0;

            return string.Format("Elapsed time: {0}, Memory Usage: {1:N2} KB", stopwatch.Elapsed,
memUsage / 1024);
        }
    }
```

عنوان: **چگونگی دسترسی به فیلد و خاصیت غیر عمومی**
نویسنده: مسعود پاکدل
تاریخ: ۱۳۹۲/۰۳/۱۵ ۱۶:۱۰
آدرس: www.dotnettips.info
گروه‌ها: C#, Reflection, read-only

یک از ابتدایی‌ترین مواردی که در یادگیری دات نت آموزش داده می‌شود مباحث مربوط به کپسوله سازی است. برای مثال فیلدها و خواص Private که به صورت خصوصی هستند یا Protected هستند از خارج کلاس قابل دسترسی نیستند. برای دسترسی به این کلاس‌ها باید از خواص یا متدهای عمومی استفاده کرد.

```
public class Book
    {
        private int code = 10;

        public int GetCode()
        {
            return code;
        }
    }
```

یا فیلدها و خواصی که به صورت فقط خواندنی هستند،(RealOnly) امکان تغییر مقدار برای اون‌ها وجود ندارد. برای مثال کد پایین کامپایل نخواهد شد.

```
public class Book
    {
        private readonly int code = 10;

        public int GetCode()
        {
            return code = 20;
        }
    }
```

اما در دات نت با استفاده از Reflectionها می‌تونیم تمام قوانین بالا رو نادیده بگیریم. یعنی می‌تونیم هم به خواص و فیلدهای غیر عمومی کلاس دسترسی پیدا کنیم و هم می‌تونیم مقدار فیلدهای فقط خواندنی رو تغییر بدیم. به مثال‌های زیر دقت کنید.
#مثال اول

```
using System.Reflection;

 public class Book
 {
        private int code = 10;
 }

 public class Program
 {
        static void Main( string[] args )
        {
            Book book = new Book();
            var codeField = book.GetType().GetField( "code", BindingFlags.NonPublic |
BindingFlags.Instance );
            codeField.SetValue( book, 20 );
            var value = codeField.GetValue( book );
        }
    }
```

ابتدا یک کلاس که دارای یک متغیر به نام کد است ساخته ایم که مقدار 10 را دارد. فیلد به صورت private است. بعد از اجرا به راحتی مقدار Code را به دست می‌آوریم.

```
class Program
{
   static void Main( string[] args )
   {
      Book book = new Book();
      var codeField = book.GetType().GetField( "code", BindingFlags.NonPublic | BindingFlags.Instance );
      var value = codeField.GetValue( book );
   }          value 10
}
```

حتی امکان تغییر مقدار فیلد private هم امکان پذیر است.

```
class Program
{
   static void Main( string[] args )
   {
      Book book = new Book();
      var codeField = book.GetType().GetField( "code", BindingFlags.NonPublic | BindingFlags.Instance );
      codeField.SetValue( book, 100 );
      var value = codeField.GetValue( book );
   }          value 100
}
```

#مثال دوم.

در این مثال قصد داریم مقدار یک فیلد، از نوع فقط خواندنی رو تغییر دهیم.

```
using System.Reflection;

 public class Book
 {
        private readonly int code = 10;
 }

 public class Program
 {
        static void Main( string[] args )
        {
            Book book = new Book();
            var codeField = book.GetType().GetField( "code", BindingFlags.NonPublic |
BindingFlags.Instance );
            codeField.SetValue( book, 50);
            var value = codeField.GetValue( book );
        }
    }
```

بعد از اجرا مقدار متغیر code به 50 تغییر می‌یابد.

```
Book book = new Book();
var codeField = book.GetType().GetField( "code
codeField.SetValue( book, 50 );
var value = codeField.GetValue( book );
}
```

value 50

مطالب تکمیلی

نظرات خوانندگان

نویسنده: وحید نصیری
تاریخ: ۱۳۹۲/۰۳/۱۷ ۱۰:۱۰

البته مباحث Reflection، تابع سطح دسترسی کد فراخوان است (همان لینک آخر بحث جهت تاکید بیشتر و همچنین تنویر مقدمه):

« Security Considerations for Reflection »

برای نمونه در حالت medium trust، گزینه ReflectionPermission غیرفعال است.

برای آزمایش این مسایل می‌شود از دو برنامه Permview و Permcalc استفاده کرد.

نویسنده: سالار خلیل زاده
تاریخ: ۱۳۹۲/۰۳/۱۸ ۹:۱۸

ReflectionMagic جهت همین کار طراحی شده

http://nuget.org/packages/ReflectionMagic

نویسنده: reza
تاریخ: ۱۳۹۳/۰۵/۲۵ ۱۷:۹

آیا می‌توان به کمک رفلکشن به خصوصیتی که مثلا Set ندارد مقدار دهی کرد. بعنوان مثال

```
class Program
{
  static void Main(string[] args)
  {
      Book book = new Book();
      var codeprop = book.GetType().GetProperty("Code", BindingFlags.Public | BindingFlags.Instance);
      codeprop.SetValue(book, 20, null);
      var value = codeprop.GetValue(book, null);
  }
}

public class Book
{
  public int code;
  public int Code
  {
      get { return code; }
  }
}
```

نویسنده: مسعود پاکدل
تاریخ: ۱۳۹۳/۰۵/۲۶ ۱۴:۰

خیر! با یک ArgumentException و پیغام Property set method not found مواجه خواهید شد. اما در مثال بالا می‌توان مقدار فیلد code را تغییر داد که در نتیجه خاصیت Code نیز مقدار جدید را برگشت می‌دهد.

عنوان: **سرعت واکشی اطلاعات در List و Dictionary**
نویسنده: شاهین کیاست
تاریخ: ۱۳۹۲/۰۳/۱۷ ۱۳:۵
آدرس: www.dotnettips.info
برچسب‌ها: C#, LINQ, .NET, Performance

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 Dictonary

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

برای پیدا کردن مقدار استفاده می‌شود با اولین تلاش مقدار مورد نظر یافت می‌شود پس پیچیدگی زمانی آن 1 O 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: ۱۳۹۲/۰۳/۱۷ ۲۱:۳۵

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

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

با تشکر از دوست خوبم ، یک سئوال 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 واکشی شده و در نتیجه جستجو میشه؟ آیا این مطلب درسته؟ اگر آره پس نتیجه به نفع Linq تغییر میکنه

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

نه. ToList یا ToDictionary اصطلاحا 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؛ فرضا 100 رکورد دریافت شده، حالا شما خواستید از این رکوردها برای استفاده، List درست کنید یا دیکشنری یا حالت‌های دیگر).

عنوان: **آشنایی با Implicit Casting و Explicit Casting**
نویسنده: مسعود پاکدل
تاریخ: ۱۳۹۲/۰۳/۱۸ ۲۳:۳۰
آدرس: www.dotnettips.info
گروه‌ها: C#, implicit casting, explicit casting

همه ما به نحوی در پروژه‌های خود مجبور به تبدیل انوع داده شده ایم و یک نوع از داده یا Object رو به نوع دیگری از داده یا Object تبدیل کرده ایم. در این پست دو روش دیگر برای تبدیل انواع داده‌ها بررسی میکنیم. برای شروع دو کلاس زیر رو در نظر بگیرید.

#1کلاس Book

```
public class Book
    {
        public int Code { get; set; }
        public string Title { get; set; }
        public string Category { get; set; }
    }
```

#2کلاس NoteBook

```
public class NoteBook
    {
        public int Code { get; set; }
        public string Title { get; set; }
    }
```

این دو کلاس هیچ ارتباطی با هم ندارند در نتیجه امکان تبدیل این دو نوع وجود ندارد یعنی اجرای هر دو دستور زیر باعث ایجاد خطای کامپایلری می‌شود.

```
static void Main( string[] args )
        {
            Book book = new Book()
            {
                Code = 1,
                Title = "Book1",
                Category = "Default"
            };

            NoteBook noteBook = new NoteBook();

            noteBook = (NoteBook)book;//Compile error
            noteBook = book as NoteBook;//Compile error
        }
```

برای حل این مشکل و تبدیل این دو نوع از Objectها می‌تونیم از دو نوع ImplicitCasting و Explicit Casting استفاده کنیم.

Explicit Casting#

```
public class Book
    {
        public int Code { get; set; }
        public string Title { get; set; }
        public string Category { get; set; }

        public static explicit operator NoteBook( Book book )
        {
            return new NoteBook()
            {
                Code = book.Code,
                Title = book.Title
            };
        }
    }
```

در Explicit یک Operator به صورت Explicit تعریف می‌کنیم که ورودی اون از نوع خود کلاس book و خروجی اون از نوع

مورد دلخواه است. Converter مورد نظر رو در بدنه این Operator می‌نویسیم. حالا به راحتی دستور زیر کامپایل می‌شود.

```
static void Main( string[] args )
        {
            Book book = new Book()
            {
                Code = 1,
                Title = "Book1",
                Category = "Default"
            };

            NoteBook noteBook = new NoteBook();

            noteBook = (NoteBook)book;//Correct
        }
```

```
static void Main( string[] args )
{
    Book book = new Book()
    {
        Code = 1,
        Title = "Book1",
        Category = "Default"
    };

    NoteBook noteBook = new NoteBook();

    noteBook = (NoteBook)book;//Correct
}
```

noteBook {ImplicitExplicitCasting.NoteBook}

Code	1
Title	"Book1"

در بالا مشاهده می‌کنید که حتما باید به طور صریح عملیات Cast رو انجام دهیددر غیر این صورت همچنان خطا خواهید داشت. اما می‌توان این مراحل رو هم نادیده گرفت و تبدیل رو به صورت Implicit انجام داد.

Implicit Casting#

```
public class Book
    {
        public int Code { get; set; }
        public string Title { get; set; }
        public string Category { get; set; }

        public static implicit operator NoteBook( Book book )
        {
```

```
            return new NoteBook()
            {
                Code = book.Code,
                Title = book.Title
            };
        }
    }
```

تنها تفاوت این روش با روش قبلی، در نوع تعریف operator است. بعد از تعریف نوع استفاده به صورت زیر خواهد بود.

```
static void Main( string[] args )
        {
            Book book = new Book()
            {
                Code = 1,
                Title = "Book1",
                Category = "Default"
            };

            NoteBook noteBook = new NoteBook();

            noteBook = book;//Correct
        }
```

در این روش نیاز به ذکر نوع Object برای Cast نیست و Object مورد نظر به راحتی به نوع داده قبل از اپراتور = تبدیل می‌شود.

نظرات خوانندگان

نویسنده: وحید نصیری
تاریخ: ۱۳۹۲/۰۳/۱۸ ۲۳:۵۸

مطلب جالبی است که کمتر مورد استفاده قرار می‌گیره. شاید ذکر مثال‌های دنیای واقعی اون در اینجا مکمل خوبی باشه. اگر دوستان هم مثالی مدنظرشون بود، لطفا عنوان کنند.

An easier way to manage INotifyPropertyChanged

#Custom Explicit and Implicit Operators in C

?Why does this code work

ASP.NET MVC Model binding with implicit operators

Forgotten C# language features: implicit operator

How to fake Enums in EF 4

نویسنده: میثم هوشمند
تاریخ: ۱۳۹۲/۰۹/۰۲ ۱۰:۲

با سلام
من این کار را میخواهم بر روی آرایه ای از یک کلاس انجام دهم؛
یعنی در واقع آرایه ای از یک کلاس را به آرایه ای از کلاسی دیگر تخصیص دهم
با جستجویی که کردم گویا امکانش نیست و نیاز به استفاده از تکنیک هایی هست
اما مثلا در دات نت ورژن دو گویا پشتیبانی نمی‌شوند.
دقیق خاطرم نیست اما فکر میکنم از LINQ استفاده شده بود
با تشکر

عنوان: Bind کردن Enum به ItemsSource در XAML
نویسنده: بزمی
تاریخ: ۱۳۹۲/۰۵/۱۱ ۸:۰
آدرس: www.dotnettips.info
برچسب‌ها: C#, Silverlight, WPF, XAML, Binding

شاید تا به حال در یک برنامه سازمانی نیاز به Bind کردن یک Enum به کنترل‌های XAML به چشمتان خورده باشد ، روشی که من برای این کار استفاده می‌کنم توسط یک Markup Extension به صورت زیر است :

```
public class ByteEnumerationExtention : MarkupExtension
    {
        public ByteEnumerationExtention(Type enumType)
        {
            this.enumType = enumType;
        }

        private Type enumType;

        public Type EnumType
        {
            get { return enumType; }
            private set
            {
                enumType = value;
            }
        }

        public override object ProvideValue(IServiceProvider serviceProvider)
        {
            return (from x in Enum.GetValues(EnumType).OfType<Enum>()
                    select new EnumerationMember
                    {
                        Value = GetValue(x),
                        Description = GetDescription(x)
                    }).ToArray();
        }

        private byte? GetValue(object enumValue)
        {
            return Convert.ToByte(enumValue.GetType().GetField("value__").GetValue(enumValue));
        }

        public object GetObjectValue(object enumValue)
        {
            return enumValue.GetType().GetField("value__").GetValue(enumValue);
        }

        public string GetDescription(object enumValue)
        {
            var descAttrib = EnumType.GetField(enumValue.ToString())
                .GetCustomAttributes(typeof(DescriptionAttribute), false)
                .FirstOrDefault() as DescriptionAttribute;
            return descAttrib != null ? descAttrib.Description : enumValue.ToString();
        }
    }

    public class EnumerationMember
    {
        public string Description { get; set; }

        public byte? Value { get; set; }
    }
```

: XAML

```
<ComboBox ItemsSource="{Binding Source={ Extensions:ByteEnumerationExtention {x:Type type:MyEnum} }}"
DisplayMemberPath="Description"
 SelectedValuePath="Value" SelectedValue="{Binding SelectedItemInViewModel}"/>
```

: Enum

```
public enum MyEnum : short
    {
        [Description("گزینه 1")]
        Item1 = 1,

        [Description("گزینه 2")]
        Item1 = 2,

        [Description("گزینه 3")]
        Item1 = 3,

        [Description("گزینه 4")]
        Item1 = 4,

        [Description("گزینه 5")]
        Item1 = 5,
.
.
.
    }
```

SelectedItem در ViewModel :

```
short? selectedItemInViewModel;
        public short? SelectedItemInViewModel
        {
            get
            {
                return selectedItemInViewModel;
            }
            set
            {
                selectedItemInViewModel = value;
                 RaisePropertyChanged("SelectedItemInViewModel");
               //do calculations if needed
            }
        }
```

عنوان: **بهینه سازی برنامه‌های وب ASP.NET برای موتورهای جستجو (SEO)**
نویسنده: محسن نجف زاده
تاریخ: ۱۳۹۲/۰۷/۲۴ ۱۴:۵۰
آدرس: www.dotnettips.info
گروه‌ها: ASP.Net, C#, MVC, SEO

می‌دانیم بهینه‌سازی موتورهای جستجو (به انگلیسی: Search engine optimization (SEO)) که گاهی در فارسی به آن سئو نیز گفته می‌شود، عملیاتی است برای بهبود دید یک وب‌گاه یا یک صفحهٔ وب، در صفحه نتایج موتورهای جستجو که می‌تواند طبیعی و یا الگوریتمی باشد. این عملیات برای وبمسترها یکی از عوامل مهم و حیاتی بدست آوردن کاربران جدید از موتورهای جستجو است.

اگر چک لیست‌های SEO وب سایت ها را مشاهده کنیم، می‌توانیم آن‌ها را در دو دسته‌ی کلی بهینه سازی درونی و برونی وب سایت در نظر بگیریم :

Off-Page Optimization یا برونی ، که بیشتر بر دوش مشاوران سئو و خود مدیران وب سایت است . (link building ، فعالیت در شبکه اجتماعی و ...)

و اما در حوزه **On-Page Optimization** یا درونی که بخش‌های مهمی از آن وظیفه‌ی مابرنامه نویس‌ها است . (H1 Tag ، URL Naming ، Meta Tags ، عنوان صفحه و ...)

[البته عامل درونی بهینه سازی محتوا (Content Optimization) که مهمترین عامل در الگوریتم‌های نسل جدید موتورهای جستجو و همچنین الگوریتم جدید گوگل (و +) به حساب می‌آید بر عهده مشاوران سئو و خود مدیران وب سایت می‌باشد]

در ادامه به ارائه چند راهکار جهت بهینه سازی برنامه‌های وب ASP.NET مان برای موتورهای جستجو می‌پردازیم:

1.متدی برای ایجاد عنوان سایت

```
private const string SeparatorTitle = " - ";
    private const int MaxLenghtTitle = 60;
    public static string GeneratePageTitle(params string[] crumbs)
    {
        var title = "";

        for (int i = 0; i < crumbs.Length; i++)
        {
            title += string.Format
                        (
                            "{0}{1}",
                            crumbs[i],
                            (i < crumbs.Length - 1) ? SeparatorTitle : string.Empty
                        );
        }

        title = title.Substring(0, title.Length <= MaxLenghtTitle ? title.Length :
MaxLenghtTitle).Trim();

        return title;
    }
```

نکته :

MaxLenghtTitle پیشنهادی برای عنوان سایت 60 می‌باشد.

2.متدی برای ایجاد متاتگ صفحات سایت

```
public enum CacheControlType
{
    [Description("public")]
    _public,
    [Description("private")]
    _private,
    [Description("no-cache")]
    _nocache,
    [Description("no-store")]
    _nostore
```

```
}



private const int MaxLenghtTitle = 60;
private const int MaxLenghtDescription = 170;
private const string FaviconPath = "~/cdn/ui/favicon.ico";
public static string GenerateMetaTag(string title, string description, bool allowIndexPage, bool
allowFollowLinks, string author = "", string lastmodified = "", string expires = "never", string
language = "fa", CacheControlType cacheControlType = CacheControlType._private)
{
        title = title.Substring(0, title.Length <= MaxLenghtTitle ? title.Length :
MaxLenghtTitle).Trim();
        description = description.Substring(0, description.Length <= MaxLenghtDescription ?
description.Length : MaxLenghtDescription).Trim();

        var meta = "";
        meta += string.Format("<title>{0}</title>\n", title);
        meta += string.Format("<link rel=\"shortcut icon\" href=\"{0}\"/>\n", FaviconPath);
        meta += string.Format("<meta http-equiv=\"content-language\" content=\"{0}\"/>\n", language);
        meta += string.Format("<meta http-equiv=\"content-type\" content=\"text/html; charset=utf-
8\"/>\n");
        meta += string.Format("<meta charset=\"utf-8\"/>\n");
        meta += string.Format("<meta name=\"description\" content=\"{0}\"/>\n", description);
        meta += string.Format("<meta http-equiv=\"Cache-control\" content=\"{0}\"/>\n",
EnumExtensions.EnumHelper<CacheControlType>.GetEnumDescription(cacheControlType.ToString()));
        meta += string.Format("<meta name=\"robots\" content=\"{0}, {1}\" />\n", allowIndexPage ?
"index" : "noindex", allowFollowLinks ? "follow" : "nofollow");
        meta += string.Format("<meta name=\"expires\" content=\"{0}\"/>\n", expires);

        if (!string.IsNullOrEmpty(lastmodified))
            meta += string.Format("<meta name=\"last-modified\" content=\"{0}\"/>\n", lastmodified);

        if (!string.IsNullOrEmpty(author))
            meta += string.Format("<meta name=\"author\" content=\"{0}\"/>\n", author);

        //-----------------------------------Google & Bing Doesn't Use Meta Keywords ...
        //meta += string.Format("<meta name=\"keywords\" content=\"{0}\"/>\n", keywords);

        return meta;
}
```

چند نکته :

MaxLenghtDescription پیشنهادی برای متاتگ توضیح سایت 170 می باشد.

آشنایی با متاتگ‌ها (Meta tags) و کاربرد آنها در صفحات وب (HTML)

برای کاربرد allowIndexPage و allowFollowLinks هم می‌توانید به لینک بالا و بررسی متاتگ robots بپردازید.

با توجه به اهمیت شبکه‌های اجتماعی متاتگ‌های شبکه‌های اجتماعی (+ و +) را هم نباید از قلم انداخت.

برای دریافت Description نوع سفارشی CacheControlType از پروژه متدهای الحاقی علیرضا اسم رام استفاده کردم.

3.متدی برای ایجاد Slug (اسلاگ آدرسی با مفهوم برای بکار بردن در URL ها است که دوست‌دار موتورهای جستجو می‌باشد)

```
private const int MaxLenghtSlug = 45;
public static string GenerateSlug(string title)
{
        var slug = RemoveAccent(title).ToLower();
        slug = Regex.Replace(slug, @"[^a-z0-9-\u0600-\u06FF]", "-");
        slug = Regex.Replace(slug, @"\s+", "-").Trim();
        slug = Regex.Replace(slug, @"-+", "-");
        slug = slug.Substring(0, slug.Length <= MaxLenghtSlug ? slug.Length : MaxLenghtSlug).Trim();

        return slug;
}

private static string RemoveAccent(string text)
{
        var bytes = Encoding.GetEncoding("UTF-8").GetBytes(text);
        return Encoding.UTF8.GetString(bytes);
}
```

نکته :

MaxLenghtSlug پیشنهادی برای عنوان سایت 45 می‌باشد.

نمونه ای از کاربرد توابع :

```
Head.InnerHtml = SEO.GenerateMetaTag
                            (
                                title: SEO.GeneratePageTitle(".NET Tips", "آرشیو مطالب", "ASP.NET MVC
#1"),
                                description: "چرا ASP.NET MVC با وجود فریم ورک پخته‌ای به نام ASP.NET web
forms، مطرح می‌شود این است: «برای چی؟». بنابراین تا به این سؤال ASP.NET MVC اولین سؤالی که حین سوئیچ به
پاسخ داده نشود، هر نوع بحث فنی در این مورد بی فایده است.",
                                allowIndexPage: true,
                                allowFollowLinks: true,
                                author: "وحید نصیری",
                                cacheControlType: SEO.CacheControlType._private
                            );
```

و خروجی در Page Source :

```
<title>.NET Tips - آرشیو مطالب - ASP.NET MVC #1</title>
<link rel="shortcut icon" href="../../cdn/images/ui/favicon.ico"/>
<meta http-equiv="content-language" content="fa"/>
<meta http-equiv="content-type" content="text/html; charset=utf-8"/>
<meta charset="utf-8"/>
<meta name="description" content="چرا ASP.NET MVC با وجود فریم ورک پخته‌ای به نام ASP.NET web forms،
مطرح می‌شود این است: &#171;برای چی؟&#187;. بن ASP.NET MVC اولین سؤالی که حین سوئیچ به ..."/>
<meta http-equiv="Cache-control" content="private"/>
<meta name="robots" content="index, follow" />
<meta name="expires" content="never"/>
<meta name="author" content="وحید نصیری"/>
```

موفق باشید

نظرات خوانندگان

نویسنده: صابر فتح الهی
تاریخ: ۱۳۹۲/۰۹/۰۸ ۱۵:۲۰

دوست من با این روش

```
Head.InnerHtml = SEO.GenerateMetaTag(....)
```

در صورتی که در قسمت head صفحه استایل یا متاتگهای دیگری تعریف شده باشد همگی حذف خواهند شد.

نویسنده: محسن نجف زاده
تاریخ: ۱۳۹۲/۰۹/۰۸ ۱۸:۴۵

بله صابر جان حتما همین طور است (برای مثال بیان شده است).
خودم به این صورت استفاده می‌کنم :

```
<head>
    <asp:Literal runat="server" ID="litHead"></asp:Literal>

    ....

</head>
```

و در قسمت *code - behind*

```
litHead.Text = SEO.GenerateMetaTag(...)
```

پ ن : چرا از کنترل Literal استفاده شده و از Label استفاده نشده ؟ به این دلیل که خروجی متن رندر شده Label توسط تگ یا <label> احاطه می‌شود. (+)

نویسنده: پوریا منفرد
تاریخ: ۱۳۹۲/۱۰/۱۴ ۱۲:۱۵

با سلام
امکان داره این خط کلاسش رو هم بزارید بنده دقیقا نتونستم با متد الحاقی که گفتین این قسمت رو پیاده سازی کنم

```
EnumExtensions.EnumHelper<CacheControlType>.GetEnumDescription(cacheControlType.ToString()));
```

نویسنده: وحید نصیری
تاریخ: ۱۳۹۲/۱۰/۱۴ ۱۲:۲۳

مراجعه کنید به پروژه DNT.Extensions و همچنین اینجا

نویسنده: پوریا منفرد
تاریخ: ۱۳۹۲/۱۰/۱۴ ۱۳:۲۸

بنده یه مشکلی دارم : از MasterPage استفاده می‌کنم در ASP.net Webform و فرمی رو بهش Connect کردم که:
1- در MPage یک سری متا تگ‌ها هست + JavaScript + CSS
2- وقتی در یک webForm متصل به Mpage کد:

```
Head.InnerHtml = SEO.GenerateMetaTag ....
```

رو قرار می‌دهم تمام تگ‌های Head حتی MasterPage هم پاک می‌شه
اینم می‌دونم این کلاس این کارو می‌کنه
راه حل چیه که تگ‌ها پاک نشه ؟ مخصوصا آدرس‌های CSS و JavaScript ؟

نویسنده: وحید نصیری
تاریخ: ۱۳۹۲/۱۰/۱۴ ۱۳:۵۱

کمی بالاتر توضیح دادند ؛ استفاده از یک Literal برای مثال.

نویسنده: مهرداد پاک دل
تاریخ: ۱۳۹۲/۱۱/۰۲ ۱۶:۱۶

من این توابع را خواندم که در مورد seo نوشته بودید ولی متاسفانه نحوه استفاده از آن هارا نفهمیدم یعنی کجا تعریف میشن و از کجا پارامترهای خود را دریافت می‌کنند. لطف کنید اگر امکان داره یه پروژه کوچک همه این‌ها استفاده شده باشه اگر هست بگذارید .
نکته دیگه شما عنوان هر پست را داخل urlها نمایش میدهید منظور اینکه جزئی از url سایت است این کار را چگونه انجام می‌دهید

```
mysite.info/post/1
```

```
dotnettips.info/post/1200/seo
```

نویسنده: محسن خان
تاریخ: ۱۳۹۲/۱۱/۰۲ ۱۶:۳۸

- پروژه Iris هست. به section metatags اون دقت کن در فایل‌های View.

- مثلا متد Head.InnerHtml = SEO.GenerateMetaTag عنوان شده در این مطلب باید در Page_Load یک وب فرم فراخوانی شود. Id مطلب رو دارید. عنوان و متن و سایر مشخصات اون رو از دیتابیس دریافت کنید و بعد فقط یک جایگذاری است در متد تهیه شده.

- به اینکار Routing و Url rewrite می‌گن. بحثش در MVC و وب فرم‌ها کمی با هم فرق می‌کنه.

نویسنده: امین قادری
تاریخ: ۱۳۹۳/۰۵/۰۹ ۳:۳۵

لطفا مثالی را در مورد نحوه‌ی استفاده از " **متدی برای ایجاد Slug** " در عمل ارسال نمائید. با تشکر

نویسنده: موسوی
تاریخ: ۱۳۹۳/۰۹/۲۲ ۱۸:۹

سلام منم کدها رو تویه فایل csبه نام سئو دخیره کردم اما اینجا وقتی ران میکنم ارور میده

```
public enum CacheControlType
{
    [Description("public")]
    _public,
```

```
    [Description("private")]
    _private,
    [Description("no-cache")]
    _nocache,
    [Description("no-store")]
    _nostore
}
```

نویسنده: محسن خان
تاریخ: ۱۳۹۳/۰۹/۲۲ ۱۹:۵

« ارور می‌ده » برای رفع مشکل کافی نیست. چه خطایی می‌ده دقیقا؟

نویسنده: موسوی
تاریخ: ۱۳۹۳/۰۹/۲۳ ۸:۴۰

درسته حق با شماست. این ارور هست؛ یه نگاهی بندازین فکر کنم با فرم وورکم مشکل داره؟

```
Attribute 'Description' is not valid on this declaration type. It is only valid on 'method'
declarations.
```

نویسنده: وحید نصیری
تاریخ: ۱۳۹۳/۰۹/۲۳ ۹:۲۹

- ابتدا باید اسمبلی استاندارد System.ComponentModel.DataAnnotations به ارجاعات پروژه اضافه شود.
- سپس باید از فضای نام مرتبط استفاده کنید (و نه از فضاهای نام مشابه)

```
enum MyEnum {
    [System.ComponentModel.Description("Blah")]
    MyValue
}
```

نویسنده: موسوی
تاریخ: ۱۳۹۳/۰۹/۲۳ ۱۵:۵۸

بازم تشکر بله مشکل همین بود و طبق دستور شما استفاده کردم و در headهم از یه لیترال استفاده کردم وخیلی خوب جواب داد.زنده باشین

نویسنده: محمدرضا امینی
تاریخ: ۱۳۹۳/۱۲/۰۳ ۱۱:۵۹

سلام.
هنگام جستجو در گوگل گاهی اوقات سایتی باز میشه که خود اون سایت هم کلمه کلیدی شما رو جستجو کرده و نتایج رو به شما نشون میده.
مثلا یک فروشگاه خوب هنگامی که در گوگل قیمت رم لپ تاپ رو جستجو کنید چنین صفحه ای داره.
به نظر شما دوستان چطور میشه چنین صفحه ای در سایت خود داشته باشیم؟

نویسنده: وحید نصیری
تاریخ: ۱۳۹۳/۱۲/۰۳ ۱۵:۷

- ابتدا یک پروفایل Google analytics ایجاد کنید و سپس اسکریپت آن‌را به سایت خودتان اضافه کنید.
- چند روز بعد که به آمار آن مراجعه کنید، می‌توانید لیست جستجوهای گوگل منتهی به سایت خودتان را به همراه واژه‌های کلیدی مرتبط، گزارش گیری کنید.

- بر اساس این واژه‌های کلیدی، برای محصولات خودتان برچسب درست کنید یا آن‌ها را گروه بندی کنید. گوگل بر این اساس در دفعات آتی، نتایج جستجوی دقیق‌تری را به کاربران ارائه می‌دهد.

نویسنده: محسن نجف زاده
تاریخ: ۱۳۹۳/۱۲/۰۳ ۱۵:۲۵

پیرو تکمیل صحبت آقای نصیری،
احتمالا جستجو در وب سایت مورد نظر در دو صفحه مجزا طراحی شده :
1.جستجوی‌های طبقه بندی شده بر اساس اطلاعات استخراجی شان از ابزارها ارزیابی وب سایت‌ها مانند Google analytics (راهنما در کامنت قبل)
2. جستجوی لحظه ای در محتوای سایت
این نکته در متا تگ‌های ایجاد شده نیز قابل مشاهده است :

www. .com/Search/Category-Laptop-RAM/#/Category-Laptop-RAM/

```
<title> جستجو در گروه رم لپ تاپ </title>
<meta content="جستجو در گروه رم لپ تاپ" name="description">
<meta content="جستجو در گروه رم لپ تاپ" name="keywords">
```

www. .com/Search/#/Keyword-رم لپ تاپ/

```
<title> جستجو </title>
```

تعداد نتایج یافت شده متفاوت این دو صفحه (با یک عنوان جستجو) نشان از تایید مطلب دارد.

نویسنده: محمدرضا امینی
تاریخ: ۱۳۹۳/۱۲/۱۹ ۱۶:۵۱

بر فرض مثال ما می‌خواهیم زمانی که از گوگل شخصی به سایت ما هدایت می‌شود با کلمه کلیدی ای که وارد سایت شده به طور خودکار در سایت جستجو کنیم.
کلا چطور میشه متوجه شد که کاربری از گوگل به سایت ما وارد شده و با چه جستجویی؟

نویسنده: وحید نصیری
تاریخ: ۱۳۹۳/۱۲/۱۹ ۱۹:۱۴

Request.UrlReferrer را آنالیز کنید. مقدار آن مشخص می‌کند که مثلا شخص از گوگل است یا سایت‌های جستجوی دیگر. بعد بر این اساس، کوئری استرینگ q آن‌را یافته و سپس شخص را به صفحه‌ی جستجو هدایت کنید:

```
// for Google
var urlReferrer = Request.UrlReferrer.ToString();
var query = HttpUtility.ParseQueryString(urlReferrer);
var searchQuery = query["q"];
```

یک نمونه‌ی کاملتر

نویسنده: علی یگانه مقدم
تاریخ: ۱۳۹۳/۱۲/۲۰ ۲۰:۲۹

یک سوال داشتم
اینکه این گونه سایت‌ها از این روش برای افزایش بازدید استفاده می‌کنند یک نوع کلاه برداری در SEO به شمار نمیاد؟
چون که عموما این سایت‌ها هیچ محتوای مرتبطی ندارن و به خاطر افزایش بازدید اینکارو می‌کنن و باعث به هم ریختن نتایج جست و جو میشن تا جایی که گاهی اوقات می‌بینی چند صفحه اول تمام لینک‌ها از این قبیل سایت هاست و گوگل هم بارها هشدار داده سایت هایی که کلاه برداری می‌کنند و محتوا ندارن رو جریمه می‌کنه ولی به تا به حال یک بار هم این حرف عملی نشده و همچنان همان سایت‌ها هم به فعالیت ادامه میدن و باعث به هم ریختگی میشن

نویسنده: محسن خان
تاریخ: ۱۳۹۳/۱۲/۲۱ ۹:۵۶

هرکسی از ظن خود شد یار من ...

عنوان: CheckBoxList برای فیلد Enum Flags مدل در ASP.Net MVC
نویسنده: پژمان پارسائی
تاریخ: ۱۳۹۲/۰۷/۳۰ ۲۳:۰
آدرس: www.dotnettips.info
گروه‌ها: C#, ASP.Net MVC, Enum, ASP.Net MVC 4, MVC

قبلا مطالبی در سایت راجع به نوع داده شمارشی یا Enum و همچنین CheckBoxList و RadioButtonList وجود دارد. اما در این مطلب قصد دارم تا یک روش متفاوت را برای تولید و بهره گیری از CheckBoxList با استفاده از نوع داده‌های شمارشی برای شما ارائه کنم.
فرض کنید بخواهید به کاربر این امکان را بدهید تا بتواند چندین گزینه را برای یک فیلد انتخاب کند. به عنوان یک مثال ساده فرض کنید گزینه ای از مدل، پارچه‌های مورد علاقه یک نفر هست. کاربر می‌تواند چندین پارچه را انتخاب کند. و این فرض را هم بکنید که به لیست پارچه‌ها گزینه دیگری اضافه نخواهد شد. پارچه (Fabric) را مثلا می‌توانیم به صورت زیر تقسیم بندی کنیم :

پنبه (Cotton)
ابریشم (Silk)
پشم (Wool)
ابریشم مصنوعی (Rayon)
پارچه‌های دیگر (Other)

با توجه به اینکه دیگر قرار نیست به این لیست گزینه دیگری اضافه شود می‌توانیم آنرا به صورت یک نوع داده شمارشی (Enum) تعریف کنیم. مثلا بدین صورت:

```
public enum Fabric
{
    [Description("پنبه")]
    Cotton,

    [Description("ابریشم")]
    Silk,

    [Description("پشم")]
    Wool,

    [Description("ابریشم مصنوعی")]
    Rayon,

    [Description("پارچه‌های دیگر")]
    Other
}
```

حال فرض کنید View Model زیر فیلدی از نوع نوع داده شمارشی Fabric دارد:

```
public class MyViewModel
{
    public Fabric Fabric { get; set; }
}
```

توجه داشته باشید که فیلد Fabric از کلاس MyViewModel باید چند مقدار را در خود نگهداری کند. یعنی می‌تواند هر کدام از گزینه‌های Cotton، Silk، Wool، Rayon، Other به صورت جداگانه یا ترکیبی باشد. اما در حال حاضر با توجه به اینکه یک فیلد Enum معمولی فقط می‌تواند یک مقدار را در خودش ذخیره کند قابلیت ذخیره ترکیبی مقادیر در فیلد Fabric از View Model بالا وجود ندارد.

اما راه حل این مشکل استفاده از پرچم (Flags) در تعریف نوع داده شمارشی هست. با استفاده از پرچم نوع داده شمارشی بالا به صورت زیر باید تعریف شود:

```
[Flags]
public enum Fabric
```

```
{
    [Description("پنبه")]
    Cotton = 1,

    [Description("ابریشم")]
    Silk = 2,

    [Description("پشم")]
    Wool = 4,

    [Description("ابریشم مصنوعی")]
    Rayon = 8,

    [Description("پارچه‌های دیگر")]
    Other = 128
}
```

همان طور که می‌بینید از عبارت [Flags] قبل از تعریف enum استفاده کرده ایم. همچنین هر کدام از مقادیر ممکن این نوع داده شمارشی با توانهایی از 2 تنظیم شده اند. در این صورت یک نمونه از این نوع داده می‌تواند چندین مقدار را در خودش ذخیره کند.

برای آشنایی بیشتر با این موضوع به کدهای زیر نگاه کنید:

```
Fabric cotWool = Fabric.Cotton | Fabric.Wool;
int cotWoolValue = (int) cotWool;
```

به وسیله عملگر | می‌توان چندین مقدار را در یک نمونه از نوع Fabric ذخیره کرد. مثلا متغیر cotWool هم دارای مقدار Fabric.Cotton و هم دارای مقدار Fabric.Wool هست. مقدار عددی معادل متغیر cotWool برابر 5 هست که از جمع مقدار عددی Fabric.Cotton و Fabric.Wool به دست آمده است.

حال فرض کنید فیلد Fabric از View Model ذکر شده (کلاس MyViewModel) را به صورت لیستی از چک باکس‌ها نمایش دهیم. مثل زیر:

شکل (الف)

سپس بخواهیم تا کاربر بعد از انتخاب گزینه‌های مورد نظرش از لیست بالا و پست کردن فرم مورد نظر، بایندر وارد عمل شده و فیلد Fabric را بر اساس گزینه هایی که کاربر انتخاب کرده مقداردهی کند.

برای این کار از پروژه MVC Enum Flags کمک خواهیم گرفت. این پروژه شامل یک Html Helper برای تبدیل یه Enum به یک CheckBoxList و همچنین شامل Model Binder مربوطه هست. البته بعضی از کدهای Html Helper آن احتیاج به تغییر داشت که

آنرا انجام دادم ولی بایندر آن بسیار خوب کار می‌کند.

خوب html helper مربوط به آن به صورت زیر می‌باشد:

```
public static IHtmlString CheckBoxesForEnumFlagsFor<TModel, TEnum>(this HtmlHelper<TModel> htmlHelper,
Expression<Func<TModel, TEnum>> expression)
{
    ModelMetadata metadata = ModelMetadata.FromLambdaExpression(expression, htmlHelper.ViewData);
    Type enumModelType = metadata.ModelType;

    // Check to make sure this is an enum.
    if (!enumModelType.IsEnum)
    {
        throw new ArgumentException("This helper can only be used with enums. Type used was: " +
enumModelType.FullName.ToString() + ".");
    }

    // Create string for Element.
    var sb = new StringBuilder();

    foreach (Enum item in Enum.GetValues(enumModelType))
    {
        if (Convert.ToInt32(item) != 0)
        {
            var ti = htmlHelper.ViewData.TemplateInfo;
            var id = ti.GetFullHtmlFieldId(item.ToString());

            //Derive property name for checkbox name
            var body = expression.Body as MemberExpression;
            var propertyName = body.Member.Name;
            var name = ti.GetFullHtmlFieldName(propertyName);

            //Get currently select values from the ViewData model
            TEnum selectedValues = expression.Compile().Invoke(htmlHelper.ViewData.Model);

            var label = new TagBuilder("label");
            label.Attributes["for"] = id;
            label.Attributes["style"] = "display: inline-block;";
            var field = item.GetType().GetField(item.ToString());

            // Add checkbox.
            var checkbox = new TagBuilder("input");
            checkbox.Attributes["id"] = id;
            checkbox.Attributes["name"] = name;
            checkbox.Attributes["type"] = "checkbox";
            checkbox.Attributes["value"] = item.ToString();

            if ((selectedValues as Enum != null) && ((selectedValues as Enum).HasFlag(item)))
            {
                checkbox.Attributes["checked"] = "checked";
            }
            sb.AppendLine(checkbox.ToString());

            // Check to see if DisplayName attribute has been set for item.
            var displayName = field.GetCustomAttributes(typeof(DisplayNameAttribute), true)
                .FirstOrDefault() as DisplayNameAttribute;
            if (displayName != null)
            {
                // Display name specified.  Use it.
                label.SetInnerText(displayName.DisplayName);
            }
            else
            {
                // Check to see if Display attribute has been set for item.
                var display = field.GetCustomAttributes(typeof(DisplayAttribute), true)
                    .FirstOrDefault() as DisplayAttribute;
                if (display != null)
                {
                    label.SetInnerText(display.Name);
                }
                else
                {
                    label.SetInnerText(item.ToDescription());
                }
            }
            sb.AppendLine(label.ToString());

            // Add line break.
            sb.AppendLine("<br />");
```

```
        }
    }

    return new HtmlString(sb.ToString());
}
```

در کدهای بالا از متد الحاقی ToDescription نیز برای تبدیل معادل انگلیسی به فارسی یک مقدار از نوع داده شمارشی استفاده کرده ایم.

```
public static string ToDescription(this Enum value)
{
    var attributes =
(DescriptionAttribute[])value.GetType().GetField(value.ToString()).GetCustomAttributes(typeof(Descripti
onAttribute), false);
    return attributes.Length > 0 ? attributes[0].Description : value.ToString();
}
```

برای استفاده از این Html Helper در View کد زیر را می‌نویسیم:

```
@Html.CheckBoxesForEnumFlagsFor(x => x.Fabric)
```

که باعث تولید خروجی که در تصویر (الف) نشان داده شد می‌شود. و همچنین مدل بایندر مربوط به آن به صورت زیر هست:

```
public class FlagEnumerationModelBinder : DefaultModelBinder
{
    public override object BindModel(ControllerContext controllerContext, ModelBindingContext
bindingContext)
    {
        if (bindingContext == null) throw new ArgumentNullException("bindingContext");

        if (bindingContext.ValueProvider.ContainsPrefix(bindingContext.ModelName))
        {
            var values = GetValue<string[]>(bindingContext, bindingContext.ModelName);

            if (values.Length > 1 && (bindingContext.ModelType.IsEnum &&
bindingContext.ModelType.IsDefined(typeof(FlagsAttribute), false)))
            {
                long byteValue = 0;
                foreach (var value in values.Where(v => Enum.IsDefined(bindingContext.ModelType, v)))
                {
                    byteValue |= (int)Enum.Parse(bindingContext.ModelType, value);
                }

                return Enum.Parse(bindingContext.ModelType, byteValue.ToString());
            }
            else
            {
                return base.BindModel(controllerContext, bindingContext);
            }
        }

        return base.BindModel(controllerContext, bindingContext);
    }

    private static T GetValue<T>(ModelBindingContext bindingContext, string key)
    {
        if (bindingContext.ValueProvider.ContainsPrefix(key))
        {
            ValueProviderResult valueResult = bindingContext.ValueProvider.GetValue(key);
            if (valueResult != null)
            {
                bindingContext.ModelState.SetModelValue(key, valueResult);
                return (T)valueResult.ConvertTo(typeof(T));
            }
        }
        return default(T);
    }
}
```

این مدل بایندر را باید به این صورت در متد Application_Start فایل Global.asax فراخوانی کنیم:

```
ModelBinders.Binders.Add(typeof(Fabric), new FlagEnumerationModelBinder());
```

مشاهده می‌کنید که در اینجا دقیقا مشخص کرده ایم که این مدل بایندر برای نوع داده شمارشی Fabric هست. اگر نیاز دارید تا این بایندر برای نوع داده‌های شمارشی دیگری نیز به کار رود نیاز هست تا این خط کد را برای هر کدام از آنها تکرار کنید. اما راه حل بهتر این هست که کلاسی به صورت زیر تعریف کنیم و تمامی نوع داده‌های شمارشی که باید از بایندر بالا استفاده کنند را در یک پراپرتی آن برگشت دهیم. مثلا بدین صورت:

```
public class ModelEnums
{
    public static IEnumerable<Type> Types
    {
        get
        {
            var types = new List<Type> { typeof(Fabric) };
            return types;
        }
    }
}
```

سپس به متد Application_Start رفته و کد زیر را اضافه می‌کنیم:

```
foreach (var type in ModelEnums.Types)
{
   ModelBinders.Binders.Add(type, new FlagEnumerationModelBinder())
}
```

اگر گزینه‌های پشم و ابریشم مصنوعی را از CheckBoxList تولید شده انتخاب کنیم، بدین صورت:

شکل (ب)

و سپس فرم را پست کنید، موردی شبیه زیر مشاهده می‌کنید:

شکل (ج)

همچنین مقدار عددی معادل در این جا برابر 12 می‌باشد که از جمع دو مقدار Wool و Rayon به دست آمده است. بدین ترتیب در یک فیلد از مدل، گزینه‌های انتخابی توسط کاربر قرار گرفته شده اند.

پروژه مربوط به این مثال را از لینک زیر دریافت کنید:
MvcEnumFlagsProjectSample.zip

پی نوشت : پوشه‌های bin و obj و packages جهت کاهش حجم پروژه از آن حذف شده اند. برای بازسازی پوشه packages لطفا به مطلب بازسازی کامل پوشه packages بسته‌های NuGet به صورت خودکار مراجعه کنید.

نظرات خوانندگان

نویسنده: علی
تاریخ: ۱۳۹۲/۰۸/۰۱ ۱۸:۵۱

من یک گرید تلریک دارم که یک فیلد چکباکس کنارش هست و مثلا لیستی از یوزرها رو این گرید شامل میشه .
آیتم‌های انتخابی رو چطوری میتونم به کنترلرم ارسال کنم (ورودی کنترلرمو چی بگیرم)...

نویسنده: وحید نصیری
تاریخ: ۱۳۹۲/۰۸/۰۱ ۱۸:۵۸

سؤال شما بیشتر به مطلب « CheckBoxList در ASP.NET MVC » مرتبط است تا enum ایی که ویژگی flag دارد. ضمنا گرید تلریک مستندات خوبی دارد که بهتر است به آن مراجعه کنید (مثال Ajax CheckBoxes هست که کدهای View و کنترلر آن نیز پیوست شدند).

نویسنده: علیرضا
تاریخ: ۱۳۹۲/۰۸/۰۲ ۱۲:۱۵

استفاده از Description برای Enumها جالب ولی به نظر من کمی مشکل سازه و مشکل اصلی اون Code Wired کردن شرح هر جزء Enum هست. اگر با یک پروژه دوزبانه طرف باشیم چی؟ اگر کلا استراتژی ترجمه رشته‌ها در پروژه ما این طور باشه که از منابع خارجی مثل DB یا اسمبلی‌های Resource استفاده کنیم چی؟

نویسنده: وحید نصیری
تاریخ: ۱۳۹۲/۰۸/۰۲ ۱۲:۳۲

مراجعه کنید به پیشنیازهای این مباحث تکمیلی. مانند:
تهیه سایت‌های چند زبانه و بومی سازی نمایش اطلاعات در ASP.NET MVC
Globalization در ASP.NET MVC
اولی در مورد کار با ریسورس‌ها است و بومی سازی ویژگی‌ها نیز در آن لحاظ شده و دومی تهیه یک فریم ورک است برای کار با بانک اطلاعاتی و تامین منبع داده از این طریق

عنوان: **مدیریت رجیستری در C#**
نویسنده: امین زمانی
تاریخ: ۱۳۹۲/۰۹/۲۳ ۱۳:۱۵
آدرس: www.dotnettips.info
گروه‌ها: C#, Windows, Registry

رجیستری یک پایگاه داده‌ی سیستمی است که برنامه‌ها، اجزای سیستم و اطلاعات پیکربندی در آن ذخیره و بازیابی می‌شود. داده‌های ذخیره شده در رجیستری مطابق با نسخه ویندوز فرق می‌کنند. نرم‌افزارها برای بازیابی، تغییر و پاک کردن رجیستری از API های مختلفی استفاده می‌کنند. خوشبختانه .NET نیز امکانات لازم برای مدیریت رجیستری را فراهم کرده است.

در صورت رخداد خطا در رجیستری، امکان خراب شدن ویندوز وجود دارد در نتیجه با احتیاط عمل کنید و قبل از هر کاری از رجیستری پشتیبان تهیه نمایید. قبل از شروع به کدنویسی قدری با ساختار رجیستری آشنا شویم تا در ادامه قادر به درک مفاهیم باشیم.

ساختار رجیستری

رجیستری اطلاعات را در ساختار درختی نگاه می‌دارد. هر گره در درخت، یک کلید (key) نامیده می‌شود. هر کلید می‌تواند شامل چندین زیرکلید (subkey) و چندین مقدار (value) باشد. در برخی موارد، وجود یک کلید تمام اطلاعاتی است که نرم افزار بدان نیاز دارد و در برخی موارد، برنامه کلید را باز کرده و مقادیر مربوط به آن کلید را می‌خواند. یک کلید می‌تواند هر تعداد مقدار داشته باشد و مقادیر به هر شکلی می‌توانند باشند. هر کلید شامل یک یا چند کاراکتر است. نام کلیدها نمی‌توانند کاراکتر "\" را داشته باشند. نام هر زیرکلید یکتاست و وابسته به کلیدی است که در سلسله مراتب، بلافاصله بالای آن می‌آید. نام کلیدها باید انگلیسی باشند اما مقادیر را به هر زبانی می‌توان نوشت. در زیر یک نمونه از ساختار رجیستری را مشاهده می‌کنید که در نرم‌افزار registry editor به نمایش در آمده است.

هر کدام از درخت‌های زیر my computer یک کلید است. HKEY_LOCAL_MACHINE دارای زیرکلیدهایی مثل SAM ، HARDWARE و SECURITY است. هر مقدار شامل یک اسم، نوع و داده‌های درون آن است. برای مثال MaxObjectNumber از مقادیر زیرکلید HKEY_LOCAL_MACHINE\HARDWARE\DEVICEMAP\VIDEO است. داده‌های درون هر مقدار می‌تواند از انواع باینری، رشته‌ای و عددی باشد؛ برای مثال MaxObjectNumber یک عدد ۳۲ بیتی است.

محدودیت‌های فنی برای نوع و اندازه‌ی اطلاعاتی که در رجیستری ذخیره می‌گردد، وجود دارد. برنامه‌ها باید اطلاعات اولیه و پیکربندی را در رجیستری نگه دارند وسایر داده‌ها را در جای دیگر ذخیره کنند. معمولا داده‌های بیش‌تر از یک یا دو کیلوبایت باید در یک فایل ذخیره شوند و با استفاده از یک کلید در رجیستری به آن فایل رجوع کرد. برای حفظ فضای ذخیره سازی باید داده‌های شبیه به هم در یک ساختار جمع آوری گردند و ساختار را به عنوان یک مقدار ذخیره کرد؛ به جای آن که هر عضو ساختار را به عنوان یک کلید ذخیره کرد. ذخیره سازی اطلاعات به صورت باینری این امکان را می‌دهد که اطلاعات را در یک مقدار ذخیره کنید.

اطلاعات رجیستری در پیج فایل (Page File) ذخیره می‌شوند. پیج فایل ناحیه‌ای از حافظه RAM است که می‌تواند در زمانی که استفاده نمی‌شود به Hard منتقل شود. اندازه‌ی پیج فایل به وسیله‌ی مقدار PagedPoolSize در کلید HKEY_LOCAL_MACHINE\SYSTEM\CurrentControlSet\Control\Session Manager\Memory Management مطابق با جدول زیر تنظیم می‌گردد.

مقدار	توضیحات
0×00000000	سیستم یک مقدار بهینه را تعیین می‌کند
0x1-0x20000000	یک اندازه مشخص برحسب بایت که در این بازه باشد
0xFFFFFFFF	سیستم بیش‌ترین مقدار ممکن را تشخیص می‌دهد

کلیدهای از پیش تعریف شده

یک برنامه قبل از آن که اطلاعاتی را در رجیستری درج کند باید یک کلید را باز کند. برای باز کردن یک کلید می‌توان از سایر کلیدهایی که باز هستند، استفاده کرد. سیستم کلیدهایی را از پیش تعریف کرده که همیشه باز هستند. در ادامه کلیدهای از پیش تعریف شده را قدری بررسی می‌کنیم.

HKEY_CLASSES_ROOT

زیرشاخه‌های این کلید، انواع اسناد و خصوصیات مربوط به آن‌ها را مشخص می‌کنند. این شاخه نباید در یک سرویس یا برنامه‌ای که کاربران متعدد دارد، مورد استفاده قرار گیرد.

HKEY_CURRENT_USER

زیرشاخه‌های این کلید، تنظیمات مربوط به کاربر جاری را مشخص می‌کنند. این تنظیمات شامل متغیرهای محیطی، اطلاعات درباره‌ی برنامه‌ها، رنگ‌ها، پرینترها، ارتباطات شبکه و تنظیمات برنامه‌هاست. به طور مثال مایکروسافت اطلاعات مربوط به برنامه‌های خود را در کلید HKEY_CURRENT_USER\Software\Microsoft ذخیره می‌کند. هر کدام از برنامه‌ها یک زیرکلید در کلید مزبور را به خود اختصاص داده‌اند. این شاخه نیز نباید در یک سرویس یا برنامه‌ای که کاربران متعدد دارد، مورد استفاده قرار گیرد.

HKEY_LOCAL_MACHINE

زیرشاخه‌های این کلید، وضعیت فیزیکی کامپیوتر را مشخص می‌کنند که شامل حافظه‌ی سیستم، سخت‌افزار و نرم‌افزارهای نصب شده بر روی سیستم، اطلاعات پیکربندی، تنظیمات ورود به سیستم، اطلاعات امنیتی شبکه و اطلاعات دیگر سیستم است.

HKEY_USERS

زیرشاخه‌های این کلید، پیکربندی کاربران پیش فرض، جدید، جاری سیستم و به طور کلی همه‌ی کاربران را مشخص می‌کند.

HKEY_CURRENT_CONFIG

زیرشاخه‌های این کلید، اطلاعاتی درباره وضعیت سخت‌افزار کامپیوتر در اختیار ما می‌گذارند. در واقع این کلید نام مستعاری برای کلید HKEY_LOCAL_MACHINE\SYSTEM\CurrentControlSet\Hardware Profiles\Current است که در ویندوزهای قبل از NT ۳.۵۱ وجود نداشته است.

کندوهای رجیستری

یک کندو (Hive) یک گروه از کلیدها، زیرکلیدها و مقادیر در رجیستری است که یک مجموعه از فایل‌های پشتیبان را به همراه دارد. در هنگام بوت ویندوز، اطلاعات از این فایل‌ها استخراج می‌شوند. شما هم چنین می‌توانید با استفاده از Import در منوی فایل registry editor به صورت دستی این کار را انجام دهید. زمانی که ویندوز را خاموش می‌کنید، اطلاعات کندوها در فایل‌های پشتیبان نوشته می‌شوند. شما می‌توانید این کار را به طور دستی با Export در منوی فایل registry editor نیز انجام دهید.

فایل‌های پشتیبان همه کندوها به جز HKEY_CURRENT_USER در شاخه‌ی Windows Root\System32\config قرار دارند. فایل‌های پشتیبان HKEY_CURRENT_USER در شاخه‌ی System Root\Documents and Settings\Username قرار دارند. پسوند فایل‌ها در این شاخه‌ها، نوع داده‌هایی که در بر دارند را نشان می‌دهند. در جدول زیر برخی کندوها و فایل‌های پشتیبانشان آمده است.

کندوی رجیستری	فایل‌های پشتیبان
HKEY_CURRENT_CONFIG	System, System.alt, System.log, System.sav
HKEY_CURRENT_USER	Ntuser.dat, Ntuser.dat.log
HKEY_LOCAL_MACHINE\SAM	Sam, Sam.log, Sam.sav
HKEY_LOCAL_MACHINE\Security	Security, Security.log, Security.sav
HKEY_LOCAL_MACHINE\Software	Software, Software.log, Software.sav
HKEY_LOCAL_MACHINE\System	System, System.alt, System.log, System.sav
HKEY_USERS\.DEFAULT	Default, Default.log, Default.sav

هر زمان که یک کاربر به کامپیوتر وارد می‌شود، یک کندوی جدید با فایل‌های مجزا برای آن کاربر ساخته می‌شود که کندوی

پروفایل کاربر نام دارد. یک کندوی کاربر، اطلاعاتی شامل تنظیمات برنامه‌های کاربر، تصویر زمینه، ارتباطات شبکه و پرینترها را در بر دارد. کندوهای پروفایل کاربر در کلید HKEY_USERS قرار دارند. مسیر فایل‌های پشتیبان این کندوها در کلید HKEY_LOCAL_MACHINE\SOFTWARE\Microsoft\Windows NT\CurrentVersion\ProfileList\SID\ ProfileImagePath مشخص شده است. مقدار ProfileImagePath مسیر پروفایل کاربر و نام کاربر را مشخص می‌کند.

دسته بندی اطلاعات

قبل از قرار دادن اطلاعات در رجیستری باید آن‌ها را به دو دسته اطلاعات کامپیوتر و اطلاعات کاربر تقسیم کرد. با این تقسیم بندی، چندین کاربر می‌توانند از یک برنامه استفاده کنند و یا اطلاعات را بر روی شبکه قرار دهند. زمانی که یک برنامه نصب می‌شود، باید اطلاعات کامپیوتری خود را در شاخه فرضی HKEY_LOCAL_MACHINE\Software\MyCompany\MyProduct\1.0 به گونه‌ای تعریف کند که نام شرکت، نام محصول و نسخه برنامه به خوبی مشخص گردند و هم چنین اطلاعات مربوط به کاربران را در شاخه فرضی HKEY_CURRENT_USER\Software\MyCompany\MyProduct\1.0 نگاه دارد.

باز کردن، ساختن و بستن کلیدها

قبل از آن که بتوانیم یک اطلاعات را در رجیستری درج کنیم، باید یک کلید بسازیم و یا یک کلید موجود را باز کنیم. یک برنامه همیشه به یک کلید به عنوان زیرکلیدی از یک کلید باز رجوع می‌کند. کلیدهای از پیش تعریف شده همیشه باز هستند.
کلاس‌های تعریف شده برای کار با رجیستری در فضانام Microsoft.Win32 قرار دارند. کلاس Microsoft.Win32.Registry مربوط به کلاس‌های از پیش تعریف شده و کلاس Microsoft.Win32.RegistryKey برای کار با رجیستری است. برای باز کردن یک کلید از متد RegistryKey.OpenSubKey استفاده می‌کنیم. به یاد داشته باشید که کلیدهای از پیش تعریف شده همیشه باز هستند و نیازی به باز کردن ندارند. برای ساختن یک کلید از متد RegistryKey.CreateSubKey استفاده می‌کنیم. دقت کنید زیرکلیدی که می‌خواهید بسازید، باید به یک کلید باز رجوع کند. برای خاتمه دسترسی به یک کلید، باید آن را ببندیم. برای بستن یک کلید از متد RegistryKey.Close استفاده می‌کنیم.

اکنون که با ساختار رجیستری و کلاس‌های مربوطه در .NET برای کار با رجیستری آشنا شدیم، به کدنویسی می‌پردازیم.

ساختن یک زیرکلید جدید

برای ساختن یک زیرکلید جدید از متد RegistryKey.CreateSubKey به صورت زیر استفاده می‌کنیم.

```
public RegistryKey CreateSubKey( string subkey);
```

subkey نام و مسیر کلیدی که می‌خواهید بسازید را مشخص می‌کند که معمولا به فرم فرضی key name\Company Name\Application Name\version است. این متد یک زیرکلید را برمی‌گرداند و در صورت بروز خطا مقدار null را برمی‌گرداند و یک exception را فرا می‌خواند. خطا به دلایلی چون عدم داشتن مجوز، وجود نداشتن مسیر درخواستی و غیره رخ می‌دهد. برای بررسی exception ها می‌توانید از بلوک try-catch استفاده کنید.

```
RegistryKey MyReg = Registry .CurrentUser.CreateSubKey( "SOFTWARE\\SomeCompany\\SomeApp\\SomeVer" );
```

مثال فوق یک زیرکلید جدید در مسیر تعیین شده در شاخه‌ی HKEY_CURRENT_USER می‌سازد.
برای دست یابی به کلیدهای از پیش تعریف شده از کلاس Registry مطابق جدول زیر استفاده می‌کنیم.

فیلد	کلید
ClassesRoot	HKEY_CLASSES_ROOT

فیلد	کلید
CurrentUser	HKEY_CURRENT_USER
LocalMachine	HKEY_LOCAL_MACHINE
Users	HKEY_USERS
CurrentConfig	HKEY_CURRENT_CONFIG

چند نکته حائز اهمیت است. اگر یک زیرکلید با نام مشابه در مسیر تعیین شده وجود داشته باشد، هیچ کلیدی ساخته نمی‌شود. حقیقت آن است که از متد CreateSubKey برای باز کردن یک کلید نیز می‌توانیم استفاده کنیم. متد CreateSubKey زیرکلید را همیشه در حالت ویرایش بازمی‌گرداند. متد CreateSubKey دو پارامتر دیگر به عنوان ورودی دریافت می‌کند که از دو کلاس RegistryKeyPermissionCheck و RegistryOptions استفاده می‌کند. RegistryKeyPermissionCheck مشخص می‌کند که درخت زیرکلید، فقط خواندنی یا قابل ویرایش باشد. RegistryOptions مشخص می‌کند که اطلاعات کلید فقط در حافظه‌ی اصلی باشد و دیگر به کندوها منتقل نشود یعنی به طور موقتی باشد یا به طور پیش فرض دائمی باشد.

باز کردن زیرکلید موجود

برای باز کردن یک زیرکلید موجود از متد RegistryKey.OpenSubKey به دو صورت استفاده می‌کنیم.

```
public RegistryKey OpenSubKey( string name);
public RegistryKey OpenSubKey( string name, bool writable);
```

صورت اول، کلید را در حالت فقط خواندنی باز می‌کند و صورت دوم، اگر true ، writable باشد کلید را در حالت ویرایش باز می‌کند و اگر false باشد کلید را در حالت فقط خواندنی باز می‌کند. در هر دو صورت name ، نام و مسیر زیرکلیدی که می‌خواهید باز کنید را مشخص می‌کند. اگر با خطا مواجه نشوید، متد زیرکلید را برمی‌گرداند، در غیر این صورت مقدار null را برمی‌گرداند.

```
RegistryKey MyReg = Registry .CurrentUser.OpenSubKey( "SOFTWARE\\SomeCompany\\SomeApp\\SomeVer" , true
);
```

مثال فوق کلید مشخص شده را در شاخه‌ی HKEY_CURRENT_USER و در حالت ویرایش باز می‌کند.

خواندن اطلاعات از رجیستری

اگر یک شیء RegistryKey سالم داشته باشید می‌توانید به مقادیر و اطلاعات درون مقادیر آن دسترسی داشته باشید. برای دست یابی به اطلاعات درون یک مقدار مشخص در کلید از متد RegistryKey.GetValue به دو صورت استفاده کنیم.

```
public object GetValue( string name);
public object GetValue( string name, object defaultValue);
```

صورت اول، اطلاعات درون مقداری با نام و مسیر name را برمی‌گرداند. اگر مقدار مذکور وجود نداشته باشد، مقدار null را برمی‌گرداند. درصورت دوم اگر مقدار خواسته شده وجود نداشته باشد، defaultValue را برمی‌گرداند. متد GetValue یک مقدار از نوع object را برمی‌گرداند در نتیجه شما برای استفاده، باید آن را به نوعی که می‌خواهید تبدیل کنید.

نوشتن اطلاعات در رجیستری

برای نوشتن اطلاعات در یک مقدار از متد RegistryKey.SetValue به صورت زیر استفاده می‌کنیم.

```
public void SetValue( string name, object value);
```

رشته name ، نام مقداری که اطلاعات باید در آن ذخیره شود و value اطلاعاتی که باید در آن مقدار ذخیره شود را مشخص می‌کنند. چون value از نوع object است می‌توانید هر مقداری را به آن بدهید. Vallue به طور اتوماتیک به DWORD یا باینری یا رشته‌ای تبدیل می‌شود. البته یک پارامتر سومی نیز وجود دارد که از کلاس RegistryValueKind استفاده کرده و نوع اطلاعات را به طور دقیق مشخص می‌کند. برای ذخیره اطلاعات در مقدار پیش فرض (Default) کافی است که مقدار name را برابر string.Empty قرار دهید. هر کلید می‌تواند یک مقدار پیش فرض داشته باشد که باید نام آن مقدار را Default قرار دهید.

بستن یک کلید

زمانی که دیگر با کلید کاری ندارید و می‌خواهید تغییرات در رجیستری ثبت گردد باید فرآیندی به نام flushing را انجام دهید. برای انجام این کار به راحتی از متد RegistryKey.Close استفاده کنید.

```
RegistryKey MyReg = Registry .CurrentUser.CreateSubKey( "SOFTWARE\\SomeCompany\\SomeApp\\SomeVer" );
int nSomeVal = ( int )MyReg.GetValue( "SomeVal" , 0);
MyReg.SetValue( "SomeValue" , nSomeVal + 1);
MyReg.Close();
```

پاک کردن یک کلید

برای پاک کردن یک زیرکلید از متد RegistryKey.DeleteSubKey به دو صورت استفاده می‌کنیم.

```
public void DeleteSubKey( string subkey);
public void DeleteSubKey( string subkey, bool throwOnMissingSubKey);
```

در صورت اول زیرکلید subkey را به شرطی حذف می‌کند که زیرکلید مذکور موجود باشد و زیرکلید دیگری در زیر آن نباشد. در صورت دوم نیز این شروط برقرار است با این تفاوت که اگر زیرکلید مذکور یافت نشود و throwOnMissingSubKey مقدار true داشته باشد یک exception فرا می‌خواند.

پاک کردن کل یک درخت

برای پاک کردن کل یک درخت با همه‌ی کلیدهای فرزند و مقادیر آن‌ها از متد RegistryKey.DeleteSubKeyTree به دو صورت استفاده می‌کنیم.

```
public void DeleteSubKeyTree( string subkey);
public void DeleteSubKeyTree( string subkey, bool throwOnMissingSubKey);
```

دیگر با پارامترهای ارسالی در این متد آشنایی دارید و لازم به توضیح نیست.

پاک کردن یک مقدار

برای پاک کردن یک مقدار از متد RegistryKey.DeleteValue به دو صورت زیر استفاده می‌کنیم.

```
public void DeleteValue( string name);
public void DeleteValue( string name, bool throwOnMissingValue);
```

لیست کردن زیرکلیدها

برای به دست آوردن یک لیست از همه زیرکلیدهای یک شیء RegistryKey از متد RegistryKey.GetSubKeyNames به صورت زیر

استفاده می‌کنیم که یک آرایه رشته‌ای از نام زیرکلیدها را برمی‌گرداند.

```
public string [] GetSubKeyNames();
```

هم چنین می‌توانید برای شمردن تعداد زیرکلیدها از خصوصیت RegistryKey. SubKeyCount استفاده نمایید.

لیست کردن نام مقادیر

برای به دست آوردن یک لیست از همه مقادیری که در یک شیء RegistryKey وجود دارند از متد RegistryKey.GetValueNames به صورت زیر استفاده می‌کنیم که یک آرایه رشته‌ای از نام مقادیر را برمی‌گرداند.

```
public string [] GetSubKeyNames();
```

هم چنین می‌توانید برای شمردن تعداد زیرکلیدها از خصوصیت RegistryKey.ValueCount استفاده نمایید.

ثبت تغییرات به صورت دستی

برای ثبت تغییرات یا به اصطلاح فلاش کردن به صورت دستی می‌توانید از متد RegistryKey.Flush به صورت زیر استفاده نمایید. زمانی که از RegistryKey.Close استفاده می‌کنید فرآیند فلاش کردن به طور اتوماتیک انجام می‌گیرد.

```
public void Flush();
```

نظرات خوانندگان

نویسنده: سید ایوب کوکبی
تاریخ: ۱۳۹۲/۰۹/۲۴ ۱۲:۳۳

ممنونم با وجود دات نت و معماری جدید برنامه ها، معمولا در چه شرایطی استفاده از رجیستری دلیل قانع کننده ای داره؟ و اون دلیل قانع کننده چیه؟

نویسنده: محسن خان
تاریخ: ۱۳۹۲/۰۹/۲۴ ۱۲:۴۵

گاهی از اوقات برنامه‌های دسکتاپ نیاز پیدا می‌کنند تا به اطلاعات سیستمی دسترسی پیدا کنند یا آن‌ها را تغییر دهند. مثلا برنامه‌ای که نیاز داره در حین آغاز ویندوز، شروع به کار کنه، نیاز به ثبت خودش در رجیستری داره. یا اگر برنامه‌ای نیاز داشت که مثلا با Adobe reader کار کنه، می‌تونه مسیر دقیق نصب اون رو از رجیستری بخونه و از این موارد سیستمی زیاد هست.

عنوان: **SQL Instance**
نویسنده: م منفرد
تاریخ: ۱۳۹۲/۱۰/۱۲ ۱۹:۲۵
آدرس: www.dotnettips.info
گروه‌ها: C#, SQL Server, Registry

ممکن است کاربر بر روی سیستم خود نسخه‌های مختلفی از SQL Server را نصب کرده باشد. برای مثال SQL Express, SQL 2005, SQL 2008. و یا نسخه ای خاص (مثلا 2012) را چند بار روی سیستم خود نصب کرده باشد. SQL برای تفکیک این نسخه‌ها و نصب‌ها از مفهومی با عنوان Instance استفاده می‌کند. یعنی به هر نسخه نصب شده نامی یکتا می‌دهد تا بتوان به تفکیک به آنها دسترسی داشت.

برای اتصال به این نسخه‌ها باید در بخش آدرس سرور، از ترکیب نام سیستم و نام Instance به این شکل استفاده کرد:

SystemName\Instance

بعضی مواقع لازم است که لیست Instanceهای نصب شده روی سیستم کاربر را به دست آوریم. ADO.NET کلاسی به همین منظور تعبیه کرده که شبکه را جستجو کرده و SQL Instanceهای مختلف را که قابل دسترسی هستند را برای شما لیست می‌کند. استفاده از این کلاس بسیار ساده است:

```
using System.Data.Sql;

class Program
{
  static void Main()
  {
    // Retrieve the enumerator instance and then the data.
    SqlDataSourceEnumerator instance =
      SqlDataSourceEnumerator.Instance;
    System.Data.DataTable table = instance.GetDataSources();

    // Display the contents of the table.
    DisplayData(table);

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

  private static void DisplayData(System.Data.DataTable table)
  {
    foreach (System.Data.DataRow row in table.Rows)
    {
      foreach (System.Data.DataColumn col in table.Columns)
      {
        Console.WriteLine("{0} = {1}", col.ColumnName, row[col]);
      }
      Console.WriteLine("============================");
    }
  }
}
```

البته با توجه به اینکه شبکه را جستجو می‌کند در نرم افزار شما وقفه خواهد انداخت. خوب اگه بخواهیم Instanceهای نصب شده روی سیستم کاربر را پیدا کنیم چی؟ ساده‌ترین و سریعترین راه استفاده از رجیستری سیستم است. نام Instanceها در رجیستری ویندوز در آدرس زیر قابل دسترسی است:

```
HKEY_LOCAL_MACHINE\SOFTWARE\Microsoft\Microsoft SQL Server\Instance Names
```

برای استفاده از این کلید در c# می‌توان از کد زیر کمک بگیرید:

```
var key = Registry.LocalMachine.OpenSubKey(@"SOFTWARE\Microsoft\Microsoft SQL Server\Instance Names");

            foreach (string sk in key.GetSubKeyNames())
            {
                var rkey = key.OpenSubKey(sk);
                foreach (string s in rkey.GetValueNames())
                {
                    MessageBox.Show("Sql instance name:" + s);
                }
            }
```

فقط دو نکته قابل توجه است. برنامه باید در Any CPU کامپایل شود تا در سیستم‌های 64 بیتی بتوانید به محل درست رجیستری دسترسی پیدا کنید. چون نرم افزارهای 32 بیت در ویندوز 64 بیت در سیستم wow64 اجرا می‌شود که دسترسی به رجیستری آن در آدرس wow64 هر قسمت رجیستری است. بنابراین کد فوق در حالت Any CPU و غیر فعال بودن Prefer 32-bit قسمت Build در Properties برنامه به درستی اجرا می‌شود.

نکته: Default Instance در SQL مقدار MSSQLSERVER می‌باشد.

عنوان: آشنایی با متدهای Abstract و Virtual
نویسنده: امیر هاشم زاده
تاریخ: ۱۳۹۲/۱۰/۱۸ ۱۱:۴۰
آدرس: www.dotnettips.info
گروه‌ها: C#, Derived-Class

وقتی از کلاسی وهله‌ای را ایجاد می‌کنید، در واقع جدولی از اشاره گرها را به پیاده سازی متدهای آن ایجاد خواهید کرد. تصویر زیر مفهوم این جمله را بیان می‌کند.

متدها به روش‌های مختلفی تعریف می‌شوند و هر کدام رفتارهای مختلفی را در زمان ارث بری از خود نشان می‌دهند. روش استفاده استاندارد از آن‌ها مانند شکل بالاست ولی در صورتیکه بخواهید این روش را تغییر دهید می‌توانید به آن‌ها کلمات کلیدی اضافه کنید.

Abstract methodها

abstract متدها به هیچ جایی اشاره نمی‌کنند. مانند شکل زیر:

درصورتی که کلاسهای شما دارای اعضای abstract باشند، باید خودشان نیز abstract باشند. شما نمی‌توانید از این کلاس‌ها وهله‌ایی ایجاد نمایید؛ ولی می‌توانید از آن‌ها در ارث بری سایر کلاسها استفاده کنید و از سایر کلاس‌ها یک وهله ایجاد نمایید.

```
public abstract class Person
{
    public abstract void ShowInfo();
}

public class Teacher : Person
{
    public override void ShowInfo()
    {
        Console.WriteLine("I am a teacher!");
    }
}

public class Student : Person
{
    public override void ShowInfo()
    {
        Console.WriteLine("I am a student!");
    }
}
```

در مثال بالا رفتار متد ()ShowInfo به پیاده سازی آن در کلاس مربوطه بستگی دارد.

```
Person person = new Teacher();
person.ShowInfo();    // Shows 'I am a teacher!'

person = new Student();
person.ShowInfo();    // Shows 'I am a student!'
```

همانگونه که مشاهده می‌کنید متد ()ShowInfo کلاس‌های Student و Teacher از کلاس Person ارث بری کرده اند ولی زمانی که از آن‌ها درخواست نمایش اطلاعات می‌کنید هر کدام رفتارهای مختلفی از خود نشان می‌دهند. خب چه اتفاقاتی در پشت صحنه رخ می‌دهد در حالیکه آن‌ها از Person ارث بری کرده اند؟ تا قبل از پیاده سازی متد ()ShowInfo، اشاره گر به جایی اشاره نمی‌کند. ولی زمانیکه به عنوان مثال یک وهله از Student ایجاد می‌کنید، اشاره گر به پیاده سازی واقعی متد ()ShowInfo در کلاس Student اشاره می‌کند.

Virtual methodها

از کلاسهای دارای Virtual method می‌توان یک وهله ایجاد کرد و حتی امکان تحریف Virtual methodهای کلاس پایه در Derived-class وجود دارد. مانند کد زیر:

```
public class Person
{
    public virtual void ShowInfo()
    {
        Console.WriteLine("I am a person!");
    }
}

public class Teacher : Person
{
    public override void ShowInfo()
    {
        Console.WriteLine("I am a teacher!");
    }
}
```

در مثال بالا هم عضو Person.ShowInfo به جایی اشاره نمی‌کند، پس چرا می‌توانیم از آن یک وهله ایجاد کنیم؟!

باید توجه داشته باشید تصویر بالا با تصویر اول تفاوتی ندارد بدلیل اینکه virtual methodها به روش استاندارد پیاده سازی اشاره می‌کنند. با استفاده از کلمه کلید virtual شما به عنوان مثال به کلاس Person می‌گویید که متد ()ShowInfo می‌تواند پیاده سازی‌های دیگری هم **شاید** داشته باشد و سایر پیاده سازی‌های دیگر این متد باید با کلمه کلیدی override در کلاس دیگر (Teacher) مشخص شوند.

در ضمن در صورتیکه پیاده سازی دیگری از آن متد ارائه نشود از پیاده سازی کلاس پایه استفاده می‌شود.

```
public class Student : Person
{
}

Person person = new Teacher();
person.ShowInfo();    // Shows 'I am a teacher!'

person = new Student();
person.ShowInfo();    // Shows 'I am a person!'
```

نکته پایانی:

کلمه کلیدی new در متدهای کلاس‌های پایه و Derived مانند shadowing عمل می‌کند. برای بیان بهتر به کد زیر توجه کنید:

```
public class Person
{
    public void ShowInfo()
    {
        Console.WriteLine("I am Person");
    }
}

public class Teacher : Person
{
    public new void ShowInfo()
    {
        Console.WriteLine("I am Teacher");
    }
}
```

همانطور که ملاحظه می‌کنید متد ()ShowInfo در کلاس پایه و Derived دارای پیاده سازی متفاوتی است برای این نوع پیاده سازی متد ()ShowInfo در کلاس Teacher از کلمه کلیدی new برای عمل shadowing استفاده کرده ایم.

Person
ShowInfo
...
Console.WriteLine("I am Person");
Student
ShowInfo
...
Console.WriteLine("I am a student!");

عنوان: مقایسه value type و reference type
نویسنده: امیر هاشم زاده
تاریخ: ۱۳۹۲/۱۰/۲۳ ۱۰:۵۵
آدرس: www.dotnettips.info
گروه‌ها: C#, class, struct

در سی شارپ دو نوع class و struct وجود دارد که تقریباً مشابه یکدیگرند در حالیکه یکی از آنها value type و دیگری reference-type است.

struct چیست؟

structها مشابه classها هستند با این تفاوت که structها finalizer ندارند و از ارث بری پشتیبانی نمی‌کنند. structها کاملا مشابه classها تعریف می‌شوند و در تعریف آنها از کلمه کلیدی struct استفاده می‌شود. آنها شامل فیلدها، متدها، خصوصیت‌ها نیز می‌شوند. در زیر نحوه تعریف آن را مشاهده می‌کنید:

```
struct Point
{
   private int x, y;             // private fields

   public Point (int x, int y)   // constructor
   {
         this.x = x;
         this.y = y;
   }

   public int X                  // property
   {
         get {return x;}
         set {x = value;}
   }

   public int Y
   {
         get {return y;}
         set {y = value;}
   }
}
```

value type و reference type

تفاوت دیگری که بین class و struct، از اهمیت ویژه‌ای برخوردار است آن است که **classها** reference-type و **structها** value-type هستند و در زمان اجرا با آنها متفاوت رفتار می‌شود و در ادامه به تشریح آن می‌پردازیم.

وقتی یک وهله از value-type ایجاد شود، یک فضای خالی از حافظه‌ی اصلی (RAM) برای ذخیره سازی مقدار آن تخصیص داده می‌شود. نوع‌های اصلی مانند int, float, bool و char از نوع value type هستند. در ضمن سرعت دسترسی به آنها بسیار بالاست.

ولی وقتی یک وهله از reference-type ایجاد شود، یک فضا برای object و فضایی دیگر برای اشاره‌گر به آن شیء در حافظه اصلی ذخیره می‌شود. در واقع دو فضا از حافظه برای ذخیره سازی آنها اشغال می‌شود. برای درک بهتر به مثال زیر توجه کنید:

```
Point p1 = new Point();          // Point is a *struct*
Form f1 = new Form();            // Form is a *class*
```

نکته: Point از نوع struct و Form از نوع reference است. در مورد اول، یک فضا از حافظه برای p1 تخصیص داده می‌شود و در مورد دوم، دو فضا از حافظه اصلی یکی برای ذخیره کردن اشاره‌گر f1 برای اشاره به Form object و دیگری برای ذخیره کردن Form object تخصیص داده می‌شود.

```
Form f1;                         // Allocate the reference
f1 = new Form();                 // Allocate the object
```

به قطعه کد زیر دقت کنید:

```
Point p2 = p1;
Form f2 = f1;
```

همانطور که قبلاً گفته شد p2، یک نوع struct است بنابراین در مورد اول مقدار p2 یک کپی از مقدار p1 خواهد بود ولی در مورد دوم، آدرس f1 را درون f2 کپی می‌کنیم در واقع f1 و f2 به یک شیء اشاره خواهند کرد. (یک شیء با 2 اشاره گر)
در سی شارپ، پارامترها (بصورت پیش فرض) بصورت یک کپی از آنها به متدها ارسال می‌شوند، یعنی اگر پارامتر از نوع value-type باشد یک کپی از آن وهله و اگر پارامتر reference-type یک کپی از آدرس ارسال خواهد شد. برای توضیح بهتر به مثال زیر توجه کنید:

```
Point myPoint = new Point (0, 0);      // a new value-type variable
Form myForm = new Form();              // a new reference-type variable
Test (myPoint, myForm);                // Test is a method defined below

void Test (Point p, Form f)
{
      p.X = 100;                       // No effect on MyPoint since p is a copy
      f.Text = "Hello, World!";        // This will change myForm’s caption since
                                       // myForm and f point to the same object
      f = null;                        // No effect on myForm
}
```

انتساب null به f درون متد Test هیچی اثری بر روی آدرس myForm ندارد چون f، یک کپی از آدرس myForm است.
حال می‌توانیم روش پیش فرض را با افزودن کلمه کلید ref تغییر دهیم. وقتی از ref استفاده کنیم متد با پارامترهای فراخوانی کننده (caller's arguments) بصورت مستقیم در تعامل است در کد زیر می‌توانیم تصور کنیم که پارامترهای p و f متد Test همان متغیرهای myPoint و myForm است.

```
Point myPoint = new Point (0, 0);      // a new value-type variable
Form myForm = new Form();              // a new reference-type variable
Test (ref myPoint, ref myForm);        // pass myPoint and myForm by reference

void Test (ref Point p, ref Form f)
{
      p.X = 100;                       // This will change myPoint’s position
      f.Text = “Hello, World!”;        // This will change MyForm’s caption
      f = null;                        // This will nuke the myForm variable!
}
```

در کد بالا انتساب null به f باعث تهی شدن myForm می‌شود بدلیل اینکه متد مستقیماً به آن دسترسی داشته است.

تخصیص حافظه

CLR اشیاء را در دو قسمت ذخیره می‌کند:

stack یا پشته

heap

ساختار stack یا پشته first-in last-out است که دسترسی به آن سریع است. زمانی که متدی فراخوانی می‌شود، CLR پشته را نشانه گذاری می‌کند. سپس متد data را به پشته جهت اجرا push می‌کند و زمانی که اجرایش به اتمام رسید، CLR پشته را تا محل نشانه گذاری شده مرحله قبل، پاک می‌کند (pop).

ولی ساختار heap بصورت تصادفی است. یعنی اشیاء در محل‌های تصادفی قرار داده می‌شوند بهمین دلیل آنها دارای 2 سربار memory manager و garbage-collector هستند.

برای آشنایی با نحوه استفاده پشته و heap به کد زیر توجه کنید:

```
void CreateNewTextBox()
{
      TextBox myTextBox = new TextBox();             // TextBox is a class
}
```

در این متد، ما یک متغیر محلی ایجاد کرده ایم که به یک شیء اشاره می‌کند.

پشته همیشه برای ذخیره سازی موارد زیر استفاده می‌شود:
قسمت reference متغیرهای محلی و پارامترهای از نوع reference-typed (مانند myTextBox)
متغیرهای محلی و پارامترهای متد از نوع value-typed (مانند integer, bool, char, DateTime و ...)

همچنین از heap برای ذخیره سازی موارد زیر استفاده می‌شود:
محتویات شیء از نوع reference-typed
هر چیزی که قرار است در شیء از نوع reference-typed ذخیره شود.

آزادسازی حافظه در heap

در کد بالا وقتی اجرای متد CreateNewTextBox به اتمام برسد متغیر myTextBox از دید (Scope) خارج می‌شود. بنابراین از پشته نیز خارج می‌شود ولی با خارج شدن myTextBox از پشته چه اتفاقی برای TextBox object رخ خواهد داد؟! پاسخ در garbage-collector نهفته است. garbage-collector بصورت خودکار عملیات پاکسازی heap را انجام می‌دهد و اشیائی که اشاره گر معتبر ندارند را حذف می‌نماید. در حالت کلی اگر شیء از حافظه خارج شد باید منابع سایر قسمت‌های اشغال شده توسط آن هم آزاد شود، که این آزاد سازی بعهده garbage-collector است.

حال آزاد سازی برای کلاسهایی که اینترفیس IDisposable را پیاده سازی می‌کنند به دو صورت انجام می‌شود:

دستی: با فراخوانی متد Dispose میسر است.

خودکار: افزودن شیء به Net Container. مانند Form, Panel, TabPage یا UserControl. این نگهدارندها این اطمینان را به ما می‌دهند در صورتیکه آنها از حافظه خارج شدند کلیه عضوهای آن هم از حافظه خارج شوند.

برای آزادسازی دستی می‌توانیم مانند کدهای زیر عمل کنیم:

```
using (Stream s = File.Create ("myfile.txt"))
{
   ...
}
```

یا

```
Stream s = File.Create ("myfile.txt");
try
{
   ...
}
finally
{
   if (s != null) s.Dispose();
```

```
}
```

مثالی از Windows Forms

فرض کنید قصد داریم فونت و اندازه یک ویندوز فرم را تغییر دهیم.

```
Size s = new Size (100, 100);          // struct = value type
Font f = new Font (“Arial”,10);        // class = reference type

Form myForm = new Form();

myForm.Size = s;
myForm.Font = f;
```

توجه کنید که ما در کد بالا از اعضای myForm استفاده کردیم نه از کلاسهای Font و Size که این دو گانگی قابل قبول است. حال به تصویر زیر که به پیاده سازی کد بالا اشاره دارد توجه کنید.

همانطور که مشاهد می‌کنید محتویات s و آدرس f را در Form object ذخیره کرده ایم که نشان می‌دهد تغییر در s برروی فرم تغییر ایجاد نمی‌کند ولی تغییر در f باعث ایجاد تغییر فرم می‌شود. Form object دو اشاره گر به Font object دارد.

In-Line Allocation (تخصیص درجا)

در قبل گفته شد برای ذخیره متغیرهای محلی از نوع value-typed از پشته استفاده می‌شود آیا شیء Size جدید هم در پشته ذخیره می‌شود؟ خیر، بدلیل اینکه آن متغیر محلی نیست و در شیء دیگر ذخیره می‌شود (در مثال بالا در یک فرم ذخیره شده است) که آن شیء هم در heap ذخیره شده است پس شیء جدید Size هم در heap ذخیر می‌شود که به این نوع ذخیره سازی In-Line گفته می‌شود.

تله (Trap)

فرض کنید کلاس Form بشکل زیر تعریف شده است:

```
class Form
{
      // Private field members
      Size size;
      Font font;

      // Public property definitions
      public Size Size
      {
            get    { return size; }
            set    { size = value; fire resizing events }
      }

      public Font Font
      {
            get    { return font;  }
            set    { font = value; }
      }
}
```

حال ما قصد داریم ارتفاع آن را دو برابر کنیم، بنابراین از کد زیر استفاده می‌کنیم:

```
myForm.ClientSize.Height = myForm.ClientSize.Height * 2;
```

ولی با خطای کامپایلر زیر روبرو می‌شویم:

```
Cannot modify the return value of 'System.Windows.Forms.Form.ClientSize' because it is not a variable
```

علت چیست؟ بدلیل اینکه myForm.ClientSize شیء Size که از نوع Struct است را بر می‌گرداند و این Struct از نوع value-typed است و این شیء یک کپی از اندازه فرم است و ما همزمان قصد دو برابر نمودن آن کپی را داریم که کامپایلر خطای بالا را نمایش می‌دهد.

برای توضیح بیشتر می‌توانید به این سوال مراجعه کنید و در تکمیل آن این لینک را هم بررسی کنید.

پس بنابراین کد بالا را به کد زیر اصلاح می‌کنیم:

```
myForm.ClientSize = new Size (myForm.ClientSize.Width, myForm.ClientSize.Height * 2);
```

برای اصلاح خطای کامپایلر، ما باید یک شیء جدیدی را برای اندازه فرم تخصیص بدهیم.

عنوان: **آشنایی با جنریک‌ها #1**
نویسنده: امیر هاشم زاده
تاریخ: ۱۳۹۲/۱۰/۲۹ ۱۶:۱۵
آدرس: www.dotnettips.info
گروه‌ها: C#, Generics, Generic Classes

طبق این معرفی ، جنریک‌ها باعث می‌شوند که نوع داده‌ای (data type) المان‌های برنامه در زمان استفاده از آن‌ها در برنامه مشخص شوند. به عبارت دیگر، جنریک به ما اجازه می‌دهد کلاس‌ها یا متدهایی بنویسیم که می‌توانند با هر نوع داده‌ای کار کنند.

نکاتی از جنریک‌ها:
برای به حداکثر رسانی استفاده مجدد از کد، type safety و کارایی است.
بیشترین استفاده مشترک از جنریک‌ها جهت ساختن کالکشن کلاس‌ها (collection classes) است.
تا حد ممکن از جنریک کالکشن کلاسها (generic collection classes) جدید فضای نام System.Collections.Generic بجای کلاس‌هایی مانند ArrayList در فضای نام System.Collections استفاده شود.
شما می‌توانید *اینترفیس جنریک* ، *کلاس جنریک* ، متد جنریک و *عامل جنریک* سفارشی خودتان تهیه کنید.
جنریک کلاس‌ها، ممکن است در دسترسی به متدهایی با نوع داده‌ای خاص محدود شود.
بوسیله reflection، می‌توانید اطلاعاتی که در یک جنریک در زمان اجرا (run-time) قرار دارد بدست آورید.

انواع جنریک ها:
کلاس‌های جنریک
اینترفیس‌های جنریک
متدهای جنریک
عامل‌های جنریک

در قسمت اول به معرفی کلاس جنریک می‌پردازیم.
کلاس‌های جنریک کلاس جنریک یعنی کلاسی که می‌تواند با چندین نوع داده کار کند برای آشنایی با این نوع کلاس به کد زیر دقت کنید:

```
using System;
using System.Collections.Generic;

namespace GenericApplication
{
    public class MyGenericArray<T>
    {
        // تعریف یک آرایه از نوع جنریک
        private T[] array;

        public MyGenericArray(int size)
        {
            array = new T[size + 1];
        }

        // بدست آوردن یک آیتم جنریک از آرایه جنریک
        public T getItem(int index)
        {
            return array[index];
        }

        // افزودن یک آیتم جنریک به آرایه جنریک
        public void setItem(int index, T value)
        {
            array[index] = value;
        }
    }
}
```

در کد بالا کلاسی تعریف شده است که می‌تواند بر روی آرایه‌هایی از نوع داده‌ای مختلف عملیات درج و حذف را انجام دهد. برای تعریف کلاس جنریک کافی است عبارت <T> بعد از نام کلاس خود اضافه کنید، سپس همانند سایر کلاس‌ها از این نوع داده ای در

کلاس استفاده کنید. در مثال بالا یک آرایه از نوع T تعریف شده است که این نوع، در زمان استفاده مشخص خواهد شد. (یعنی در زمان استفاده از کلاس مشخص خواهد شد که چه نوع آرایه ای ایجاد می‌شود)

در کد زیر نحوه استفاده از کلاس جنریک نشان داده شده است، همانطور که مشاهده می‌کنید نوع کلاس int و char در نظر گرفته شده است (نوع کلاس، زمان استفاده از کلاس مشخص می‌شود) و سپس آرایه هایی از نوع int و char ایجاد شده است و 5 آیتم از نوع int و char به آرایه‌های هم نوع افزوده شده است.

```
class Tester
{
        static void Main(string[] args)
        {
            // تعریف یک آرایه از نوع عدد صحیح
            MyGenericArray<int> intArray = new MyGenericArray<int>(5);

            // افزودن اعداد صحیح به آرایه ای از نوع عدد صحیح
            for (int c = 0; c < 5; c++)
            {
                intArray.setItem(c, c*5);
            }

            // بدست آوردن آیتم‌های آرایه ای از نوع عدد صحیح
            for (int c = 0; c < 5; c++)
            {
                Console.Write(intArray.getItem(c) + " ");
            }
            Console.WriteLine();

            // تعریف یک آرایه از نوع کاراکتر
            MyGenericArray<char> charArray = new MyGenericArray<char>(5);

            // افزودن کاراکترها به آرایه ای از نوع کاراکتر
            for (int c = 0; c < 5; c++)
            {
                charArray.setItem(c, (char)(c+97));
            }

            // بدست آوردن آیتم‌های آرایه ای از نوع کاراکتر
            for (int c = 0; c< 5; c++)
            {
                Console.Write(charArray.getItem(c) + " ");
            }
            Console.WriteLine();
            Console.ReadKey();
        }
}
```

زمانی که کد بالا اجرا می‌شود خروجی زیر بدست می‌آید:

```
0 5 10 15 20
a b c d e
```

عنوان: آشنایی با جنریک‌ها #2
نویسنده: امیر هاشم زاده
تاریخ: ۱۳۹۲/۱۰/۳۰ ۱۲:۰
آدرس: www.dotnettips.info
گروه‌ها: C#, Generics, Generic Interfaces

قبل از ادامه آموزش مفاهیم جنریک، در نظر داشتن این نکته ضروری است که مطالبی که در این سری مقالات ارائه می‌شود در سطح مقدماتی است و قصد من آشنا نمودن برنامه نویسانی است که با این مفاهیم ناآشنا هستند ولی با مطالعه این مقاله می‌توانند کدهای تمیزتر و بهتری تولید کنند و همینطور این مفاهیم ساده، پایه‌ای باشد برای فراگیری سایر نکات تکمیلی و پیچیده‌تر جنریک‌ها.

در قسمت قبلی ، نحوه تعریف کلاس جنریک شرح داده شد و در سری دوم اشاره‌ای به مفاهیم و نحوه پیاده سازی اینترفیس جنریک می‌پردازیم.
مفهوم اینترفیس جنریک همانند مفهوم اینترفیس در دات نت است. با این تفاوت که برای آن‌ها یک نوع عمومی تعریف می‌شود و نوع آن‌ها در زمان اجرا تعیین خواهد شد و کلاس بر اساس نوع اینترفیس، اینترفیس را پیاده سازی می‌کند. برای درک بهتر به نحوه تعریف اینترفیس جنریک زیر دقت کنید:

```
public interface IBinaryOperations<T>
{
   T Add(T arg1, T arg2);
   T Subtract(T arg1, T arg2);
   T Multiply(T arg1, T arg2);
   T Divide(T arg1, T arg2);
}
```

در کد بالا اینترفیسی از نوع جنریک تعریف شده است که دارای چهار متد با چهار خروجی و پارامترهای چنریک می‌باشد که نوع خروجی‌ها و نوع پارامترهای ورودی در زمان استفاده از اینترفیس تعیین می‌شوند که البته در بالا بطور خاص بیان شده است.
اینترفیسی داریم که دو ورودی از هر نوعی دریافت می‌کند و چهار عملی اصلی را بر روی آن‌ها انجام داده و خروجی آن‌ها را از همان نوع پارامتر ورودی تولید می‌کند. (بجای اینترفیس‌های مختلف عملیات چهار عمل اصلی برای هر نوع داده (data type)، یک اینترفیس کلی برای تمام data typeها)
در کلاس زیر نحوه پیاده سازی اینترفیس از نوع int را مشاهده می‌کنید که چهار عملی اصلی را برروی داده هایی از نوع int انجام می‌شود و چهار خروجی از نوع int تولید می‌شود.

```
public class BasicMath : IBinaryOperations<int>
{
   public int Add(int arg1, int arg2)
   { return arg1 + arg2; }

   public int Subtract(int arg1, int arg2)
   { return arg1 - arg2; }

   public int Multiply(int arg1, int arg2)
   { return arg1 * arg2; }

   public int Divide(int arg1, int arg2)
   { return arg1 / arg2; }
}
```

بعد از پیاده سازی اینترفیس حال نوبت به استفاده از کلاس می‌رسد که زیر نیز نحوه استفاده از کلاس نمایش داده شده است:

```
static void Main(string[] args)
{
   Console.WriteLine("***** Generic Interfaces *****\n");
   BasicMath m = new BasicMath();
   Console.WriteLine("1 + 1 = {0}", m.Add(1, 1));
   Console.ReadLine();
}
```

و در صورتیکه بخواهید کلاسی چهار عمل اصلی را بر روی نوع داده double انجام دهد کافیست کلاسی اینترفیس نوع double را

پیاده سازی کرده باشد. مانند کد زیر:

```
public class BasicMath : IBinaryOperations<double>
{
   public double Add(double arg1, double arg2)
   { return arg1 + arg2; }
   ...
}
```

برداشتی آزاد از این مقاله .

عنوان: ایجاد سرویس چندلایه‌ی WCF با Entity Framework در قالب پروژه - 9
نویسنده: حامد قنادی
تاریخ: ۱۳۹۲/۱۱/۰۱ ۲۲:۰
آدرس: www.dotnettips.info
گروه‌ها: C#, Entity framework, WCF

یک Windows Form جدید ایجاد کنید و نام آن را frmAddEditNews بگذارید.

برابر با شکل ویژگی‌های tblCategory، IsDeleted و tblNewsId را برابر با None کنید و tblCategoryId را از نوع Combobox انتخاب کنید. سپس با فشار فلش کنار tblNews گزینه‌ی Details را انتخاب کنید.

روی tblNews کلیک کرده آن‌را بکشید و روی فرم رها کنید. آن‌گاه ظاهر فرم و چیدمان کنترل‌ها را تنظیم کنید و دو دکمه ذخیره و لغو برابر با شکل در فرم ایجاد کنید:

کد روی‌داد دو دکمه را این‌گونه بنویسید:

```
private void btnCancel_Click(object sender, EventArgs e)
        {
            this.Close();
        }

        private void btnSave_Click(object sender, EventArgs e)
        {
            this.DialogResult = System.Windows.Forms.DialogResult.OK;
        }
```

در پایین فرم روی tblNewsBindingSource کلیک کنید و از قسمت Properties ویژگی Modifiers آن‌را برابر با Public کنید.

روی Combobox کلیک کنید، سپس ویژگی DataBinding -> Text آن‌را خالی کنید. سپس روی فلش بالای Combobox دسته خبر کلیک کنید و تنظیمات آن‌را مانند شکل زیر انجام دهید.

برای پرشدن آن کد زیر را در روی‌داد Load فرم این‌گونه بنویسید:

```
private void frmAddEditNews_Load(object sender, EventArgs e)
        {
            MyNewsService.MyNewsServiceClient MyNews = new MyNewsService.MyNewsServiceClient();
            tblCategoryIdComboBox.DataSource = MyNews.GetAllCategory();
        }
```

به فرم اصلی بازگردید و برای روی‌داد دکمه‌ی ویرایش چنین بنویسید:

```
private void btnEdit_Click(object sender, EventArgs e)
        {
            if (tblNewsDataGridView.CurrentRow == null)
            {
                MessageBox.Show("سطری برای ویرایش انتخاب کنید");
            }
            else
            {
                //tblNews news = tblNewsDataGridView.CurrentRow.DataBoundItem as tblNews;
                tblNews news  =
MyNews.GetNews(Convert.ToInt32(tblNewsDataGridView.CurrentRow.Cells["tblNewsId"].Value));
                frmAddEditNews frmAdd = new frmAddEditNews();
                frmAdd.tblNewsBindingSource.DataSource = news;
                if (frmAdd.ShowDialog() == DialogResult.OK)
                {
                    MyNews.EditNews(news);
                    tblNewsBindingSource.DataSource = MyNews.GetAllNews().Select(p => new {
p.tblNewsId, p.tblCategory.CatName, p.Title, p.Description, RegDate = MiladiToShamsi(p.RegDate) });
                }
            }
        }
```

در صورتی که متد GetAllNews را به صورت ساده به ویژگی DataSource دیتاگرید نسبت داده بودیم می‌توانستید از کد زیر برای مقداردهی به متغیر news بهره ببریم. ولی در حال حاضر این خط کد پیغام خطا می‌دهد. البته راه‌های دیگری برای حل این مشکل وجود دارد که در این درس قصد پرداختن به آن‌را ندارم.

```
tblNews news = tblNewsDataGridView.CurrentRow.DataBoundItem as tblNews;
```

کد مربوط به روی‌داد دکمه‌ی افزودن و حذف را نیز به صورت زیر بنویسید:

```
private void btnAdd_Click(object sender, EventArgs e)
        {
            tblNews news = new tblNews();
            frmAddEditNews frmAdd = new frmAddEditNews();
```

```
            frmAdd.tblNewsBindingSource.DataSource = news;
            if (frmAdd.ShowDialog() == DialogResult.OK)
            {
                MyNews.AddNews(news);
                tblNewsBindingSource.DataSource = MyNews.GetAllNews().Select(p => new { p.tblNewsId,
p.tblCategory.CatName, p.Title, p.Description, RegDate = MiladiToShamsi(p.RegDate) });
            }
        }

        private void btnRemove_Click(object sender, EventArgs e)
        {
            if (MessageBox.Show("هشدار","آیا با حذف این سطر اطمینان دارید؟",MessageBoxButtons.YesNo) ==
System.Windows.Forms.DialogResult.Yes)
            {

MyNews.DeleteNews(Convert.ToInt32(tblNewsDataGridView.CurrentRow.Cells["tblNewsId"].Value));
                tblNewsBindingSource.DataSource = MyNews.GetAllNews().Select(p => new { p.tblNewsId,
p.tblCategory.CatName, p.Title, p.Description, RegDate = MiladiToShamsi(p.RegDate) });
            }
        }
```

برنامه را اجرا کنید. کار ما کم و بیش به پایان رسیده است. شما یک پروژه‌ی ویندوز ساده با استفاده از WCF ای که از Entity Framework برای اتصال به پایگاه داده بهره می‌برد؛ ایجاد کردید. WCF بسیار گسترده‌تر از این است و در این‌جا تنها به بخشی از آن پرداختیم. احتمالاً در صورت استقبال خوانندگان در آینده درباره‌ی تنظیمات ریز WCF برای امنیت، سرعت، محدودیت و استفاده در محیط‌های مختلف خواهم نوشت.

شاد و پیروز باشید.

نظرات خوانندگان

نویسنده: وحید
تاریخ: ۱۳۹۲/۱۱/۰۳ ۶:۴۲

بسیار عالی بود

نویسنده: پژمان
تاریخ: ۱۳۹۲/۱۱/۰۳ ۱۸:۵۱

مرسی ، خیلی عالی بود. اگه میشه در مورد security in WCF مقاله بگذارید ممنون میشم. باز هم ممنون

نویسنده: پوریا منفرد
تاریخ: ۱۳۹۲/۱۱/۰۶ ۰:۴۱

سلام مطالب فوق العاده کاربردی هستند مشتاقانه منتظر ادامه این بحث هستیم
سوال:
در صورتی که بخوام از سرویس WCF روی یک سرور جدا استفاده کنم چطور با WinApp خودم به سرویس‌های WCF Server وصل شم؟ بدون واسطه Web App ?
و اینکه سرعت واکشی اطلاعات (رکوردهای زیاد 2 ، 3 هزارتا یا بیشتر) چگونه هست؟ با WCF و WinApp واسه نرم افزارهای سازمانی که تحت شبکه محلی و وایرلس داخل شهری هستن بخوام ازین روش استفاده شه آیا در بلند مدت با افزایش رکوردها به مشکل برخورد نمی‌کنم از نظر کار با دیتابیس و داده ها؟

نویسنده: پوریا منفرد
تاریخ: ۱۳۹۲/۱۱/۰۶ ۱:۱۸

تستی که من با تعداد رکوردها برای واکشی از دیتابیس انجام دادم به یه مشکل برخورد کردم:
زمانی که تعداد رکوردها زیر 100 تا باشه خب win app به راحتی اطلاعات رو بارگزاری می‌کنه
ولی وقتی بیش از این مقدار مثلا 288 رکورد در زمان اجرای پروژه به مشکل برخورد می‌کنم که فرم بارگزاری نمیشه و از حالت Start می‌پره بیرون
دلیلش چی می‌تونه باشه؟ محدودیت‌های وب سرویس؟ چطور و چگونه این مشکل رو برطرف کنیم؟
پیام Catch :

```
The maximum message size quota for incoming messages (65536) has been exceeded.
 To increase the quota, use the MaxReceivedMessageSize property on the appropriate binding element.
```

از پیام معلومه که از حداکثر مقدار دریافتی یک واکشی بیش از حد درخواست کردیم ...
یک راه حل جامع چی می‌تونه باشه؟

نویسنده: پوریا منفرد
تاریخ: ۱۳۹۲/۱۱/۰۶ ۱:۴۳

راه حلی که بنده پیدا کردم؛ تغییراتی در مقدار سایز پیام دریافتی به شکل زیر در appConfig مربوط به پروژه WinApp از این شکل :

```
<?xml version="1.0" encoding="utf-8" ?>
<configuration>
    <system.serviceModel>
        <bindings>
            <basicHttpBinding>
                <binding name="BasicHttpBinding_IMyNewsService" />
            </basicHttpBinding>
        </bindings>
```

```
        <client>
            <endpoint address="http://localhost:4636/SedaService.svc" binding="basicHttpBinding"
                bindingConfiguration="BasicHttpBinding_IMyNewsService"
contract="MyNewsService.IMyNewsService"
                name="BasicHttpBinding_IMyNewsService" />
        </client>
    </system.serviceModel>
</configuration>
```

به این شکل تغییر دهید :

```
<bindings>
            <basicHttpBinding>
                <binding maxReceivedMessageSize="2147483647" name="BasicHttpBinding_IMyNewsService" />
            </basicHttpBinding>
        </bindings>
```

حالا امنیت رو نمیدونم اینجا نقض کردم یا نه؟ لطفا اگر اطلاعاتی دارید راهنمایی بفرمایید که امنیت نقض شده یا نه؟ و کلا با یه عدد این شکلی که Max رو مشخص می‌کنه بنظرم نسبت به آینده نگری یک نرم افزار تجاری منطقی نیست...

نویسنده: حامد قنادی
تاریخ: ۱۳۹۲/۱۱/۰۶ ۶:۵۹

با درود و سپاس از همه‌ی همراهان.
همان‌سان که پیش‌تر هم نوشته ام می‌توانید سرویس WCF را در IIS یک سرور دیگر راه‌اندازی کنید و آدرس آی‌پی و یا DNS مربوط به آن‌را در WinApp خود استفاده کنید.
هنوز به تنظیمات خاص Web.Config نرسیده ایم در آن‌جا به امنیت و محدودیت‌ها خواهم پرداخت.
پیروز باشید.

نویسنده: علیرضا طهوری
تاریخ: ۱۳۹۲/۱۲/۲۴ ۱۱:۱

سلام
با تشکر از این آموزش. فقط یه خواهش دارم . اگر براتون مقدوره در مورد امنیت برای تبادل داده‌ها در wcf این مبحث رو ادامه بدید.

ممنون

نویسنده: حسین پاکدل
تاریخ: ۱۳۹۳/۰۱/۱۷ ۱۵:۱۴

با عرض سلام؛ آیا برای استفاده از یک وب سرویس هم باید مبحث "Dependency Injection" در نظر گرفته بشه؟ اگر پاسخ مثبت است لطفا با مثالی ساده توضیح دهید روش کار به چه صورت است؟ ممنون

نویسنده: وحید نصیری
تاریخ: ۱۳۹۳/۰۱/۱۷ ۱۶:۵۴

- برای تولید سرویس: « پیاده سازی InstanceProvider برای سرویس‌های WCF »
- برای استفاده از سرویس: در همان لایه سرویس برنامه از آن استفاده کنید. مباحث و مفاهیم تزریق وابستگی‌های آن تفاوتی با حالت استفاده از یک دیتابیس یا یک WebClient ندارد و یکی است .

نویسنده: حسین پاکدل
تاریخ: ۱۳۹۳/۰۱/۲۶ ۱۴:۳۴

مشکلی که در استفاده از وب سرویس دارم اینه که وب سرویس در ازای بعضی از درخواست‌ها خطایی از نوع System.ServiceModel.FaultException بر میگردونه. این خطا رو میتوان در Controller با HandleError به View ی خاصی هدایت کرد. اما من قصد دارم پیام بازگردانده شده از نوع FaultException رو به کاربر نمایش بدم. برای این کار چه باید کرد؟
ممنون

نویسنده: وحید نصیری
تاریخ: ۱۳۹۳/۰۱/۲۶ ۱۴:۴۲

- به دلایل امنیتی نباید جزئیات خطاها را به کاربران نمایش داد. صرفا به نمایش صفحات و پیام‌های عمومی بسنده کنید.
+ در مورد MVC و مدیریت خطاها در آن بحث مجزایی در سایت وجود دارد (^)؛ قسمت «دسترسی به اطلاعات استثناء در صفحه نمایش خطاها»

نویسنده: خلوت گزیده
تاریخ: ۱۳۹۳/۰۴/۱۵ ۹:۱

سلام ممنون از مطالب خوب و ارزشمندی که گذاشتید
فقط یه سوال دارم که هر چی گشتم نتونستم حل کنم
اونم نحوه پابلیش و خروجی گرفتن از برنامه برای IIS هست
ممنون میشم راهنمایی کنید که پروژه ای رو که ساختید چطوری میشه پابلیش کرد
بازهم ممنون

نویسنده: محسن خان
تاریخ: ۱۳۹۳/۰۴/۱۵ ۹:۸

- در قسمت هفتم ، تنظیمات برنامه‌های وب آن بحث شده. پابلیش آن کپی و پیست پروژه در یک دایرکتوری مجازی در IIS است (یعنی فرقی با راه اندازی یک وب سایت معمولی ASP.NET نداره در اساس).
- اگر به خطایی برخوردید در این بین، عین خطا را ارسال کنید تا بیشتر بشود بحث کرد.

نویسنده: خلوت گزیده
تاریخ: ۱۳۹۳/۰۴/۱۵ ۱۲:۲۲

سلام
فکر می‌کنم ایراد از تنظیمات IIS ویندوز باشه و ربطی به برنامه نویسی نداره
اول که IIS تنظیم می‌کردم این Error میداد

```
HTTP Error 404.3 - Not Found The page you are requesting cannot be served because of the
extension configuration. If the page is a script, add a handler. If the file should be downloaded,
add a MIME map
```

که کارهایی که در وبلاگ زیر گفته شده انجام دادم
http://blogs.msdn.com/b/ericwhite/archive/2010/05/11/getting-started-building-a-wcf-web-service.aspx
الان پیغام زیر رو میده

```
Server Error in '/MyNewService' Application.
Could not load type 'System.ServiceModel.Activation.HttpModule' from assembly 'System.ServiceModel,
Version=3.0.0.0, Culture=neutral, PublicKeyToken=b77a5c561934e089'.
```

ممنون میشم اگه بتونی مشکل منو حل کنی
با تشکر

نویسنده: محسن خان
تاریخ: ۱۳۹۳/۰۴/۱۵ ۱۲:۴۱

خطای آخری رو که ارسال کردید اینجا توضیح داده شده: http://support.microsoft.com/kb/2015129

خلاصه‌اش اینکه باید دستور aspnet_regiis.exe /iru رو در خط فرمان اجرا کنید. محل قرارگیری برنامه aspnet_regiis.exe در پوشه ویندوز هست (فایل‌ها رو جستجو کنید تا یافت بشه).

نویسنده: خلوت گزیده
تاریخ: ۱۳۹۳/۰۴/۱۵ ۱۳:۲۲

ممنون دوست عزیز

درضمن باید تنظیمات زیر روا هم اعمال کنید

```
Everywhere the problem to this solution was mentioned as re-registering aspNet by using
aspnet_regiis.exe. But this did not work for me.
Though this is a valid solution (as explained beautifully here)
but it did not work with Windows 8.
For Windows 8 you need to Windows features and enable everything under ".Net Framework 3.5" and ".Net
Framework 4.5 Advanced Services".
```

عنوان: **فراخوانی داینامیک اسمبلی‌های آفیس در برنامه**
نویسنده: مهران ر
تاریخ: ۱۳۹۲/۱۱/۰۷ ۱۶:۲۵
آدرس: www.dotnettips.info
گروه‌ها: C#, Reflection, Dynamic Programming

در بسیاری از پروژه‌های دات نت، نیاز به استفاده از فایلهای نرم افزار آفیس، از قبیل ورد و اکسل و ... وجود دارد. برای مثال گاهی لازم است اطلاعات یک گرید، یا هر منبع داده‌ای، در قالب اکسل به کاربر نمایش داده شود. بدین شکل که این فایلها در زمان اجرا ساخته شده و به کاربر نمایش داده شود .حال فرض کنید شما روی سیستم خودتان Office2007 را نصب کرده اید و به اسمبلی‌های این ورژن دسترسی دارید. البته بدون نیاز به نصب آفیس نیز میتوان به این توابع دسترسی داشت و از آنها در برنامه استفاده کرد که همان استفاده از Primary Interop Assemblies میباشد. مشکلی که ممکن است پیش آید این است که در کامپیوترهای کاربران ممکن است ورژن‌های مختلفی از آفیس نصب باشد مانند 2003 -2007-2010-2013 و اگر با ورژن اسمبلی‌هایی که فراخوانی‌های فایلهای اکسل از طریق آن انجام شده باشد متفاوت باشد، برنامه اجرا نمی‌شود .

در حالت معمول برای نمایش یک فایل آفیس مثل اکسل در برنامه، ابتدا اسمبلی مربوطه را (اکسل در این مثال) که به نام Microsoft.Office.Interop.Excel میباشد به اسمبلی‌های برنامه اضافه کرده (از طریق add reference) و برای نمایش یک فایل اکسل در زمان اجرا از کدهای زیر استفاده مینماییم :

```
try
            {
                var application =
(Microsoft.Office.Interop.Excel.Application)Activator.CreateInstance(Type.GetTypeFromProgID("Excel.Appl
ication"));
               Workbook  wrkBook;
                var  wbk = application.Workbooks;
                 wrkBook = wbk.Add();
                wrkBook.Activate();
                application.Visible = true;
             }
            catch (Exception ex)
            {
                Error(ex.Message);
            }
```

حال اگر آفیس 2010 به عنوان مثال در سیستم ما نصب باشد، ورژن این اسمبلی 14 می‌باشد و اگر این برنامه را در کامپیوتر کلاینتی که آفیس 2007 بر روی آن نصب باشد انتشار دهیم اجرا نمیشود. برای حل این مشکل بنده با استفاده از روش dynamic این موضوع را حل کردم و بنظر می‌رسد راه‌های دیگری نیز برای حل آن وجود داشته باشد.

در این روش با توجه به ورژن آفیسی که بر روی سیستم کاربر نصب شده اسمبلی مربوطه را از سیستم کاربر لود کرده و فایلهای آفیس را اجرا مینماییم. در ابتدا تشخیص میدهیم چه ورژنی از آفیس بر روی سیستم کاربر نصب است :

```
string strVersion = null;
                dynamic objEApp =Activator.CreateInstance(Type.GetTypeFromProgID("Excel.Application"));
                if (objEApp.Version == "12.0")
                {
                    strVersion = "2007";
                }
                else if (objEApp.Version == "14.0")
                {
                    strVersion = "2010";
                }
```

روش دیگر برای انجام اینکار استفاده از اطلاعات رجیستری ویندوز است :

```
string strEVersionSubKey = "\\Excel.Application\\CurVer";
            string strValue = null; //Value Present In Above Key
            string strVersion = null; //Determines Excel Version
            RegistryKey rkVersion = null; //Registry Key To Determine Excel Version
            rkVersion = Registry.ClassesRoot.OpenSubKey(strEVersionSubKey, false); //Open Registry Key
```

```
            if ((rkVersion != null)) //If Key Exists
            {
                strValue = (string)rkVersion.GetValue(string.Empty); //Get Value
                strValue = strValue.Substring(strValue.LastIndexOf(".") + 1); //Store Value
                switch (strValue) //Determine Version
                {
                    case "11":
                        strVersion = "2003";
                        break;

                    case "12":
                        strVersion = "2007";
                        break;

                    case "14":
                        strVersion = "2010";
                        break;

                }
```

حال با استفاده از تابع assembly.load() اسمبلی مورد نیاز را لود کرده و در برنامه استفاده مینماییم :

```
if (strVersion == "2007")
                {
                    string strAssemblyOff2007 =
                        "Microsoft.Office.Interop.Excel, Version=12.0.0.0, Culture=neutral,
PublicKeyToken=71e9bce111e9429c";
                    try
                    {
                        Assembly xslExcelAssembly = Assembly.Load(strAssemblyOff2007); //Load Assembly
                        Type type = xslExcelAssembly.GetTypes().Single(t => t.Name ==
"ApplicationClass");
                        dynamic application = Activator.CreateInstance(type);
                        var workbooks = application.Workbooks;
                        var workbook = workbooks.Add();
                        var worksheet = workbook.Worksheets[1];
                        workbook.Activate();
                        application.Visible = true;
                    }
                    catch (Exception ex)
                    {
                    }
                }
```

در این حالت بدون اینکه بدانیم بر روی سیستم کاربر چه ورژنی از آفیس نصب است میتوان فایلهای آفیس را در زمان اجرا لود کرده و استفاده کرد .

نظرات خوانندگان

نویسنده: حسین
تاریخ: ۱۳۹۲/۱۱/۱۹ ۲۰:۳۳

با تشکر از مقاله مفید شما
من یه بار توی یه پروژه یک تمپلت ورد ایجاد کردم و توش انواع اقسام چارت‌ها و جدول‌ها رو توش رسم کردم و کلی هم روش کار کردم تا گزارش خوبی از کار در بیارد
واقعا اطلاع نداشتم با ورژن‌ها مختلف اجرا نمیشه!
الان عذاب وجدان گرفتم :)

عنوان: **مدیریت AccessViolationException در برنامه‌های دات نت 4 به بعد**
نویسنده: وحید نصیری
تاریخ: ۱۳۹۲/۱۱/۱۴ ۱:۲۰
آدرس: www.dotnettips.info
گروه‌ها: C#, C++, exception handling

فرض کنید که از یک برنامه‌ی native ویندوز برای تهیه تصاویر سایت‌ها در یک برنامه‌ی وب استفاده می‌کنید و صبح که به سایت سر زده‌اید پیام در دسترس نبودن سایت قابل مشاهده است. مشکل از کجا است؟!

یک مثال ساده

```
using System;

namespace AccessViolationExceptionSample
{
    class Program
    {
        private static unsafe void AccessViolation()
        {
            byte b = *(byte*)(8762765876);
        }

        static void Main(string[] args)
        {
            try
            {
                AccessViolation();
            }
            catch (Exception ex)
            {
                Console.WriteLine(ex);
            }

            Console.WriteLine("Press a key...");
            Console.ReadKey();
        }
    }
}
```

برنامه‌ی کنسول فوق را پس از فعال سازی Allow unsafe code در قسمت تنظیمات پروژه، کامپایل کرده و سپس آن‌را خارج از VS.NET اجرا کنید. احتمالا انتظار دارید که قسمت catch این کد حداقل اجرا شود و سپس سطر «کلیدی را فشار دهید» ظاهر گردد. اما ... خیر! کل پروسه کرش کرده و هیچ پیام خطایی را دریافت نخواهید کرد. اگر به لاگ‌های ویندوز مراجعه کنید پیام زیر قابل مشاهده است:

```
System.AccessViolationException.  Attempted to read or write protected memory.
 This is often an indication that other memory is corrupt.
```

و این نوع مسایل هنگام کار با کتابخانه‌های C و C++ زیاد ممکن است رخ دهند. نمونه‌ی آن استفاده از WebControl دات نت است یا هر برنامه‌ی native دیگری. در این حالت اگر برنامه‌ی شما یک برنامه‌ی وب باشد، عملا سایت از کار افتاده‌است. به عبارتی پروسه‌ی ویندوزی آن کرش کرده و بلافاصله از طرف ویندوز خاتمه یافته است.

چرا قسمت catch اجرا نشد؟

از دات نت 4 به بعد ، زمانیکه دسترسی غیرمجازی به حافظه صورت گیرد، برای مثال دسترسی به یک pointer آزاد شده، استثنای حاصل، توسط برنامه catch نشده و اجازه داده می‌شود تا برنامه کلا کرش کند. به این نوع استثناءها Corrupted State Exceptions یا CSE گفته می‌شود. اگر نیاز به مدیریت آن‌ها توسط برنامه باشد، باید به یکی از دو طریق زیر عمل کرد:
الف) از ویژگی HandleProcessCorruptedStateExceptions بر روی متد فراخوان کتابخانه‌ی native باید استفاده شود. برای مثال در کدهای فوق خواهیم داشت:

```
[HandleProcessCorruptedStateExceptions]
  static void Main(string[] args)
  {
```

ب) و یا فایل کانفیگ برنامه را ویرایش کرده و چند سطر ذیل را به آن اضافه کنید:

```
<?xml version="1.0" encoding="utf-8" ?>
<configuration>
  <runtime>
     <legacyCorruptedStateExceptionsPolicy enabled="true" />
  </runtime>
</configuration>
```

در این حالت مدیریت اینگونه خطاها در کل برنامه همانند برنامه‌های تا دات نت 3.5 خواهد شد.

نظرات خوانندگان

نویسنده: سوین
تاریخ: ۱۳۹۲/۱۱/۱۶ ۱۹:۴۶

با سلام
من در فایل کانفیگ یه WPF App ، تغییرات گفته شده را قرار دادم و خطای زیر رو در vs داد
.The type initializer for 'System.Windows.Application' threw an exception

و بیرون از vs اصلا اجرا نشد خیلی نیاز دارم به این مورد ، چون یه پروژه دارم که درست اجرا میشه اما بعضی مواقع برنامه کرش میکنه و نمی‌تونم catch کنم . با تشکر

نویسنده: وحید نصیری
تاریخ: ۱۳۹۲/۱۱/۱۶ ۱۹:۵۹

- مطلب فوق بیشتر مرتبط است به استثناهای کتابخانه‌های native استفاده شده در برنامه‌های دات نت. برای سایر موارد باید در فایل App.xaml.cs موارد ذیل را بررسی کنید:

```
public partial class App
    {
        public App()
        {
            this.DispatcherUnhandledException += appDispatcherUnhandledException;
            AppDomain.CurrentDomain.UnhandledException += CurrentDomain_UnhandledException;
        }
```

+ نمونه تنظیم زیر در فایل app.config یک برنامه WPF کار می‌کند (آزمایش شد):

```
<?xml version="1.0"?>
<configuration>
  <startup>
    <supportedRuntime version="v4.0" sku=".NETFramework,Version=v4.0"/>
  </startup>
  <runtime>
     <legacyCorruptedStateExceptionsPolicy enabled="true" />
  </runtime>
</configuration>
```

نویسنده: سوین
تاریخ: ۱۳۹۲/۱۲/۰۲ ۹:۳۸

با سلام؛ من یه پروژه با WPF نوشتم اما یه ایراد داره و اونم اینه که مثلا فرم1 رو 20 بار اجرا می‌کنی خطا نمی‌ده اما بار 21 ام برنامه کرش میکنه و اصلا نمیشه catch کرد. متن خطا در Log ویندوز اینه

```
Error 01 :
Application: MyWPFApp.exe
Framework Version: v4.0.30319
Description: The process was terminated due to an unhandled exception.
Exception Info: exception code c0000005, exception address 77D52239

Error 02 :
Faulting application name: MyWPFApp.exe, version: 1.0.0.0, time stamp: 0x52d550ac
Faulting module name: ntdll.dll, version: 6.1.7601.17514, time stamp: 0x4ce7b96e
Exception code: 0xc0000005
Fault offset: 0x00032239
Faulting process id: 0xa28
Faulting application start time: 0x01cf113ae6813d88
Faulting application path: R:\Source\MyWPFApp\bin\Debug\MyWPFApp.exe
Faulting module path: C:\Windows\SYSTEM32\ntdll.dll
Report Id: 460eda62-7d33-11e3-a572-ac220bc99cf8

Information :
```

```
Fault bucket , type 0
Event Name: APPCRASH
Response: Not available
Cab Id: 0

Problem signature:
P1: MyWPFApp.exe
P2: 1.0.0.0
P3: 52d550ac
P4: ntdll.dll
P5: 6.1.7601.17514
P6: 4ce7b96e
P7: c0000005
P8: 00032239
P9:
P10:

Attached files:
C:\Users\Administrator\AppData\Local\Temp\WERE9B3.tmp.WERInternalMetadata.xml
C:\Users\Administrator\AppData\Local\Temp\WER16AC.tmp.appcompat.txt
C:\Users\Administrator\AppData\Local\Temp\WER18A1.tmp.hdmp
C:\Users\Administrator\AppData\Local\Temp\WER3BFA.tmp.mdmp

These files may be available here:
C:\Users\Administrator\AppData\Local\Microsoft\Windows\WER\ReportQueue\AppCrash_MyWPFApp.exe_125fc667a6
9fcc31c463a5e1b4032657c4ce830_cab_0ac03d3e

Analysis symbol:
Rechecking for solution: 0
Report Id: 460eda62-7d33-11e3-a572-ac220bc99cf8
```

نویسنده: محسن خان
تاریخ: ۱۳۹۲/۱۲/۰۲ ۱۱:۳۷

از چه کامپوننتی استفاده کردی ؟ بهتره اون فایل‌های کرش دامپ dmp رو براشون ارسال کنی.

نویسنده: سوین
تاریخ: ۱۳۹۲/۱۲/۰۲ ۱۹:۲۵

با سلام

از کامپوننت شرکت‌های ثالث استفاده نکردم . آیا راه حل کلی برای پیدا کردن چنین خطاهایی وجود نداره ، اینترنت رو هم سرچ کردم اما کمک زیادی نکرد که بشه فهمید مشکل از چیه و قبل ارسال این پست 2 ساعت تمام آزمایش کردم خطا نداد اما بعضی مواقع این اتفاق می‌افته .

در ضمن این برنامه WPF App که برای اوتوماسیون اداری نوشته شده و از EF 6.2 ، قفل سخت افزاری (که بدون قفل هم این ایراد رو می‌ده)

نویسنده: وحید نصیری
تاریخ: ۱۳۹۲/۱۲/۰۳ ۰:۴۶

مثال WPF ایی که AccessViolation عمدی دارد: WpfApplicationC0000005.zip

به فایل‌های App.config و App.xaml.cs آن دقت کنید.

پروژه را کامپایل کرده و خارج از VS.NET اجرا کنید. خطا را نمایش می‌دهد ولی کرش نمی‌کند.

نویسنده: سوین
تاریخ: ۱۳۹۲/۱۲/۰۳ ۹:۲۱

با سلام

من تگ startup رو به صورت زیر نوشته بودم آیا می‌تونه تاثیر داشته باشه

```
<startup useLegacyV2RuntimeActivationPolicy="true">
    <supportedRuntime version="v4.0" sku=".NETFramework,Version=v4.0"/>
    <requiredRuntime version="v4.0.20506"/>
</startup>
```

باز تست می‌کنم ببینم چی میشه ، انشاالله که درست بشه . با تشکر

نویسنده: وحید نصیری
تاریخ: ۱۳۹۲/۱۲/۰۳ ۹:۳۰

تنظیم یاد شده مربوط به تگ runtime است.

عنوان: **ذخیره سریع ده‌ها هزار رکورد دیتاتیبل در اکسل**
نویسنده: مجتبی فخاری
تاریخ: ۱۳۹۲/۱۱/۱۶ ۱۹:۳۵
آدرس: www.dotnettips.info
گروه‌ها: C#, Excel

یکی از امکاناتی که در نرم افزارهای اتوماسیون مورد نیاز است، ذخیره اطلاعات، داخل فایل اکسل است و در صورتی که حجم این اطلاعات زیاد باشد زمان زیادی صرف این عمل خواهد شد. در زیر کلاسی را برای شما آماده نموده‌ام که 20 هزار رکورد را در 4 ثانیه، در فایل اکسل ذخیره می‌نماید. در این روش با استفاده از یک آرایه به نام rawdata این عمل انجام شده.
توضیحات کدها نیز به صورت comment در کنار کدها آورده شده است.

```
//using System;
    //using System.Data;
    //using Microsoft.Office.Interop.Excel;

    class FastExportingMethod
    {
        //System.Data.DataTable dt= دیتاتیبل که حاوی اطلاعات می‌باشد
        //outputPath= مسیر ذخیره شدن
        public static string ExportToExcel(System.Data.DataTable dt, string outputPath)
        {
            try
            {
                // ساخت یک شی اکسل
                ApplicationClass excelApp = new ApplicationClass();

                // ساخت یک WorkBook جدید
                Workbook excelWorkbook = excelApp.Workbooks.Add(Type.Missing);

                int sheetIndex = 0;

                // ساخت آرایه به طول تعداد سطرهای دیتاتیبل+1 و تعداد ستونهای دیتاتیبل
                object[,] rawData = new object[dt.Rows.Count + 1, dt.Columns.Count];

                // کپی نام ستون‌های دیتاتیبل به عنوان هدر برای فایل اکسل در اولین سطر از آرایه
                for (int col = 0; col < dt.Columns.Count; col++)
                {
                    rawData[0, col] = dt.Columns[col].ColumnName;
                }

                // کپی اطلاعات دیتاتیبل به داخل آرایه
                for (int col = 0; col < dt.Columns.Count; col++)
                {
                    for (int row = 0; row < dt.Rows.Count; row++)
                    {
                        rawData[row + 1, col] = dt.Rows[row].ItemArray[col].ToString();
                    }
                }

                // محاسبه نام ستونهای اکسل

                string finalColLetter = string.Empty;
                string colCharset = "ABCDEFGHIJKLMNOPQRSTUVWXYZ";
                int colCharsetLen = colCharset.Length;

                if (dt.Columns.Count > colCharsetLen)
                {
                    finalColLetter = colCharset.Substring((dt.Columns.Count - 1) / colCharsetLen - 1,
1);
                }

                finalColLetter += colCharset.Substring((dt.Columns.Count - 1) % colCharsetLen, 1);

                // ساخت یک Sheet
                Worksheet excelSheet = (Worksheet)excelWorkbook.Sheets.Add(
                    excelWorkbook.Sheets.get_Item(++sheetIndex),
                    Type.Missing, 1, XlSheetType.xlWorksheet);
                //تنظیم نام شیت به نام دلخواه
                excelSheet.Name = "List";
                //تنظیم خاصیت راست به چپ برای نمایش اطلاعات
                excelSheet.DisplayRightToLeft = true;


                // تعیین محدوده سطرها و ستونها
                string excelRange = string.Format("A1:{0}{1}",finalColLetter, dt.Rows.Count + 1);
                //انتقال اطلاعات از آرایه به شیت مورد نظر
```

```
                excelSheet.get_Range(excelRange, Type.Missing).Value2 = rawData;

                // ضخیم کردن اولین سطر برای عنوان ستونها
                ((Range)excelSheet.Rows[1, Type.Missing]).Font.Bold = true;

                // تنظیم عرض ستونها به اندازه محتوای ستونها
                for (int col = 0; col < dt.Columns.Count; col++)
                {
                    ((Range)excelSheet.Columns[col + 1]).EntireColumn.AutoFit();
                }


                //ذخیره و بستن  Workbook
                excelWorkbook.SaveAs(outputPath, XlFileFormat.xlWorkbookNormal, Type.Missing,
                    Type.Missing, Type.Missing, Type.Missing, XlSaveAsAccessMode.xlExclusive,
                    Type.Missing, Type.Missing, Type.Missing, Type.Missing, Type.Missing);
                excelWorkbook.Close(true, Type.Missing, Type.Missing);
                excelWorkbook = null;

                excelApp.Quit();
                excelApp = null;

                // Collect the unreferenced objects
                GC.Collect();
                GC.WaitForPendingFinalizers();

                return "اطلاعات شما در مسیر انتخاب شده ذخیره گردید";
            }
            catch (Exception ex)
            {
                //بدست آوردن کد خطا برای مدیریت خطاها
                int code = System.Runtime.InteropServices.Marshal.GetExceptionCode();

                return ex.Message + code;
            }
        }
    }
```

نظرات خوانندگان

نویسنده: راضیه
تاریخ: ۱۳۹۲/۱۱/۱۷ ۱۹:۵۶

کتابخانه NPOI نیز دارای سرعت بسیار بالایی است.پیشنهاد می‌کنم حتما امتحانش کنید

http://npoi.codeplex.com/

نمونه: http://www.zachhunter.com/2010/05/getting-started-with-npoi/

نویسنده: وحید نصیری
تاریخ: ۱۳۹۲/۱۱/۱۷ ۲۰:۲

برای کار با اکسل، اگر 2007 به بعد مدنظر است، بدون نیاز به نصب اکسل می‌شود با EPPlus هم کار کرد.

عنوان: **خواندن سریع اطلاعات فایل اکسل و ذخیره در بانک SQL**
نویسنده: مجتبی فخاری
تاریخ: ۱۳۹۲/۱۱/۱۷ ۱۸:۳۵
آدرس: www.dotnettips.info
گروه‌ها: C#, SQL Server, Excel

یکی از زمانبرترین عملیاتها در نرم افزارهای اتوماسیون، خواندن اطلاعات از فایل‌های اکسل با حجم بالا است. در صورتی که این کار را می‌توان با استفاده از کلاس SqlBulkCopy به سرعت انجام داد. در ادامه نحوه استفاده از این کلاس، همراه نمونه کدها آورده شده است.

توضیحات به صورت Comment است.

```
try
            {
//انتخاب فایل اکسل
                OpenFileDialog ofd = new OpenFileDialog();
                ofd.Title = "انتخاب فایل اکسل حاوی اطلاعات";
                ofd.Filter = "2003 فایل اکسل (*.xls)|*.xls|فایل اکسل 2007 به بعد(*.xlsx)|*.xlsx";
                if (ofd.ShowDialog() == DialogResult.OK)
                {
                    string excelConnectionString = "";
                    string SourceFilePath = ofd.FileName;
//ایجاد کانکشن استرینگ برا خواندن کل اطلاعات از فایل اکسل و ریختن آنها در یک دیتاتیبل به نام dt
                    if (System.IO.Path.GetExtension(ofd.FileName) == ".xlsx")
//تشخیص نوع فایل اکسل برای ایجاد کانکشن استرینگ برای نسخه‌های مختلف اکسل
                        excelConnectionString = "Provider=Microsoft.ACE.OLEDB.12.0;Data Source=" +
SourceFilePath + ";Extended Properties=Excel 12.0";
                    else if (System.IO.Path.GetExtension(ofd.FileName) == ".xls")
                        excelConnectionString = "Provider=Microsoft.Jet.Oledb.4.0;Data Source=" +
SourceFilePath + ";Extended Properties=Excel 8.0";

                    DataTable dt = new DataTable("tblinfos");


                    using (System.Data.OleDb.OleDbConnection connection = new
System.Data.OleDb.OleDbConnection(excelConnectionString))
                    {

                        connection.Open();

                        System.Data.OleDb.OleDbDataAdapter da = new
System.Data.OleDb.OleDbDataAdapter("SELECT * FROM [List$]", connection);//List نام شیت در فایل اکسل
است

                        da.Fill(dt);
                        connection.Close();

                    }

                    using (System.Data.OleDb.OleDbConnection connection = new
System.Data.OleDb.OleDbConnection(excelConnectionString))
                    {
                        this.Cursor = Cursors.WaitCursor;

                        connection.Open();
//ایجاد ارتباط با بانک اس کیو ال
                        string sqlConnectionString = @"Data
Source=.\SQLEXPRESS;AttachDbFilename=|DataDirectory|\BankDPR.mdf;Max Pool Size=6000; Connection
Timeout=50;Integrated Security=True;User Instance=True";


                        Dataaccess db = new Dataaccess();
                        DataTable dtr = db.select("Select Top(1) * From tblinfos");//بدست آورن نام
ستون‌های جدول مورد نظر برای تطبیق با ستونهای فایل اکسل
                        //*******************   Fast Copy
                        using (System.Data.SqlClient.SqlBulkCopy bulkCopy = new
System.Data.SqlClient.SqlBulkCopy(sqlConnectionString,
System.Data.SqlClient.SqlBulkCopyOptions.KeepIdentity))//تعریف یک شی از کلاس SqlBulkCopy
                        {
                            for (int i = 1; i < dtr.Columns.Count; i++)
                            {
                                bulkCopy.ColumnMappings.Add(dt.Columns[i - 1].Caption,
dtr.Columns[i].Caption);//نام ستونهای فایل اکسل با نام ستونهای جدول ColumnMappings با استفاده از خاصیت
تطبیق داده می‌شود sql مورد نظر بانک
                            }

                            bulkCopy.DestinationTableName = "tblinfos";//مشخص نمودن نام جدول که قرار
```

```
است اطلاعات درون ان کپی گردد
                            bulkCopy.WriteToServer(dt);//به dt انجام عملیات کپی اطلاعات از دیتاتیبل با نام
بانک

                        }

                        this.Cursor = Cursors.Arrow;
                        MessageBox.Show("اطلاعات از فایل شما خوانده شد");

                    }
                }
            }
            catch (Exception ex)
            {
                this.Cursor = Cursors.Arrow;

                MessageBox.Show(ex.Message, "error");
            }
```

نظرات خوانندگان

نویسنده: محسن خان
تاریخ: ۱۳۹۲/۱۱/۱۷ ۲۱:۲۲

احتمالا این قطعه کد مستقیما از داخل سورس یکی از پروژه‌های شما بیرون آمده (مثال نیست؛ واقعی هست). می‌شد کمی اون رو refactor کرد مثلا یک متد از داخلش بیرون آورد که این متد داخلش مسیج باکس نباشه یا باز کردن یک صفحه دیالوگ و تغییر کرسر. try و catch هم نداشته باشه چون باید در یک سطح بالاتر catch بشه مشکلاتش. اون selectها مثلا می‌شدند چند پارامتر، برای اینکه این کد قابلیت استفاده مجدد بهتری پیدا کنه. یا مثلا اون sqlConnectionString از داخل کدها بیرون می‌اومد و می‌شد یک پارامتر جدید. نمایش کرسر هم داخل این متد قرار نمی‌گرفت. نام جدول نهایی هم مثلا یک پارامتر دیگر می‌شد برای سهولت استفاده مجدد و همچنین تست بهتر یک قطعه کوچک از کار. خود متد اصلی هم می‌شد دو متد کوچک‌تر؛ یکی کار load رو انجام می‌داد و دیگری کار insert سریع.

نویسنده: پالادین
تاریخ: ۱۳۹۲/۱۱/۱۹ ۱۳:۳۸

من هم با دوستمون محسن خان موافقم. اتفاق‌ها زیادی در مورد این کد باید بیافته تا به یک کد خوب تبدیل بشه. شما کاملا به صورت Smart UI کد زدید. امیدوارم وقت بزارید و این کد رو بهینه کنید. توی همین سایت مثال‌های خوبی واسه‌ی یادگیری هست..

نویسنده: مجتبی فخاری
تاریخ: ۱۳۹۲/۱۱/۲۰ ۱۱:۳۶

بله این کد رو من دقیقا از یکی از برنامه هام آوردم و مال چندین سال پیشه.
اما خوبیش اینه که بدون خطا جواب میده و فقط کافیه اسم جداول رو عوض کنید.

نویسنده: محمدی
تاریخ: ۱۳۹۲/۱۲/۱۲ ۱۰:۲۹

با سلام و تشکر. این برنامه پیغام میدهد List وجود ندارد.

```
SELECT * FROM [List$]
```

لطفا راهنمایی بفرمایید من چطور میتوانم نام شیت‌ها را بدست بیاورم؟

نویسنده: وحید نصیری
تاریخ: ۱۳۹۲/۱۲/۱۲ ۱۰:۴۷

در کد فوق اکثر قسمت‌ها بر اساس یک سری پیش فرض مشخص تهیه شده‌اند و آن‌ها را تبدیل به پارامتر متغیر نکرده‌اند. نام شیت، همان نام برگه جاری است:

نویسنده: احمد زواری
تاریخ: ۱۳۹۳/۰۴/۱۱ ۲۳:۴۰

سلام. من از این روش برای خواندن فایل موجود در آدرس http://www.site.com/market/export.aspx?type=dtod&date=20120405 استفاده کردم، ولی متاسفانه فیلدهای عددی را NULL مقدار میگیرد. مشکل از چیست و چه جوری برطرف میشود؟

نویسنده: وحید نصیری
تاریخ: ۱۳۹۳/۰۴/۱۲ ۰:۲۴

این فایل استاندارد اکسل نیست. خروجی آن را با نوت پد باز کنید؛ یک فایل HTML معمولی است.

نویسنده: احمد زواری
تاریخ: ۱۳۹۳/۰۴/۱۲ ۳:۳۰

الان من نیاز شدید دارم این فایل رو تبدیل به SQL کنم، و چون هر روز این فایل تغییر میکنه نمیتونم توی فایل تغییری بدم، چه راه حلی هست برای این کار؟

نویسنده: وحید نصیری
تاریخ: ۱۳۹۳/۰۴/۱۲ ۹:۱۹

فایل‌های HTML را با استفاده از کتابخانه‌ی HTML Agility Pack پردازش می‌کنند.

عنوان: **حرکت دادن و تغییر اندازه کنترل‌های فرم در زمان اجرا با استفاده از ماوس**
نویسنده: زمرد2020
تاریخ: ۱۳۹۲/۱۱/۲۰ ۱۷:۲۰
آدرس: www.dotnettips.info
گروه‌ها: C#, Windows forms, Controls

برخی اوقات نیاز است در یک فرم ویندوزی، کنترل‌های آن‌را در حال اجرا با استفاده از ماوس جابجا کنیم و یا اندازه‌ی آن‌ها را تغییر بدیم.
در وب راهکارهای مختلفی برای این کار ارائه شده، ولی این راه‌ها معمولا یا فقط برای تغییر مکان و یا فقط برای تغییر اندازه کنترل‌ها ارائه شده‌اند. من یکی از مقالات کد پروجکت را که به جابجا کردن کنترل‌ها پرداخته بود، توسعه دادم که امکان تغییر اندازه هم به آن اضافه شود. مقاله‌ی من (به زبان انگلیسی) در اینجا قرار دارد.
چون از کلاس و متدهای استاتیک استفاده کردم، روش استفاده از این کلاس ساده بوده و افزودن قابلیت تغییر اندازه و جابجایی زمان اجرا با ماوس برای هر کنترل فقط با یک خط کد قابل انجام است:

```
ControlMoverOrResizer.Init(button1);
```

نحوه‌ی استفاده از کلاس:
برای فعال کردن قابلیت تغییر اندازه و جابجایی یک کنترل در حال اجرای برنامه با موس ما باید متد Init از کلاس MoveAndResizeControls را فراخوانی کنیم و کنترل را به عنوان پارامتر به آن بفرستیم.

```
ControlMoverOrResizer.Init(button1);
```

اگر که ما بخواهیم به همراه تغییر کنترل ، خواص container آن را هم تغییر دهیم. باید کنترل container را به عنوان پارامتر دوم به متد مذکور ارسال کنیم.

```
ControlMoverOrResizer.Init(button2,panel1);
```

برخی اوقات ممکن است که ما فقط بخواهیم که یا کنترل‌ها را جابجا کنیم و یا اندازه‌ی آنها را تغییر دهیم؛ در این مواقع ما باید خاصیت WorkType کلاس MoveAndResizeControls را تغییر دهیم به یکی از مقادیر ذیل تغییر دهیم .

```
internal enum MoveOrResize
        {
            Move,
            Resize,
            MoveAndResize
        }
```

مثالی از نحوه‌ی کار با کلاس :

```
using System;
using System.Windows.Forms;
using ControlManager;
namespace MoveAndResizeControls
{
    public partial class Form1 : Form
    {
        public Form1()
        {
            InitializeComponent();
        }

        private void Form1_Load(object sender, EventArgs e)
        {
            ControlMoverOrResizer.Init(button1);
            ControlMoverOrResizer.Init(groupBox1);
            ControlMoverOrResizer.Init(textBox1);
            ControlMoverOrResizer.Init(button2,panel1);
            comboBox1.SelectedIndex = 0;
        }
```

```
        private void comboBox1_SelectedIndexChanged(object sender, EventArgs e)
        {
            switch (comboBox1.SelectedIndex)
            {
                case 0:
                    ControlMoverOrResizer.WorkType=ControlMoverOrResizer.MoveOrResize.MoveAndResize;
                    break;
                case 1:
                    ControlMoverOrResizer.WorkType = ControlMoverOrResizer.MoveOrResize.Move;
                    break;
                case 2:
                    ControlMoverOrResizer.WorkType = ControlMoverOrResizer.MoveOrResize.Resize;
                    break;
            }
        }
    }
}
```

نکته :بعد از انجام تغییرات، جهت ذخیره وضعیت کنترل‌ها و بازیابی مجدد آنها می‌توان از متدهای زیر استفاده کرد:

```
GetSizeAndPositionOfControlsToString  , SetSizeAndPositionOfControlsFromString
```

دانلود سورس

شکل حالت اولیه:

شکل حالت نتیجه:

Demo
select Work:
Move and Resize
save in string
load from string
groupBox1
textbox1
button1
button2 (dock=fill in panel1)

عنوان: Design Pattern: Factory
نویسنده: مهسا حسن کاشی
تاریخ: ۱۳۹۲/۱۱/۲۶ ۲۲:۱۰
آدرس: www.dotnettips.info
گروه‌ها: C#, Design patterns, SOLID, Object Oriented Design, ServiceFactory, Patterns, Software architecture

الگوهای طراحی، سندها و راه حلهای از پیش تعریف شده و تست شده‌ای برای مسائل و مشکلات روزمره‌ی برنامه نویسی می‌باشند که هر روزه ما را درگیر خودشان می‌کنند. هر چقدر مقیاس پروژه وسیعتر و تعداد کلاسها و اشیاء بزرگتر باشند، درگیری برنامه نویس و چالش برای مرتب سازی و خوانایی برنامه و همچنین بالا بردن کارآیی و امنیت افزون‌تر می‌شود. از همین رو استفاده از ساختارهایی تست شده برای سناریوهای یکسان، امری واجب تلقی می‌شود.

الگوهای طراحی از لحاظ سناریو، به سه گروه عمده تقسیم می‌شوند:

1- تکوینی: هر چقدر تعداد کلاسها در یک پروژه زیاد شود، به مراتب تعداد اشیاء ساخته شده از آن نیز افزوده شده و پیچیدگی و درگیری نیز افزایش می‌یابد. راه حل‌هایی از این دست، تمرکز بر روی مرکزیت دادن به کلاسها با استفاده از رابطها و کپسوله نمودن (پنهان سازی) اشیاء دارد.

2- ساختاری: گاهی در پروژه‌ها پیش می‌آید که می‌خواهیم ارتباط بین دو کلاس را تغییر دهیم. از این رو امکان از هم پاشی اجزایِ دیگر پروژه پیش می‌آید. راه حلهای ساختاری، سعی در حفظ انسجام پروژه در برابر این دست از تغییرات را دارند.

3- رفتاری: گاهی بنا به مصلحت و نیاز مشتری، رفتار یک کلاس می‌بایستی تغییر نماید. مثلا چنانچه کلاسی برای ارائه صورتحساب داریم و در آن میزان مالیات 30% لحاظ شده است، حال این درصد باید به عددی دیگر تغییر کند و یا پایگاه داده به جای مشاهده‌ی تعدادِ معدودی گره از درخت، حال می‌بایست تمام گره‌ها را ارائه نماید.

الگوی فکتوری:

الگوی فکتوری در دستهء اول قرار می‌گیرد. من در اینجا به نمونه‌ای از مشکلاتی که این الگو حل می‌نماید، اشاره می‌کنم:

فرض کنید یک شرکت بزرگ قصد دارد تا جزییات کامل خرید هر مشتری را با زدن دکمه چاپ ارسال نماید. چنین شرکت بزرگی بر اساس سیاستهای داخلی، بر حسب میزان خرید، مشتریان را به چند گروه مشتری معمولی و مشتری ممتاز تقسیم می‌نماید. در نتیجه نمایش جزییات برای آنها با احتساب میزان تخفیف و به عنوان مثال تعداد فیلدهایی که برای آنها در نظر گرفته شده است، تفاوت دارد. بنابراین برای هر نوع مشتری یک کلاس وجود دارد.

یک راه این است که با کلیک روی دکمه‌ی چاپ، نوع مشتری تشخیص داده شود و به ازای نوع مشتری، یک شیء از کلاس مشخص شده برای همان نوع ساخته شود.

```
// Get Customer Type from Customer click on Print Button
            int customerType = 0;

            // Create Object without instantiation
            object obj;


            //Instantiate obj according to customer Type
            if (customerType == 1)
            {
                obj = new Customer1();
            }
            else if (customerType == 2)
            {
                obj = new Customer2();
            }
            // Problem:
            //          1: Scattered New Keywords
            //          2: Client side is aware of Customer Type
```

همانگونه که مشاهده می‌نمایید در این سبک کدنویسی غیرحرفه‌ای، مشکلاتی مشهود است که قابل اغماض نیستند. در ابتدا سمت کلاینت دسترسی مستقیم به کلاسها دارد و همانگونه که در شکل بالا قابل مشاهده است کلاینت مستقیما به کلاس وصل است. مشکل دوم عدم پنهان سازی کلاس از دید مشتری است.

راه حل: این مشکل با استفاده از الگوی فکتوری قابل حل است. با استناد به الگوی فکتوری، کلاینت تنها به کلاس فکتوری و یک اینترفیس دسترسی دارد و کلاسهای فکتوری و اینترفیس، حق دسترسی به کلاسهای اصلی برنامه را دارند.

گام نخست: در ابتدا یک class library به نام Interface ساخته و در آن یک کلاس با نام ICustomer می سازیم که متد Report() را معرفی می‌نماید.

Interface//

```
namespace Interface
{
    public interface ICustomer
    {
        void Report();
    }
}
```

گام دوم: یک class library به نام MainClass ساخته و با Add Reference کلاس Interface را اضافه نموده، در آن دو کلاس با نام Customer1, Customer2 می‌سازیم و using Interface را Import می‌نماییم. هر دو کلاس از ICustomer ارث می‌برند و سپس متد Report() را در هر دو کلاس Implement می‌نماییم.

```
// Customer1
using System;
```

```
using Interface;

namespace MainClass
{
    public class Customer1 : ICustomer
    {
        public void Report()
        {
            Console.WriteLine("این گزارش مخصوص مشتری نوع اول است");
        }
    }
}

//Customer2
using System;
using Interface;

namespace MainClass
{
   public class Customer2 : ICustomer
    {
        public void Report()
        {
            Console.WriteLine("این گزارش مخصوص مشتری نوع دوم است");
        }
    }
}
```

گام سوم: یک class library به نام FactoryClass ساخته و با Add Reference کلاس Interface, MainClass را اضافه نموده، در آن یک کلاس با نام clsFactory می سازیم و using Interface, using MainClass را Import می‌نماییم. پس از آن یک متد با نام getCustomerType ساخته که ورودی آن نوع مشتری از نوع int است و خروجی آن از نوع Interface-ICustomer و بر اساس کد نوع مشتری object را از کلاس Customer1 و یا Customer2 می‌سازیم و آن را return می نماییم.

```
//Factory
using System;
using Interface;
using MainClass;

namespace FactoryClass
{
    public class clsFactory
    {
        static public ICustomer getCustomerType(int intCustomerType)
        {
            ICustomer objCust;
            if (intCustomerType == 1)
            {
                objCust = new Customer1();
            }
            else if (intCustomerType == 2)
            {
                objCust = new Customer2();
            }
            else
            {
                return null;
            }
            return objCust;
        }
    }
}
```

گام چهارم (آخر): در قسمت UI Client، کد نوع مشتری را از کاربر دریافت کرده و با Add Reference کلاس Interface, FactoryClass را اضافه نموده (دقت نمایید هیچ دسترسی به کلاس‌های اصلی وجود ندارد)، و using Interface, using FactoryClass را Import می‌نماییم. از clsFactory تابع getCustomerType را فراخوانی نموده (به آن کد نوع مشتری را پاس می‌دهیم) و خروجی آن را که از نوع اینترفیس است به یک object از نوع ICustomer نسبت می‌دهیم. سپس از این object متد Report را فراخوانی می‌نماییم. همانطور که از شکل و کدها مشخص است، هیچ رابطه ای بین UI(Client) و کلاسهای اصلی برقرار نیست.

```
//UI (Client)
using System;
using FactoryClass;
using Interface;

namespace DesignPattern
{
    class Program
    {
        static void Main(string[] args)
        {
            int intCustomerType = 0;
            ICustomer objCust;
            Console.WriteLine("نوع مشتری را وارد نمایید");
            intCustomerType = Convert.ToInt16(Console.ReadLine());
            objCust = clsFactory.getCustomerType(intCustomerType);
            objCust.Report();
            Console.ReadLine();
        }
    }
}
```

عنوان: **ایجاد نقشه سایت (Site Map) داینامیک**
نویسنده: شقایق اشتری
تاریخ: ۱۳۹۲/۱۲/۰۲ ۰:۳۰
آدرس: www.dotnettips.info
گروه‌ها: ASP.Net, C#, sitemap

همان طور که می‌دانید نقشه‌ی سایت علاوه بر استفاده از MetaTagها و Url Routingها و ... یکی از نکات اصلی برای سایت شماست که در نتایج گوگل برای جستجو کنندگان نمایش داده شود .

در این مقاله من قصد دارم که به شما چگونگی ایجاد و کار با یک SiteMap داینامیک را آموزش دهم .
منظور از SiteMap داینامیک این است که به ازای هر مطلبی که شما در سایت تان ایجاد ، ویرایش یا حذف می‌کنید چنانچه دارای صفحه ای در سایت باشد SiteMap تغییر می‌کند .

ساختار یک SiteMap به صورت زیر است :

```
<?xml version="1.0" encoding="utf-8"?>
<urlset xmlns="http://www.sitemaps.org/schemas/sitemap/0.9">
  <url>
<loc>Url Page 1</loc>
<lastmod>2014-02-20</lastmod>
<changefreq>always</changefreq>
<priority>1</priority>
  </url>
  <url>
<loc>Url Page 2</loc>
<lastmod>2014-02-20</lastmod>
<changefreq>always</changefreq>
<priority>1</priority>
  </url>
</urlset>
```

به ازای هر مطلبی که به سایتتان اضافه می‌کنید چنانچه آن مطلب جهت نمایش دارای Url باشد ، باید یک تگ Url به SiteMap اضافه شود.
تگ loc موجود در تگ url آدرس صفحه را مشخص می‌کند .
تگ lastmod تاریخ اضافه کردن یا آخرین ویرایش را نمایش می‌دهد .
تگ changefreq که دوره‌ی بروز رسانی صفحه را مشخص می‌کند .
تگ priority الویت صفحه را مشخص می‌کند .

که من در کد نویسی تگ changefreq را always و تگ priority را 1 قرار دادم.

در فایل ضمیمه یک کلاس به اسم updateSiteMap.cs وجود دارد که تابع آن شامل دو پارامتر ورودی مانند زیر است :

```
public void UpdateSiteMap(string Addr,string NewOpr)
```

پارامتر Addr که آدرس صفحه ای است که شما می‌خواهید به SiteMap اضافه شود .
پارامتر NewOpr که می‌تواند شامل یکی از سه مقدار زیر باشد: add ، edit ، delete

اگر پارامتر NewOpr دارای مقدار add باشد یعنی مقدار موجود در پارامتر Addr را بهSiteMap اضافه کن . یعنی یک تگ url به SiteMap اضافه کن .
اگر پارامتر NewOpr دارای مقدار edit باشد یعنی مقدار موجود در تگ <lastmod> را ویرایش کن . یعنی تاریخ آخرین بروزرسانی تغییر می‌کند .
اگر پارامتر NewOpr دارای مقدار delete باشد یعنی تگ url ایی که محتوای تگ loc آن برابر است با مقدار موجود در پارامتر

Addr را از SiteMap حذف کن.

این بخش از کد موجود در فایل ضمیمه uupdateSiteMaop.cs قسمت edit و delete نقشه‌ی سایت را انجام می‌دهد .

```
if (NewOpr != "add")
{
XmlElement xmlElement = xmlDoc.DocumentElement;
if (xmlElement.ChildNodes != null)
{
  foreach (XmlElement myElement in xmlDoc.DocumentElement)
  {
if (myElement.ChildNodes[0].InnerText == Addr)
{
  if (NewOpr != "delete")
myElement.ChildNodes[1].InnerText = DateTime.Now.ToString("yyyy-MM-dd");
  else
myElement.ParentNode.RemoveChild(myElement);
break;
}
  }
}
}
```

و بخش else دستور بالا قسمت Add را انجام می‌دهد .یعنی کدهای زیر:

```
else
{
string ns="http://www.sitemaps.org/schemas/sitemap/0.9";
XmlNode url = xmlDoc.CreateNode(XmlNodeType.Element, "url",ns );
XmlNode loc = xmlDoc.CreateNode(XmlNodeType.Element, "loc", ns);
XmlNode lastmod = xmlDoc.CreateNode(XmlNodeType.Element, "lastmod", ns);
XmlNode changefreq = xmlDoc.CreateNode(XmlNodeType.Element, "changefreq", ns);
XmlNode priority = xmlDoc.CreateNode(XmlNodeType.Element, "priority", ns);
loc.InnerText = Addr;
lastmod.InnerText = DateTime.Now.ToString("yyyy-MM-dd");
changefreq.InnerText = "always";
priority.InnerText = "1";
url.AppendChild(loc);
url.AppendChild(lastmod);
url.AppendChild(changefreq);
url.AppendChild(priority);
xmlDoc.DocumentElement.AppendChild(url);
}
```

اگر اطلاعاتی را به جدول اضافه می‌کنید و می‌خواهید Url صفحه‌ی مربوط به آن اطلاعات برای شما در SiteMap اضافه شود بعد از ذخیره شدن اطلاعات در جدول بلافاصله کد زیر را اضافه می‌کنید :

```
//
Add Info In Table
//
updateSiteMap updateSiteMap = new updateSiteMap();
updateSiteMap.UpdateSiteMap("Url Page", "add");
```

برای قسمت ویرایش هم پس از آنکه اطلاعات را ویرایش کردید چنانچه برای آن اطلاعات صفحه ای را در SiteMap درج کرده اید کد زیر را می‌نویسید :

```
updateSiteMap updateSiteMap = new updateSiteMap();
updateSiteMap.UpdateSiteMap("Url Page", "edit");
```

برای قسمت حذف هم اگر شما اطلاعاتی را از جدول حذف می‌کنید چنانچه برای آن اطلاعات صفحه ای در SiteMap درج کرده اید کد زیر را می‌نویسید :

```
updateSiteMap updateSiteMap = new updateSiteMap();
updateSiteMap.UpdateSiteMap("Url Page", "delete");
```

موفق باشید .

Files.zip

نظرات خوانندگان

نویسنده: وحید نصیری
تاریخ: ۱۳۹۲/۱۲/۰۲ ۰:۴۶

چند نکته تکمیلی

- نسخه‌ی MVC این بحث را در اینجا می‌توانید مطالعه کنید. این نسخه را می‌توان تبدیل به یک فایل ashx وب فرم‌ها نیز کرد. context آن قابل انتقال است و با کد انتهای بحث یکی است.

- در وب فرم‌ها الزاما نیازی نیست تا حتما یک فایل XML ثابت را روی سخت دیسک ذخیره کنید. می‌شود با استفاده از مباحث routing در وب فرم‌ها ، فایلی‌را که وجود خارجی ندارد، ایجاد کرد:

```
void Application_Start(object sender, EventArgs e)
{
    RouteTable.Routes.Add(new Route(
        "sitemap.xml", new PageRouteHandler("~/sitemap.ashx")));
}
```

در اینجا مسیر sitemap.ashx که یک Generic handler است، به مسیر sitemap.xml نگاشت می‌شود (کاربر اگر sitemap.xml را در مرورگر وارد کند، مدیریت آن توسط sitemap.ashx انجام خواهد شد).

- سپس برای تولید فایل XML به صورت پویا فقط از sitemap.ashx استفاده کنید به همراه مباحث caching اطلاعات :

```
public class Sitemap : IHttpHandler
    {
        public void ProcessRequest(HttpContext context) {
            context.Response.ContentType = "text/xml";
            //مابقی نکات بحث در اینجا
            context.Response.End();
        }

        public bool IsReusable {
            get {  return false; }
        }
    }
```

نویسنده: صابر فتح الهی
تاریخ: ۱۳۹۲/۱۲/۰۲ ۲۳:۲۹

متشکر از زحمتی که برای این مطلب کشیدید.

یک ایراد ویرایشی کوچک:

که من در کد نویسی تگ changefreq را 1 و تگ priority را always قراار دادم.

ظاهرا جابجا نوشتین

نویسنده: سلمان کاظمی
تاریخ: ۱۳۹۲/۱۲/۲۸ ۱:۴۴

من یه مشکلی با سایت مپ دارم و اون اینه که من یک فرم دارم که ممکنه چند فرم مختلف این فرم را نمایش میدهند. حالا این روش شما مشکل را حل می‌کنه؟مثلا اگه اسم فرم من A باشه می‌خوام وقتی از فرم B فرم A را نمایش میدهم سایت مپ بصورت B-->A نشون داده بشه و اگه از فرم C فرم A را نشون میدم شکل سایت مپ بصورت C-->A باشه

نویسنده: وحید نصیری
تاریخ: ۱۳۹۲/۱۲/۲۸ ۹:۶

sitemap معرفی شده در مطلب جاری، صرفا برای موتورهای جستجو نوشته شده است و کاربرد مدنظر شما را ندارد.

نویسنده: محسن نجف زاده

تاریخ: ۱۳۹۳/۰۵/۲۸ ۹:۴۶

من از این روش (تولید فایل XML به صورت پویا از طریق sitemap.ashx) استفاده کردم ولی متاسفانه خطای زیر رو میده *(از دات نت 4.5 استفاده می‌کنم)*

```
Type 'sitemap' does not inherit from 'System.Web.UI.Page'.
```

نویسنده: وحید نصیری
تاریخ: ۱۳۹۳/۰۵/۲۸ ۱۰:۱۳

نیاز به کمی اصلاح دارد. جنریک هندلرها برای معرفی به سیستم مسیریابی باید از طریق یک IRouteHandler معرفی شوند:

```
// نحوه‌ی معرفی جنریک هندلر به سیستم مسیریابی
public class RouteHandler : System.Web.Routing.IRouteHandler
{
    public IHttpHandler GetHttpHandler(System.Web.Routing.RequestContext requestContext)
    {
        return new Sitemap();
    }
}

// Global.asax file
void Application_Start(object sender, EventArgs e)
{
    RouteTable.Routes.Add(new Route("sitemap.xml", new RouteHandler()));
}
```

خلاصه آن به صورت یک متد الحاقی

عنوان: **ایجاد سیستم وضعیت آب و هوا مانند گوگل (بخش اول)**
نویسنده: شقایق اشتری
تاریخ: ۱۳۹۲/۱۲/۱۰ ۱۱:۳۰
آدرس: www.dotnettips.info
گروه‌ها: ASP.Net, C#, ASP.Net MVC, MVC, xml

در این آموزش قصد دارم چگونگی ایجاد یک سیستم اعلام وضعیت آب و هوا را مشابه آنچه که در سایت گوگل می‌بینید برای شما توضیح دهم. باید توجه داشت من این آموزش را با ASP.NET MVC نوشتم ولی شما می‌توانید با اندک تغییراتی در کدها، آنرا در ASP.NET وب فرمز نیز استفاده کنید. برای گرفتن آب و هوای هر شهر از Rssهای اعلام وضعیت آب و هوای یاهو استفاده می‌کنم و توضیح خواهم داد که چگونه با Rss آن کار کنید.
Rss آب و هوای هر شهر در یاهو به صورت یک لینک یکتا می‌باشد؛ به شکل زیر :

```
http://weather.yahooapis.com/forecastrss?w=WOEID&u=c
```

حال می‌خواهم کوئری استرینگ‌های این لینک را برای شما توضیح دهم. هر شهری بر روی کره‌ی زمین یک WOEID یکتا و منحصر بفرد دارد که شما به پارامتر w عدد WOEID شهر موردنظر خود را می‌دهید. بعد از مقداردهی پارامتر w، وقتی این لینک را در آدرس بار مرورگر خود می‌زنید، RSS مربوط به آب و هوای آن شهر را به شما می‌دهد. مثلا WOEID تهران عدد 28350859 می‌باشد.
و این لینک http://weather.yahooapis.com/forecastrss?w=28350859&u=c اطلاعات آب و هوای تهران را در قالب یک RSS به شما نمایش خواهد داد.

خوب، حالا پارامتر دوم یعنی پارامتر u چکاری را انجام می‌دهد؟
* چنانچه مقدار پارامتر u برابر c باشد، یعنی شما دمای آب و هوای شهر مد نظر را بر اساس سانتیگراد می‌خواهید.
* اگر مقدار پارامتر u برابر f باشد، یعنی شما دمای آب و هوای آن شهر مورد نظر را بر اساس فارنهایت می‌خواهید.

برای گرفتن WOEID شهرها هم به این سایت بروید http://woeid.rosselliot.co.nz و اسم هر شهری که می‌خواهید بزنید تا WOEID را به شما نمایش دهد.

در این مثال من از یک DropDown استفاده کردم که کاربر با انتخاب هر شهر از DropDown، آب و هوای آن شهر را مشاهده می‌کند.
Action مربوط به صفحه‌ی Index به صورت زیر می‌باشد :

```
[HttpGet]
        public ActionResult Index()
        {
           ViewBag.ProvinceList = _RPosition.Positions;
            ShowWeatherProvince(8);
            return View();
        }
```

در اینجا من لیست شهرها را از جدول می‌خوانم که البته این جدول را چون بخش مهمی نبود و فقط شامل ID و نام شهرها بود در فایل ضمیمه قرار ندادم و نام شهرها و ID آنها را بر عهده‌ی خودتان گذاشتم.
حال تابعی را که آب و هوای مربوط به هر شهر را نمایش می‌دهد، به شرح زیر است:

```
public ActionResult ShowWeatherProvince(int dpProvince)
        {
            XDocument rssXml=null;
            CountryName CountryName = new CountryName();
            if (dpProvince != 0)
            {
                switch (dpProvince)
                {
                    case 1:
                        {
                            rssXml =
XDocument.Load("http://weather.yahooapis.com/forecastrss?w=2345768&u=c");
```

```
                            CountryName = new CountryName() { Country = "Iran", City = "Azarbayejan-e
Sharqhi" };
                            break;
                        }
                    case 2:
                        {
                            rssXml =
XDocument.Load("http://weather.yahooapis.com/forecastrss?w=2345767&u=c");
                            CountryName = new CountryName() { Country = "Iran", City = "Azarbayejan-e
Qarbi" };
                            break;
                        }
                    case 3:
                        {
                            rssXml =
XDocument.Load("http://weather.yahooapis.com/forecastrss?w=2254335&u=c");
                            CountryName = new CountryName() { Country = "Iran", City = "Ardebil" };
                            break;
                        }
                    case 4:
                        {
                            rssXml =
XDocument.Load("http://weather.yahooapis.com/forecastrss?w=28350859&u=c");
                            CountryName = new CountryName() { Country = "Iran", City = "Alborz" };
                            break;
                        }
                    case 5:
                        {
                            rssXml =
XDocument.Load("http://weather.yahooapis.com/forecastrss?w=2345787&u=c");
                            CountryName = new CountryName() { Country = "Iran", City = "Esfahan" };
                            break;
                        }
                    case 6:
                        {
                            rssXml =
XDocument.Load("http://weather.yahooapis.com/forecastrss?w=2345775&u=c");
                            CountryName = new CountryName() { Country = "Iran", City = "Ilam" };
                            break;
                        }
                    case 7:
                        {
                            rssXml =
XDocument.Load("http://weather.yahooapis.com/forecastrss?w=2254463&u=c");
                            CountryName = new CountryName() { Country = "Iran", City = "Bushehr" };
                            break;
                        }
                    case 8:
                        {
                            rssXml =
XDocument.Load("http://weather.yahooapis.com/forecastrss?w=28350859&u=c");
                            CountryName = new CountryName() { Country = "Iran", City = "Tehran" };
                            break;
                        }
                    case 9:
                        {
                            rssXml =
XDocument.Load("http://weather.yahooapis.com/forecastrss?w=2345769&u=c");
                            CountryName = new CountryName() { Country = "Iran", City = "Chahar Mahall
va Bakhtiari" };
                            break;
                        }
                    case 10:
                        {
                            rssXml =
XDocument.Load("http://weather.yahooapis.com/forecastrss?w=56189824&u=c");
                            CountryName = new CountryName() { Country = "Iran", City = "Razavi
Khorasan" };
                            break;
                        }
                    case 11:
                        {
                            rssXml =
XDocument.Load("http://weather.yahooapis.com/forecastrss?w=2345789&u=c");
                            CountryName = new CountryName() { Country = "Iran", City = "Shomali
Khorasan" };
                            break;
                        }
                    case 12:
                        {
                            rssXml =
```

```
XDocument.Load("http://weather.yahooapis.com/forecastrss?w=2345789&u=c");
                            CountryName = new CountryName() { Country = "Iran", City = "Jonubi
Khorasan" };
                            break;
                        }
                    case 13:
                        {
                            rssXml =
XDocument.Load("http://weather.yahooapis.com/forecastrss?w=2345778&u=c");
                            CountryName = new CountryName() { Country = "Iran", City = "Khuzestan" };
                            break;
                        }
                    case 14:
                        {
                            rssXml =
XDocument.Load("http://weather.yahooapis.com/forecastrss?w=2255311&u=c");
                            CountryName = new CountryName() { Country = "Iran", City = "Zanjan" };
                            break;
                        }
                    case 15:
                        {
                            rssXml =
XDocument.Load("http://weather.yahooapis.com/forecastrss?w=2345784&u=c");
                            CountryName = new CountryName() { Country = "Iran", City = "Semnan" };
                            break;
                        }
                    case 16:
                        {
                            rssXml =
XDocument.Load("http://weather.yahooapis.com/forecastrss?w=2345770&u=c");
                            CountryName = new CountryName() { Country = "Iran", City = "Sistan va
Baluchestan" };
                            break;
                        }
                    case 17:
                        {
                            rssXml =
XDocument.Load("http://weather.yahooapis.com/forecastrss?w=2345772&u=c");
                            CountryName = new CountryName() { Country = "Iran", City = "Fars" };
                            break;
                        }
                    case 18:
                        {
                            rssXml =
XDocument.Load("http://weather.yahooapis.com/forecastrss?w=20070200&u=c");
                            CountryName = new CountryName() { Country = "Iran", City = "Qazvin" };
                            break;
                        }
                    case 19:
                        {
                            rssXml =
XDocument.Load("http://weather.yahooapis.com/forecastrss?w=2255062&u=c");
                            CountryName = new CountryName() { Country = "Iran", City = "Qom" };
                            break;
                        }
                    case 20:
                        {
                            rssXml =
XDocument.Load("http://weather.yahooapis.com/forecastrss?w=2345779&u=c");
                            CountryName = new CountryName() { Country = "Iran", City = "Kordestan" };
                            break;
                        }
                    case 21:
                        {
                            rssXml =
XDocument.Load("http://weather.yahooapis.com/forecastrss?w=2254796&u=c");
                            CountryName = new CountryName() { Country = "Iran", City = "Kerman" };
                            break;
                        }
                    case 22:
                        {
                            rssXml =
XDocument.Load("http://weather.yahooapis.com/forecastrss?w=2254797&u=c");
                            CountryName = new CountryName() { Country = "Iran", City = "Kermanshah" };
                            break;
                        }
                    case 23:
                        {
                            rssXml =
XDocument.Load("http://weather.yahooapis.com/forecastrss?w=2345771&u=c");
                            CountryName = new CountryName() { Country = "Iran", City = "Kohgiluyeh va
```

```
Buyer Ahmad" };
                            break;
                        }
                    case 24:
                        {
                            rssXml =
XDocument.Load("http://weather.yahooapis.com/forecastrss?w=20070201&u=c");
                            CountryName = new CountryName() { Country = "Iran", City = "Golestan" };
                            break;
                        }
                    case 25:
                        {
                            rssXml =
XDocument.Load("http://weather.yahooapis.com/forecastrss?w=2345773&u=c");
                            CountryName = new CountryName() { Country = "Iran", City = "Gilan" };
                            break;
                        }
                    case 26:
                        {
                            rssXml =
XDocument.Load("http://weather.yahooapis.com/forecastrss?w=2345782&u=c");
                            CountryName = new CountryName() { Country = "Iran", City = "Lorestan" };
                            break;
                        }
                    case 27:
                        {
                            rssXml =
XDocument.Load("http://weather.yahooapis.com/forecastrss?w=2345783&u=c");
                            CountryName = new CountryName() { Country = "Iran", City = "Markazi" };
                            break;
                        }
                    case 28:
                        {
                            rssXml =
XDocument.Load("http://weather.yahooapis.com/forecastrss?w=2345780&u=c");
                            CountryName = new CountryName() { Country = "Iran", City = "Mazandaran" };
                            break;
                        }
                    case 29:
                        {
                            rssXml =
XDocument.Load("http://weather.yahooapis.com/forecastrss?w=2254664&u=c");
                            CountryName = new CountryName() { Country = "Iran", City = "Hamedan" };
                            break;
                        }
                    case 30:
                        {
                            rssXml =
XDocument.Load("http://weather.yahooapis.com/forecastrss?w=2345776&u=c");
                            CountryName = new CountryName() { Country = "Iran", City = "Hormozgan" };
                            break;
                        }
                    case 31:
                        {
                            rssXml =
XDocument.Load("http://weather.yahooapis.com/forecastrss?w=2253355&u=c");
                            CountryName = new CountryName() { Country = "Iran", City = "Yazd" };
                            break;
                        }
                }
                ViewBag.Location = CountryName;
                XNamespace yWeatherNS = "http://xml.weather.yahoo.com/ns/rss/1.0";
                List<YahooWeatherRssItem> WeatherList = new List<YahooWeatherRssItem>();
                for (int i = 0; i < 4; i++)
                {
                    YahooWeatherRssItem YahooWeatherRssItem = new YahooWeatherRssItem()
                    {
                        Code = Convert.ToInt32(rssXml.Descendants("item").Elements(yWeatherNS +
"forecast").ElementAt(i).Attribute("code").Value),
                        Day = rssXml.Descendants("item").Elements(yWeatherNS +
"forecast").ElementAt(i).Attribute("day").Value,
                        Low = rssXml.Descendants("item").Elements(yWeatherNS +
"forecast").ElementAt(i).Attribute("low").Value,
                        High = rssXml.Descendants("item").Elements(yWeatherNS +
"forecast").ElementAt(i).Attribute("high").Value,
                        Text = rssXml.Descendants("item").Elements(yWeatherNS +
"forecast").ElementAt(i).Attribute("text").Value,
                    };

                    WeatherList.Add(YahooWeatherRssItem);
                }
```

```
                ViewBag.FeedList = WeatherList;
            }

                return PartialView("_Weather");
        }
```

قسمت SwitchCase، مقدار و Value مربوط به هر آیتم DropDown را که شامل یک اسم شهر است، میگیرد و RSS مربوط به آن شهر را بر می‌گرداند.
حالا کد مربوط به خواندن فایل Rss را برایتان توضیح می‌دهم : حلقه‌ی for 0 تا 4 (که در کد بالا مشاهده می‌کنید)یعنی اطلاعات 4 روز آینده را برایم برگردان.
من تگ‌های High ، Low ، Day ، Code و text فایل RSS را در این حلقه For می‌خوانم که البته مقادیر این 4 روز را در لیستی اضافه می‌کنم که نوع این لیست هم از نوع YahooWeatherRssItem می‌باشد. من این کلاس را در فایل ضمیمه قرار دادم. اکنون هر کدام از این تگ‌ها را برایتان توضیح می‌دهم:

code : هر آب و هوا کدی دارد .مثلا آب و هوای نیمه ابری یک کد ، آب و هوای آفتابی کدی دیگر و ...
Low: حداقل دمای آن روز را به ما می‌دهد .
High: حداکثر دمای آن روز را به می‌دهد .
day: نام روز از هفته را بر می‌گرداند مثلا شنبه ، یکشنبه و ....
text: که توضیحاتی می‌دهد مثلا اگر هوا آفتابی باشد مقدار sunny را بر می‌گرداند و ...

خوب، تا اینجا ما Rss مربوط به هر شهر را خواندیم حالا در قسمت Design باید چکار کنیم .
کدهای html صفحه‌ی Index ما شامل کدهای زیر است :

```
@{

    ViewBag.Title = "Weather";
}

<link href="~/Content/User/Weather/Weather.css" rel="stylesheet" />
@section scripts{
    <script src="@Url.Content("~/Scripts/jquery-1.6.2.min.js")" type="text/javascript"></script>
        <script src="@Url.Content("~/Scripts/jquery.unobtrusive-ajax.min.js")"
type="text/javascript"></script>
       <script type="text/javascript">
           $("#dpProvince").change(function () {
               $(this).parents("form").submit();
           });
    </script>
}
<h2>Weather</h2>
<div id="Progress">
    <img src="~/Images/User/Other/ajax-loader.gif" />
</div>
<div id="BoxContent"> @Html.Partial("_Weather")</div>

  @using (Ajax.BeginForm(actionName: "ShowWeatherProvince", ajaxOptions: new AjaxOptions {
UpdateTargetId = "BoxContent", LoadingElementId = "Progress", InsertionMode = InsertionMode.Replace }))
                {
<div style="padding-top:15px;">
        <div style="float:left; width:133px; ">Select Your Province</div>
        <div style="float:left">   @Html.DropDownList("dpProvince", new
SelectList(ViewBag.ProvinceList, "Id", "Name"),"Select Your Province", new { @class =
"webUserDropDown", @style = "width:172px" })</div>
    </div>
  }
```

و کدهای _Weather که Partial است به صورت زیر است:

```
@{
    List<Weather.YahooWeatherRssItem> Feeds = ViewBag.FeedList;
}
<div>
```

```
    @{
        HtmlString StartTable = new HtmlString("<table class='WeatherTable' cellspacing='0'
cellpadding='0'><tbody><tr>");
        HtmlString EndTable = new HtmlString("</tr></tbody></table>");
        HtmlString StartTD = new HtmlString("<td>");
        HtmlString EndTD = new HtmlString("</td>");
    }
    <div style="width: 300px;">
        @{
            @StartTable
            foreach (var item in Feeds)
            {
            @StartTD
            <div>@item.Day</div>
            <div>
                @{
                string FileName = "";
                switch (item.Code)
                {
                    case 0: { FileName = "/Images/User/Weather/Tornado.png"; break; }
                    case 1: { FileName = "/Images/User/Weather/storm2.gif"; break; }
                    case 2: { FileName = "/Images/User/Weather/storm2.gif"; break; }
                    case 3: { FileName = "/Images/User/Weather/storm2.gif"; break; }
                    case 4: { FileName = "/Images/User/Weather/15.gif"; break; }
                    case 5: { FileName = "/Images/User/Weather/29.gif"; break; }
                    case 6: { FileName = "/Images/User/Weather/29.gif"; break; }
                    case 7: { FileName = "/Images/User/Weather/29.gif"; break; }
                    case 8: { FileName = "/Images/User/Weather/26.gif"; break; }
                    case 9: { FileName = "/Images/User/Weather/drizzle.png"; break; }
                    case 10: { FileName = "/Images/User/Weather/26.gif"; break; }
                    case 11: { FileName = "/Images/User/Weather/18.gif"; break; }
                    case 12: { FileName = "/Images/User/Weather/18.gif"; break; }
                    case 13: { FileName = "/Images/User/Weather/19.gif"; break; }
                    case 14: { FileName = "/Images/User/Weather/19.gif"; break; }
                    case 15: { FileName = "/Images/User/Weather/19.gif"; break; }
                    case 16: { FileName = "/Images/User/Weather/22.gif"; break; }
                    case 17: { FileName = "/Images/User/Weather/Hail.png"; break; }
                    case 18: { FileName = "/Images/User/Weather/25.gif"; break; }
                    case 19: { FileName = "/Images/User/Weather/dust.png"; break; }
                    case 20: { FileName = "/Images/User/Weather/fog_icon.png"; break; }
                    case 21: { FileName = "/Images/User/Weather/hazy_icon.png"; break; }
                    case 22: { FileName = "/Images/User/Weather/2017737395.png"; break; }
                    case 23: { FileName = "/Images/User/Weather/32.gif"; break; }
                    case 24: { FileName = "/Images/User/Weather/32.gif"; break; }
                    case 25: { FileName = "/Images/User/Weather/31.gif"; break; }
                    case 26: { FileName = "/Images/User/Weather/7.gif"; break; }
                    case 27: { FileName = "/Images/User/Weather/38.gif"; break; }
                    case 28: { FileName = "/Images/User/Weather/6.gif"; break; }
                    case 29: { FileName = "/Images/User/Weather/35.gif"; break; }
                    case 30: { FileName = "/Images/User/Weather/7.gif"; break; }
                    case 31: { FileName = "/Images/User/Weather/33.gif"; break; }
                    case 32: { FileName = "/Images/User/Weather/1.gif"; break; }
                    case 33: { FileName = "/Images/User/Weather/34.gif"; break; }
                    case 34: { FileName = "/Images/User/Weather/2.gif"; break; }
                    case 35: { FileName = "/Images/User/Weather/freezing_rain.png"; break; }
                    case 36: { FileName = "/Images/User/Weather/30.gif"; break; }
                    case 37: { FileName = "/Images/User/Weather/15.gif"; break; }
                    case 38: { FileName = "/Images/User/Weather/15.gif"; break; }
                    case 39: { FileName = "/Images/User/Weather/15.gif"; break; }
                    case 40: { FileName = "/Images/User/Weather/12.gif"; break; }
                    case 41: { FileName = "/Images/User/Weather/22.gif"; break; }
                    case 42: { FileName = "/Images/User/Weather/22.gif"; break; }
                    case 43: { FileName = "/Images/User/Weather/22.gif"; break; }
                    case 44: { FileName = "/Images/User/Weather/39.gif"; break; }
                    case 45: { FileName = "/Images/User/Weather/thundershowers.png"; break; }
                    case 46: { FileName = "/Images/User/Weather/19.gif"; break; }
                    case 47: { FileName = "/Images/User/Weather/thundershowers.png"; break; }
                    case 3200: { FileName = "/Images/User/Weather/1211810662.png"; break; }
                }
                }
                <img alt='@item.Text' title='@item.Text' src='@FileName'>
            </div>
            <div>
                <span>@item.High°</span>
                <span>@item.Low°</span>
            </div>
            @EndTD
            }
        }
        @EndTable
    </div>
```

```
</div>
```

من عکس‌های مربوط به وضعیت آب و هوا را در فایل ضمیمه قرار دادم.
چنانچه در مورد RSS وضعیت آب و هوای یاهو اطلاعات دقیق‌تری را می‌خواهید بدانید به این لینک بروید.
در آموزش بعدی قصد دارم برایتان این بخش را توضیح دهم که بر اساس IP بازدید کننده سایت شما، اطلاعات آب و هوایی شهر بازدید کننده را برایش در سایت نمایش دهد.

Files-06bf65bac63d4dd694b15fc24d4cb074.zip

موفق باشید

نظرات خوانندگان

نویسنده: وحید نصیری
تاریخ: ۱۳۹۲/۱۲/۱۱ ۰:۵

ممنون از شما. چند نکته جزئی در مورد بهبود کیفیت کدها. در Viewهای MVC باید کدنویسی صرفا به نمایش اطلاعات View خلاصه شود. یعنی switch داخل آن بهتر است تبدیل به یک extension method شده و نهایتا استفاده شود (برای تمیز کردن View).
همچنین استفاده از switch 1 و 2 و 10000 اصطلاحا به magic code مشهور هستند. یعنی مشخص نیست معنای این اعداد چی هست. این‌ها را عموما بهتر است تبدیل به enum کرد و بعد استفاده نهایی.

نویسنده: شقایق اشتری
تاریخ: ۱۳۹۲/۱۲/۱۲ ۸:۳۹

ممنونم از شما که مشکل کد نویسی من را گفتید . extension methodها منظورتون همون helperها هست ؟ لینکی دارید که در مورد این extension methodها آموزشی داده باشد چون من می‌خوام کدهامو تا جایی که می‌شود بهینه بنویسم .

نویسنده: شاهین کیاست
تاریخ: ۱۳۹۲/۱۲/۱۲ ۹:۱۵

یک مثال در اینجا
قسمت آخر (**Ifهای شرطی را در Viewها را در متدهای کمکی کپسوله کنید)**

نویسنده: وحید نصیری
تاریخ: ۱۳۹۲/۱۲/۱۲ ۹:۵۱

قسمت switch ایی را که در View نوشتید، تبدیل کنید به یک متد کمکی در کلاسی خارج از View: (مهم نیست متد الحاقی باشد یا خیر؛ فقط داخل View نباشد)

```
public static string GetFileName(int code)
{
    switch (code)
    {
       case 0: return "/Images/User/Weather/Tornado.png";
  //...
}
```

بعد یک خاصیت محاسباتی به نام FileName به مدل مورد استفاده اضافه کنید:

```
public class YahooWeatherRssItem
    {
        public int Code { get; set; }
        //...
        public string FileName
        {
          get { return Util.GetFileName(Code); }
        }
    }
```

به این صورت View از کدهای محاسبات یافتن FileName خالی می‌شود.

نویسنده: علی
تاریخ: ۱۳۹۲/۱۲/۱۲ ۱۰:۷

یه راه دیگه هم استفاده از Yahoo Query Language هست، می تونید با کوئری زیر اطلاعات آب و هوای شهر مورد نظر رو در قالب JSON دریافت کنید

```
select * from weather.forecast where woeid in (select woeid from geo.placefinder where text="CityName")
```

نویسنده: شقایق اشتری
تاریخ: ۱۳۹۲/۱۲/۱۲ ۱۰:۱۳

یه سوال دیگه که داشتم اگر بین کدهایمان ، کد html وجود داشت باز هم باید آنها را از view جدا کنیم و به صورت extension method بنویسیم؟

یا اینکه جداسازی کدها از View را زمانی انجام می‌دهیم که چندین خط کد پشت سر هم نوشته باشیم و بین آنها کد html یی نباشد ؟

نویسنده: وحید نصیری
تاریخ: ۱۳۹۲/۱۲/۱۲ ۱۰:۴۱

- نه الزاما. اگر چندین جا استفاده و تکرار می‌شود یا منطق طولانی دارد، روش « نوشتن HTML Helpers ویژه، به کمک امکانات Razor » می‌تواند مفید باشد.
- منطق قرار گرفته در View فقط باید کار «نهایی» نمایشی را انجام دهد. اگر در View مثلا مستقیما با دیتابیس کار می‌کنید، محاسبات مرتبط با فیلد خاصی را انجام می‌دهید و کلا هر منطقی که «نهایی» بودن نمایش اطلاعات را زیر سؤال ببرد، باید از View جدا شده و به کنترلر و زیرساخت آن منتقل شود.

نویسنده: محسن تقی پور
تاریخ: ۱۳۹۲/۱۲/۲۰ ۰:۸

میشه راجبه این روش کمی بیشتر توضیح بدین

نویسنده: Anaritius
تاریخ: ۱۳۹۲/۱۲/۲۱ ۰:۴۴

سلام
ممنون از مقاله خوبتون
امکانش هست سورس کد webForm ش رو هم بذارید واسه دانلود؟
ممنون

نویسنده: علی
تاریخ: ۱۳۹۲/۱۲/۲۱ ۱۰:۸

اینجا یک مثال از نحوه کار با YQL رو توضیح داده

نویسنده: ع پارسایی
تاریخ: ۱۳۹۳/۱۱/۱۸ ۱۶:۳۶

با سلام
امکان داره بگید در اکشن، قسمت ViewBag.ProvinceList = _RPosition.Positions چه کار میکنه و RPosition کارش چیه؟

نویسنده: ع پارسایی
تاریخ: ۱۳۹۳/۱۱/۲۱ ۱۹:۳۴

با سلام؛ در مطلب گفته شده برای گرفتن WOEID شهرها به فلان سایت بروید و هر کشوری رو بخواییم باید بریم و دستی واردش اما من دیتابیس یا هر چیز دیگری رو میخوام که قابلیت این رو داشته باشه که کاربر با وارد کردن کشور و شهر بتونه آب و هوا رو

ببینه. شما میتونید راهنمایی ام کنید که چه کار باید بکنم و یا منبعی برای دیتابیس مختصات جغرافیایی کشورها وجود داره؟ ممنون میشم کمکم کنید. با تشکر

نویسنده: شقایق اشتری
تاریخ: ۱۳۹۳/۱۱/۲۶ ۲:۱۳

سلام ، نام استان‌ها را در ViewBag قرار دادم.

نویسنده: شقایق اشتری
تاریخ: ۱۳۹۳/۱۱/۲۶ ۲:۱۴

به دو لینک زیر نگاهی بیندازید:

https://developer.yahoo.com/yql/

http://dev.maxmind.com/geoip/legacy/geolite/

عنوان: آشنایی با جنریک‌ها #3
نویسنده: امیر هاشم زاده
تاریخ: ۱۳۹۳/۰۱/۲۱ ۲۳:۴۰
آدرس: www.dotnettips.info
گروه‌ها: C#, Generics

متدهای جنریک

متدهای جنریک، دارای پارامترهایی از نوع جنریک هستند و بوسیله‌ی آنها می‌توانیم نوع‌های (type) متفاوتی را به متد ارسال نمائیم. در واقع از متد، یک نمونه پیاده سازی کرده‌ایم، در حالیکه این متد را برای انواع دیگر هم می‌توانیم فراخوانی کنیم.

تعریف ساده دیگر

جنریک متدها اجازه می‌دهند متدهایی با نوع هایی که در زمان فراخوانی مشخص کرده ایم، داشته باشیم.

نحوه تعریف یک متد جنریک بشکل زیر است:

```
return-type method-name<type-parameters>(parameters)
```

قسمت مهم syntax بالا، **type-parameters** است. در آن قسمت می‌توانید یک یا چند نوع که بوسیله کاما از هم جدا می‌شوند را تعریف کنید. این typeها در return-value و نوع برخی یا همه پارامترهای ورودی جنریک متد، قابل استفاده هستند. به کد زیر توجه کنید:

```
public T1 PrintValue<T1, T2>(T1 param1, T2 param2)
{
    Console.WriteLine("values are: parameter 1 = " + param1 + " and parameter 2 = " + param2);

    return param1;
}
```

در کد بالا، دو پارامتر ورودی بترتیب از نوع T1 و T2 و پارامتر خروجی (return-type) از نوع T1 تعریف کرده‌ایم.

اعمال محدودیت بر روی جنریک متدها

در زمان تعریف یک جنریک کلاس یا جنریک متد، امکان اعمال محدودیت بر روی typeهایی را که قرار است به آن‌ها ارسال شود، داریم. یعنی می‌توانیم تعیین کنیم جنریک متد چه typeهایی را در زمان ایجاد یک وهله‌ی از آن بپذیرد یا نپذیرد. اگر نوعی که به جنریک متد ارسال می‌کنیم جزء محدودیت‌های جنریک باشد با خطای کامپایلر روبرو خواهیم شد. این محدودیت‌ها با کلمه کلیدی where اعمال می‌شوند.

```
public void MyMethod< T >()
       where T : struct
{
  ...
}
```

محدودیت‌های قابل اعمال بر روی جنریک ها

struct: نوع آرگومان ارسالی باید value-type باشد؛ بجز مقادیر غیر NULL.

```
class C<T> where T : struct {} // value type
```

class: نوع آرگومان ارسالی باید reference-type (کلاس، اینترفیس، عامل، آرایه) باشد.

```
class D<T> where T : class {} // reference type
```

new(): آرگومان ارسالی باید یک سازنده عمومی بدون پارامتر باشد. وقتی این محدوده کننده را با سایر محدود کننده‌ها به صورت همزمان استفاده می‌کنید، این محدوده کننده باید در آخر ذکر شود.

```
class H<T> where T : new() {} // no parameter constructor
```

```
public void MyMethod< T >()
       where T : IComparable, MyBaseClass, new ()
{
  ...
}
```

<base class name>: نوع آرگومان ارسالی باید از کلاس ذکر شده یا کلاس مشتق شده آن باشد.

```
class B {}
class E<T> where T : B {} // be/derive from base class
```

<interface name>: نوع آرگومان ارسالی باید اینترفیس ذکر شده یا پیاده ساز آن اینترفیس باشد.

```
interface I {}
class G<T> where T : I {} // be/implement interface
```

U: نوع آرگومان ارسالی باید از نوع یا مشتق شده U باشد.

```
class F<T, U> where T : U {} // be/derive from U
```

توجه: در مثال‌های بالا، محدوده کننده‌ها را برای جنریک کلاس‌ها اعمال کردیم که روش تعریف این محدودیت‌ها برای جنریک متدها هم یکسان است.

اعمال چندین محدودیت همزمان

برای اعمال چندین محدودیت همزمان بر روی یک آرگومان فقط کافی است محدودیت‌ها را پشت سرهم نوشته و آنها را بوسیله کاما از یکدیگر جدا نمایید.

```
interface I {}
class J<T>
  where T : class, I
```

در کلاس J بالا، برای آرگومان محدودیت **class** و **اینترفیس I** را اعمال کرده‌ایم.

این روش قابل تعمیم است:

```
interface I {}
class J<T, U>
  where T : class, I
  where U : I, new() {}
```

در کلاس J، آرگومان T با محدودیت‌های class و اینترفیس I و آرگومان U با محدودیت اینترفیس I و new() تعریف شده است و البته تعداد آرگومان‌ها قابل گسترش است.

حال سوال این است: **چرا از محدود کننده‌ها استفاده می‌کنیم؟**

کد زیر را در نظر بگیرید:

```
//this method returns if both the parameters are equal
public static bool Equals< T > (T t1, Tt2)
{
  return (t1 == t2);
}
```

متد بالا برای مقایسه دو نوع یکسان استفاده می‌شود. در مثال بالا در صورتیکه دو مقدار از نوع int با هم مقایسه نماییم جنریک متد بدرستی کار خواهد کرد ولی اگر بخواهیم دو مقدار از نوع string را مقایسه کنیم با خطای کامپایلر مواجه خواهیم شد. عمل مقایسه دو مقدار از نوع string که مقادیر در heap نگهداری می‌شوند بسادگی مقایسه دو مقدار int نیست. چون همانطور که می‌دانید int یک value-type و string یک reference-type است و برای مقایسه دو reference-type با استفاده از عملگر ==

تمهیداتی باید در نظر گرفته شود.
برای حل مشکل بالا 2 راه حل وجود دارد:
Runtime casting
استفاده از محدود کننده‌ها

casting در زمان اجرا، بعضی اوقات شاید مناسب باشد. در این مورد، CLR نوع‌ها را در زمان اجرا بدلیل کارکرد صحیح بصورت اتوماتیک cast خواهد کرد اما مطمئناً این روش همیشه مناسب نیست مخصوصاً زمانی که نوع‌های مورد استفاده در حال تحریف رفتار طبیعی عملگرها باشند (مانند آخرین نمونه بالا).

عنوان: **خلاص شدن از شر deep null check**
نویسنده: رفیعی
تاریخ: ۱۳۹۳/۰۱/۳۰ ۱۲:۴۵
آدرس: www.dotnettips.info
گروه‌ها: C#, Tips

آیا تا به حال مجبور به نوشتن کدی شبیه قطعه کد زیر شده اید؟

```
var store = GetStore();
string postCode = null;
if (store != null && store.Address != null && store.Address.PostCode != null)
     postCode = store.Address.PostCode.ToString();
```

بله! من مطمئن هستم برای شما هم پیش آمده است.

هدف بازیابی و یا محاسبه یک مقدار است، اما برای انجام این کار می‌بایست به چندین شیء میانی دسترسی پیدا کنیم که البته ممکن است در حالت پیش فرض خود قرار داشته باشند و حاوی هیچ مقداری نباشند. بنابراین برای جلوگیری از وقوع NullException ، مجبوریم تمامی اشیائی که در مسیر قرار دارند را بررسی کنیم که null نباشند. مثال بالا کاملا گویا ست. گاهی اوقات حتی ممکن است فراخوانی یک متد، تبدیل نوع با استفاده از as و یا دسترسی به عناصر یک مجموعه وجود داشته باشد. متاسفانه مدیریت تمامی این حالات باعث حجیم شدن کدها و در نتیجه کاهش خوانایی آنها می‌شود. بنابراین باید به دنبال یک راه حل مناسب بود.

استفاده از یک متد الحاقی شرطی (Conditional extensions)

از نظر بسیاری از برنامه نویس‌ها راه حل، استفاده از یک متد الحاقی شرطی است. اگر عبارت "c# deep null check" را گوگل کنید، پیاده سازی‌های متنوعی را پیدا خواهید کرد. اگر چه این متدها نام‌های متفاوتی دارند اما همه آن‌ها از یک ایده کلی مشترک استفاده می‌کنند:

```
public static TResult IfNotNull<TResult, TSource>(
    this TSource source,
    Func<TSource, TResult> onNotDefault)
    where TSource : class
{
    if (onNotDefault == null) throw new ArgumentNullException("onNotDefault");
    return source == null ? default(TResult) : onNotDefault(source);
}
```

همانطور که می‌بینید این متد الحاقی مقداری از نوع TResult را بر می‌گرداند. اگر source که در اینجا با توجه به الحاقی بودن متد به معنای شی جاری است، null باشد مقدار پیش فرض نوع خروجی(TResult) بازگردانده می‌شود و در غیر این صورت دیلیگیت onNotDefault فراخوانی می‌گردد.

بعد از افزودن متد الحاقی IfNotNull به پروژه می‌توانیم مثال ابتدای مطلب را به صورت زیر بنویسیم :

```
var postCode =
    GetStore()
        .IfNotNull(x => x.Address)
        .IfNotNull(x => x.PostCode)
        .IfNotNull(x => x.ToString());
```

این روش مزایای بسیاری دارد اما در موارد پیچیده دچار مشکل می‌شویم. برای مثال در نظر بگیرید قصد داریم در طول مسیر، متدی را فراخوانی کنیم و مقدار بازگشتی را در یک متغیر موقتی ذخیره کنیم و بر اساس آن ادامه مسیر را طی کنیم. متاسفانه این کارها هم اکنون امکان پذیر نیست. پس به نظر می‌رسد باید کمی متد الحاقی IfNotNull را بهبود ببخشیم.

برای بهبود عملکرد متد الحاقی IfNotNull علاوه بر موارد ذکر شده حداقل دو مورد به نظر من می‌رسد:
این متد فقط با انواع ارجاعی (reference types) کار می‌کند و می‌بایست برای کار با انواع مقداری (value types) اصلاح شود.

با انواع داده ای مثل string چه باید کرد؟ در مورد این نوع داده‌ها تنها مطمئن شدن از null نبودن کافی نیست. برای مثال در مورد string ، گاهی اوقات ما می‌خواهیم از خالی نبودن آن نیز مطمئن شویم. و یا در مورد collectionها تنها null نبودن کافی نیست بلکه زمانی که نیاز به محاسبه مجموع و یا یافتن بزرگترین عضو است، باید از خالی نبودن مجموعه و وجود حداقل یک عضو در آن مطمئن باشیم.

برای حل این مشکلات می‌توانیم متد الحاقی IfNotNull را به متد الحاقی IfNotDefault تبدیل کنیم:

```
public static TResult IfNotDefault<TResult, TSource>(
    this TSource source,
    Func<TSource, TResult> onNotDefault,
    Predicate<TSource> isNotDefault = null)
{
    if (onNotDefault == null) throw new ArgumentNullException("onNotDefault");
    var isDefault = isNotDefault == null
        ? EqualityComparer<TSource>.Default.Equals(source, default(TSource))
        : !isNotDefault(source);
   return isDefault ? default(TResult) : onNotDefault(source);
}
```

تعریف این متد خیلی با تعریف متد قبلی متفاوت نیست. به منظور پشتیبانی از struct ها، قید where TSource : class حذف شده است. بنابراین دیگر نمی‌توان از مقایسه‌ی ساده null با استفاده از عملگر == استفاده کرد چراکه structها nullable نیستند. پس مجبوریم از EqualityComparer<TSource>.Default بخواهیم که این کار را انجام دهد. متد الحاقی IfNotDefault همچنین شامل یک predicate اختیاری با نام isNotDefault است. در صورتی که مقایسه پیش فرض کافی نبود می‌توان از این predicate استفاده کرد.
در انتها اجازه بدهید چند مثال کاربردی را مرور کنیم:
1- انجام یک سری اعمال بر روی string در صورتی که رشته خالی نباشد:

```
return person
        . IfNotDefault(x => x.Name)
        . IfNotDefault(SomeOperation, x => !string.IsNullOrEmpty(x));
```

محاسبه‌ی مقدار میانگین. متد الحاقی IfNotDefault به زیبایی در یک زنجیره‌ی LINQ کار می‌کند:

```
var avg = students
        .Where(IsNotAGraduate)
        .FirstOrDefault()
        .IfNotDefault(s => s.Grades)
        .IfNotDefault(g => g.Average(), g => g != null && g.Length > 0);
```

برای مطالعه بیشتر

Get rid of deep null checks

Chained null checks and the Maybe monad

Maybe or IfNotNull using lambdas for deep expressions

Dynamically Check Nested Values for IsNull Values

نظرات خوانندگان

نویسنده: وحید نصیری
تاریخ: ۱۳۹۳/۰۱/۳۰ ۱۳:۱۱

با تشکر از شما. در سی‌شارپ 6 ، برای این مورد ویژه قرار است عملگر جدیدی اضافه شود:

Null propagation	customer?.Orders?[5]?.$price	Planned	Planned

بحثی در این مورد

نویسنده: رفیعی
تاریخ: ۱۳۹۳/۰۱/۳۰ ۱۴:۷

بله همین طوره!ممنون از دقت نظرتون. اتفاقاً پیشنهاد این کار رو هم فردی به نام جمشید اسدزاده که احتمالا ایرانیه در قسمت پیشنهادهای مایکروسافت مطرح کرده.

نویسنده: حمید سامانی
تاریخ: ۱۳۹۳/۰۱/۳۰ ۱۷:۱۱

احتمالا ایشون SafeNavigationOperator توی زبان Groovy را قبلا دیده بودن :) .

نویسنده: یاسر مرادی
تاریخ: ۱۳۹۳/۰۱/۳۰ ۱۸:۲۶

متاسفانه روش فوق کد نویسی را تا حد زیادی تحت تاثیر قرار می‌دهد، مگر این که روش استفاده از متد الحاقی شما را به خوبی متوجه نشده باشم
به مثال زیر دقت کنید:

```
public class Customer
    {
        public CustomerInfo Info { get; set; }
        public Int32 GetNameLength()
        {
            return this.IfNotDefault(city => city.Info)
                .IfNotDefault(info => info.CityInfo)
                .IfNotDefault(cityInfo => cityInfo.Name)
                .IfNotDefault(name => name.Length);
        }
    }
    public class CustomerInfo
    {
        public CustomerCityInfo CityInfo { get; set; }
    }
    public class CustomerCityInfo
    {
        public String Name { get; set; }
    }
```

و برای استفاده داریم:

```
Customer customer = new Customer();
            String cityName = customer
                .IfNotDefault(cust => cust.Info)
                .IfNotDefault(info => info.CityInfo)
```

```
                .IfNotDefault(city => city.Name);
            Int32 length = customer.GetNameLength();
```

در حالی که با متد الحاقی زیر داریم

```
public static TValue GetValue<TObj, TValue>(this TObj obj, Func<TObj, TValue> member, TValue
defaultValueOnNull = default(TValue))
        {
            if (member == null)
                throw new ArgumentNullException("member");

            if (obj == null)
                throw new ArgumentNullException("obj");

            try
            {
                return member(obj);
            }
            catch (NullReferenceException)
            {
                return defaultValueOnNull;
            }
        }
```

تعریف ساده‌تر کلاس

```
public class Customer
    {
        public CustomerInfo Info { get; set; }

        public Int32 GetNameLength()
        {
            return this.Info.CityInfo.Name.Length;
        }
    }

    public class CustomerInfo
    {
        public CustomerCityInfo CityInfo { get; set; }
    }

    public class CustomerCityInfo
    {
        public String Name { get; set; }
    }
```

و سادگی در استفاده

```
Customer customer = new Customer();

            String cityName = customer.GetValue(cust => cust.Info.CityInfo.Name, "Not Selected");

            Int32 i = customer.GetValue(cust => cust.GetNameLength());
```

شاید بگویید استفاده از Try-Catch سیستم را کند می‌کند، البته نه در آن حدی که فکر می‌کنید، و اگر قسمتی از کد شما به تعداد زیادی در بازه‌ی زمانی کوتاه فراخوانی می‌شود، می‌توانید آنرا به صورت کاملا عادی بنویسید، چون واقعا تعداد این شرایط زیاد نیست و این مورد سناریوی فراگیری نیست، در عوض خوانایی کد بسیار بسیار بالاتر از حالات عادی است.
در ضمن دقت کنید که تا زمانی که خطای NullReference رخ ندهد، سرعت سیستم در حد همان حداقل نیز کاهش نمی‌یابد، بدین جهت که بسیاری از افراد فکر می‌کنند Try-Catch نوشتن به خودی خود برنامه را کند می‌کند، ولی این رخ دادن خطا و جمع آوری StackTrace و ... است که برنامه را کند می‌کند، که شاید در خیلی از موارد اصلا رخ ندهد.
البته کدهای نوشته صرفا نمونه کد است، به هیچ وجه اصول طراحی در آن رعایت نشده است، بلکه سعی کرده ام مثال واضح‌تری بزنم
موفق و پایدار باشید

نویسنده: رضا عرب
تاریخ: ۱۳۹۳/۰۱/۳۱ ۹:۲۶

با تشکر از شما من یک Extension Method نوشتم که به نظرم کار کردن باهاش راحتتره. GetValueOrDefault

نویسنده: رفیعی
تاریخ: ۱۳۹۳/۰۱/۳۱ ۱۰:۰

بدیهی است راه‌های زیادی برای این کار وجود دارد اگرچه هسته همه اون‌ها خیلی شبیه...
متد الحاقی IfNotDefault چند ویژگی مهم دارد :

همانطور که در متن ذکر شده، بحث فقط چک برای null نبودن نیست بلکه چک برای قرار نداشتن در حالت پیش فرضه! که در انواعی مثل string و collectionها خیلی مهمه.
گاهی اوقات هر کدام از اشیاء در طول زنجیره برای ما مهم هستند. متد الحاقی IfNotDefault این امکان را دارد که هر کدام از اشیاء جداگانه بررسی شوند. روش ارایه شده در C# 6.0 هم همینگونه است.

نویسنده: وحید نصیری
تاریخ: ۱۳۹۳/۰۳/۰۲ ۰:۳۲

دو روش دیگر برای حل این مساله

- استفاده از روش‌های AOP مانند Minimizing the null ref with dynamic proxies
- استفاده از expression trees مانند Avoiding nulls with expression trees

عنوان: **انجام اعمال ریاضی بر روی Generics**
نویسنده: وحید نصیری
تاریخ: ۱۳۹۳/۰۴/۰۱ ۰:۴۵
آدرس: www.dotnettips.info
گروه‌ها: C#, Generics

کامپایلر سی‌شارپ اگر نتواند نوع‌های عملوندها را در حین بکارگیری عملگرها تشخیص دهد، اجازه‌ی استفاده از عملگر را نخواهد داد و کار کامپایل، با یک خطا خاتمه می‌یابد. برای نمونه مثال زیر را در نظر بگیرید:

```
public interface ICalculator<T>
    {
        T Add(T operand1, T operand2);
    }

    public class Calculator<T> : ICalculator<T>
    {
        public T Add(T operand1, T operand2)
        {
            return operand1 + operand2;
        }
    }
```

در اینجا چون کامپایلر نمی‌داند که عملگر + بر روی چه نوع‌هایی قرار است اعمال شود (به علت جنریک تعریف شدن این نوع‌ها و مشخص نبودن اینکه آیا این نوع، اصلا عملگر + دارد یا خیر)، با صدور خطای زیر، عملیات کامپایل را متوقف می‌کند:

```
Operator '+' cannot be applied to operands of type 'T' and 'T'
```

برای حل این مساله، چندین روش مطرح شده‌است که در ادامه تعدادی از آن‌ها را مرور خواهیم کرد.

روش اول: واگذار کردن استراتژی عملیات ریاضی به یک کلاس خارجی

این راه حلی است که توسط اعضای تیم سی‌شارپ در روزهای ابتدایی معرفی جنریک‌ها مطرح شده‌است. فرض کنید می‌خواهیم لیستی از جنریک‌ها را با هم جمع بزنیم:

```
public class Calculator2<T>
    {
        public T Sum(List<T> list)
        {
            T sum = 0;
            for (int i = 0; i < list.Count; i++)
                sum += list[i];
            return sum;
        }
    }
```

این کد نیز قابل کامپایل نبوده و امکان اعمال عملگر + بر روی نوع ناشناخته‌ی T میسر نیست.

```
public interface ICalculator<T>
    {
        T Add(T operand1, T operand2);
    }

    public class Int32Calculator : ICalculator<int>
    {
        public int Add(int operand1, int operand2)
        {
            return operand1 + operand2;
        }
    }

    public class AlgorithmLibrary<T> where T : new()
    {
        private readonly ICalculator<T> _calculator;
        public AlgorithmLibrary(ICalculator<T> calculator)
        {
```

```
            _calculator = calculator;
        }

        public T Sum(List<T> items)
        {
            var sum = new T();
            for (var i = 0; i < items.Count; i++)
            {
                sum = _calculator.Add(sum, items[i]);
            }
            return sum;
        }
    }
```

در راه حل ارائه شده، یک اینترفیس عمومی که متد جمع را تعریف کرده‌است، مشاهده می‌کنیم. سپس این اینترفیس در سازنده‌ی کتابخانه‌ی الگوریتم‌های برنامه تزریق شده‌است. اکنون کدهای AlgorithmLibrary بدون مشکل کامپایل می‌شوند. هر زمان که نیاز به استفاده از آن بود، بر اساس نوع T، پیاده سازی خاصی را باید ارائه داد. برای مثال در اینجا Int32Calculator پیاده سازی نوع int را انجام داده‌است. برای استفاده از آن نیز خواهیم داشت:

```
var result = new AlgorithmLibrary<int>(new Int32Calculator()).Sum(new List<int> { 1, 2, 3 });
```

البته این نوع پیاده سازی را که کار اصلی آن واگذاری عملیات جمع، به یک کلاس خارجی است، توسط Func نیز می‌توان خلاصه‌تر کرد:

```
public class Algorithms<T> where T : new()
    {
        public T Calculate(Func<T, T, T> add, IEnumerable<T> numbers)
        {
            var sum = new T();
            foreach (var number in numbers)
            {
                sum = add(sum, number);
            }
            return sum;
        }
    }
```

استفاده از Action و Func نیز یکی دیگر از روش‌های تزریق وابستگی‌ها است که در اینجا بکار گرفته شده‌است. برای استفاده از آن خواهیم داشت:

```
var result = new Algorithms<int>().Calculate((a, b) => a + b, new[] { 1, 2, 3 });
```

آرگومان اول روش جمع زدن را مشخص می‌کند و آرگومان دوم، لیستی است که باید اعضای آن جمع زده شوند.

روش دوم: استفاده از واژه‌ی کلیدی dynamic

با استفاده از واژه‌ی کلیدی dynamic می‌توان بررسی نوع داده‌ها را به زمان اجرا موکول کرد. به این ترتیب دیگر کامپایلر مشکلی با کامپایل قطعه کد ذیل نخواهد داشت:

```
public class Calculator<T> : ICalculator<T>
    {
        public T Add(T operand1, T operand2)
        {
            return (dynamic)operand1 + operand2;
        }
    }
```

و مثال زیر نیز به خوبی کار می‌کند:

```
var test = new Calculator<int>().Add(1, 2);
```

البته بدیهی است که نوع تعریف شده در اینجا باید دارای عملگر + باشد. در غیر اینصورت در زمان اجرا برنامه با یک خطا خاتمه خواهد یافت.

روش فوق نسبت به حالتی که بر اساس نوع T تصمیم‌گیری شود و از عملگر + متناظری استفاده گردد، خوانایی بهتری دارد:

```
public T Add(T t1, T t2)
{
    if (typeof(T) == typeof(double))
    {
        var d1 = (double)t1;
        var d2 = (double)t2;
        return (T)(d1 + d2);
    }
    else if (typeof(T) == typeof(int)){
        var i1 = (int)t1;
        var i2 = (int)t2;
        return (T)(i1 + i2);
    }
    else ...
}
```

روش سوم: استفاده از Expression Trees

روش زیر بسیار شبیه است به حالتیکه از Func در روش اول استفاده شد. در اینجا این Func به صورت پویا تولید و سپس صدا زده می‌شود:

```
using System;
using System.Linq.Expressions;

namespace GenericsArithmetic
{
    public class Solution3
    {
        public T Add<T>(T a, T b)
        {
            var paramA = Expression.Parameter(typeof(T), "a");
            var paramB = Expression.Parameter(typeof(T), "b");

            var body = Expression.Add(paramA, paramB);
            var add = Expression.Lambda<Func<T, T, T>>(body, paramA, paramB).Compile();
            return add(a, b);
        }
    }
}
```

البته این مثال، یک مثال ابتدایی در این مورد است. بر همین مبنا و ایده، یک کتابخانه‌ی با کارآیی بالا، تحت عنوان Generic Operators که جزو Misc utils می‌باشد، تهیه شده‌است.

به کمک کتابخانه‌ی Generic Operators، کدهای جمع زدن اعضای یک لیست جنریک به صورت ذیل خلاصه می‌شوند:

```
public static T Sum<T>(this IEnumerable<T> source)
{
    T sum = Operator<T>.Zero;
    foreach (T value in source)
    {
            sum = Operator.Add(sum, value);
    }
    return sum;
}
```

نظرات خوانندگان

نویسنده: سجاد
تاریخ: ۱۳۹۳/۰۴/۰۸ ۱۲:۵۶

c++ احتیاجی به این نوع پیاده سازی‌های دشوار با استفاده از روش‌های غیرمعمول رو نداره. هرچند خودم هم یکی از طرفدارهای پروپا قرص c# هستم ولی genericهای c# در مقابل templateهای c++ کمبود دارند. هرچند همیشه **عاشق** c# بودم ولی generic های c# هیچوقت انتظارات منو برآورده نکرد.

نویسنده: وحید نصیری
تاریخ: ۱۳۹۳/۰۴/۰۸ ۱۳:۱۸

C# generics مانند C++ templates نیستند . آرگومان‌های C# generics در زمان اجرا دریافت و پردازش می‌شوند، در حالیکه C++ templates مانند یک compiler macro عمل کرده و در زمان کامپایل و پیش از اجرا به صورت کامل دریافت، بررسی و الحاق خواهند شد. به همین جهت است که C++ templates می‌توانند برای مثال تشخیص دهند، آرگومان مورد استفاده، دارای عملگر + هست یا خیر.

پردازش در زمان اجرای آرگومان‌های جنریک این مزیت را به همراه دارد که بتوانید بدون نیاز به الحاق سورس آرگومان‌های مورد استفاده (چون برخلاف C++ templates ، ریز اطلاعات آن‌ها کامپایل نمی‌شوند)، کتابخانه‌ای را برای عموم منتشر کنید.

عنوان: **ایجاد «خواص الحاقی»**
نویسنده: وحید نصیری
تاریخ: ۱۳۹۳/۰۵/۱۷ ۱۱:۳۵
آدرس: www.dotnettips.info
گروه‌ها: #C

حتما با متدهای الحاقی یا Extension methods آشنایی دارید؛ می‌توان به یک شیء، که حتی منبع آن در دسترس ما نیست، متدی را اضافه کرد. سؤال: در مورد خواص چطور؟ آیا می‌شود به وهله‌ای از یک شیء موجود از پیش طراحی شده، یک خاصیت جدید را اضافه کرد؟

احتمالا شاید عنوان کنید که با اشیاء dynamic می‌توان چنین کاری را انجام داد. اما سؤال در مورد اشیاء غیر dynamic است.

یا نمونه‌ی دیگر آن Attached Properties در برنامه‌های مبتنی بر Xaml هستند. می‌توان به یک شیء از پیش موجود Xaml، خاصیتی را افزود که البته پیاده سازی آن منحصر است به همان نوع برنامه‌ها.

راه حل پیشنهادی

یک Dictionary را ایجاد کنیم تا ارجاعی از اشیاء، به عنوان کلید، در آن ذخیره شده و سپس key/valueهایی به عنوان value هر شیء، در آن ذخیره شوند. این key/valueها همان خواص و مقادیر آن‌ها خواهند بود. هر چند این راه حل به خوبی کار می‌کند اما ... مشکل نشتی حافظه دارد.

شیء Dictionary یک ارجاع قوی را از اشیاء، درون خودش نگه داری می‌کند و تا زمانیکه در حافظه باقی است، سیستم GC مجوز رهاسازی منابع آن‌ها را نخواهد یافت؛ چون عموما این نوع Dictionaryها باید استاتیک تعریف شوند تا طول عمر آن‌ها با طول عمر برنامه یکی گردد. بنابراین اساسا اشیایی که به این نحو قرار است پردازش شوند، هیچگاه dispose نخواهند شد. راه حلی برای این مساله در دات نت 4 به صورت توکار به دات نت فریم ورک اضافه شده‌است؛ به نام ساختار داده‌ای ConditionalWeakTable.

معرفی ConditionalWeakTable

ConditionalWeakTable جزو ساختارهای داده‌ای کمتر شناخته شده‌ی دات نت است. این ساختار داده، اشاره‌گرهایی را به ارجاعات اشیاء، درون خود ذخیره می‌کند. بنابراین چون ارجاعاتی قوی را به اشیاء ایجاد نمی‌کند، مانع عملکرد GC نیز نشده و برنامه در دراز مدت دچار مشکل نشتی حافظه نخواهد شد. هدف اصلی آن ایجاد ارتباطی بین CLR و DLR است. توسط آن می‌توان به اشیاء دلخواه، خواصی را افزود. به علاوه طراحی آن به نحوی است که thread safe است و مباحث قفل گذاری بر روی اطلاعات، به صورت توکار در آن پیاده سازی شده‌است. کار DLR فراهم آوردن امکان پیاده سازی زبان‌های پویایی مانند Ruby و Python برفراز CLR است. در این نوع زبان‌ها می‌توان به وهله‌هایی از اشیاء موجود، خاصیت‌های جدیدی را متصل کرد.

به صورت خلاصه کار ConditionalWeakTable ایجاد نگاشتی است بین وهله‌هایی از اشیاء CLR (اشیایی غیرپویا) و خواصی که به آن‌ها می‌توان به صورت پویا انتساب داد. در کار GC اخلال ایجاد نمی‌کند و همچنین می‌توان به صورت همزمان از طریق تردهای مختلف، بدون مشکل با آن کار کرد.

پیاده سازی خواص الحاقی به کمک ConditionalWeakTable

در اینجا نحوه‌ی استفاده از ConditionalWeakTable را جهت اتصال خواصی جدید به وهله‌های موجود اشیاء مشاهده می‌کنید:

```
using System.Collections.Generic;
using System.Runtime.CompilerServices;

namespace ConditionalWeakTableSamples
{
    public static class AttachedProperties
    {
        public static ConditionalWeakTable<object,
            Dictionary<string, object>> ObjectCache = new ConditionalWeakTable<object,
                Dictionary<string, object>>();

        public static void SetValue<T>(this T obj, string name, object value) where T : class
        {
            var properties = ObjectCache.GetOrCreateValue(obj);
```

```
            if (properties.ContainsKey(name))
                properties[name] = value;
            else
                properties.Add(name, value);
        }

        public static T GetValue<T>(this object obj, string name)
        {
            Dictionary<string, object> properties;
            if (ObjectCache.TryGetValue(obj, out properties) && properties.ContainsKey(name))
                return (T)properties[name];
            return default(T);
        }

        public static object GetValue(this object obj, string name)
        {
            return obj.GetValue<object>(name);
        }
    }
}
```

ObjectCache تعریف شده از نوع استاتیک است؛ بنابراین در طول عمر برنامه زنده نگه داشته خواهد شد، اما اشیایی که به آن منتسب می‌شوند، خیر. هرچند به ظاهر در متد GetOrCreateValue، یک وهله از شیءایی موجود را دریافت می‌کند، اما در پشت صحنه صرفا IntPtr یا اشاره‌گری به این شیء را ذخیره سازی خواهد کرد. به این ترتیب در کار GC اخلالی صورت نخواهد گرفت و شیء مورد نظر، تا پایان کار برنامه به اجبار زنده نگه داشته نخواهد شد.

کاربرد اول

اگر با ASP.NET کار کرده باشید حتما با IPrincipal آشنایی دارید. خواصی مانند Identity یک کاربر در آن ذخیره می‌شوند. سؤال: چگونه می‌توان یک خاصیت جدید به نام مثلا Disclaimer را به وهله‌ای از این شیء افزود:

```
public static class ISecurityPrincipalExtension
    {
        public static bool Disclaimer(this IPrincipal principal)
        {
            return principal.GetValue<bool>("Disclaimer");
        }

        public static void SetDisclaimer(this IPrincipal principal, bool value)
        {
            principal.SetValue("Disclaimer", value);
        }
    }
```

در اینجا مثالی را از کاربرد کلاس AttachedProperties فوق مشاهده می‌کنید. توسط متد SetDisclaimer یک خاصیت جدید به نام Disclaimer به وهله‌ای از شیءایی از نوع IPrincipal قابل اتصال است. سپس توسط متد Disclaimer قابل دستیابی خواهد بود.

اگر صرفا قرار است یک خاصیت به شیءایی متصل شود، روش ذیل نیز قابل استفاده می‌باشد (بجای استفاده از دیکشنری از یک کلاس جهت تعریف خاصیت اضافی جدید استفاده شده‌است):

```
using System.Runtime.CompilerServices;

namespace ConditionalWeakTableSamples
{
    public static class PropertyExtensions
    {
        private class ExtraPropertyHolder
        {
            public bool IsDirty { get; set; }
        }

        private static readonly ConditionalWeakTable<object, ExtraPropertyHolder> _isDirtyTable
                = new ConditionalWeakTable<object, ExtraPropertyHolder>();
```

```
        public static bool IsDirty(this object @this)
        {
            return _isDirtyTable.GetOrCreateValue(@this).IsDirty;
        }

        public static void SetIsDirty(this object @this, bool isDirty)
        {
            _isDirtyTable.GetOrCreateValue(@this).IsDirty = isDirty;
        }
    }
}
```

کاربرد دوم

ایجاد Id منحصربفرد برای اشیاء برنامه.

فرض کنید در حال نوشتن یک Entity framework profiler هستید. طراحی فعلی سیستم Interception آن به نحو زیر است:

```
public void Closed(DbConnection connection, DbConnectionInterceptionContext interceptionContext)
{
}
```

سؤال: اینجا رویداد بسته شدن یک اتصال را دریافت می‌کنیم؛ اما ... دقیقا کدام اتصال؟ رویداد Opened را هم داریم اما چگونه این اشیاء را به هم مرتبط کنیم؟ شیء DbConnection دارای Id نیست. متد GetHashCode هم الزامی ندارد که اصلا پیاده سازی شده باشد یا حتی یک Id منحصربفرد را تولید کند. این متد با تغییر مقادیر خواص یک شیء می‌تواند مقادیر متفاوتی را ارائه دهد. در اینجا می‌خواهیم به ازای ارجاعی از یک شیء، یک Id منحصربفرد داشته باشیم تا بتوانیم تشخیص دهیم که این اتصال بسته شده، دقیقا کدام اتصال باز شده‌است؟

راه حل: خوب ... یک خاصیت Id را به اشیاء موجود متصل کنید!

```
using System;
using System.Runtime.CompilerServices;

namespace ConditionalWeakTableSamples
{
    public static class UniqueIdExtensions
    {
        static readonly ConditionalWeakTable<object, string> _idTable =
                                    new ConditionalWeakTable<object, string>();

        public static string GetUniqueId(this object obj)
        {
            return _idTable.GetValue(obj, o => Guid.NewGuid().ToString());
        }

        public static string GetUniqueId(this object obj, string key)
        {
            return _idTable.GetValue(obj, o => key);
        }
    }
}
```

در اینجا مثالی دیگر از پیاده سازی و استفاده از ConditionalWeakTable را ملاحظه می‌کنید. اگر در کش آن ارجاعی به شیء مورد نظر وجود داشته باشد، مقدار Guid آن بازگشت داده می‌شود؛ اگر خیر، یک Guid به ارجاعی از شیء، انتساب داده شده و سپس بازگشت داده می‌شود. به عبارتی به صورت پویا یک خاصیت UniqueId به وهله‌هایی از اشیاء اضافه می‌شوند. به این ترتیب به سادگی می‌توان آن‌ها را ردیابی کرد و تشخیص داد که اگر این Guid پیشتر جایی به اتصال باز شده‌ای منتسب شده‌است، در چه زمانی و در کجا بسته شده است یا اصلا ... خیر. جایی بسته نشده‌است.

برای مطالعه بیشتر

The Conditional Weak Table: Enabling Dynamic Object Properties

How to create mixin using C# 4.0

Disclaimer Page using MVC

Extension Properties Revised

Easy Modeling

Providing unique ID on managed object using ConditionalWeakTable

عنوان: **اثبات قانون مشاهده‌گر در برنامه نویسی**
نویسنده: میثم نوایی
تاریخ: ۱۳۹۳/۰۷/۲۷ ۱:۳۰
آدرس: www.dotnettips.info
گروه‌ها: C#, Threading, Random

امروز حین کدنویسی به یک مشکل نادر برخورد کردم. کلاسی پایه داشتم (مثلا Person) که یک سری کلاس دیگر از آن ارث بری میکردند (مثلا کلاس‌های Student و Teacher).در اینجا در کلاس پایه بصورت اتوماتیک یک ویژگی(Property) را روی کلاس‌های مشتق شده مقدار دهی میکردم؛ مثلا به این شکل:

```
public class Person
    {
        public Person()
        {
            personId= this.GetType().Name + (new Random()).Next(1, int.MaxValue);
        }
     }
```

سپس در یک متد مجموعه‌ای از Studentها و teacherها را ایجاد کرده و به لیستی از Personها اضافه میکنم:

```
var student1=new Student(){Name="Iraj",Age=21};
var student1=new Student(){Name="Nima",Age=20};
var student1=new Student(){Name="Sara",Age=25};
var student1=new Student(){Name="Mina",Age=22};
var student1=new Student(){Name="Narges",Age=26};
var teacher1=new Student(){Name="Navaei",Age=45};
var teacher2=new Student(){Name="Imani",Age=50};
```

اما در نهایت اتفاقی که رخ میداد این بود که PersonId همه Studentها یکسان می‌شد ولی قضیه به همین جا ختم نشد؛ وقتی خط به خط برنامه را Debug و مقادیر را Watch می‌کردم، مشاهده می‌کردم که PersonId به درستی ایجاد می‌شود.
در فیزیک نوین اصلی هست به نام عدم قطعیت هایزنبرگ که به زبان ساده میتوان گفت نحوه رخداد یک اتفاق، با توجه به وجود یا عدم وجود یک مشاهده‌گر خارجی نتیجه‌ی متفاوتی خواهد داشت.
کم کم داشتم به وجود قانون مشاهده‌گر در برنامه نویسی هم ایمان پیدا میکردم که این کد فقط در صورتیکه آنرا مرحله به مرحله بررسی کنم جواب خواهد داد!
جالب اینکه زمانیکه personId را نیز ایجاد میکردم، یک دستور برای دیدن خروجی نوشتم مثل این

```
public class Person
    {
        public Person()
        {
            personId= this.GetType().Name + (new Random()).Next(1, int.MaxValue);
            Debug.Print(personId)
        }
     }
```

در این حالت نیز دستورات درست عمل میکردند و personId متفاوتی ایجاد می‌شد! قبل از خواندن ادامه مطلب شما هم کمی فکر کنید که مشکل کجاست؟
این مشکل ربطی به قانون مشاهده‌گر و یا دیگر قوانین فیزیکی نداشت. بلکه بدلیل سرعت بالای ایجاد وهله ها(instance) از کلاسی‌های مطروحه (مثلا در زمانی کمتر از یک میلی ثانیه) زمانی در بازه یک کلاک CPU رخ می‌داد.

هر نوع ایجاد کندی (همچون نمایش مقادیر در خروجی) باعث می‌شود کلاک پردازنده نیز تغییر کند و عدد اتفاقی تولید شده فرق کند.
همچنین برای حل این مشکل میتوان از کلاس تولید کننده اعداد اتفاقی، شبیه زیر استفاده کرد:

```
using System;
using System.Threading;

public static class RandomProvider
```

```
{
    private static int seed = Environment.TickCount;

    private static ThreadLocal<Random> randomWrapper = new ThreadLocal<Random>(() =>
        new Random(Interlocked.Increment(ref seed))
    );

    public static Random GetThreadRandom()
    {
        return randomWrapper.Value;
    }
}
```

نظرات خوانندگان

نویسنده: رحمت اله رضایی
تاریخ: ۱۳۹۳/۰۷/۲۷ ۱۴:۴

به جای کلاس Random از جایگزین بهتر آن RNGCryptoServiceProvider استفاده کنید و دوباره برنامه رو تست کنید.

نویسنده: سعید
تاریخ: ۱۳۹۳/۰۸/۰۶ ۱۲:۴۸

شما اگر برای تولید عدد تصادفی از یک آبجکت کلاس Random استفاده می‌کردید به چنین مشکلی بر نمی‌خوردید.

نویسنده: میثم نوایی
تاریخ: ۱۳۹۳/۰۸/۰۶ ۱۳:۲

اگه مطلب را کامل مطالعه بفرمایید و شبیه سازی کنید به مشکل مطروحه برخواهید خورد.

نویسنده: سعید
تاریخ: ۱۳۹۳/۰۸/۰۶ ۱۴:۱۵

من نیاز مسئله شما را به صورت زیر نوشتم و برای هر شی Person یک مقدار متفاوت دارم.

```
class Program
    {
        static void Main(string[] args)
        {
            var lst = new List<Person>();
            Random rnd = new Random();
            for (int i = 0; i < 50; i++)
            {
                lst.Add(new Person(rnd));
            }


            foreach (var item in lst)
            {
                Console.WriteLine(item.PersonId);
            }

            Console.ReadKey();
        }
    }

    public class Person
    {
        public Person(Random rnd)
        {
            PersonId = this.GetType().Name + rnd.Next(1, int.MaxValue);
        }

        public string PersonId { get; set; }
    }
```

البته من فکر میکنم اگر شما نیاز به یک ای دی منحصر به فرد دارید راه بهتر استفاده از GUID باشد.

نویسنده: میثم نوایی
تاریخ: ۱۳۹۳/۰۸/۰۶ ۱۴:۴۱

ممنون بابت نظراتتان.
عرض کردم مطلب را کامل بخوانید.این مشکل در سیستم هایی با پردازش بالا و در زمان‌های هزارم ثانیه رخ میدهد.به این معنی که کلاس Random مقادیر متفاوتی ایجاد نمیکند.و عملا کارایی ندارد.

عنوان: **روش صحیح مقایسه دو عدد اعشاری با هم**
نویسنده: وحید نصیری
تاریخ: ۱۳۹۳/۰۸/۰۳ ۱۱:۴۵
آدرس: www.dotnettips.info
گروه‌ها: C#, Math

استفاده از عملگر == برای مقایسه اعداد اعشاری عموما جواب نخواهد داد و کار صحیحی نیست. از این جهت که اعداد، اساسا به صورت یک سری صفر و یک ذخیره شده و امکان ذخیره سازی کامل و دقیق قسمت اعشاری وجود ندارد.
برای مثال نوع‌های double و float امکان ذخیره سازی دقیق عدد یک دهم را ندارند. عدد 10/1 به صورت 0.000110011001100... ذخیره می‌شود (در حالت باینری) و مقایسه دقیق مقادیر ثابت 0.00011 یا 0.00011001100 با آن میسر نیست؛ چون دقت نهایی این اعداد متفاوت است.
در زبان #C، نوع double، مطابق استاندارد IEEE-754 تهیه شده‌است و تنها 15 رقم اعشار دقت دارد و ذخیره سازی اعداد اعشاری در آن، به یک گرد سازی نهایی ختم خواهد شد. بنابراین به دلیل وجود این rounding error طبیعی، امکان استفاده از عملگری مانند == جهت مقایسه‌ی اعداد اعشاری همیشه پاسخ صحیحی را به همراه نخواهد داشت.
برای نمونه مثال زیر را بررسی کنید:

```
double d1 = 12.14;
double d2 = 12.13;
double d3 = d1 - d2; // Should be 0.01
bool check = (d3 == 0.01); // should be true
```

که یک چنین خروجی را به همراه دارد (حاصل آن برخلاف تصور مساوی 0.01 نیست):

روشی که برای حل این مساله پیشنهاد شده‌است، تفریق دو عدد از هم و مقایسه‌ی نتیجه‌ی آن با epsilon است؛ بجای مقایسه با صفر:

```
public static bool ApproxEquals(double d1, double d2)
        {
            const double epsilon = 2.2204460492503131E-16;

            if (d1 == d2)
                return true;

            double tolerance = ((Math.Abs(d1) + Math.Abs(d2)) + 10.0) * epsilon;
            double difference = d1 - d2;

            return (-tolerance < difference && tolerance > difference);
        }
```

متد فوق را در فایل MathUtils کتابخانه‌ی JSON.NET می‌تواند مشاهده کنید.

با این خروجی:

در این حالت می‌توان نتیجه گرفت که d3 و 0.01 بسیار بسیار نزدیک به هم هستند؛ یا تقریبا مساوی.

نظرات خوانندگان

نویسنده: سوین
تاریخ: ۱۳۹۳/۰۸/۰۳ ۱۶:۴۷

با سلام
من خودم در یه نرم افزار به این مشکل برخورده بودم و برای حل این مشکل ، قبل از اعمال جبری نوع‌های double اون‌ها رو تبدیل به decimal کرده و مقایسه یا تفریق می‌کنم .

نویسنده: عثمان رحیمی
تاریخ: ۱۳۹۳/۰۸/۰۳ ۱۹:۵۸

سلام .
آیا این قسمت

```
if (d1 == d2)
        return true;
```

در تابع ApproxEquals اصلا اجرا می‌شود ، یعنی true برگردانده می‌شود ؟ اگر بله چه زمانی ؟
و چه فرقی با

```
(d3 == 0.01)
```

دارد مگه هر دو بررسی دو مقدار double نیستند ؟ پس وقتی (d3 == 0.01) نتیجه‌ی مورد نظر را ندهد if هم true را (تقریبا هیچ وقت) بر نمیگرداند.

نویسنده: وحید نصیری
تاریخ: ۱۳۹۳/۰۸/۰۳ ۲۱:۴۴

زمانیکه d1 و d2 حاصل هیچ نوع عملیات ریاضی خاصی نباشند. برای مثال اگر 0.33 را با 0.33 مقایسه کنید. اما مقایسه (double)1/3 == (double)0.33333 هرچند صحیح به نظر می‌رسد اما حاصل false است چون دقت اعشار دو طرف یکی نیست. سمت چپ حداکثر دقت را دارد و سمت راست یک عدد ثابت غیر محاسباتی است. همچنین در بسیاری از محاسبات، نتیجه‌ی نهایی در یک double جای داده می‌شود ؛ مانند d3 در تصاویر فوق. علت اینجا است که مطابق استاندارد IEEE 754، نوع double یک عدد **binary** floating-point است و علت اینکه d3 حاصل از محاسبات در اینجا دقیقا مساوی 0.01 نشده این است که تمام بیت‌های حاصل از عملیات ریاضی محاسبه‌ی آن در double ایی که در کل 64 بیتی است، جای نمی‌گیرد و نتیجه‌ی نهایی، خیلی جزئی کمتر است از 0.01 (rounding error).
اطلاعات بیشتر

نویسنده: وحید نصیری
تاریخ: ۱۳۹۳/۰۸/۰۳ ۲۲:۰

- نوع double در دات نت 64 بیتی و نوع decimal دارای 128 بیت است. نوع double توسط CPU به صورت مستقیم پشتیبانی می‌شود اما نوع decimal خیر. به همین جهت کار کردن با double چندین برابر سریعتر است از decimal.
- نوع double به صورت باینری ذخیره می‌شود؛ اما نوع decimal دقیقا در مبنای 10. به همین جهت نوع decimal برای کارهای رومزه تجاری دارای اعشار، بسیار مناسب‌تر است.

عنوان: **ایجاد یک Abstract Factory با استفاده از جنریک‌ها**
نویسنده: احمد نواصری
تاریخ: ۱۳۹۳/۰۸/۱۱ ۱۵:۲۵
آدرس: www.dotnettips.info
گروه‌ها: C#, Design patterns

همان طور که میدانید از الگوی Factory به عنوان روشی برای کاهش وابستگی اجزای یک سیستم استفاده میشود. در این مقاله میخواهیم با استفاده از جنریک‌ها، الگوی Abstract Factory را پیاده سازی کنیم.

1) ایجاد یک کلاس به نام AbstractFactory و یک متد جنریک به نام CreateObject

```
public class AbstractFactory
    {
        public static T CreateObject<T>() where T : class , new()
        {
            return new T();
        }
    }
```

2) ساخت کلاسهای مورد نظر

```
public class Product
    {
        public void DisplayInfo()
        {
            Console.WriteLine("Product Class Craeted. ");
        }
    }
```

```
public class Category
    {
        public void DisplayInfo()
        {
            Console.WriteLine("Category Class Created.");
        }
    }
```

3) حال در یک برنامه‌ی کنسول ویندوز، از کلاس AbstractFactory به شکل زیر استفاده میکنیم

```
static void Main(string[] args)
        {
            var p = AbstractFactory.CreateObject< Product>();
            p.DisplayInfo();
            Console.WriteLine("======");

            var c = AbstractFactory.CreateObject<Category>();
            c.DisplayInfo();
            Console.WriteLine("======");

            Console.ReadKey();
        }
```

خروجی کد بالا

file:///c:/users/cloud/documents/
Product Class Craeted.
======
Category Class Created.
======

نظرات خوانندگان

نویسنده: سعید
تاریخ: ۱۳۹۳/۰۸/۱۲ ۲۱:۱۶

تو این حالت میشه به صورت مستقیم هم از کلاس Product، آبجکت تعریف کرد. پس در این حالت کاهش وابستگی دقیقا به چه صورتی هست؟ چه فرقی هست بین AbstractFactory.CreateObject<Product>() و new Product ()؟

نویسنده: محسن خان
تاریخ: ۱۳۹۳/۰۸/۱۲ ۲۱:۳۵

مطلب فوق کمی خلاصه شده هست. یک مثال عملی اون رو برای کاهش وابستگی‌ها، می‌تونید اینجا مطالعه کنید: استفاده از Factories برای حذف Service locators در برنامه‌های WinForms

نویسنده: احمد نواصری
تاریخ: ۱۳۹۳/۰۸/۲۲ ۱۰:۲۸

درسته . اما در این حالت دیگر کلاینت شما مستقیما به کار ساخت Object را انجام نمیدهد و کار را به کلاس دیگری واگذار کرده.

عنوان: **آشنایی با Saltarelle کامپایلر قدرتمند C# به جاوااسکریپت**
نویسنده: رضا بازرگان
تاریخ: ۱۳۹۳/۰۸/۱۶ ۱۹:۵۵
آدرس: www.dotnettips.info
گروه‌ها: C#, JavaScript, Javascript extensions, Compiler, Saltarelle

شاید ساده‌ترین تعریف برای Saltarelle این باشد که «کامپایلریست که کدهای C# را به جاوا اسکریپت تبدیل می‌کند». محاسن زیادی را می‌توان برای اینگونه کامپایلرها نام برد؛ مخصوصا در پروژه‌های سازمانی که نگهداری از کدهای جاوا اسکریپت بسیار سخت و گاهی خارج از توان است و این شاید مهمترین عامل ظهور ابزارهای جدید از قبیل Typescript باشد.

در هر صورت اگر حوصله و وقت کافی برای تجهیز تیم نرم افزاری، به دانش یک زبان جدید مانند Typescript نباشد، استفاده از توان و دانش تیم تولید، از زبان C# ساده‌ترین راه حل است و اگر ابزاری مطمئن برای استفاده از حداکثر قدرت JavaScript همراه با امکانات نگهداری و توسعه کدها وجود داشته باشد، بی شک Saltarelle یکی از بهترین‌های آنهاست.

قبلا کامپایلر هایی از این دست مانند #Script وجود داشتند، اما فاقد همه امکانات C# بوده وعملا قدرت کامل C# در کد نویسی وجود نداشت. اما با توجه به ادعای توسعه دهندگان این کامپایلر سورس باز در استفاده‌ی حداکثری از کلیه ویژگی‌های C# 5 و با وجود Library های متعدد می‌توان Saltarelle را عملا یک کامپایلر موفق در این زمینه دانست.

برای استفاده از Saltarelle در یک برنامه وب ساده باید یک پروژه Console Application به Solution اضافه کرد و پکیج Saltarelle.Compiler را از nuget نصب نمایید. بعد از نصب این پکیج، کلیه Reference ها از پروژه حذف می‌شوند و هر بار Build توسط کامپایلر Saltarelle انجام می‌شود. البته با اولین Build ، مقداری Error را خواهید دید که برای از بین بردنشان نیاز است پکیج Saltarelle.Runtime را نیز در این پروژه نصب نمایید:

```
PM> Install-Package Saltarelle.Compiler
PM> Install-Package Saltarelle.Runtime
```

در صورتیکه کماکان Build نهایی با Error همرا بود، یکبار این پروژه را Unload و سپس مجددا Load نمایید

UI یک پروژه وب MVC است و Client یک Console Application که پکیج‌های مورد نیاز Saltarelle روی آن نصب شده است.

در صورتیکه پروژه را Build نماییم و نگاهی به پوشه‌ی Debug بیاندازیم، یک فایل JavaScript همنام پروژه وجود دارد:

Name	Date modified	Type	Size
mscorlib.dll	11/7/2014 13:16	Application extens...	
mscorlib	11/7/2014 13:16	XML Document	
SaltarelleSample.Client	11/7/2014 13:32	Application	
SaltarelleSample.Client.exe	11/7/2014 13:15	XML Configuratio...	
SaltarelleSample.Client	11/7/2014 13:17	JavaScript File	
SaltarelleSample.Client	11/7/2014 13:32	Program Debug D...	
SaltarelleSample.Client.vshost	11/7/2014 13:16	Application	
SaltarelleSample.Client.vshost.exe	11/7/2014 13:15	XML Configuratio...	
SaltarelleSample.Client.vshost.exe.manifest	6/18/2013 16:58	MANIFEST File	

برای اینکه بعد از هر بار Build ، فایل اسکریپت به پوشه‌ی مربوطه در پروژه UI منتقل شود کافیست کد زیر را در Post Build پروژه Client بنویسیم:

```
copy "$(TargetDir)$(TargetName).js" "$(SolutionDir)SalratelleSample.UI\Scripts"
```

اکنون پس از هر بار Build ، فایل اسکریپت مورد نظر در پوشه‌ی Scripts پروژه UI آپدیت می‌شود:

در ادامه کافیست فایل اسکریپت را به layout اضافه کنیم.

```
<script src="~/Scripts/SaltarelleSample.Client.js"></script>
```

در پوشه‌ی Saltarelle.Runtime در پکیج‌های نصب شده، یک فایل اسکریپت به نام mscorlib.min.js نیز وجود دارد که حاوی اسکریپت‌های مورد نیاز Saltarelle در هنگام اجراست. آن را به پوشه اسکریپت‌های پروژه UI کپی نمایید و سپس به Layout اضافه کنید.

```
<script src="~/Scripts/mscorlib.min.js"></script>
<script src="~/Scripts/SaltarelleSample.Client.js"></script>
```

حال نوبت به اضافه نمودن libraryهای مورد نیازمان است. برای دسترسی به آبجکت هایی از قبیل document, window, element و غیره در جاوااسکریپت می‌توان پکیج Saltarelle.Web را در پروژه‌ی Client نصب نمود و برای دسترسی به اشیاء و فرمانهای jQuery، پکیج Salratelle.jQuery را نصب نمایید.

```
> Install-Package Saltarelle.Web
> Install-Package Saltarelle.jQuery
```

به این libraryها imported library می‌گویند. در واقع، در زمان کامپایل، برای این libraryها فایل اسکریپتی تولید نمی‌شود و فقط آبجکت‌های #C هستند که که هنگام کامپایل تبدیل به کدهای ساده اسکریپت می‌شوند که اگر اسکریپت مربوط به آنها به صفحه اضافه نشده باشد، اجرای اسکریپت با خطا مواجه می‌شود.

به طور ساده‌تر وقتی از jQuery library استفاده می‌کنید هیچ فایل اسکریپت اضافه‌ای تولید نمی‌شود، اما باید اسکریپت jQuery به صفحه شما اضافه شده باشد.

```
<script src="~/Scripts/jquery-1.10.2.min.js"></script>
```

مثال ما یک اپلیکیشن ساده برای خواندن فیدهای همین سایت است. ابتدا کدهای سمت سرور را در پروژه UI می نویسیم.

کلاس‌های مورد نیاز ما برای این فید ریدر:

```
public class Feed
    {
        public string FeedId { get; set; }
        public string Title { get; set; }
        public string Address { get; set; }

    }
    public class Item
    {
        public string Title { get; set; }
        public string Link { get; set; }
        public string Description { get; set; }
    }
```

و یک کلاس برای مدیریت منطق برنامه

```
public class SiteManager
    {
        private static List<Feed> _feeds;
        public static List<Feed> Feeds
        {
            get
```

```
            {
                if (_feeds == null)
                    _feeds = CreateSites();
                return _feeds;
            }
        }
        private static List<Feed> CreateSites()
        {
            return new List<Feed>() {
                new Feed(){
                    FeedId = "1",
                    Title = "آخرین تغییرات سایت",
                    Address = "http://www.dotnettips.info/rss.xml"
                },
                 new Feed(){
                    FeedId = "2",
                    Title = "مطالب سایت",
                    Address = "http://www.dotnettips.info/feeds/posts"
                },
                 new Feed(){
                    FeedId = "3",
                    Title = "نظرات سایت",
                    Address = "http://www.dotnettips.info/feeds/comments"
                },
                 new Feed(){
                    FeedId = "4",
                    Title = "خلاصه اشتراک ها",
                    Address = "http://www.dotnettips.info/feed/news"
                },
            };
        }

        public static IEnumerable<Item> GetNews(string id)
        {
            XDocument feedXML = XDocument.Load(Feeds.Find(s=> s.FeedId == id).Address);
            var feeds = from feed in feedXML.Descendants("item")
                        select new Item
                        {
                            Title = feed.Element("title").Value,
                            Link = feed.Element("link").Value,
                            Description = feed.Element("description").Value
                        };
            return feeds;
        }

    }
```

کلاس SiteManager فقط یک لیست از فیدها دارد و متدی که با گرفتن شناسه‌ی فید ، یک لیست از آیتم‌های موجود در آن فید ایجاد می‌کند.

حال دو ApiController برای دریافت داده‌ها ایجاد می‌کنیم

```
public class FeedController : ApiController
{
        // GET api/<controller>
        public IEnumerable<Feed> Get()
        {
            return SiteManager.Feeds;
        }
    }

public class ItemsController : ApiController
    {
        // GET api/<controller>/5
        public IEnumerable<Item> Get(string id)
        {
            return SiteManager.GetNews(id);
        }
    }
```

در View پیش‌فرض که Index از کنترلر Home است، یک Html ساده برای فرم صفحه اضافه می‌کنیم

```
<div>
    <div>
        <h2>Feeds</h2>
        <ul id="Feeds">

        </ul>
    </div>
    <div>
        <h2>Items</h2>
        <p id="FeedItems">
        </p>
    </div>

</div>
```

در المنت Feeds لیست فیدها را قرار می‌دهیم و در FeedItems آیتم‌های مربوط به هر فید. حال به سراغ کدهای سمت کلاینت می‌رویم و به جای جاوا اسکریپت از Saltarelle استفاده می‌کنیم.

کلاس Program را از پروژه Client باز می‌کنیم و متد Main را به شکل زیر تغییر می‌دهیم:

```
static void Main()
        {
            jQuery.OnDocumentReady(() => {
                FillFeeds();
            });
        }
```

بعد از کامپایل شدن، کد C# شارپ بالا به صورت زیر در می‌آید:

```
$SaltarelleSample_Client_$Program.$main = function() {
$(function() {
$SaltarelleSample_Client_$Program.$fillFeeds();
});
};
$SaltarelleSample_Client_$Program.$main();
```

و این همان متد معروف jQuery است که Saltarelle.jQuery برایمان ایجاد کرده است.

متد FillFeeds را به شکل زیر پیاده سازی می‌کنیم

```
private static void FillFeeds()
        {
            jQuery.Ajax(new jQueryAjaxOptions()
            {
                Url = "/api/feed",
                Type = "GET",
                Success = (d,t,r) => {

                    // Fill
                    var ul = jQuery.Select("#Feeds");
                    jQuery.Each((List<Feed>)d, (idx,i) => {
                        var li = jQuery.Select("<li>").Text(i.Title).CSS("cursor", "pointer");
                        li.Click(eve => {
                            FillData(i.FeedId);
                        });
                        ul.Append(li);
                    });
                }
            });
        }
```

آبجکت jQuery، متدی به نام Ajax دارد که یک شی از کلاس jQueryAjaxOptions را به عنوان پارامتر می‌پذیرد. این کلاس کلیه خصوصیات متد Ajax در jQuery را پیاده سازی می‌کند. نکته شیرین آن توانایی نوشتن lambda برای Delegate هاست.

خاصیت Success یک Delegate است که 3 پارامتر ورودی را می‌پذیرد.

```
public delegate void AjaxRequestCallback(object data, string textStatus, jQueryXmlHttpRequest request);
```

data همان مقداریست که api باز می‌گرداند که یک لیست از Feed هاست. برای زیبایی کار، من یک کلاس Feed در پروژه Client اضافه می‌کنم که خصوصیاتی مشترک با کلاس اصلی سمت سرور دارد و مقدار برگشتی Ajax را به آن تبدیل می‌کنم.

کلاس Feed و Item

```
[PreserveMemberCase()]
    public class Feed
    {
        //[ScriptName("FeedId")]
        public string FeedId;

        //[ScriptName("Title")]
        public string Title;

        //[ScriptName("Address")]
        public string Address;

    }

    [PreserveMemberCase()]
    public class Item
    {
        // [ScriptName("Title")]
        public string Title;

        // [ScriptName("Link")]
        public string Link;

        // [ScriptName("Description")]
        public string Description;
    }
```

Attributeهای زیادی در Saltarelle وجود دارند و از آنجایی که کامپایلر اسم فیلدها را camelCase کامپایل می‌کند من برای جلوگیری از آن از PreserveMemberCase بر روی هر کلاس استفاده کردم. می‌توانید اسم هر فیلد را سفارشی کامپایل نمایید.

```
jQuery.Each((List<Feed>)d, (idx,i) => {
                        var li = jQuery.Select("<li>").Text(i.Title).CSS("cursor", "pointer");
                        li.Click(eve => {
                            FillData(i.FeedId);
                        });
                        ul.Append(li);
                    });
```

به ازای هر آیتمی که در شیء بازگشتی وجود دارد، با استفاد از متد each در jQuery یک li ایجاد می‌کنیم. همان طور که می‌بینید کلیه خواص، به شکل Fluent قابل اضافه شدن می‌باشد. سپس برای li یک رویداد کلیک که در صورت وقوع، متد FillData را با شناسه فید کلیک شده فراخوانی می‌کند و در آخر li را به المنت ul اضافه می‌کنیم.

برای هر کلیک هم مانند مثال بالا api را با شناسه‌ی فید مربوطه فراخوانی کرده و به ازای هر آیتم، یک سطر ایجاد می‌کنیم.

```
private static void FillData(string p)
        {
            jQuery.Ajax(new jQueryAjaxOptions()
            {
                Url = "/api/items/" + p,
                Type = "GET",
                Success = (d, t, r) => {
                    var content = jQuery.Select("#FeedItems");
                    content.Html("");
                    foreach (var item in (List<Item>)d)
                    {
```

```
                var row = jQuery.Select("<div>").AddClass("row").CSS("direction", "rtl");
                var link = jQuery.Select("<a>").Attribute("href", item.Link).Text(item.Title);
                row.Append(link);
                content.Append(row);
            }
        }
    });
}
```

خروجی برنامه به شکل زیر است:

در این مثال ما از Saltarelle.jQuery برای استفاده از jQuery.js استفاده نمودیم. libraryهای متعددی برای Saltarelle از قبیل linq,angular,knockout,jQueryUI,nodeJs ایجاد شده و همچنین قابلیت‌های زیادی برای نوشتن imported libraryهای سفارشی نیز وجود دارد.

مطمئنا استفاده از چنین کامپایلرهایی راه حلی سریع برای رهایی از مشکلات متعدد کد نویسی با جاوا اسکریپت در نرم افزارهای بزرگ مقیاس است. اما مقایسه آنها با ابزارهایی از قبیل typescript احتیاج به زمان و تجربه کافی در این زمینه دارد.

فایل پروژه ضمیمه

نظرات خوانندگان

نویسنده: یاسر مرادی
تاریخ: ۱۳۹۳/۰۸/۱۶ ۲۳:۴۶

با تکمیل شدن Roslyn و آسانتر شدن امور، روز به روز شاهد تعداد بیشتری از چنین مبدل هایی خواهیم بود، اما به صورت بنیادی هرگونه مبدل کد CSharp به JavaScript یا هر زبان دیگری محکوم به شکست است، و آنچه که به صورت بنیادی مشکل ندارد، تبدیل IL به سایر زبانها است، چرا که فرض کنید شما یک DLL تقریبا ساده مانند Humanizer را که برای کار با رشته‌ها و ... به کار می‌رود را در مثالتان استفاده کنید، در این صورت دیگر برنامه شما کار نخواهد کرد، حتی اگر در حد یک Pluralize کردن باشد، اما اگر تبدیل IL به JavaScript باشد، هر رفتاری را که با DLL شما داشته باشد، همان رفتار را با Humanizer خواهد داشت، برای همین است که امروزه تبدیلگرهای قدرتمند از IL استفاده می‌کنند، مانند JSIL، و SharpKit که اوایل از تبدیل CSharp به JavaScript استفاده می‌کرد و هم اکنون تازه به این نتیجه رسیده که آب در هاون می‌کوبیده و الآن با استفاده از Cecil، به تبدیل IL به JavaScript روی آورده است.

همچنین تبدیل گر مربوطه، باید یکسری کتابخانه جاوا اسکریپتی پایه که امکانات پایه و اولیه .NET را ارائه دهد داشته باشد، که باز دقیقا تبدیلگرهای حرفه ای همین رفتار را دارند، چون همه‌ی امکانات .NET در JavaScript موجود نیست که صرف تبدیل کد کافی باشد و لااقل بعضی امکانات پایه باید ارائه شوند، مثلا برای ایجاد کردن اعداد تصادفی معادل کلاس Random در .NET

نویسنده: رضا بازرگان
تاریخ: ۱۳۹۳/۰۸/۱۸ ۱۹:۳۳

با تشکر از نوشتارتون ذکر دو نکته را لازم می‌دونم.

اول اینکه هدف از این مطلب الزام به استفاده یا عدم استفاده از این نوع کامپایلرها نیست و فقط برای آشنایی با این گونه ابزارها بوده که در حال حاضر در بسیاری از نرم افزارهای اینترپراز در حال استفاده اند.

و دوم اینکه تفاوت ساختاری و ماهیتی سی شارپ و جاوا اسکریپت اونقدر واضح هست که نباید از این دو انتظار یکسان داشت. و مهمترین عامل به وجود اومدن چنین کامپایلرهایی استفاده از سینتکس سی شارپ بوده و نه قدرت دات نت فریم ورک. بنا براین فکر می‌کنم لزومی به وجود مبدل هایی از زبان میانی وجود ندارد و همچنین واضح است کلاس هایی از قبیل Random و غیره که نه توانایی زبان سی شارپ بلکه امکانات درونی دات نت فریم ورک است برای همچین ابزاری بی معناست.

و فکر می‌کنم برای چنین کامپایلری لازم نیست جاوا اسکریپت همه امکانات سی شارپ را داشته باشد. و اینکه سی شارپ بتواند قسمت زیادی از امکانات جاوا اسکریپت را در اختیار برنامه ساز قرار دهد کافیست.

باز هم تشکر می‌کنم

عنوان: **نحوه ایجاد الگوی Singleton به صورت جنریک**
نویسنده: احمد نواصری
تاریخ: ۱۳۹۳/۰۸/۲۲ ۱۳:۰
آدرس: www.dotnettips.info
گروه‌ها: C#, Design patterns

در برخی از مواقع، ایجاد یک وهله از یک کلاس کاری هزینه بر می‌باشد. بنابراین نیاز است تا فقط یک وهله از آن کلاس را ایجاد و تا آخر اجرای برنامه از آن استفاده کرد. این راه حل در قالب یک الگوی طراحی به نام Singleton معرفی شده است. حال می‌خواهیم با استفاده از امکانات جنریک، کلاسی را طراحی کنیم تا عملیات ساخت وهله‌ها را انجام دهد.
نکاتی که در طراحی یک الگوی Singleton باید مد نظر داشت این است که:
دسترسی سازنده کلاس Singleton را از نوع Private تعیین کنیم.
یک فیلد استاتیک از نوع کلاس Singleton تعریف کنیم.
یک خاصیت از نوع استاتیک فقط خواندنی (یعنی فقط get داشته باشد) تعریف کرده تا فیلد استاتیک را مقداردهی و Return کند. به جای پروپرتی میتوان از یک متد استاتیک نیز استفاده کرد.

```
public class SingletonClassCreator<T> where T:class , new()
    {
        private static T _singletoneInstance;
        private static readonly object Lock = new object();

        public static T SingletoneInstance
        {
            get
            {
                lock (Lock)
                {
                    if (_singletoneInstance == null)
                    {
                        _singletoneInstance = new T();
                    }
                }
                return _singletoneInstance;
            }
        }

        private SingletonClassCreator()
        {
        }
    }
```

برای ایجاد حالت Tread-Safe در برنامه هایی که امکان دسترسی همزمان به یک شیء (مثلا در برنامه‌های وب) وجود دارد، از یک بلاک Lock استفاده شده است تا در هر لحظه فقی یک نخ قادر به ایجاد Singleton شود.
حال برای ایجاد وهله‌های Singleton از کلاسهای مورد نظر به صورت زیر عمل میکنیم

```
public class FirstSingleton
    {
        public int Square(int input)
        {
            return input*input;
        }
    }
```

```
static void Main(string[] args)
        {
            var firstSingletone = SingletonClassCreator<FirstSingleton>.SingletoneInstance ;
            Console.WriteLine(firstSingletone.Square(12));
            Console.ReadKey();
        }
```

در خط اول، با تعریف یک متغیر و قرار دادن وهله استاتیک که بوسیله پروپرتی استاتیک SingletoneInstance برگشت داده میشود، یک شی Singleton از کلاس FirstSingleton را ایجاد میکنیم.

نظرات خوانندگان

نویسنده: مصطفی
تاریخ: ۱۳۹۳/۰۸/۲۷ ۹:۳۳

یه روش بهتر برای استفاده در حالت Thread Safe که به نظرم بهینه‌تر هستش در زمان اجرا, بهینه سازی کد به این شکل هستش

```
if (instance == null)
         {
            lock (Lock)
            {
               if (instance == null)
                  instance = new Singleton();
            }
         }
```

با این حالت در صورتی که شی قبلا ایجاد شده باشه هیچ کدوم از Threadها رو بلوک نمیکنه.

نویسنده: مصطفی
تاریخ: ۱۳۹۳/۰۸/۲۷ ۹:۳۴

همچنین میشد این کلاس رو با این لینک تلفیق کرد که یه خروجی جالبتری به وجود بیاد.
با تشکر از زحمت شما

عنوان: **تاثیر فرهنگ جاری سیستم بر روی اعداد در دات نت**
نویسنده: وحید نصیری
تاریخ: ۱۳۹۳/۰۹/۰۵ ۸:۵
آدرس: www.dotnettips.info
گروه‌ها: C#, Globalization, Persian, Culture

در ویندوز 8، مایکروسافت سعی کرده‌است تا تنظیمات بومی مرتبط با ایران، با واقعیت انطباق بیشتری داشته باشد. برای مثال در فرهنگ فارسی سیستم، علامت ممیز آن / است؛ بجای . معمول.

برای آزمایش آن، سعی کنید چنین برنامه‌ای را در ویندوز 8 اجرا کنید:

```
using System;
namespace CultureAndNumbers
```

```
{
    class Program
    {
        static void Main(string[] args)
        {
            var number = Convert.ToDecimal("12.32");
            Console.WriteLine(number);
        }
    }
}
```

در اینجا سعی شده‌است یک عدد دسیمال رشته‌ای به معادل عددی آن تبدیل شود.
خروجی آن به نحو ذیل است:

بله! چون در فرهنگ جاری سیستم، علامت ممیز دیگر . نیست، رشته‌ی 12.32 نیز بی‌معنا است و قابل تبدیل به یک عدد دسیمال نخواهد بود.
همچنین باید دقت داشت تاثیر فرهنگ جاری سیستم بر روی متدهای Convert.ToDecimal و decimal.Parse یکسان است.

روشی برای آزمایش موقت فرهنگ‌های مختلف

برای اینکه بتوان فرهنگ‌های مختلف را به سادگی مورد آزمایش قرار داد، نیاز است خاصیت CurrentCulture ترد جاری برنامه را تغییر داد و پس از پایان کار، مجددا این ترد را به فرهنگ پیش از آزمایش تنظیم کرد. برای این منظور می‌توان از پیاده سازی الگوی IDisposable کمک گرفت:

```
public class CultureScope : IDisposable
    {
        private readonly CultureInfo _originalCulture;

        public CultureScope(string culture)
        {
            _originalCulture = Thread.CurrentThread.CurrentCulture;
            Thread.CurrentThread.CurrentCulture = new CultureInfo(culture);
        }
```

```
        public void Dispose()
        {
            Thread.CurrentThread.CurrentCulture = _originalCulture;
        }
    }
```

در این حالت برای آزمایش فرهنگ فارسی نصب شده در سیستم می‌توان به صورت ذیل عمل کرد. این فرهنگ تنها در چارچوب قطعه کد using، تنظیم می‌شود و پس از آن، مجددا برنامه با فرهنگ اصلی پیش از اجرای این قطعه کد به کار خود ادامه خواهد داد:

```
using (var cultureScope = new CultureScope("fa-IR"))
{
   Console.WriteLine(Thread.CurrentThread.CurrentCulture.NumberFormat.NumberDecimalSeparator);
   var number = decimal.Parse("12.32");
   Console.WriteLine(number);
}
```

تعیین صریح فرهنگ مورد استفاده

یک راه حل برای رفع این مشکل، قید صریح فرهنگ مورد استفاده است. برای مثال اگر اعداد در بانک اطلاعاتی به صورت 12.32 ثبت شده‌اند، می‌توان نوشت:

```
using (var cultureScope = new CultureScope("fa-IR"))
{
   Console.WriteLine(Thread.CurrentThread.CurrentCulture.NumberFormat.NumberDecimalSeparator);
   var number = decimal.Parse("12.32", new CultureInfo("en"));
   Console.WriteLine(number);
}
```

در اینجا فرهنگ انگلیسی به صورت صریح ذکر شده‌است و دیگر فرهنگ تنظیم شده‌ی fa-IR مورد استفاده قرار نخواهد گرفت. اما این روش هم قابل اطمینان نیست. زیرا کاربر می‌تواند در کنترل پنل سیستم، به سادگی علامت ممیز را مثلا به # تغییر دهد و در این حالت باز هم برنامه کرش خواهد کرد. راه حلی که برای این مساله در دات نت وجود دارد، فرهنگی است به نام Invariant که یک کپی فقط خواندنی از فرهنگ انگلیسی را به همراه دارد و در این حالت تنظیمات اختصاصی کاربر در کنترل پنل، ندید گرفته خواهند شد:

```
using (var cultureScope = new CultureScope("fa-IR"))
{
   Console.WriteLine(Thread.CurrentThread.CurrentCulture.NumberFormat.NumberDecimalSeparator);
   var number = decimal.Parse("12.32", CultureInfo.InvariantCulture);
   Console.WriteLine(number);
}
```

اینبار هر چند فرهنگ ترد جاری به fa-IR تنظیم شده‌است اما چون فرهنگ مورد استفاده CultureInfo.InvariantCulture است، از یک فرهنگ انگلیسی فقط خواندنی که تنظیمات محلی کاربر بر روی آن بی تاثیر است، استفاده خواهد شد. یک چنین کدی در تمام سیستم‌ها بدون مشکل کار می‌کند.

نظرات خوانندگان

نویسنده: سالار خلیل زاده
تاریخ: ۱۳۹۳/۰۹/۰۵ ۱۷:۲۷

با تشکر، نکته مهمی بود.

و اینجاست که برنامه نویس‌ها میگن در سیستم من که درست کار می‌کنه! :)

عنوان: پیش پردازنده ها Preprocessors
نویسنده: علی یگانه مقدم
تاریخ: ۱۳۹۳/۰۹/۱۸ ۲۲:۰
آدرس: www.dotnettips.info
گروه‌ها: C#, preprocessors

احتمالا شما با پیش پردازنده ها کم و بیش آشنایی دارید؛ برای آشنایی با پیش پردازنده‌های موجود در سی شارپ می‌توانید به این آدرس بروید.
البته این پیش پردازنده‌ها به قدرتمندی سایر پیش پردازنده هایی که در زبان‌های دیگر مانند سی یا سی پلاس پلاس دیده‌اید نیستند. مثلا نمی‌توانند مقدار دیگری جز مقدارهای بولین دریافت کنند، یا از حافظه‌ی مصرفی استفاده کنند. همچنین باید به یاد داشته باشید که حتما باید قبل از شروع کد، از پیش پردازنده‌های استفاده کنید.

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;
using System.Threading.Tasks;

#define DEBUG
```

Cannot define/undefine preprocessor symbols after first token in file

برای تعریف یک سمبل symbol می‌توانید از پیش پردازنده‌ی #define استفاده و برای حذف آن هم از #undef استفاده کنید. رسم هست که سمبل‌ها با حروف بزرگ تعریف شوند.
عبارات #if,#else,#elif,#endif هم عبارات شرطی هستند که می‌توان برای چک کردن یک سمبل از آن‌ها استفاده کرد:

```
#define DEBUG
...
#if DEBUG
    Console.WriteLine("You have defined DEBUG symbol");
#endif
```

نتیجه آن را می‌توانید در تصویر زیر مشاهده کنید:

بدیهی است که همین سمبل DEBUG را undef کنید متن بالا نمایش داده نخواهد شد.
بهتر است به پیش پردازنده‌های دیگر هم نگاهی بیندازیم:

```
#if STANDARD
    Console.WriteLine("You have defined STANDARD symbol");
#elif PROFESSIONAL
    Console.WriteLine("You have defined PROFESSIONAL symbol");
#elif ULTIMATE
    Console.WriteLine("You have defined ULTIMATE symbol");
#endif
```

حتی می‌توانید از عملگرهای شرطی چون && یا || یا == یا != و... هم استفاده کنید. تکه کد زیر، از این عملگرها بهره جسته است:

```
#if STANDARD && EVAL
    Console.WriteLine("You have defined STANDARD and EVAL symbols");
#endif
```

پیش پردازنده‌های #warning و #error

در پیش پردازنده #warning می‌توانید یک پیام هشدار یا اخطار را به پنجره‌ی warning ارسال کنید؛ ولی برنامه کماکان به اجرای خود ادامه می‌دهد. اما با #error برنامه هم پیام خطا را در پنجره مربوطه نمایش می‌دهد و هم باعث halt شدن برنامه می‌شود.

```
#if STANDARD && EVAL
    Console.WriteLine("You have defined STANDARD and EVAL symbols");
#endif
```

در کد بالا #warning را با #error جابجا می‌کنیم:

region# و endregion#

از این دو عبارت در بین کدها استفاده می‌کنیم. برای بلوک بندی کدها می‌توان از آن‌ها استفاده کرد. برای مثال دسته بندی کدهای نوشته شده مثل جدا کردن propertyها یا رویدادها یا متدها و ...، با محصور شدن تکه کدهای بین این دو، یک علامت + یا - برای انجام عمل expand و collapsed ایجاد می‌شود.

```
#region Public Properties
    public string CustomerID { get; set; }
    public string CompanyName { get; set; }
    public string ContactName { get; set; }
    public string Country { get; set; }
#endregion
```

line#

برای تغییر نام فایل و شماره خطوط در هنگام دیباگ (نمایش خطا و هشدارها در پنجره‌ی نمایش خطاها) به کار می‌رود.

مثلا به تکه کد زیر دقت کنید و همچنین به تصویر بعد از آن، بدون نوشتن line# دقت کنید:

```
namespace CSPreProcessorDirectivesDemo
{
    class Program
    {
        static void Main(string[] args)
        {
            inta a = 100;
            Console.ReadLine();
        }
    }
}
```

Error List

1 Error | 0 Warnings | 0 Messages | Search Error List

	Description	File	Line	Column	Project
1	The type or namespace name 'inta' could not be found (are you missing a using directive or an assembly reference?)	Program.cs	14	13	CSPreProcessorDirectivesDemo

خطای ما در خط 14 فایل program.cs رخ داده است. در تکه کد زیر پیش پردازنده line# را اضافه کردیم:

```
#line 400 "MyFile.cs"
inta a = 100;
```

همانطور که می‌بینید آدرس تکه کد یا خط بعد از آن تغییر پیدا کرد و از آنجا به بعد از 400 به بعد شمرده می‌شود.
طبق منابع نوشته شده این پیش پردازنده موقعی بیشتر سودمند هست که تکه کد، توسط ابزارهای خارجی یا سیستمی ویرایش شده باشد.
در صورتیکه از #line default استفاده کنید، از آن نقطه به بعد، نام فایل و شماره خطاها به صورت عادی اعلام می‌شوند و #line قبلی در نظر گرفته نمی‌شود تا شاید اگر دوباره به #line جدیدی برخورد کند.
#line hidden هم تکه کدهای مربوطه را از دید دیباگر مخفی می‌کند مثل موقعیکه برنامه نویس، کد به کد یا خط به خط برنامه را دیباگ می‌کند ولی از اینجا به بعد از روی این خطوط رد می‌شود تا به یک #line دیگر برسد. منظور از رد شدن، عدم اجرای خطوط نیست؛ بلکه دیباگ خط به خط می‌باشد.

#progma

این پیش پردازنده از دو بخش نام دستور و آگومان‌ها تشکیل شده است:

```
#pragma pragma-name pragma-arguments
```

دات نت از دو نام دستور warning و checksum پشتیبانی می‌کند؛ آرگومان‌هایی که با دستور warning می‌پذیرد:

```
#pragma warning disable
#pragma warning restore
```

با آرگومان disabled تمامی هشدارهای خطوط بعد از آن نادیده گرفته شده و اعلام نمی‌شوند و از restore برای بازگشت از حالت disabled به کار می‌رود. همچنین برای غیر فعال کردن هشدار برای خط یا خطوط خاص هم میتوانید به صورت زیر بنویسید:

```
#pragma warning disable 414
#pragma warning disable 414, 3021
```

#checksum

```
#pragma checksum "filename" "{guid}" "checksum bytes"
```

از این یکی برای ذخیره هشدارها و خطاها در program database یا PDB استفاده می‌شود (برای مواقعیکه پروژه شما قرار است به یک com یا dll تبدیل شود؛ کاربردی زیادی دارد). آرگومان اول نام فایل که بعدا برای مانیتور کردن به راحتی بین کلاس‌ها تشخیص داده شود و دومی که GUID است و همین GUID را باید برای فایل مشخص کنید.

```
// Guid for the interface IMyInterface.
[Guid("F9168C5E-CEB2-4faa-B6BF-329BF39FA1E4")]
interface IMyInterface
{
    void MyMethod();
```

```
}

// Guid for the coclass MyTestClass.
[Guid("936DA01F-9ABD-4d9d-80C7-02AF85C822A8")]
public class MyTestClass : IMyInterface
{
    public void MyMethod() {}
}
```

و checksum _bytes که باید به صورت هگزادسیمال در حالت رشته‌ای نوشته شود و باید بیانگر یک عدد زوج باشد؛ در صورتیکه یک عدد فرد را مشخص کنید، کمپایلر پیش پردازنده شما را در نظر نمی‌گیرد. نهایتا به صورت زیر نوشته می‌شود:

```
class TestClass
{
    static int Main()
    {
        #pragma checksum "file.cs" "{3673e4ca-6098-4ec1-890f-8fceb2a794a2}" "{012345678AB}" // New
checksum
    }
}
```

منابع : http://www.codeguru.com/csharp/.net/using-preprocessor-directives-in-c.htm

http://msdn.microsoft.com/en-us/library/ms173226.aspx

عنوان: **کمی درباره دستورات using**
نویسنده: علی یگانه مقدم
تاریخ: ۱۳۹۳/۰۹/۲۲ ۰:۵۵
آدرس: www.dotnettips.info
گروه‌ها: C#, Visual Studio

شخصی سازی using directives موقعی که یک کلاس جدید را در VS.NET باز میکنید، فضاهای نامی مشخص و تکراری، همیشه به صورت پیش فرض صدا زده شده‌اند و این فضاهای نام را مایکروسافت بر اساس بیشترین کاربرد و استفاده توسط برنامه نویسان قرار داده است؛ ولی در خیلی از اوقات این فضاهای نام پیش فرض، چنان هم برای خیلی از برنامه نویسان کاربردی نداشته یا با توجه به برنامه هایی که می‌نویسند همیشه متفاوت هست و هربار مجبورند فضاهای نام خاصی را صدا بزنند.
برای مثال فضای نام System.ComponentModel.DataAnnotations را در نظر بگیرید که برنامه نویس میخواهد برای مدل‌های نوشته شده خود از تگ‌های متا استفاده کند و باید در هرکلاس ساخته شده، یکبار مورد بالا را صدا بزند که بیشتر باعث کند شدن کار برنامه نویس می‌شود. پس باید کاری کنیم که پیش فرض‌های فضای نام به آنچه خودمان میخواهیم تغییر پیدا کند.
برای این منظور، به محل نصب ویژوال استودیو رفته و مسیر زیر را دنبال کنید (به مسیر دقت کنید ،در اینجا زبان سی شارپ انتخاب شده است):

```
X:\...\IDE\ItemTemplates\CSharp\Code\1033
```

در اینجا تعدادی دایرکتوری با اسامی آشنا می‌بینید که داخل هر کدام از آن‌های یک فایل به اسم class.cs هست و اگر آن را باز کنید یک نمونه یا قالب برای usingها قابل مشاهده است. برای مثال ما وارد دایرکتوری class می‌شویم و فایل class.cs را باز می‌کنیم:

```
using System;
using System.Collections.Generic;
$if$ ($targetframeworkversion$ >= 3.5)using System.Linq;
$endif$using System.Text;
$if$ ($targetframeworkversion$ >= 4.5)using System.Threading.Tasks;
$endif$
namespace $rootnamespace$
{
    class $safeitemrootname$
    {
    }
}
```

الان باید با یک نگاه به الگو، مشخص باشد که چکار باید بکنید.
یک سری از فضاهای نام که در تمامی فریمورک‌ها استفاده میشوند به همان شکل عادی نوشته شده‌اند. ولی آنهایی که از نسخه‌ی خاصی از یک فریم ورک اضافه شده‌اند باید توسط شرط مورد نظر اضافه شده و اعلام شود که این فضای نام از چه نسخه‌ی فریم ورکی به بعد باید اضافه گردد:

```
$if$ ($targetframeworkversion$ >= 3.5)using System.Linq;//فضای نام مورد نظر
$endif$
```

حالا تغییرات را ذخیره کنید و در VS.NET یک کلاس جدید را ایجاد کنید. همانطور که خواهید دید، تغییرات شما اعمال شده‌است.
برای اعمال تغییرات نیازی به بستن و باز کردن مجدد VS.NET نمی‌باشد. در لحظه ایجاد کلاس الگو خوانده می‌شود.
حال در همان دایرکتوری سی شارپ دقت کنید، می‌بینید که برای موارد دیگری هم فایل هایی وجود دارند. برای مثال برای اینترفیس‌ها یا silverlight و ... که هر کدام را می‌توانید جداگانه تغییر دهید. *نکته:احتمال دارد در نسخه‌های متفاوت به خصوص پایین‌تر مثل نسخه 8 ویژوال استودیو ، فایل class.cs به صورت zip باشد که بعد از تغییرات باید دوباره به حالت zip بازگردانده شود.*

حذف فضای نام‌های اضافی

هر موقع که کلاس جدیدی میسازیم، namespaceها به صورت پیش فرض که در بالا اشاره کردیم وجود دارند و شاید اصلا در آن کلاس از آن‌ها استفاده نمی‌کنیم یا حتی خودمان در حین نوشتن کدها چند namespace خاص را اضافه می‌کنیم که شاید در طول برنامه نویسی چندتایی را بلا استفاده بگذاریم. برای همین همیشه فضای نام هایی صدا زده شده‌اند که اصلا در آن کلاس استفاده

نشده‌اند. پس برای همین بهتر هست که این رفرنس‌های بلا استفاده را پیدا کرده و آن‌ها را حذف کنیم.
شاید این سوال برای بعضی‌های پدید بیاد که چرا باید این‌ها را حذف کنیم، چون کاری هم با ما ندارند و ما هم کاری با آن‌ها نداریم؟
این کار چند علت میتواند داشته باشد:
تمیزکاری کد و خلوت شدن فضای کدنویسی
ممکن هست بعدها گیج کننده شود که من چرا از این‌ها استفاده کردم؟ در آینده با نگاه به یک کد تمیزتر متوجه میشوید یک کد از چه چیزهایی برای انجام کارش بهره‌مند شده و هم اینکه در کارهای گروهی و تیمی هم این مورد به شدت تاثیرگذار هست.
باعث کند شدن تحلیل‌های ایستا میشه (اینجا و اینجا)
کمپایل شدن کد کندتر میشه
موقع تست برنامه، اجرای اولیه کندتر خواهد بود چون CLR باید این نوع موارد را شناسایی و حذف کند
همه موارد بالا در مورد رفرنس‌های موجود یا همان dllهای موجود در شاخه‌ی Bin و References هم صدق می‌کند.
برای حذف فضاهای نام اضافی در یک صفحه می‌توانید از طریق این مسیر انجام بدید:

```
Edit>
```

IntelliSense >Organize Usings>Remove Unused using برای مرتب سازی هم گزینه Sort Usings و انجام هر دو کار Remove and Sort موجود هست. البته اگه روی صفحه هم راست کلیک کنید گزینه Organize Usings هم وجود دارد. می توانید از ابزارهایی چون Power tools Extensions هم استفاده کنید (در صورتی که ویژوال استودیوی شما گزینه‌های مورد نظر را ندارد، این ابزار را نصب نمایید)
در صورتی که از ابزارهایی چون telerik یا devexpress استفاده می‌کنید یا از هر ابزار اضافی که بر روی IDE نصب می‌شود، عموما چنین گزینه هایی حتی با امکانات وسیعتر وجود دارند. مثلا whole tomato هم یکی از این ابزارهاست.
این نکته را هم خاطر نشان کنم در صورتیکه فضاهای نامی بین پیش پردازنده ها که در قبل توضیح دادیم محصور شده باشند، حذف نخواهند شد و همانطور باقی خواهند ماند.
در مورد کامنت‌های بین usingها به قطعه کد زیر نگاه کنید:

```
using System;
/* Comment before remains */
using /* Comment between removed */ System.Linq;
// Comment after remains
namespace ConsoleApplication1
{
class Program
{
static void Main(string[] args)
{
Console.WriteLine("My Example");
}
}
}
```

و حالا بعد از حذف فضای نام‌های اضافی

```
using System;
/* Comment before remains */
// Comment after remains
namespace ConsoleApplication1
{
class Program
{
static void Main(string[] args)
{
Console.WriteLine("My Example");
}
}
}
```

برای اینکه این عمل را بتونید در کل صفحات اعمال کنید می‌توانید از *cleanup selected code* هم استفاده کنید؛ به جز اینکه فضاهای نام اضافی را هم پاک می‌کند، کلیه کدهای شما را در قالبی شکیل‌تر و خواناتر قرار خواهد داد.

با کلیدهای *Ctrl+k+d* سند انتخابی و با کلیدهای ترکیبی *Ctrl+k+f* هم محدوده انتخاب شده قالب بندی می‌شود. یکی دیگر از ابزارهایی که می‌توان با آن‌ها به کد سر و سامان بهتری داد، افزونه‌ی codemaid هست.

ویژگی سی شارپ 6 در مورد Using فرض کنید ما یک کلاس ایستا به نام utilities ایجاد کردیم که یک متد به اسم addints دارد. حالا و این کلاس در namespace به نام SomeNamespace قرار دارد. مطمئنا در این حالت ما ابتدا فضای نام را using میکنیم و سپس در کد کلاس، متد را به شکل زیر صدا میزنیم:

```
using System;
using SomeNamespace;

namespace ConsoleApplication1
{
    class Program
    {
        static void Main(string[] args)
        {
            int sum = Utility.AddInts(5, 2);

            Console.ReadLine();
        }
    }
}
```

ولی در سی شارپ 6 میتوانید بعد از فضای نام، یک . گذاشته و سپس اسم کلاس ایستا static را بیاورید و در کد مستقیما متد دلخواه خود را صدا بزنید.

به شکل زیر دقت کنید:

```
using System;
using SomeNamespace.Utility;

namespace ConsoleApplication1
{
    class Program
    {
        static void Main(string[] args)
        {
            int sum = AddInts(5, 2);

            Console.ReadLine();
        }
    }
}
```

نکته پایانی:در visual studio 2014 فضاهای نام اضافی به رنگ خاکستری نمایش داده می‌شوند.

منابع: http://blogs.msdn.com/b/steve/archive/2007/04/10/changing-the-default-using-directives-in-visual-studio.aspx

http://stackoverflow.com/questions/629667/why-remove-unused-using-directives-in-c

http://stackoverflow.com/questions/5755942/how-do-you-auto-format-code-in-visual-studio

/http://csharp.2000things.com

عنوان: **استفاده از Interop.word برای جایگزین کردن مقادیر در تمامی فایل (Header - Footer - ...)**
نویسنده: امیر عزیزخانی
تاریخ: ۱۳۹۳/۱۰/۲۰ ۱۹:۱۰
آدرس: www.dotnettips.info
گروه‌ها: C#, PDF, MS Office, MS Word

یکی از متداول‌ترین کارهایی که با اسناد می‌توان انجام داد، تهیه خروجی pdf از word و پر کردن یک فایل word با مقادیر ورودی است که سعی داریم یک نمونه از آن را اینجا بررسی کنیم. کد عمومی برای جایگزین کردن:

```
public void MsInteropReplace(Microsoft.Office.Interop.Word.Application doc, object findText, object
replaceWithText)
        {
            object matchCase = false;
            object matchWholeWord = true;
            object matchWildCards = false;
            object matchSoundsLike = false;
            object matchAllWordForms = false;
            object forward = true;
            object format = false;
            object matchKashida = false;
            object matchDiacritics = false;
            object matchAlefHamza = false;
            object matchControl = false;
            object read_only = false;
            object visible = true;
            object replace = 2;
            object wrap = 1;
            //execute find and replace
            doc.Selection.Find.Execute(ref findText, ref matchCase, ref matchWholeWord,
                ref matchWildCards, ref matchSoundsLike, ref matchAllWordForms, ref forward, ref wrap,
ref format, ref replaceWithText, ref replace,
                ref matchKashida, ref matchDiacritics, ref matchAlefHamza, ref matchControl);
        }
```

و یا این مورد:

```
private static void MsInteropReplace2()
        {
            var doc = new
Microsoft.Office.Interop.Word.Application().Documents.Open(@"D:\temp\te1.docx");

            doc.Content.Find.Execute("@levelOrder", false, true, false, false, false, true, 1, false,
"12345", 2,
            false, false, false, false);
            object missing = System.Reflection.Missing.Value;
            doc.SaveAs(@"D:\temp\out.docx", ref missing, ref missing, ref missing, ref missing
                , ref missing, ref missing, ref missing, ref missing, ref missing, ref missing
                , ref missing, ref missing, ref missing, ref missing, ref missing);
}
```

که هر دو مورد را در stackoverflow میتوانید پیدا کنید. به شخصه از این مورد برای replace کردن مقادیر در یک فایل template.docx استفاده میکردم؛ ولی بعد از مدتی فهمیدم که Footerها و Header را نمیتواند پردازش کند. کد زیر در تمامی قسمت‌هایی که در یک فایل word می‌توان متغیر تعریف کرد را گشته و عمل پر کردن مقادیر را بر روی فایل نمونه، انجام می‌دهد و شامل سه متد ذیل است:

```
private static void repAll()
        {
            object Missing = System.Reflection.Missing.Value;

            Application app = null;
            Microsoft.Office.Interop.Word.Document doc = null;
            try
            {
                app = new Microsoft.Office.Interop.Word.Application();

                doc = app.Documents.Open(@"D:\temp\te1.docx", Missing, Missing, Missing, Missing,
Missing, Missing, Missing, Missing, Missing);

                FindReplaceAnywhere(app, "@levelOrder", "محرمانه");
```

```
                doc.SaveAs(@"D:\temp\out.docx", Missing, Missing, Missing, Missing, Missing, Missing,
Missing, Missing, Missing);
            }
            finally
            {
                try
                {
                    if (doc != null) ((Microsoft.Office.Interop.Word._Document)doc).Close(true,
Missing, Missing);
                }
                finally { }
                if (app != null) ((Microsoft.Office.Interop.Word._Application)app).Quit(true, Missing,
Missing);
            }
        }

        private static void searchAndReplaceInStory(Microsoft.Office.Interop.Word.Range rngStory,
string strSearch, string strReplace)
        {
            rngStory.Find.ClearFormatting();
            rngStory.Find.Replacement.ClearFormatting();
            rngStory.Find.Text = strSearch;
            rngStory.Find.Replacement.Text = strReplace;
            rngStory.Find.Wrap = WdFindWrap.wdFindContinue;
            object Missing = System.Reflection.Missing.Value;

            object arg1 = Missing; // Find Pattern
            object arg2 = Missing; //MatchCase
            object arg3 = Missing; //MatchWholeWord
            object arg4 = Missing; //MatchWildcards
            object arg5 = Missing; //MatchSoundsLike
            object arg6 = Missing; //MatchAllWordForms
            object arg7 = Missing; //Forward
            object arg8 = Missing; //Wrap
            object arg9 = Missing; //Format
            object arg10 = Missing; //ReplaceWith
            object arg11 = WdReplace.wdReplaceAll; //Replace
            object arg12 = Missing; //MatchKashida
            object arg13 = Missing; //MatchDiacritics
            object arg14 = Missing; //MatchAlefHamza
            object arg15 = Missing; //MatchControl

            rngStory.Find.Execute(ref arg1, ref arg2, ref arg3, ref arg4, ref arg5, ref arg6, ref arg7,
ref arg8, ref arg9, ref arg10, ref arg11, ref arg12, ref arg13, ref arg14, ref arg15);
        }

        // Main routine to find text and replace it,
        //   var app = new Microsoft.Office.Interop.Word.Application();
        public static void FindReplaceAnywhere(Microsoft.Office.Interop.Word.Application app, string
findText, string replaceText)
        {
            // http://forums.asp.net/p/1501791/3739871.aspx
            var doc = app.ActiveDocument;

            // Fix the skipped blank Header/Footer problem
            //    http://msdn.microsoft.com/en-us/library/aa211923(office.11).aspx
            Microsoft.Office.Interop.Word.WdStoryType lngJunk =
doc.Sections[1].Headers[WdHeaderFooterIndex.wdHeaderFooterPrimary].Range.StoryType;

            // Iterate through all story types in the current document
            foreach (Microsoft.Office.Interop.Word.Range rngStory in doc.StoryRanges)
            {

                // Iterate through all linked stories
                var internalRangeStory = rngStory;

                do
                {
                    searchAndReplaceInStory(internalRangeStory, findText, replaceText);

                    try
                    {
                        //   6 , 7 , 8 , 9 , 10 , 11 -- http://msdn.microsoft.com/en-
us/library/aa211923(office.11).aspx
                        switch (internalRangeStory.StoryType)
                        {
                            case Microsoft.Office.Interop.Word.WdStoryType.wdEvenPagesHeaderStory: // 6
                            case Microsoft.Office.Interop.Word.WdStoryType.wdPrimaryHeaderStory:   // 7
                            case Microsoft.Office.Interop.Word.WdStoryType.wdEvenPagesFooterStory: // 8
                            case Microsoft.Office.Interop.Word.WdStoryType.wdPrimaryFooterStory:   // 9
                            case Microsoft.Office.Interop.Word.WdStoryType.wdFirstPageHeaderStory: //
```

```
10
                            case Microsoft.Office.Interop.Word.WdStoryType.wdFirstPageFooterStory: //
11

                                if (internalRangeStory.ShapeRange.Count > 0)
                                {
                                    foreach (Microsoft.Office.Interop.Word.Shape oShp in
internalRangeStory.ShapeRange)
                                    {
                                        if (oShp.TextFrame.HasText != 0)
                                        {
                                            searchAndReplaceInStory(oShp.TextFrame.TextRange, findText,
replaceText);
                                        }
                                    }
                                }
                                break;

                            default:
                                break;
                        }
                    }
                    catch
                    {
                        // On Error Resume Next
                    }

                    // ON ERROR GOTO 0 -- http://www.harding.edu/fmccown/vbnet_csharp_comparison.html

                    // Get next linked story (if any)
                    internalRangeStory = internalRangeStory.NextStoryRange;
                } while (internalRangeStory != null); //
http://www.harding.edu/fmccown/vbnet_csharp_comparison.html
            }

        }
```

برای تهیه pdf نیز می‌توانید به کد زیر مراجعه کنید:

```
public static void getFileDocxInPdf()
        {
            // Create a new Microsoft Word application object
            Microsoft.Office.Interop.Word.Application word = new
Microsoft.Office.Interop.Word.Application();

            // C# doesn't have optional arguments so we'll need a dummy value
            object oMissing = System.Reflection.Missing.Value;

            // Get list of Word files in specified directory
            DirectoryInfo dirInfo = new DirectoryInfo(@"D:\temp");
            FileInfo[] wordFiles = dirInfo.GetFiles("*.docx");

            word.Visible = false;
            word.ScreenUpdating = false;

            foreach (FileInfo wordFile in wordFiles)
            {
                // Cast as Object for word Open method
                Object filename = (Object)wordFile.FullName;

                // Use the dummy value as a placeholder for optional arguments
                Microsoft.Office.Interop.Word.Document doc = word.Documents.Open(ref filename, ref
oMissing,
                    ref oMissing, ref oMissing, ref oMissing, ref oMissing, ref oMissing,
                    ref oMissing, ref oMissing, ref oMissing, ref oMissing, ref oMissing,
                    ref oMissing, ref oMissing, ref oMissing, ref oMissing);
                doc.Activate();

                object outputFileName = wordFile.FullName.Replace(".docx", ".pdf");
                object fileFormat = WdSaveFormat.wdFormatPDF;

                // Save document into PDF Format
                doc.SaveAs(ref outputFileName,
                    ref fileFormat, ref oMissing, ref oMissing,
                    ref oMissing, ref oMissing, ref oMissing, ref oMissing,
                    ref oMissing, ref oMissing, ref oMissing, ref oMissing,
                    ref oMissing, ref oMissing, ref oMissing, ref oMissing);
```

```
                // Close the Word document, but leave the Word application open.
                // doc has to be cast to type _Document so that it will find the
                // correct Close method.
                object saveChanges = WdSaveOptions.wdDoNotSaveChanges;
                ((_Document)doc).Close(ref saveChanges, ref oMissing, ref oMissing);
                doc = null;
            }

            // word has to be cast to type _Application so that it will find
            // the correct Quit method.
            ((_Application)word).Quit(ref oMissing, ref oMissing, ref oMissing);
            word = null;
        }
```

نظرات خوانندگان

نویسنده: امیر نوروزیان
تاریخ: ۱۳۹۳/۱۰/۲۱ ۸:۳۸

سلام؛ من ازهمین روش شما استفاده کردم چند وقت پیش به وسیله bookmark:

```
private Document oDoc;
        public void createdoc1()
        {
            var realpath="~/template";
            var filePath = Path.Combine(HttpContext.Current.Server.MapPath("~/template"),
Lcourseid.Text + ".doc");
            var oWordApplication = new Application();
            DirectoryInfo dir = new DirectoryInfo(Server.MapPath(realpath));
            foreach (FileInfo files in dir.GetFiles())
            {
                files.Delete();
            }
            // To invoke MyMethod with the default argument value, pass
            // Missing.Value for the optional parameter.
            object missing = System.Reflection.Missing.Value;

            //object fileName = ConfigurationManager.AppSettings["DocxPath"];@"C:\DocXExample.docx";
            string fileName = @"D:\template1.dot";
            //string fileName1 = @"D:\sss.doc";
            object newTemplate = false;

 object docType = 0;
            object isVisible = true;

            //System.Reflection.Missing.Value is used here for telling that method to use default
parameter values when method execution
            oDoc = oWordApplication.Documents.Open(fileName, newTemplate, docType, isVisible, ref
missing, ref missing, ref missing, ref missing, ref missing, ref missing,
                ref missing, ref missing, ref missing, ref missing, ref missing, ref missing);
           // usable in earlier versions of Microsoft Word v2003  v11
          //  if(Convert.ToInt16(oWordApplication.Version) >=11)
            {
             //Sets or returns a Boolean that represents whether a document is being viewed in reading
layout view.
              oDoc.ActiveWindow.View.ReadingLayout = false;
           }

            //The active window is the window that currently has focus.If there are no windows open, an
exception is thrown.
            //microsoft.office.tools.word.
            oDoc.Activate();

            if (oDoc.Bookmarks.Exists("Title"))
            {
                oDoc.Bookmarks["Title"].Range.Text = "Test Field Entry from webform";
                oDoc.Bookmarks["Address"].Range.Text = "Address Field Entry from webform";
            }

            oDoc.SaveAs(filePath, ref missing);
            oWordApplication.Documents.Close(ref missing, ref missing, ref missing);
            //oWordApplication.Quit(ref SaveChanges, ref missing, ref missing, ref missing);
            ProcessRequest(filePath, Lcourseid.Text);
```

ولی این روش مشکلاتی هم داره. اول اینکه باید روی سرور تنظیمات خاصی رو انجام بدی. البته از تنظیمات منظور تنظیمات دسترسی کاربران هست. ولی استفاده از داک ایکس بیشتر استقبال میشه چون دردسرش کمتره.

نویسنده: امیر عزیزخانی
تاریخ: ۱۳۹۳/۱۰/۲۱ ۲۰:۶

البته Docxابزار خوبیه ولی توی یک مثالی که خواستم ازش استفاده کنم به EXCEPTION خورد و علتش تا اونجایی که من متوجه شدم گیر دادن به محتویات اضافه XML داخل فایل بود

عنوان: **فراخوانی متدهای Controllerها در Viewهای ASP.NET MVC**
نویسنده: تایماز قبادی
تاریخ: ۱۳۹۳/۱۱/۰۶ ۱۵:۱۵
آدرس: www.dotnettips.info
گروه‌ها: C#, MVC, Delegate, Func

شاید در ابتدا فراخوانی متدی از یک کنترلر در یک View کار سختی به نظر برسد، ولی در واقع با استفاده از مفاهیم Lambda expressions و Delegateها این کار بسیار راحت خواهد بود.

برای این کار میتوانیم متد مورد نظر را به صورت یک delegate تعریف کرده و به view ارسال کنیم. فرض کنیم متدی داریم برای برگرداندن مجموع 2 عدد به صورت string:

```
public string Sum(int a,int b)
{
    return (a + b).ToString();
}
```

حال برای اینکه بتوانیم این متد را بصورت یک delegate به view ارسال کنیم لازم است تا یک delegate را بصورت public و در خارج از تعریف کلاسها و درون یک namespace مشخصی تعریف کنیم. در اینجا برای راحتی در همان MvcTest.Controllers namespace یک delegate را بصورت زیر (MvcTest نام پروژه است) تعریف می‌کنیم:

```
public delegate string SumOf2Number(int a, int b);
```

حال میتوانیم بصورت زیر این متد را از طریق ViewBag به View ارسال کنیم:

```
SumOf2Number sum2numbers = Sum;
ViewBag.SumFunc3 = sum2numbers;
```

در روش دوم، می‌توانیم متد مورد نظر را بصورت Func به View ارسال کنیم. این کار را میتوانیم به دو صورت انجام دهیم، که هر دو را در تکه کد زیر خواهید دید:

```
ViewBag.SumFunc =(Func<int,int,string>) Sum;//way 1
ViewBag.SumFunc2 = (Func<int, int, string>)((int a, int b) => { return (a + b).ToString(); });//way 2
```

همانطور که متوجه شدید، در روش اول تنها کاری که کردیم متد Sum را از طریق TypeCasting به یک Func تبدیل کردیم و در روش دوم هم یک Lambda expression را بصورت مستقیم به Func تبدیل کرده و استفاده کردیم.

میتوانیم یک Lambda expression را به یک متغیر delegate نیز ربط دهیم؛ به این صورت:

```
SumOf2Number sum2numbers2 = (int a, int b) => { return (a + b).ToString(); };
```

در نهایت کد بخش کنترلر کلاً به اینصورت خواهد بود:

```
namespace MvcTest.Controllers
{
    public  delegate string SumOf2Number(int a, int b);
```

```
    public class HomeController : Controller
    {
        public ActionResult Index()
        {
            SumOf2Number sum2numbers = Sum;
            SumOf2Number sum2numbers2 = (int a, int b) => { return (a + b).ToString(); };

            ViewBag.SumFunc =(Func<int,int,string>) Sum;
            ViewBag.SumFunc2 = (Func<int, int, string>)((int a, int b) => { return (a + b).ToString();
});
            ViewBag.SumFunc3 = sum2numbers;
            ViewBag.SumFunc4 = sum2numbers2;
            return View();
        }
        public string Sum(int a,int b)
        {
            return (a + b).ToString();
        }
        public ActionResult About()
        {
            ViewBag.Message = "Your application description page.";

            return View();
        }

        public ActionResult Contact()
        {
            ViewBag.Message = "Your contact page.";

            return View();
        }
    }
}
```

و در Index View خواهیم داشت: (البته اصولاً استفاده از controller namespace در سمت view کار درستی نیست، منتها اینجا فقط یک مثال کاربردی ساده است)

```
@using MvcTest.Controllers;
@{
    ViewBag.Title = "Home Page";
}

<h1>
    @ViewBag.SumFunc(7,8)
</h1>
<h1>
    @ViewBag.SumFunc2(9, 10)
</h1>
<h1>
    @ViewBag.SumFunc3(5, 1)
</h1>
<h1>
    @ViewBag.SumFunc4(2, 3)
</h1>
```

عنوان: **نمایش تاریخ بر حسب تعداد روزهای گذشته**
نویسنده: محمد محمدی
تاریخ: ۱۳۹۳/۱۱/۲۳ ۱۵:۵
آدرس: www.dotnettips.info
گروه‌ها: C#, DateTime, Persian

در بیشتر وب سایت‌های شاهد نمایش تاریخ بر حسب تعداد روز/ ماه و یا سال گذشته شده از آن تاریخ هستیم. برای نمونه در سایت جاری تاریخ را بر همین اساس نمایش می‌دهند. نمونه‌ای از آن مانند «در ۲ سال قبل، چهار شنبه ۲۷ دی ۱۳۹۱، ساعت ۰۳:۳۵» می‌باشد. در این مقاله قصد دارم کدهایی را جهت انجام این کار ارائه کنم. در این مثال که در ادامه شاهد آن خواهیم بود، از یک پروژه‌ی Win form ساده، جهت نمایش بهتر استفاده کرده‌ام.

جهت اینکه درک کد و یا توضیح آن نیز ساده‌تر صورت بگیرد، به نظرم ابتدا متدهای مورد استفاده و کلاس‌هایی را که از آنها استفاده کرده‌ام، معرفی کنم بهتر باشد:

از کلاس Persiancalender جهت گرفتن روز/ماه و سال استفاده شده و سه متد به شرح زیر دارد:

GetHour : از این متد برای گرفتن ساعت تاریخ مورد نظر استفاده می‌شود. کد این متد به صورت زیر است:

```
public string GetHour(DateTime lastdate)
        {
            PersianCalendar pc = new PersianCalendar();
            string result = " ساعت " + (((pc.GetHour(lastdate)) < 10) ? ("0" +
pc.GetHour(lastdate).ToString()) : (pc.GetHour(lastdate)).ToString()) + ":" +
(((pc.GetMinute(lastdate)) < 10) ? ("0" + pc.GetMinute(lastdate).ToString()) :
(pc.GetMinute(lastdate)).ToString());
            return result;
        }
```

توضیح: اگر ساعت یا دقیقه تک رقمی باشد، یعنی کمتر از 10، برای نمایش بهتر آن یک صفر را به ابتدای آن اضافه می‌کنیم. یعنی ساعت 1:5 تبدیل می‌شود به 01:05

متد getDay : از این متد برای گرفتن نام روز مورد نظر استفاده می‌شود. ورودی این متد یک enum از نوع DayOfWeek است:

```
public string getDay(DayOfWeek day)
        {
            string Result = "";
            switch (day)
            {
                case DayOfWeek.Friday:
                    Result = "جمعه";
                    break;
                case DayOfWeek.Monday:
                    Result = "دوشنبه";
                    break;
                case DayOfWeek.Saturday:
                    Result = "شنبه";
                    break;
                case DayOfWeek.Sunday:
                    Result = "یکشنبه";
                    break;
                case DayOfWeek.Thursday:
                    Result = "پنج شنبه";
                    break;
                case DayOfWeek.Tuesday:
                    Result = "سه شنبه";

                    break;
                case DayOfWeek.Wednesday:
                    Result = "چهارشنبه";
                    break;
                default:
                    break;
            }
            return Result;
        }
```

و در هر جایی نیاز به گرفتن تاریخ باشد، به صورت زیر عمل خواهیم کرد:

```
getDay(pc.GetDayOfWeek(LastDate))
```

pc یک متغییر از نوع persianclander می‌باشد.

متد GetMounth : همانطور که از نام این متد معلوم است، کار آن بازگشت نام ماه مورد استفاد است. کد آن نیز به صورت زیر می‌باشد:

```
public string GetMounth(int month)
        {
            string[] monthInYear =
{"اسفند","بهمن","دی","آذر","آبان","مهر","شهریور","مرداد","تیر","خرداد","اردیبهشت","فروردین" };
            return monthInYear[month-1];
        }
```

وجود -1 در هنگام return به این دلیل است که زمانیکه قصد دریافت شماره ماه را از شیء PersianClander داشته باشیم، از یک شروع می‌شود. یعنی برای ماه اسفند مقدار 12 و برای ماه فروردین مقدار 1 و در یک آرایه، ایندکس‌ها از صفر شروع می‌شوند.

و اما کد کامل آن برای تبدیل تاریخ، به صورت رشته مورد نظر، به صورت زیر است:

```
private void btnGetDate_Click(object sender, EventArgs e)
        {
            DateTime LastDate = DateTime.Parse(txtLastDate.Text);
            TimeSpan ts = DateTime.Now - LastDate;
            PersianCalendar pc = new PersianCalendar();
            int DifferenceYear = DateTime.Now.Year - LastDate.Year;
            int DiffernceMounth = DateTime.Now.Month - LastDate.Month;
            if(DateTime.Now.Month>LastDate.Month)
                DiffernceMounth = DateTime.Now.Month - LastDate.Month;
            else
            DiffernceMounth=LastDate.Month-DateTime.Now.Month;
            int DifferenceDays = ts.Days;

            StringBuilder Result = new System.Text.StringBuilder("");

            if(DifferenceYear>0)
            {
                Result.Append(DifferenceYear.ToString() + " سال پیش"+" ، 
"+getDay(pc.GetDayOfWeek(LastDate))+" "+pc.GetDayOfMonth(LastDate).ToString()+" " +
GetMounth(pc.GetMonth(LastDate))+" " +pc.GetYear(LastDate)+GetHour(LastDate));
            }
            else if(DiffernceMounth>0)
            {
                Result.Append(DiffernceMounth.ToString() + " ماه پیش" + " ، " +
getDay(pc.GetDayOfWeek(LastDate)) + " " + pc.GetDayOfMonth(LastDate).ToString() + " " +
GetMounth(pc.GetMonth(LastDate)) + " " + pc.GetYear(LastDate) + GetHour(LastDate));
            }
            else if(DifferenceDays>0)
                Result.Append(DifferenceDays.ToString() + " روز پیش" + " ، " +
getDay(pc.GetDayOfWeek(LastDate)) + " " + pc.GetDayOfMonth(LastDate).ToString() + " " +
GetMounth(pc.GetMonth(LastDate)) + " " + pc.GetYear(LastDate) + GetHour(LastDate));
            else if(DifferenceDays==0)
                Result.Append(" امروز" + " ، " + getDay(pc.GetDayOfWeek(LastDate)) + " " +
pc.GetDayOfMonth(LastDate).ToString() + " " + GetMounth(pc.GetMonth(LastDate)) + " " +
pc.GetYear(LastDate) + GetHour(LastDate));

            lblResult.Text = Result.ToString();
        }
```

کد زیر برای دریافت تعداد اختلاف بین ماه‌ها، از تاریخی گذشته تا تاریخ جاری است:

```
if(DateTime.Now.Month>LastDate.Month)
                DiffernceMounth = DateTime.Now.Month - LastDate.Month;
            else
            DiffernceMounth=LastDate.Month-DateTime.Now.Month;
```

چرا از if استفاده شده است؟ فرض کنید تاریخ امروز 2015/12/2 باشد و تاریخی که قصد تبدیل آن را داریم :2014/12/10 است. تعداد اختلافی که بین تعداد ماه‌ها است 8 ماه است و اگر این بررسی چک کردن بزرگ بودن آن دو انجام نشود، مقدار 8- را بر میگرداند که برای کار ما نادرست است. نمونه‌ای از این تبدیل :

دانلود کدهای این مقاله

نظرات خوانندگان

نویسنده: محسن خان
تاریخ: ۱۳۹۳/۱۱/۲۳ ۱۵:۲۰

یک نکته: **("00")** number.ToString به صورت خودکار 0 ماقبل اعداد دو رقمی رو اضافه می‌کنه.

نویسنده: عثمان رحیمی
تاریخ: ۱۳۹۳/۱۱/۲۳ ۱۶:۵۰

جهت تکمیل >>

در صورت نیاز به نمایش دقیقه :

```
double DifferenceMinute = ts.TotalMinutes;
```

و قبل از آخرین else if :

```
else if (DateTime.Now.Date == LastDate.Date && (DifferenceMinute<60 && (DateTime.Now.Hour-
LastDate.Hour<=1)) )
                Result.Append("در " + ConvertMinuteToString((int)DifferenceMinute) + " دقیقه قبل " + " ،
"  + GetHour(LastDate));
```

و تبدیل دقیقه به رشته :

```
public string ConvertMinuteToString(int minute)
        {
            string Result = "";
            string[] minLessTen = { "","نه" ,"هشت" ,"هفت" ,"شش" ,"پنج" ,"چهار" ,"سه" ,"دو" ,"یک"
ده"" };
            string[] minLesstwenty = { "","هفده" ,"شانزده" ,"پانزده" ,"چهارده" ,"سیزده" ,"دوازده" ,"یازده"
نوزده" ,"هجده"" };

            Dictionary<int,string> minmore=new Dictionary<int,string>
                {
                    {2,"بیست"},
                    {3,"سی"},
                    {4,"چهل"},
                    {5,"پنجاه"}
                };

            if (minute <= 10)
                Result = minLessTen[minute - 1];
            else if (minute < 20)
                Result = minLesstwenty[(minute%10)-1];
            else if ((minute / 10) >= 2)
            {
                Result = minmore[minute / 10];
                if (minute % 10 > 0)
                    Result += " و " + minLessTen[(minute % 10) - 1];

            }
            return Result;
        }
```

* بهتر است بجای String از StringBuilder استفاده شود .

عنوان: **مقید سازی پارامترهای نوع جنریک**
نویسنده: عثمان رحیمی
تاریخ: ۱۳۹۳/۱۱/۲۵ ۱۰:۱۵
آدرس: www.dotnettips.info
گروه‌ها: C#, Generics

احتمالا در بیشتر مقالات (فارسی/انگلیسی) عبارات هایی مثل نمونه‌های زیر را دیده اید :

```
where T:clas
where T:struc
...
```

در این مقاله قصد داریم بپردازیم به «مقید سازی پارامترهای نوع جنریک» و اینکه چه کاربردی دارند و در چه زمانی بهتر است از آن‌ها استفاده کنیم و نحوه استفاده از آنها چگونه است. فرض میکنیم که خواننده‌ی محترم با مفاهیم جنریک آشنایی دارد. در صورتیکه با جنریک‌ها آشنا نیستید ابتدا مروری داشته باشید بر ج نریک‌ها و بعد این مقاله را مطالعه فرمایید؛ به این دلیل که موضوع مورد بحث بر پایه‌ی جنریک‌ها می‌باشد.

همانطور که مطلع هستید هر عنصری جنریکی را که تعریف میکنید حداقل دارای یک پارامتر نوع هست و در زمان بکارگیری آن جنریک باید نوع آن را مشخص نمایید. برای نمونه مثال زیر را در نظر بگیرید :

```
public  class MyCollection<T>
        {
            private List<T> collections = new List<T>();
            public void Add(T value)
            {
                collections.Add(value);
            }
        }
```

کلاس فوق یک کلاس جنریک است که در هنگام ساخت نمونه‌ای از آن، باید ابتدا data type نوعی را که که می‌خواهیم با آن کار کنیم، تعیین کنیم. برای مثال در کد فوق در هنگام ساخت نمونه‌ای از آن، نوع int را برای آن مشخص میکنیم و هر وقت بخواهیم متد Add آن را فراخوانی کنیم، فقط نوعی را قبول خواهد کرد که در ابتدا برای آن تعیین کرده ایم (int):

```
MyCollection<int> myintObj = new MyCollection<int>();
            myintObj.Add(12);
            myintObj.Add(33);
             myintObj.Add(33.3);// ERROR z
```

سؤال: می‌خواهیم فقط نوع‌هایی را بتوان به T نسبت داد که از نوع ارجاعی (reference type) هستن و یا فقط نوع هایی را به T نسبت داد که یک سازنده دارند؛ چگونه؟

ایجاد قیدها یا محدودیت‌ها بر روی پارامترهای جنریک‌ها شامل پنج حالت می‌باشد:

حالت اول : Where T:struct
در این حالت T باید یک ساختار باشد .

حالت دوم : where T:class
T باید یک نوع ارجاعی باشد. اگر در مثال فوق این قید را به آن اضافه کنیم، در هنگام ساخت نمونه‌ای از کلاس فوق، اگر یک نوع value type را به T نسبت دهیم، در هنگام وارد کردن یک نوع value type با خطا مواجه خواهیم شد. مثال:

```
public  class MyCollection<T> where T:class
        {
            private List<T> collections = new List<T>();
            public void Add(T value)
            {
                collections.Add(value);
            }
        }
```

و برای استفاده :

```
MyCollection<int> myintObj = new MyCollection<int>(); // ERROR , int is value type
```

حالت سوم : Where T:new()
نوعی که به T نسبت داده می‌شود باید یک سازنده‌ی پیش فرض داشته باشد.
داخل پرانتز : سازنده‌ی پیش فرض: زمانی که شما یک کلاس می‌نویسید اگر آن کلاس دارای هیچ سازنده‌ای نباشد، کامپایلر یک سازنده‌ی بدون پارامتر را به کلاس فوق اضافه می‌کند که کار آن مقدار دهی به فیلدهای کلاس است. در اینجا از مقادیر پیش فرض استفاده می‌شود. مثلا برای int مقدار صفر و برای string مقدار "" و به همین ترتیب.
اگر از مقدار دهی پیش فرض توسط کامپایلر خرسند نیستید، می‌توانید سازنده پیش فرض را تغییر داده و مطابق میل خود فیلدها را مقدار دهی اولیه کنید .

حالت چهارم : where T:NameOfBaseClass
نوعی که به T نسبت داده می‌شود باید از کلاس NameOfBaseClass ارث بری کرده باشد.

حالت پنجم : where T:NameOfInterface
همانند حالت چهارم می‌باشد؛ با این تفاوت: نوعی که به T نسبت داده می‌شود باید واسط NameOfInterface را پیاده سازی کرده باشد.

پنج حالت فوق نمونه‌هایی از ایجاد محدودیت بر روی پرامتر نوع اعضای جنریک بودند و اما در ادامه قصد داریم نکاتی را در این باب، بیان کنیم:

نکته اول : می‌توانید محدودیت‌های فوق را با هم ترکیب کنید برای اینکار آنها را با کاما از هم جدا کنید :

```
public  class MyCollection<T> where T:class,IDisposable,new()
        {
   //content
}
```

نوعی که به T نسبت داده می‌شود
باید از نوع ارجاعی باشد.
باید واسط IDisposable را پیاده سازی کرده باشد.
باید یک سازنده‌ی پیش فرض داشته باشد.

نکته دوم : زمانیکه از چندین محدودیت استفاده می‌کنید مثل مثال فوق، باید محدودیت ()new در آخرین جایگاه محدودیت‌ها قرار گیرد؛ در غیر اینصورت با خطای زمان ترجمه روبه رو خواهید شد .

نکته سوم : می‌توان محدودیت‌های فوق را علاوه بر کلاس، بر روی متدهای جنریک نیز اعمال کنید:

```
public void Swap<T>(ref T val1,ref T val2) where T:struct
            {
//content
            }
```

نکته چهارم : زمانیکه کلاس و یا متدهای شما بیش از یک نوع پارامتر از نوع جنریک را دریافت می‌کنند، باید محدودیت‌های مورد نظر را برای هر کدام به صورت جداگانه قید کنید. به طور مثال به کلاس زیر که دو پارمتر T و K را دارد، باید برای هر کدام جداگانه محدودیت‌های مورد نظر را اعمال کنیم (در صورت نیاز):

```
public  class MyCollection<T,K> where T:class where K:IDisposable,new()
```

```
        {
//content
}
```

عنوان: **رشته ها و پردازش متن در دات نت به زبان ساده**
نویسنده: علی یگانه مقدم
تاریخ: ۱۳۹۳/۱۱/۲۸ ۱۹:۳۵
آدرس: www.dotnettips.info
گروه‌ها: C#, Regular expressions, .NET, string, Performance

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:

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 برشمارد:
خانه‌های آن یک ضرب پر نمیشوند بلکه به ترتیب، خانه به خانه پر می‌شوند.
قبل از انتساب متن باید 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 اصلا 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"; //مسیردهی- \ چاپ خود علامت
```

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 مستقیما 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 را با #C 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 compareها استفاده نمایید. دلیل آن هم این است که برای مقایسه از فرهنگ culture فعلی سیستم استفاده میشود و نظم جدول یونیکد را رعایت نمی‌کنند و ممکن است بعضی رشته‌های 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 رشته‌ها پیدا کنند. برای مثال الگوی [A-Z0-9]+ مشخص می‌کند که رشته مورد نظر نباید خالی باشد و حداقل با یکی از حروف بزرگ یا اعداد پرشده باشد. این الگوها میتوانند برای واکشی داده‌ها یا قالب‌های خاص در رشته‌ها به کار بروند. برای مثال شماره تماس‌ها ، پست الکترونیکی و ...

در اینجا میتواند نحوه‌ی الگوسازی را بیاموزید. کد زیر بر اساس یک الگو، شماره تماس‌های مورد نظر را یافته و البته با فیلتر گذاری آن‌ها را نمایش می‌دهد:

```
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 رشته‌ی بالا با الگوی ما همخوانی دارند که بعد با استفاده از متد replace در شی Regex عبارات دلخواه خودمان را جایگزین شماره تماس‌ها خواهیم کرد. الگوی بالا شماره تماس‌هایی را میابد که با 08 آغاز شده‌اند و بعد از آن 8 عدد دیگر از 0 تا 9 قرار گرفته‌اند. بعد از اینکه متن مطابق الگو یافت شد، ما آن را با الگوی $1******** جایگزین می‌کنیم 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 رشته‌ها در Loop

برای اتصال رشته‌ها ما از علامت + یا متد ایستای string.concat استفاده می‌کنیم ولی استفاده‌ی از آن در داخل یک حلقه باعث کاهش کارآیی برنامه خواهد شد. برای همین بیایید ببینم در حین اتتقال رشته‌ها در حافظه چه اتفاقی رخ میدهد. ما در اینجا دو رشته str1 و str2 داریم که عبارات "super" و "star" را نگه داری می‌کنند و در واقع دو متغیر هستند که به حافظه‌ی پویای Heap اشاره می‌کنند. اگر این دو را با هم جمع کنیم و نتیجه را در متغیر result قرار دهیم، سه متغیر میشوند که هر کدام به حافظه‌ای جداگانه در heap اشاره می‌کنند. در واقع برای این اتصال، قسمت جدیدی از حافظه تخصصیص داده شده و مقدار جدید در آن نشسته‌است. در این حالت یک متغیر جدید ساخته شد که به آدرس آن اشاره می‌کند. کل این فرآیند یک فرآیند کاملا زمانبر است که با تکرار این عمل موجب از دست دادن کارآیی برنامه می‌شود؛ 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 می‌کند. موقعی‌که متغیری به قسمتی از حافظه اشاره می‌کند که دیگر بلا استفاده است، سیستم GC به صورت خودکار آنها را پاکسازی می‌کند که این عمل زمان بر هم خودش موجب کاهش کارآیی می‌شود. همچنین انتقال رشته‌ها 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 تاز 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 مجددا تست بزنید. در این حالت چند دقیقه ای بسته به قدرت سیستم زمان خواهد برد؛ مثلا 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 وجایی برای خود در حافظه Heap میگیرد.

حافظه رشته قدیمی و بافر دیگر بلا استفاده شده‌اند 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 رشته‌ها 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، مثلا 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 مثلا 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 خانه هایی 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: شهروز جعفری
تاریخ: ۱۳۹۳/۱۱/۲۹ ۱۸:۱۰

یک سوال منظور از Gac اینجا چیه؟

نویسنده: علی یگانه مقدم
تاریخ: ۱۳۹۳/۱۱/۲۹ ۱۸:۴۸

ممنون که گوشزد کردید؛ عذر میخوام. مطلب ویرایش شد. منظور GC بود. بنده اشتباها نوشتم GAC.

عنوان: **آشنایی با الگوی طراحی Iterator**
نویسنده: وحید فرهمندیان
تاریخ: ۱۳۹۳/۱۱/۲۹ ۱۸:۵۰
آدرس: www.dotnettips.info
گروه‌ها: C#, Design patterns, IEnumerable, Iterator methods

فرض کنید قبلا کلاسی بنام CollectionClass را داشته‌اید که در آن یک آرایه از نوع String[] تعریف کرده‌اید. همچنین n تا کلاس هم دارید که از آرایه‌ی تعریف شده‌ی در CollectionClass استفاده می‌کنند. تا اینجا مشکلی نیست. مشکل زمانی شروع می‌شود که متوجه می‌شوید دیگر این آرایه کارآیی ندارد و باید آن را با List<string> جایگزین کنید. واضح است که نمی‌توانید همه کلاس‌هایی را که از CollectionClass استفاده کرده‌اند، بیابید و آنها را تغییر دهید؛ چرا که شاید برخی از کلاس‌ها اصلا در دسترس شما نباشند یا هر دلیل دیگری.

راهگشای این مشکل، استفاده از الگوی طراحی Iterator است. در این الگو، باید کلاس CollectionClass ابتدا واسط IEnumerable را پیاده سازی نماید. این واسط متدی بنام GetEnumerator دارد که می‌توان به کمک آن، درون آرایه یا هر نوع کالکشن دیگری حرکت کرده و آیتم‌های آن را برگرداند.(مطالعه بیشتر)

اول این الگو را پیاده سازی می‌کنیم و در ادامه توضیح می‌دهیم که چگونه مشکل ما را حل میکند:

ابتدا باید کلاس CollectionClass واسط IEnumerable را پیاده سازی نماید. در ادامه بدنه متد GetEnumerator را می‌نویسیم:

```
public class CollectionClass : IEnumerable
    {
        private string[] mySet = { "Array of String 1", "Array of String 2", "Array of String 3" };
        public IEnumerator GetEnumerator()
        {
            //return arrayStrings.GetEnumerator();
            foreach (var element in mySet )
            {
                yield return element;
            }
        }
    }
```

در اینجا یک آرایه رشته‌ای را بنام mySet تعریف کرده‌ایم و مقادیر مختلفی را در آن قرار داده‌ایم. سپس در متد GetEnumerator اعضای این آرایه را خوانده و return می‌کنیم.(yield چیست؟)

وقتی از این کلاس می‌خواهیم استفاده کنیم، داریم:

```
CollectionClass c = new CollectionClass();
foreach (var element in c)
{
       Console.WriteLine(element);
}
```

در این حالت مهم نیست که مجموعه‌ی مورد نظر، آرایه هست یا هر نوع کالکشن دیگری. لذا وقتی بخواهیم نوع mySet را تغییر دهیم، نگران نخواهیم بود؛ چراکه فقط کافی‌است کلاس CollectionClass را تغییر دهیم. بصورت زیر:

```
public class CollectionClass : IEnumerable
    {
        //private readonly string[] arrayStrings = { "Array of String 1", "Array of String 2", "Array
of String 3" };
        private List<string> mySet= new List<string>() { "Array of String 1", "Array of String 2",
"Array of String 3" };
        public IEnumerator GetEnumerator()
        {
            foreach (var element in mySet )
            {
                yield return element;
            }
        }
    }
```

عنوان: **ساختار داده‌های خطی Linear Data Structure قسمت اول**
نویسنده: علی یگانه مقدم
تاریخ: ۱۳۹۳/۱۱/۳۰ ۱:۲۰
آدرس: www.dotnettips.info
گروه‌ها: C#, Data Structures, Linked List

بعضی از داده‌ها ساختارهای ساده‌ای دارند و به صورت یک صف یا یک نوار ضبط به ترتیب پشت سر هم قرار می‌گیرند؛ مثل ساختاری که صفحات یک کتاب را نگهداری می‌کند. یکی از نمونه‌های این ساختارها، List، صف، پشته و مشتقات آن‌ها می‌باشند.

ساختار داده‌ها چیست؟

در اغلب اوقات، موقعی‌که ما برنامه‌ای را می‌نویسیم با اشیاء یا داده‌های زیادی سر و کار داریم که گاهی اوقات اجزایی را به آن‌ها اضافه یا حذف می‌کنیم و در بعضی اوقات هم آن‌ها را مرتب سازی کرده یا اینکه پردازش دیگری را روی آن‌ها انجام میدهیم. به همین دلیل بر اساس کاری که قرار است انجام دهیم، باید داده‌ها را به روش‌های مختلفی ذخیره و نگه داری کنیم و در اکثر این روش‌ها داده‌ها به صورت منظم و پشت سر هم در یک ساختار قرار می‌گیرند.

ما در این مقاله، مجموعه‌ای از داده‌ها را در قالب ساختارهای متفاوتی بر اساس منطق و قوانین ریاضیات مدیریت می‌کنیم و بدیهی است که انتخاب یک ساختار مناسب برای هرکاری موجب افزایش کارآیی و کارآمدی برنامه خواهد گشت. می‌توانیم در مقدار حافظه‌ی مصرفی و زمان، صرفه جویی کنیم و حتی گاهی تعداد خطوط کدنویسی را کاهش دهیم.

نوع داده انتزاعی ADT- Abstraction Data Type

به زبان خیلی ساده لایه انتزاعی به ما تنها یک تعریف از ساختار مشخص شده‌ای را می‌دهد و هیچگونه پیاده سازی در آن وجود ندارد. برای مثال در لایه انتزاعی، تنها خصوصیت و عملگرها و ... مشخص می‌شوند. ولی کد آن‌ها را پیاده سازی نمی‌کنیم و این باعث می‌شود که از روی این لایه بتوانیم پیاده سازی‌های متفاوت و کارآیی‌های مختلفی را ایجاد کنیم.

ساختار داده‌های مختلف در برنامه نویسی:

خطی یا Linear: شامل ساختارهایی چون لیست و صف و پشته است: List ,Queue,Stack

درختی یا Tree-Like: درخت باینری ، درخت متوازن و B-Trees

Dictionary : شامل یک جفت کلید و مقدار است در جدول هش

بقیه: گراف‌ها، صف الویت، bags, Multi bags, multi sets

در این مقاله تنها ساختارهای خطی را دنبال می‌کنیم و در آینده ساختارهای پیچیده‌تری را نیز بررسی خواهیم کرد و نیاز است بررسی کنیم کی و چگونه باید از آن‌ها استفاده کنیم.

ساختارهای لیستی از محبوبترین و پراستفاده‌ترین ساختارها هستند که با اشیاء زیادی در دنیای واقعی سازگاری دارند. مثال زیر را در نظر بگیرید:

قرار است که ما از فروشگاهی خرید کنیم و هر کدام از اجناس (المان‌ها) فروشگاه را که در سبد قرار دهیم، نام آن‌ها در یک لیست ثبت خواهد شد و اگر دیگر المان یا جنسی را از سبد بیرون بگذاریم، از لیست خط خواهد خورد.

همان که گفتیم یک ADT میتواند ساختارهای متفاوتی را پیاده سازی کند. یکی از این ساختارها اینترفیس system.collection.IList است که پیاده سازی آن منجر به ایجاد یک کلاس جدید در سیستم دات نت خواهد شد. پیاده سازی اینترفیس‌ها در سی شارپ، قوانین و قراردادهای خاص خودش را دارد و این قوانین شامل مجموعه‌ای از متدها و خصوصیت‌هاست. برای پیاده سازی هر کلاسی از این اینترفیس‌ها باید این متدها و خصوصیت‌ها را هم در آن پیاده کرد.

با ارث بری از اینترفیس system.collection.IList باید رابطهای زیر در آن پیاده سازی گردد:

void Add(object)	افزودن المان به آخر لیست
void Remove(object)	حذف یک المان خاص از لیست
void Clear()	حذف کلیه المان‌ها
bool Contains(object)	شامل این داده میشود یا خیر؟
void RemoveAt(int)	حذف یک المان بر اساس جایگاه یا اندیسش
void Insert(int, object)	افزودن یک المان در جایگاهی (اندیس) خاص بر اساس مقدار position

void Add(object)	افزودن المان به آخر لیست
int IndexOf(object)	اندیس یا جایگاه یک عنصر را بر می‌گرداند
this[int]	ایندکسر ، برای دسترسی به عنصر در اندیس مورد نظر

لیست‌های ایستا static Lists

آرایه‌ها می‌توانند بسیاری از خصوصیات ADT را پیاده کنند ولی تفاوت بسیار مهم و بزرگی با آن‌ها دارند و آن این است که لیست به شما اجازه می‌دهد به هر تعدادی که خواستید، المان‌های جدیدی را به آن اضافه کنید؛ ولی یک آرایه دارای اندازه‌ی ثابت Fix است. البته این نکته قابل تامل است که پیاده سازی لیست با آرایه‌ها نیز ممکن است و باید به طور خودکار طول آرایه را افزایش دهید. دقیقا همان اتفاقی که برای stringbuilder در این مقاله توضیح دادیم رخ می‌دهد. به این نوع لیست‌ها، *لیست‌های ایستایی که به صورت آرایه ای توسعه پذیر پیاده سازی میشوند* می‌گویند. کد زیر پیاده سازی چنین لیستی است:

```
public class CustomArrayList<T>
{
    private T[] arr;
    private int count;

    public int Count
    {
        get
        {
            return this.count;
        }
    }

    private const int INITIAL_CAPACITY = 4;

    public CustomArrayList(int capacity = INITIAL_CAPACITY)
    {
        this.arr = new T[capacity];
        this.count = 0;
    }
```

در کد بالا یک آرایه با طول متغیر INITIAL_CAPACITY که پیش فرض آن را 4 گذاشته ایم می‌سازیم و از متغیر count برای حفظ تعداد عناصر آرایه استفاده می‌کنیم و اگر حین افزودن المان جدید باشیم و count بزرگتر از INITIAL_CAPACITY رسیده باشد، باید طول آرایه افزایش پیدا کند که کد زیر نحوه‌ی افزودن المان جدید را نشان می‌دهد. استفاده از حرف T بزرگ مربوط به مباحث Generic هست. به این معنی که المان ورودی می‌تواند هر نوع داده‌ای باشد و در آرایه ذخیره شود.

```
public void Add(T item)
{
    GrowIfArrIsFull();
    this.arr[this.count] = item;
    this.count++;
}

public void Insert(int index, T item)
{
    if (index > this.count || index < 0)
    {
        throw new IndexOutOfRangeException(
            "Invalid index: " + index);
    }
    GrowIfArrIsFull();
    Array.Copy(this.arr, index,
        this.arr, index + 1, this.count - index);
    this.arr[index] = item;
    this.count++;
}

private void GrowIfArrIsFull()
{
    if (this.count + 1 > this.arr.Length)
    {
        T[] extendedArr = new T[this.arr.Length * 2];
        Array.Copy(this.arr, extendedArr, this.count);
        this.arr = extendedArr;
    }
```

```
}

public void Clear()
{
    this.arr = new T[INITIAL_CAPACITY];
    this.count = 0;
}
```

در متد Add خط اول با تابع GrowIfArrIsFull بررسی می‌کند آیا خانه‌های آرایه کم آمده است یا خیر؟ اگر جواب مثبت باشد، طول آرایه را دو برابر طول فعلی‌اش افزایش می‌دهد و خط دوم المان جدیدی را در اولین خانه‌ی جدید اضافه شده قرار می‌دهد. همانطور که می‌دانید مقدار count همیشه یکی بیشتر از آخرین اندیس است. پس به این ترتیب مقدار count همیشه به خانه‌ی بعدی اشاره می‌کند و سپس مقدار count به روز میشود. متد دیگری که در کد بالا وجود دارد insert است که المان جدیدی را در اندیس داده شده قرار می‌دهد. جهت این کار از سومین سازنده‌ی array.copy استفاده می‌کنیم. برای این کار آرایه مبدا و مقصد را یکی در نظر می‌گیریم و از اندیس داده شده به بعد در آرایه فعلی، یک کپی تهیه کرده و در خانه‌ی بعد اندیس داده شده به بعد قرار می‌دهیم. با این کار آرایه ما یک واحد از اندیس داده شده یک خانه، به سمت جلو حرکت می‌کند و الان خانه index و index+1 دارای یک مقدار هستند که در خط بعدی مقدار جدید را داخل آن قرار می‌دهیم و متغیر count را به روز می‌کنیم. باقی موارد را چون پردازش‌های جست و جو، پیدا کردن اندیس یک المان و گزینه‌های حذف، به خودتان واگذار می‌کنم.

لیست‌های پیوندی Linked List - پیاده سازی پویا

همانطور که دیدید لیست‌های ایستا دارای مشکل بزرگی هستند و آن هم این است که با انجام هر عملی بر روی آرایه‌ها مانند افزودن، درج در مکانی خاص و همچنین حذف (خانه ای در آرایه خالی خواهد شد و خانه‌های جلوترش باید یک گام به عقب برگردند) نیاز است که خانه‌های آرایه دوباره مرتب شوند که هر چقدر میزان داده‌ها بیشتر باشد این مشکل بزرگتر شده و ناکارآمدی برنامه را افزایش خواهد داد.

این مشکل با لیست‌های پیوندی حل می‌گردد. در این ساختار هر المان حاوی اطلاعاتی از المان بعدی است و در لیست‌های پیوندی دوطرفه حاوی المان قبلی است. شکل زیر نمایش یک لیست پیوندی در حافظه است:

برای پیاده سازی آن به دو کلاس نیاز داریم. کلاس ListNode برای نگهداری هر المان و اطلاعات المان بعدی به کار می‌رود که از این به بعد به آن Node یا گره می‌گوییم و دیگری کلاس DynamicList<T> برای نگهداری دنباله ای از گره‌ها و متدهای پردازشی آن.

```
public class DynamicList<T>
{
    private class ListNode
    {
        public T Element { get; set; }
        public ListNode NextNode { get; set; }

        public ListNode(T element)
        {
            this.Element = element;
            NextNode = null;
        }

        public ListNode(T element, ListNode prevNode)
        {
            this.Element = element;
            prevNode.NextNode = this;
        }
    }

    private ListNode head;
    private ListNode tail;
    private int count;

    // …
```

```
}
```

از آن جا که نیازی نیست کاربر با کلاس ListNode آشنایی داشته باشد و با آن سر و کله بزند، آن را داخل همان کلاس اصلی به صورت خصوصی استفاده می‌کنیم. این کلاس دو خاصیت دارد؛ یکی برای المان اصلی و دیگر گره بعدی. این کلاس دارای دو سازنده است که اولی تنها برای عنصر اول به کار می‌رود. چون اولین بار است که یک گره ایجاد می‌شود، پس باید خاصیت NextNode یعنی گره بعدی در آن Null باشد و سازنده‌ی دوم برای گره‌های شماره 2 به بعد به کار می‌رود که همراه المان داده شده، گره قبلی را هم ارسال می‌کنیم تا خاصیت NextNode آن را به گره جدیدی که می‌سازیم مرتبط سازد. سه خاصیت کلاس اصلی به نام‌های Count,Tail,Head به ترتیب برای اشاره به اولین گره، آخرین گره و تعداد گره‌ها، به کار می‌روند که در ادامه کد آن‌را در زیر می‌بینیم:

```
public DynamicList()
{
    this.head = null;
    this.tail = null;
    this.count = 0;
}

public void Add(T item)
{
    if (this.head == null)
    {
        this.head = new ListNode(item);
        this.tail = this.head;
    }
    else
    {
        ListNode newNode = new ListNode(item, this.tail);
        this.tail = newNode;
    }
    this.count++;
}
```

سازنده مقدار دهی پیش فرض را انجام می‌دهد. در متد Add المان جدیدی باید افزوده شود؛ پس چک می‌کند این المان ارسالی قرار است اولین گره باشد یا خیر؟ اگر head که به اولین گره اشاره دارد Null باشد، به این معنی است که این اولین گره است. پس اولین سازنده‌ی کلاس ListNode را صدا می‌زنیم و آن را در متغیر Head قرار می‌دهیم و چون فقط همین گره را داریم، پس آخرین گره هم شناخته می‌شود که در tail نیز قرار می‌گیرد. حال اگر فرض کنیم المان بعدی را به آن بدهیم، اینبار دیگر Head برابر Null نخواهد بود. پس دومین سازنده‌ی ListNode صدا زده می‌شود که به غیر از المان جدید، باید آخرین گره قبلی هم با آن ارسال شود و گره جدیدی که ایجاد می‌شود در خاصیت NextNode آن نیز قرار بگیرد و در نهایت گره ایجاد شده به عنوان آخرین گره لیست در متغیر Tail نیز قرار می‌گیرد. در خط پایانی هم به هر مدلی که المان جدید به لیست اضافه شده باشد متغیر Count به روز می‌شود.

```
public T RemoveAt(int index)
{
    if (index >= count || index < 0)
    {
        throw new ArgumentOutOfRangeException(
            "Invalid index: " + index);
    }

    int currentIndex = 0;
    ListNode currentNode = this.head;
    ListNode prevNode = null;
    while (currentIndex < index)
    {
        prevNode = currentNode;
        currentNode = currentNode.NextNode;
        currentIndex++;
    }


    RemoveListNode(currentNode, prevNode);

    return currentNode.Element;
}
```

```
private void RemoveListNode(ListNode node, ListNode prevNode)
{
    count--;
    if (count == 0)
    {
        this.head = null;
        this.tail = null;
    }
    else if (prevNode == null)
    {
        this.head = node.NextNode;
    }
    else
    {
        prevNode.NextNode = node.NextNode;
    }

    if (object.ReferenceEquals(this.tail, node))
    {
        this.tail = prevNode;
    }
}
```

برای حذف یک گره شماره اندیس آن گره را دریافت می‌کنیم و از Head، گره را بیرون کشیده و با خاصیت nextNode آنقدر به سمت جلو حرکت می‌کنیم تا متغیر currentIndex یا اندیس داده شده برابر شود و سپس گره دریافتی و گره قبلی آن را به سمت تابع RemoveListNode ارسال می‌کنیم. کاری که این تابع انجام می‌دهد این است که مقدار NextNode گره فعلی که قصد حذفش را داریم به خاصیت Next Node گره قبلی انتساب می‌دهد. پس به این ترتیب پیوند این گره از لیست از دست می‌رود و گره قبلی به جای اشاره به این گره، به گره بعد از آن اشاره می‌کند. مابقی کد از قبیل جست و برگردان اندیس یک عنصر و ... را به خودتان وگذار می‌کنم.

در روش‌های بالا ما خودمان 2 عدد ADT را پیاده سازی کردیم و متوجه شدیم برای دخیره داده‌ها در حافظه روش‌های متفاوتی وجود دارند که بیشتر تفاوت آن در مورد استفاده از حافظه و کارآیی این روش هاست.

لیست‌های پیوندی دو طرفه Doubly Linked_List

لیست‌های پیوندی بالا یک طرفه بودند و اگر ما یک گره را داشتیم و می‌خواستیم به گره قبلی آن رجوع کنیم، اینکار ممکن نبود و مجبور بودیم برای رسیدن به آن از ابتدای گره حرکت را آغاز کنیم تا به آن برسیم. به همین منظور مبحث لیست‌های پیوندی دو طرفه آغاز شد. به این ترتیب هر گره به جز حفظ ارتباط با گره بعدی از طریق خاصیت NextNode، ارتباطش را با گره قبلی از طریق خاصیت PrevNode نیز حفظ می‌کند.

این مبحث را در اینجا می‌بندیم و در قسمت بعدی آن را ادامه می‌دهیم.

عنوان: ساختار داده‌های خطی Linear Data Structure قسمت دوم
نویسنده: علی یگانه مقدم
تاریخ: ۱۳۹۳/۱۲/۰۱ ۲:۱۰
آدرس: www.dotnettips.info
گروه‌ها: C#, Generics, Data Structures, Linked List

در قسمت قبلی به مقدمات و ساخت لیست‌های ایستا و پویا به صورت دستی پرداختیم و در این قسمت (مبحث پایانی) لیست‌های آماده در دات نت را مورد بررسی قرار می‌دهیم.

کلاس ArrayList

این کلاس همان پیاده سازی لیست‌های ایستایی را دارد که در مطلب پیشین در مورد آن صحبت کردیم و نحوه کدنویسی آن نیز بیان شد و امکاناتی بیشتر از آنچه که در جدول مطلب پیشین گفته بودیم در دسترس ما قرار می‌دهد. از این کلاس با اسم untyped dynamically-extendable array به معنی *آرایه پویا قابل توسعه بدون نوع* هم اسم می‌برند چرا که به هیچ نوع داده‌ای مقید نیست و می‌توانید یکبار به آن رشته بدهید، یکبار عدد صحیح، یکبار اعشاری و یکبار زمان و تاریخ، کد زیر به خوبی نشان دهنده‌ی این موضوع است و نحوه استفاده‌ی از این آرایه‌ها را نشان می‌دهد.

```
using System;
using System.Collections;

class ProgrArrayListExample
{
    static void Main()
    {
        ArrayList list = new ArrayList();
        list.Add("Hello");
        list.Add(5);
        list.Add(3.14159);
        list.Add(DateTime.Now);

        for (int i = 0; i < list.Count; i++)
        {
            object value = list[i];
            Console.WriteLine("Index={0}; Value={1}", i, value);
        }
    }
}
```

نتیجه کد بالا:

```
Index=0; Value=Hello
Index=1; Value=5
Index=2; Value=3.14159
Index=3; Value=29.02.2015 23:17:01
```

البته برای خواندن و قرار دادن متغیرها از آنجا که فقط نوع Object را برمی‌گرداند، باید یک تبدیل هم انجام داد یا اینکه از کلمه‌ی کلیدی dynamic استفاده کنید:

```
ArrayList list = new ArrayList();
list.Add(2);
list.Add(3.5f);
list.Add(25u);
list.Add(" ریال");
dynamic sum = 0;
for (int i = 0; i < list.Count; i++)
{
    dynamic value = list[i];
    sum = sum + value;
}
Console.WriteLine("Sum = " + sum);
// Output: Sum = 30.5ریال
```

مجموعه‌های جنریک Generic Collections

مشکل ما در حین کار با کلاس arrayList و همه کلاس‌های مشتق شده از system.collection.IList این است که نوع داده‌ی ما

تبدیل به Object می‌شود و موقعی‌که آن را به ما بر می‌گرداند باید آن را به صورت دستی تبدیل کرده یا از کلمه‌ی کلیدی dynamic استفاده کنیم. در نتیجه در یک شرایط خاص، هیچ تضمینی برای ما وجود نخواهد داشت که بتوانیم کنترلی بر روی نوع داده‌های خود داشته باشیم و به علاوه عمل تبدیل یا casting هم یک عمل زمان بر هست.

برای حل این مشکل، از جنریک‌ها استفاده می‌کنیم. جنریک‌ها می‌توانند با هر نوع داده‌ای کار کنند. در حین تعریف یک کلاس جنریک نوع آن را مشخص می‌کنیم و مقادیری که از آن به بعد خواهد پذیرفت، از نوعی هستند که ابتدا تعریف کرده‌ایم.

یک ساختار جنریک به صورت زیر تعریف می‌شود:

```
GenericType<T> instance = new GenericType<T>();
```

نام کلاس و به جای T نوع داده از قبیل int,bool,string را می‌نویسیم. مثال‌های زیر را ببینید:

```
List<int> intList = new List<int>();
List<bool> boolList = new List<bool>();
List<double> realNumbersList = new List<double>();
```

کلاس جنریک List<T>

این کلاس مشابه همان کلاس ArrayList است و فقط به صورت جنریک پیاده سازی شده است.

```
List<int> intList = new List<int>();
```

تعریف بالا سبب ایجاد ArrayList ـی می‌باشد که تنها مقادیر int را دریافت می‌کند و دیگر نوع Object ـی در کار نیست. یک آرایه از نوع int ایجاد می‌کند و مقدار خانه‌های پیش فرضی را نیز در ابتدا، برای آن در نظر می‌گیرد و با افزودن هر مقدار جدید می‌بیند که آیا خانه‌ی خالی وجود دارد یا خیر. اگر وجود داشته باشد مقدار جدید، به خانه‌ی بعدی آخرین خانه‌ی پر شده انتقال می‌یابد و اگر هم نباشد، مقدار خانه از آن چه هست 2 برابر می‌شود. درست است عملیات resizing یا افزایش طول آرایه عملی زمان بر محسوب میشود ولی همیشه این اتفاق نمی‌افتد و با زیاد شدن مقادیر خانه‌ها این عمل کمتر هم می‌شود. هر چند با زیاد شدن خانه‌ها حافظه مصرفی ممکن است به خاطر زیاد شدن خانه‌های خالی بدتر هم بشود. فرض کنید بار اول خانه‌ها 16 تایی باشند که بعد می‌شوند 32 تایی و بعدا 64 تایی. حالا فرض کنید به خاطر یک عنصر، خانه‌ها یا ظرفیت بشود 128 تایی در حالی که طول آرایه (خانه‌های پر شده) 65 تاست و حال این وضعیت را برای موارد بزرگتر پیش بینی کنید. در این نوع داده اگر منظور زمان باشد نتجه خوبی را در بر دارد ولی اگر مراعات حافظه را هم در نظر بگیرید و داده‌ها زیاد باشند، باید تا حدامکان به روش‌های دیگر هم فکر کنید.

چه موقع از List<T> استفاده کنیم؟

استفاده از این روش مزایا و معایبی دارد که باید در توضیحات بالا متوجه شده باشید ولی به طور خلاصه:

استفاده از index برای دسترسی به یک مقدار، صرف نظر از اینکه چه میزان داده‌ای در آن وجود دارد، بسیار سریع انجام میگیرد.

جست و جوی یک عنصر بر اساس مقدار: جست و جو خطی است در نتیجه اگر مقدار مورد نظر در آخرین خانه‌ها باشد بدترین وضعیت ممکن رخ می‌دهد و بسیار کند عمل می‌کند. داده هر چی کمتر بهتر و هر چه بیشتر بدتر. البته اگر بخواهید مجموعه‌ای از مقدارهای برابر را برگردانید هم در بدترین وضعیت ممکن خواهد بود.

حذف و درج (منظور insert) المان‌ها به خصوص موقعی که انتهای آرایه نباشید، شیفت پیدا کردن در آرایه عملی کاملا کند و زمانبر است.
موقعی که عنصری را بخواهید اضافه کنید اگر ظرفیت آرایه تکمیل شده باشد، نیاز به عمل زمانبر افزایش ظرفیت خواهد بود که البته این عمل به ندرت رخ می‌دهد و عملیات افزودن Add هم هیچ وابستگی به تعداد المان‌ها ندارد و عملی سریع است.

با توجه به موارد خلاصه شده بالا، موقعی از لیست اضافه می‌کنیم که عملیات درج و حذف زیادی نداریم و بیشتر برای افزودن مقدار به انتها و دسترسی به المان‌ها بر اساس اندیس باشد.

LinkedList<T>

یک کلاس از پیش آماده در دات نت که لیست‌های پیوندی دو طرفه را پیاده سازی می‌کند. هر المان یا گره یک متغیر جهت ذخیره مقدار دارد و یک اشاره گر به گره قبل و بعد. **چه موقع باید از این ساختار استفاده کنیم؟**

از مزایا و معایب آن :
افزودن به انتهای لیست به خاطر این که همیشه گره آخر در tail وجود دارد بسیار سریع است.
عملیات درج insert در هر موقعیتی که باشد اگر یک اشاره گر به آن محل باشد یک عملیات سریع است یا اینکه درج در ابتدا یاانتهای لیست باشد.
جست و جوی یک مقدار چه بر اساس اندیس باشد و چه مقدار، کار جست و جو کند خواهد بود. چرا که باید تمامی المان‌ها از اول به آخر اسکن بشن.
عملیات حذف هم به خاطر اینکه یک عمل جست و جو در ابتدای خود دارد، یک عمل کند است.

استفاده از این کلاس موقعی خوب است که عملیات‌های درج و حذف ما در یکی از دو طرف لیست باشد یا اشاره‌گری به گره مورد نظر وجود داشته باشد. از لحاظ مصرف حافظه به خاطر داشتن فیلدهای اشاره‌گر به جز مقدار، زیادتر از نوع List می‌باشد. در صورتی که دسترسی سریع به داده‌ها برایتان مهم باشد استفاده از List باز هم به صرفه‌تر است.

پشته Stack
یک سری مکعب را تصور کنید که روی هم قرار گرفته اند و برای اینکه به یکی از مکعب‌های پایینی بخواهید دسترسی داشته باشید باید تعدادی از مکعب‌ها را از بالا بردارید تا به آن برسید. یعنی بر خلاف موقعی که آن‌ها روی هم می‌گذاشتید و آخرین مکعب روی همه قرار گرفته است. حالا همان مکعب‌ها به صورت مخالف و معکوس باید برداشته شوند.
یک مثال واقعی‌تر و ملموس‌تر، یک کمد لباس را تصور کنید که مجبورید برای آن که به لباس خاصی برسید، باید آخرین لباس‌هایی را که در داخل کمد قرار داده‌اید را اول از همه از کمد در بیاورید تا به آن لباس برسید.
در واقع پشته چنین ساختاری را پیاده می‌کند که اولین عنصری که از پشته بیرون می‌آید، آخرین عنصری است که از آن درج شده است و به آن LIFO گویند که مخفف عبارت Last Input First Output آخرین ورودی اولین خروجی است. این ساختار از قدیمی‌ترین ساختارهای موجود است. حتی این ساختار در سیستم‌های داخل دات نت CLR هم به عنوان نگهدارنده متغیرها و پارامتر متدها استفاده می‌شود که به آن Program Execution Stack می‌گویند.
پشته سه عملیات اصلی را پیاده سازی می‌کند: **Push** جهت قرار دادن مقدار جدید در پشته، **POP** جهت بیرون کشیدن مقداری که آخرین بار در پشته اضافه شده و **Peek** جهت برگرداندن آخرین مقدار اضافه شده به پشته ولی آن مقدار از پشته حذف نمی‌شود.
این ساختار میتواند پیاده سازی‌های متفاوتی را داشته باشد ولی دو نوع اصلی که ما بررسی می‌کنیم، ایستا و پویا بودن آن است.
ایستا بر اساس آرایه است و پویا بر اساس لیست‌های پیوندی. شکل زیر پشته‌ای را به صورت استفاده از پیاده‌سازی ایستا با آرایه‌ها نشان می‌دهد و کلمه Top به بالای پشته یعنی آخرین عنصر اضافه شده اشاره می‌کند.

استفاده از لیست پیوندی برای پیاده سازی پشته:

لیست پیوندی لازم نیست دو طرفه باشد و یک طرف برای کار با پشته مناسب است و دیگر لازم نیست که به انتهای لیست پیوندی عمل درج انجام شود؛ بلکه مقدار جدید به ابتدای آن اضافه شده و برای حذف گره هم اولین گره باید حذف شود و گره دوم به عنوان head شناخته می‌شود. همچنین لیست پیوندی نیازی به افزایش ظرفیت مانند آرایه‌ها ندارد.

ساختار پشته در دات نت توسط کلاس Stack از قبل آماده است:

```
Stack<string> stack = new Stack<string>();
stack.Push("A");
stack.Push("B");
stack.Push("C");
 while (stack.Count > 0)
    {
        string letter= stack.Pop();
        Console.WriteLine(letter);
    }
//خروجی
//C
//B
//A
```

صف Queue

ساختار صف هم از قدیمی‌ترین ساختارهاست و مثال آن در همه جا و در همه اطراف ما دیده می‌شود؛ مثل صف نانوایی، صف چاپ پرینتر، دسترسی به منابع مشترک توسط سیستمها. در این ساختار ما عنصر جدید را به انتهای صف اضافه می‌کنیم و برای دریافت مقدار، عنصر را از ابتدا حذف می‌کنیم. به این ساختار FIFO مخفف First Input First Output به معنی اولین ورودی و اولین خروجی هم می‌گویند.

ساختار ایستا که توسط آرایه‌ها پیاده سازی شده است:

ابتدای آرایه مکانی است که عنصر از آنجا برداشته می‌شود و Head به آن اشاره می‌کند و tail هم به انتهای آرایه که جهت درج عنصر جدید مفید است. با برداشتن هر خانه‌ای که head به آن اشاره می‌کند، head یک خانه به سمت جلو حرکت می‌کند و زمانی که Head از tail بیشتر شود، یعنی اینکه دیگر عنصری یا المانی در صف وجود ندارد و head و Tail به ابتدای صف حرکت می‌کنند. در این حالت موقعی که المان جدیدی قصد اضافه شدن داشته باشد، افزودن، مجددا از اول صف آغاز می‌شود و به این صف‌ها، صف حلقوی می‌گویند.

عملیات اصلی صف دو مورد هستند enqueue که المان جدید را در انتهای صف قرار می‌دهد و dequeue اولین المان صف را بیرون می‌کشد.

پیاده سازی صف به صورت پویا با لیست‌های پیوندی

برای پیاده سازی صف، لیست‌های پیوندی یک طرفه کافی هستند:

در این حالت عنصر جدید مثل سابق به انتهای لیست اضافه می‌شود و برای حذف هم که از اول لیست کمک می‌گیریم و با حذف عنصر اول، متغیر Head به عنصر یا المان دوم اشاره خواهد کرد.

کلاس از پیش آمده صف در دات نت Queue<T> است و نحوه‌ی استفاده آن بدین شکل است:

```
static void Main()
{
    Queue<string> queue = new Queue<string>();
    queue.Enqueue("Message One");
    queue.Enqueue("Message Two");
    queue.Enqueue("Message Three");
    queue.Enqueue("Message Four");

    while (queue.Count > 0)
    {
        string msg = queue.Dequeue();
        Console.WriteLine(msg);
    }
```

```
}
//خروجی
//Message One
//Message Two
//Message Thre
//Message Four
```

عنوان: **درخت‌ها و گراف‌ها قسمت اول**
نویسنده: علی یگانه مقدم
تاریخ: ۱۳۹۳/۱۲/۰۲ ۱:۳۵
آدرس: www.dotnettips.info
گروه‌ها: C#, Data Structures, Tree

در این مقاله یکی از ساختارهای داده را به نام ساختارهای درختی و گراف‌ها معرفی کردیم و در این مقاله قصد داریم این نوع ساختار را بیشتر بررسی نماییم. این ساختارها برای بسیاری از برنامه‌های مدرن و امروزی بسیار مهم هستند. هر کدام از این ساختارهای داده به حل یکی از مشکلات دنیای واقعی می‌پردازند. در این مقاله قصد داریم به مزایا و معایب هر کدام از این ساختارها اشاره کنیم و اینکه کی و کجا بهتر است از کدام ساختار استفاده گردد. تمرکز ما بر **درخت هایی دودویی** ، **درخت‌های جست و جوی دو دویی** و **درخت‌های جست و جوی دو دویی متوازن** خواهد بود. همچنین ما به تشریح گراف و انواع آن خواهیم پرداخت. اینکه چگونه آن را در حافظه نمایش دهیم و اینکه گراف‌ها در کجای زندگی واقعی ما یا فناوری‌های کامپیوتری استفاده می‌شوند.

ساختار درختی

در بسیاری از مواقع ما با گروهی از اشیاء یا داده‌هایی سر و کار داریم که هر کدام از آن‌ها به گروهی دیگر مرتبط هستند. در این حالت از ساختار خطی نمی‌توانیم برای توصیف این ارتباط استفاده کنیم. پس بهترین ساختار برای نشان دادن این ارتباط **ساختار شاخه ای Branched Structure** است.

یک ساختار درختی یا یک ساختار شاخه‌ای شامل المان‌هایی به اسم گره Node است. هر گره می‌تواند به یک یا چند گره دیگر متصل باشد و گاهی اوقات این اتصالات مشابه یک *سلسه مراتب hierarchically* می‌شوند.

درخت‌ها در برنامه نویسی جایگاه ویژه‌ای دارند به طوری که استفاده‌ی از آن‌ها در بسیاری از برنامه‌ها وجود دارد و بسیاری از مثال‌های واقعی پیرامون ما را پشتیبانی می‌کنند.

در نمودار زیر مثالی وجود دارد که در آن یک تیم نرم افزاری نمایش داده شده‌است. در اینجا هر یک از بخش‌ها وظایف و مسئولیت‌هایی را بر دوش خود دارند که این مسئولیت‌ها به صورت سلسله مراتبی در تصویر زیر نمایش داده شده‌اند.

ما در ساختار بالا متوجه می‌شویم که چه بخشی زیر مجموعه‌ی چه بخشی است و سمت بالاتر هر بخش چیست. برای مثال ما متوجه شدیم که مدیر توسعه دهندگان، "سرپرست تیم" است که خود نیز مادون "مدیر پروژه" است و این را نیز متوجه می‌شویم که مثلا توسعه دهنده‌ی شماره یک هیچ مادونی ندارد و مدیر پروژه در راس همه است و هیچ مدیر دیگری بالای سر او قرار ندارد.

اصطلاحات درخت

برای اینکه بیشتر متوجه روابط بین اشیا در این ساختار بشویم، به شکل زیر خوب دقت کنید:

در شکل بالا دایره‌هایی برای هر بخش از اطلاعات کشیده شده و ارتباط هر کدام از آن‌ها از طریق یک خط برقرار شده است. اعداد داخل هر دایره تکراری نیست و همه منحصر به فرد هستند. پس وقتی از اعداد اسم ببریم متوجه می‌شویم که در مورد چه چیزی صحبت می‌کنیم.

در شکل بالا به هر یک از دایره‌ها یک **گره Node** می‌گویند و به هر خط ارتباط دهنده بین گره‌ها **لبه Edge** گفته می‌شود. گره‌های 19 و 21 و 14 زیر گره‌های گره 7 محسوب می‌شوند. گره‌هایی که به صورت مستقیم به زیر گره‌های خودشان اشاره می‌کنند را **گره‌های والد Parent** می‌گویند و زیرگره‌های 7 را گره‌های **فرزند ChildNodes** . پس با این حساب می‌توانیم بگوییم گره‌های 1 و 12 و 31 را هم فرزند گره 19 هستند و گره 19 والد آن هاست. همچنین گره‌های یک والد را مثل 19 و 21 و 14 که والد مشترک دارند، گره‌های **خواهر و برادر** یا **حتی همنژاد Sibling** می‌گوییم. همچنین ارتباط بین گره 7 و گره‌های سطح دوم و الی آخر یعنی 1 و 12 و 31 و 23 و 6 را که والد بودن آن به صورت غیر مستقیم است را **جد** یا **ancestor** می‌نامیم و نوه‌ها و نتیجه‌های آن‌ها را **نسل descendants** .

ریشه Root : به گره‌ای می‌گوییم که هیچ والدی ندارد و خودش در واقع اولین والد محسوب می‌شود؛ مثل گره 7.

برگ Leaf : به گره‌هایی که هیچ فرزندی ندارند، برگ می‌گوییم. مثال گره‌های 1 و12 و 31 و 23 و 6

گره‌های داخلی Internal Nodes : گره هایی که نه برگ هستند و نه ریشه. یعنی حداقل یک فرزند دارند و خودشان یک گره فرزند محسوب می‌شوند؛ مثل گره‌های 19 و 14.

مسیر Path : راه رسیدن از یک گره به گره دیگر را مسیر می‌گویند. مثلا گره‌های 1 و 19 و 7 و 21 به ترتیب یک مسیر را تشکیل می‌دهند ولی گره‌های 1 و 19 و 23 از آن جا که هیچ جور اتصالی بین آن‌ها نیست، مسیری را تشکیل نمی‌دهند.

طول مسیر Length of Path : به تعداد لبه‌های یک مسیر، طول مسیر می‌گویند که می‌توان از تعداد گره‌ها -1 نیز آن را به دست آورد. برای نمونه : مسیر 1 و19 و 7 و 21 طول مسیرشان 3 هست.

عمق Depth : طول مسیر یک گره از ریشه تا آن گره را عمق درخت می‌گویند. عمق یک ریشه همیشه صفر است و برای مثال در درخت بالا، گره 19 در عمق یک است و برای گره 23 عمق آن 2 خواهد بود.

تعریف خود درخت Tree : درخت یک ساختار داده برگشتی recursive است که شامل گره‌ها و لبه‌ها، برای اتصال گره‌ها به

یکدیگر است.

جملات زیر در مورد درخت صدق می‌کند:

هر گره می‌تواند فرزند نداشته باشد یا به هر تعداد که می‌خواهد فرزند داشته باشد.
هر گره یک والد دارد و تنها گره‌ای که والد ندارد، گره ریشه است (البته اگر درخت خالی باشد هیچ گره ای وجود ندارد).
همه گره‌ها از ریشه قابل دسترسی هستند و برای دسترسی به گره مورد نظر باید از ریشه تا آن گره، مسیری را طی کرد.
ار تفاع درخت Height: به حداکثر عمق یک درخت، ارتفاع درخت می‌گویند. **درجه گره Degree** : به تعداد گره‌های فرزند یک گره، درجه آن گره می‌گویند. در درخت بالا درجه گره‌های 7 و 19 سه است. درجه گره 14 دو است و درجه برگ‌ها صفر است. **ضریب انشعاب Branching Factor:** به حداکثر درجه یک گره در یک درخت، ضریب انشعاب آن درخت گویند.

پیاده سازی درخت

برای پیاده سازی یک درخت، از دو کلاس یکی جهت ساخت گره که حاوی اطلاعات است TreeNode<T> و دیگری جهت ایجاد درخت اصلی به همراه کلیه متدها و خاصیت هایش Tree<T> کمک می‌گیریم.

```
public class TreeNode<T>
{
    // شامل مقدار گره است
    private T value;

    // مشخص می‌کند که آیا گره والد دارد یا خیر
    private bool hasParent;

    // در صورت داشتن فرزند ، لیست فرزندان را شامل می‌شود
    private List<TreeNode<T>> children;

    /// <summary>سازنده کلاس </summary>
    /// <param name="value">مقدار گره</param>
    public TreeNode(T value)
    {
        if (value == null)
        {
            throw new ArgumentNullException(
                "Cannot insert null value!");
        }
        this.value = value;
        this.children = new List<TreeNode<T>>();
    }

    /// <summary>خاصیتی جهت مقداردهی گره</summary>
    public T Value
    {
        get
        {
            return this.value;
        }
        set
        {
            this.value = value;
        }
    }

    /// <summary>تعداد گره‌های فرزند را بر میگرداند</summary>
    public int ChildrenCount
    {
        get
        {
            return this.children.Count;
        }
    }

    /// <summary>به گره یک فرزند اضافه می‌کند</summary>
    /// <param name="child">آرگومان این متد یک گره است که قرار است به فرزندی گره فعلی در آید</param>
    public void AddChild(TreeNode<T> child)
    {
        if (child == null)
        {
            throw new ArgumentNullException(
                "Cannot insert null value!");
        }

        if (child.hasParent)
```

```
        {
            throw new ArgumentException(
                "The node already has a parent!");
        }

        child.hasParent = true;
        this.children.Add(child);
    }

    /// <summary>
    /// گره ای که اندیس آن داده شده است بازگردانده می‌شود
    /// </summary>
    /// <param name="index">اندیس گره</param>
    /// <returns>گره بازگشتی</returns>
    public TreeNode<T> GetChild(int index)
    {
        return this.children[index];
    }
}

/// <summary>این کلاس ساختار درخت را به کمک کلاس گره‌ها که در بالا تعریف کردیم میسازد</summary>
/// <typeparam name="T">نوع مقادیری که قرار است داخل درخت ذخیره شوند</typeparam>
public class Tree<T>
{
    // گره ریشه
    private TreeNode<T> root;

    /// <summary>سازنده کلاس</summary>
    /// <param name="value">مقدار گره اول که همان ریشه می‌شود</param>
    public Tree(T value)
    {
        if (value == null)
        {
            throw new ArgumentNullException(
                "Cannot insert null value!");
        }

        this.root = new TreeNode<T>(value);
    }

    /// <summary>سازنده دیگر برای کلاس درخت</summary>
    /// <param name="value">مقدار گره ریشه مثل سازنده اول</param>
    /// <param name="children">آرایه ای از گره‌ها که فرزند گره ریشه می‌شوند</param>
    public Tree(T value, params Tree<T>[] children)
        : this(value)
    {
        foreach (Tree<T> child in children)
        {
            this.root.AddChild(child.root);
        }
    }

    /// <summary>
    /// ریشه را بر میگرداند ، اگر ریشه ای نباشد نال بر میگرداند
    /// </summary>
    public TreeNode<T> Root
    {
        get
        {
            return this.root;
        }
    }

    /// <summary>پیمودن عرضی و نمایش درخت با الگوریتم دی اف اس </summary>
    /// <param name="root">ریشه (گره ابتدایی) درختی که قرار است پیمایش از آن شروع شود</param>
    /// <param name="spaces">یک کاراکتر جهت جداسازی مقادیر هر گره</param>
    private void PrintDFS(TreeNode<T> root, string spaces)
    {
        if (this.root == null)
        {
            return;
        }

        Console.WriteLine(spaces + root.Value);

        TreeNode<T> child = null;
        for (int i = 0; i < root.ChildrenCount; i++)
        {
            child = root.GetChild(i);
            PrintDFS(child, spaces + "   ");
        }
```

```
    }

    /// <summary>متد پیمایش درخت به صورت عمومی که تابع خصوصی که در بالا توضیح دادیم را صدا
می‌زند</summary>
    public void TraverseDFS()
    {
        this.PrintDFS(this.root, string.Empty);
    }
}

/// <summary>
/// کد استفاده از ساختار درخت
/// </summary>
public static class TreeExample
{
    static void Main()
    {
        // Create the tree from the sample
        Tree<int> tree =
            new Tree<int>(7,
                new Tree<int>(19,
                    new Tree<int>(1),
                    new Tree<int>(12),
                    new Tree<int>(31)),
                new Tree<int>(21),
                new Tree<int>(14,
                    new Tree<int>(23),
                    new Tree<int>(6))
            );

        // پیمایش درخت با الگوریتم دی اف اس یا عمقی
        tree.TraverseDFS();

        // خروجی
        // 7
        //       19
        //        1
        //        12
        //        31
        //       21
        //       14
        //        23
        //        6
    }
}
```

کلاس TreeNode وظیفه‌ی ساخت گره را بر عهده دارد و با هر شیءایی که از این کلاس می‌سازیم، یک گره ایجاد می‌کنیم که با خاصیت Children و متد AddChild آن می‌توانیم هر تعداد گره را که می‌خواهیم به فرزندی آن گره در آوریم که باز خود آن گره می‌تواند در خاصیت Children یک گره دیگر اضافه شود. به این ترتیب با ساخت هر گره و ایجاد رابطه از طریق خاصیت children هر گره درخت شکل می‌گیرد. سپس گره والد در ساختار کلاس درخت Tree قرار می‌گیرد و این کلاس شامل متدهایی است که می‌تواند روی درخت، عملیات پردازشی چون پیمایش درخت را انجام دهد.

پیمایش درخت به روش عمقی DFS (Depth First Search)

هدف از پیمایش درخت ملاقات یا بازبینی (تهیه لیستی از همه گره‌های یک درخت) تنها یکبار هر گره در درخت است. برای این کار الگوریتم‌های زیادی وجود دارند که ما در این مقاله تنها دو روش DFS و BFS را بررسی می‌کنیم.

روش DFS: هر گره‌ای که به تابع بالا بدهید، آن گره برای پیمایش، گره ریشه حساب خواهد شد و پیمایش از آن آغاز می‌گردد. در الگوریتم DFS روش پیمایش بدین گونه است که ما از گره ریشه آغاز کرده و گره ریشه را ملاقات می‌کنیم. سپس گره‌های فرزندش را به دست می‌آوریم و یکی از گره‌ها را انتخاب کرده و دوباره همین مورد را رویش انجام می‌دهیم تا نهایتا به یک برگ برسیم. وقتی که به برگی می‌رسیم یک مرحله به بالا برگشته و این کار را آنقدر تکرار می‌کنیم تا همه‌ی گره‌های آن ریشه یا درخت پیمایش شده باشند.

همین درخت را در نظر بگیرید:

پیمایش درخت را از گره 7 آغاز می‌کنیم و آن را به عنوان ریشه در نظر می‌گیریم. حتی می‌توانیم پیمایش را از گره مثلا 19 آغاز کنیم و آن را برای پیمایش ریشه در نظر بگیریم ولی ما از همان 7 پیمایش را آغاز می‌کنیم:

ابتدا گره 7 ملاقات شده و آن را می‌نویسیم. سپس فرزندانش را بررسی می‌کنیم که سه فرزند دارد. یکی از فرزندان مثل گره 19 را انتخاب کرده و آن را ملاقات می‌کنیم (با هر بار ملاقات آن را چاپ می‌کنیم) سپس فرزندان آن را بررسی می‌کنیم و یکی از گره‌ها را انتخاب می‌کنیم و ملاقاتش می‌کنیم؛ برای مثال گره 1. از آن جا که گره یک، برگ است و فرزندی ندارد یک مرحله به سمت بالا برمی‌گردیم و برگ‌های 12 و 31 را هم ملاقات می‌کنیم. حالا همه‌ی فرزندان گره 19 را بررسی کردیم، بر می‌گردیم یک مرحله به سمت بالا و گره 21 را ملاقات می‌کنیم و از آنجا که گره 21 برگ است و فرزندی ندارد به بالا باز می‌گردیم و بعد گره 14 و فرزندانش 23 و 6 هم بررسی می‌شوند. پس ترتیب چاپ ما اینگونه می‌شود:

```
7-19-1-12-31-21-14-23-6
```

پیمایش درخت به روش BFS (Breadth First Search)

در این روش (پیمایش سطحی) گره والد ملاقات شده و سپس همه گره‌های فرزندش ملاقات می‌شوند. بعد از آن یک گره انتخاب شده و همین پیمایش مجددا روی آن انجام می‌شود تا آن سطح کاملا پیمایش شده باشد. سپس به همین مرحله برگشته و فرزند بعدی را پیمایش می‌کنیم و الی آخر. نمونه‌ی این پیمایش روی درخت بالا به صورت زیر نمایش داده می‌شود:

```
7-19-21-14-1-12-31-23-6
```

اگر خوب دقت کنید می‌بینید که پیمایش سطحی است و هر سطح به ترتیب ملاقات می‌شود. به این الگوریتم، پیمایش موجی هم می‌گویند. دلیل آن هم این است که مثل سنگی می‌ماند که شما برای ایجاد موج روی دریاچه پرتاب می‌کنید.

برای این پیمایش از صف کمک گرفته می‌شود که مراحل زیر روی صف صورت می‌گیرد:

ریشه وارد صف Q می‌شود.
دو مرحله زیر مرتبا تکرار می‌شوند:

اولین گره صف به نام V را از Q در یافت می‌کنیم و آن را چاپ می‌کنیم.
فرزندان گره V را به صف اضافه می‌کنیم.

این نوع پیمایش، پیاده سازی راحتی دارد و همیشه نزدیک‌ترین گره‌ها به ریشه را می‌خواند و در هر مرحله گره‌هایی که می‌خواند از ریشه دورتر و دورتر می‌شوند.

نظرات خوانندگان

نویسنده: آقا ابراهیم
تاریخ: ۱۳۹۳/۱۲/۰۲ ۲۲:۴۶

سلام. ممنون بابت مطلب تون. کلا چند کاربرد سنگین و روز مره این ساختارهای درختی رو میخواستم.

نویسنده: محسن خان
تاریخ: ۱۳۹۳/۱۲/۰۲ ۲۳:۳۱

همین نظر تو در تویی که الان ذیل مطلب شما ارسال شد یک ساختار درختی هست.

نویسنده: علی یگانه مقدم
تاریخ: ۱۳۹۳/۱۲/۰۳ ۰:۵۶

کاربرد این موارد زیاد هست و در قسمت‌های بعدی مواردی رو هم نام خواهیم برد
نمونه‌های این مثال مثل دیکشنری‌ها و جست و جوها ، نقشه‌های شهر و مسیریابی و بازی ها
سیستم‌های برق کشی و لوله کشی و .. در بعضی کشورها روی سیستم‌ها نظارت میشه و با ایجاد یک نقص فنی روی نقشه به اونها نشون میده
یا حتی سیستم فایل یا سیستم‌های جست و جو گر
همین موتور گوگل یا حتی موتورهای جدید که با روش خاص از گراف برای جست و جوی‌های مرتبط استفاده میکنن و داده‌ها مرتبط به هم متصل میشن رو میشه نمونه از این موارد دونست
در خیلی از موارد هم شما دارین ازشون استفاده می‌کنین ولی شاید به خاطر قابلیت‌های فریم ورک‌های جدید و پیشرفت زبان‌ها چنان محسوس نبودن

عنوان: **آشنایی با الگوی طراحی Builder**
نویسنده: وحید فرهمندیان
تاریخ: ۱۳۹۳/۱۲/۰۲ ۲۰:۵
آدرس: www.dotnettips.info
گروه‌ها: C#, OOP, DesignPatterns

سناریوی زیر را در نظر بگیرید:
از شما خواسته شده است تا نحوه‌ی ساخت تلفن همراه را پیاده سازی نمایید. شما در گام اول 2 نوع تلفن همراه را شناسایی نموده‌اید (Android و Windows Phone). پس از شناسایی، احتمالا هر کدام از این انواع را یک کلاس در نظر می‌گیرید و به کمک یک واسط یا کلاس انتزاعی، شروع به ساخت کلاس می‌نمایید، تا در آینده اگر تلفن همراه جدیدی شناسایی شد، راحت‌تر بتوان آن را در پیاده سازی دخیل نمود.
اگر چنین فکر کرده اید باید گفت که 90% با الگوی طراحی Builder آشنا هستید و از آن نیز استفاده می‌کنید؛ بدون اینکه متوجه باشید از این الگو استفاده کرده‌اید. در کدهای زیر این الگو را قدم به قدم بررسی خواهیم نمود. **قدم 1:** تلفن همراه چه بخش هایی می‌تواند داشته باشد؟ (برای مثال یک OS دارند، یک Name دارند و یک Screen) همچنین برای اینکه تلفن همراهی بتواند ساخته شود ابتدا بایستی نام آن‌را بدانیم. کدهای زیر همین رویه را تصدیق می‌نمایند:

```
public class Product
{
        public Product(string name)
        {
            Name = name;
        }
        public string Name { get; set; }
        public string Screen { get; set; }
        public string OS { get; set; }
        public override string ToString()
        {
            return string.Format(Screen + "/" + OS + "/" + Name);
        }
}
```

یک کلاس ساخته‌ایم و نام آن را Product گذاشتیم. بخش‌های مختلفی را نیز در آن تعریف نموده‌ایم. تابع ToString را برای استفاده‌های بعدی override کرده‌ایم (فعلا نیازی بدان نداریم). **قدم 2:** برای ساخت تلفن همراه چه کارهایی باید انجام شود؟ (برای مثال بایستی OS روی آن نصب شود، Screen آن مشخص شود. همچنین بایستی به طریقی بتوانم تلفن همراه ساخته شده‌ی خود را نیز پیدا کنم). کدهای زیر همین رویه را تصدیق می‌نمایند:

```
public interface IBuilder
    {
        void BuildScreen();
        void BuildOS();
        Product Product { get; }
    }
```

یک واسط تعریف کرده‌ایم تا به کمک آن هر تلفن همراهی را که خواستیم بسازیم. **قدم 3:** از آنجا که فقط دو نوع تلفن همراه را فعلا شناسایی کرده‌ایم (Android و Windows Phone) نیاز داریم تا این دو تا را بسازیم.
ابتدا تلفن همراه Android را می‌سازیم:

```
public class ConcreteBuilder1 : IBuilder
    {
        public Product p;
        public ConcreteBuilder1()
        {
            p = new Product("Android Cell Phone");
        }
        public void BuildScreen()
        {
            p.Screen = "Touch Screen 16 Inch!";
        }

        public void BuildOS()
        {
            p.OS = "Android 4.4";
        }
```

```
        public Product Product
        {
            get { return p; }
        }
    }
```

سپس تلفن همراه Windows Phone را می‌سازیم:

```
public class ConcreteBuilder2 : IBuilder
    {
        public Product p;

        public ConcreteBuilder2()
        {
            p = new Product("Windows Phone");
        }
        public void BuildScreen()
        {
            p.Screen = "Touch Screen 32 Inch!";
        }

        public void BuildOS()
        {
            p.OS = "Windows Phone 2014";
        }
        public Product Product
        {
            get { return p; }
        }
    }
```

قدم 4: اول باید OS نصب شود یا Screen مشخص شود؟ برای اینکه توالی کار را مشخص سازم نیاز به یک کلاس دیگر دارم تا اینکار را انجام دهد:

```
public class Director
    {
        public void Construct(IBuilder builder)
        {
            builder.BuildScreen();
            builder.BuildOS();
        }
    }
```

این کلاس در متد Construct خود یک ورودی از نوع IBuilder می‌گیرد و براساس توالی مورد نظر، شروع به ساخت آن می‌کند.

قدم 5: نهایتا میخواهم به برنامه‌ی خود بگویم که تلفن همراه Android را بسازد:

```
Director d = new Director();
ConcreteBuilder1 cb1 = new ConcreteBuilder1();
d.Construct(cb1);
Console.WriteLine(cb1.p.ToString());
```

و به این صورت تلفن همراه من آماده است!

متد ToString در اینجا، همان ToString ابتدای بحث است که آن را Override کردیم.

به این نکته توجه کنید که اگر یک تلفن همراه جدید شناسایی شود، چه مقدار تغییری در کدها نیاز دارید؟ برای مثال تلفن همراه BlackBerry شناسایی شده‌است. تنها کاری که لازم است این است که یک کلاس بصورت زیر ساخته شود:

```
public class BlackBerry: IBuilder
    {
        public Product p;

        public BlackBerry ()
        {
            p = new Product("BlackBerry");
        }
        public void BuildScreen()
        {
            p.Screen = "Touch Screen 8 Inch!";
        }
```

```
        public void BuildOS()
        {
            p.OS = "BlackBerry XXX";
        }
        public Product Product
        {
            get { return p; }
        }
    }
```

عنوان: **درخت‌ها و گراف‌ها قسمت دوم**
نویسنده: علی یگانه مقدم
تاریخ: ۱۳۹۳/۱۲/۰۳ ۱:۲۵
آدرس: www.dotnettips.info
گروه‌ها: C#, Data Structures, Tree

در قسمت قبلی ما به بررسی درخت و اصطلاحات فنی آن پرداختیم و اینکه چگونه یک درخت را پیمایش کنیم. در این قسمت مطلب قبل را با درخت‌های دودویی ادامه می‌دهیم.

درخت‌های دودویی Binary Trees

همه‌ی موضوعات و اصطلاحاتی را که در مورد درخت‌ها به کار بردیم، در مورد این درخت هم صدق می‌کند؛ تفاوت درخت دودویی با یک درخت معمولی این است که درجه هر گره نهایتا دو خواهد بود یا به عبارتی ضریب انشعاب این درخت 2 است. از آن جایی که هر گره در نهایت دو فرزند دارد، می‌توانیم فرزندانش را به صورت فرزند چپ Left Child و فرزند راست Right Child صدا بزنیم. به گره‌هایی که فرزند ریشه هستند اینگونه می‌گوییم که گره فرزند چپ با همه فرزندانش می‌شوند زیر درخت چپ Left SubTree و گره سمت راست ریشه با تمام فرزندانش زیر درخت راست Right SubTree صدا زده می‌شوند.

نحوه پیمایش درخت دودویی

این درخت پیمایش‌های گوناگونی دارد ولی سه تای آن‌ها اصلی‌تر و مهمتر هستند:

In-order یا LVR (چپ، ریشه، راست) : در این حالت ابتدا گره‌های سمت چپ ملاقات (چاپ) می‌شوند و سپس ریشه و بعد گره‌های سمت راست.

Pre-Order یا VLR (ریشه، چپ، راست) : در این حالت ابتدا گره‌های ریشه ملاقات می‌شوند. بعد گره‌های سمت چپ و بعد گره‌های سمت راست.

Post_Order یا LRV (چپ، راست، ریشه) : در این حالت ابتدا گره‌های سمت چپ، بعد راست و نهایتا ریشه، ملاقات می‌شوند.

حتما متوجه شده‌اید که منظور از v در اینجا ریشه است و با تغییر و جابجایی مکان این سه حرف RLV میتوانید به ترکیب‌های مختلفی از پیمایش دست پیدا کنید.

اجازه دهید روی شکل بالا پیمایش LVR را انجام دهیم: همانطور که گفتیم باید اول گره‌های سمت چپ را خواند، پس از 17 به سمت 9 حرکت می‌کنیم و می‌بینیم که 9، خود والد است. پس به سمت 6 حرکت می‌کنیم و می‌بینیم که فرزند چپی ندارد؛ پس خود 6 را ملاقات می‌کنیم، سپس فرزند راست را هم بررسی می‌کنیم که فرزند راستی ندارد پس کار ما اینجا تمام است و به سمت بالا حرکت می‌کنیم. 9 را ملاقات می‌کنیم و بعد عدد 5 را و به 17 بر می‌گردیم. 17 را ملاقات کرده و سپس به سمت 15 می‌رویم و الی آخر ...

```
6-9-5-17-8-15-10
```

:VLR

```
17-9-6-5-15-8-10
```

:LRV

```
6-5-9-8-10-15-17
```

نحوه پیاده سازی درخت دودویی:

```
public class BinaryTree<T>
{
    /// <summary>مقدار داخل گره</summary>
    public T Value { get; set; }

    /// <summary>فرزند چپ گره</summary>
    public BinaryTree<T> LeftChild { get; private set; }

    /// <summary>فرزند راست گره</summary>
    public BinaryTree<T> RightChild { get; private set; }

    /// <summary>سازنده کلاس</summary>
    /// <param name="value">مقدار گره</param>
    /// <param name="leftChild">فرزند چپ</param>
    /// <param name="rightChild">فرزند راست
    /// </param>
    public BinaryTree(T value,
        BinaryTree<T> leftChild, BinaryTree<T> rightChild)
    {
        this.Value = value;
        this.LeftChild = leftChild;
        this.RightChild = rightChild;
    }

    /// <summary>سازنده بدون فرزند
    /// </summary>
    /// <param name="value">the value of the tree node</param>
    public BinaryTree(T value) : this(value, null, null)
    {
    }

    /// <summary>LVR پیمایش</summary>
    public void PrintInOrder()
    {
        // ملاقات فرزندان زیر درخت چپ
        if (this.LeftChild != null)
        {
            this.LeftChild.PrintInOrder();
        }

        // ملاقات خود ریشه
        Console.Write(this.Value + " ");

        // ملاقات فرزندان زیر درخت راست
        if (this.RightChild != null)
        {
            this.RightChild.PrintInOrder();
        }
    }
```

```
}

/// <summary>
/// نحوه استفاده از کلاس بالا
/// </summary>
public class BinaryTreeExample
{
    static void Main()
    {
        BinaryTree<int> binaryTree =
            new BinaryTree<int>(14,
                    new BinaryTree<int>(19,
                          new BinaryTree<int>(23),
                          new BinaryTree<int>(6,
                                  new BinaryTree<int>(10),
                                  new BinaryTree<int>(21))),
                    new BinaryTree<int>(15,
                          new BinaryTree<int>(3),
                          null));

        binaryTree.PrintInOrder();
        Console.WriteLine();

        // خروجی
        // 23 19 10 6 21 14 3 15
    }
}
```

تفاوتی که این کد با کد قبلی که برای یک درخت معمولی داشتیم، در این است که قبلا لیستی از فرزندان را داشتیم که با خاصیت Children شناخته می‌شدند، ولی در اینجا در نهایت دو فرزند چپ و راست برای هر گره وجود دارند. برای جست و جو هم از الگوریتم In_Order استفاده کردیم که از همان الگوریتم DFS آمده‌است. در آنجا هم ابتدا گره‌های سمت چپ به صورت بازگشتی صدا زده می‌شدند. بعد خود گره و سپس گره‌های سمت راست به صورت بازگشتی صدا زده می‌شدند.

برای باقی روش‌های پیمایش تنها نیاز است که این سه خط را جابجا کنید:

```
// ملاقات فرزندان زیر درخت چپ
        if (this.LeftChild != null)
        {
            this.LeftChild.PrintInOrder();
        }

        // ملاقات خود ریشه
        Console.Write(this.Value + " ");

        // ملاقات فرزندان زیر درخت راست
        if (this.RightChild != null)
        {
            this.RightChild.PrintInOrder();
        }
```

درخت دودویی مرتب شده Ordered Binary Search Tree

تا این لحظه ما با ساخت درخت‌های پایه آشنا شدیم: درخت عادی یا کلاسیک و درخت دو دویی. ولی در بیشتر موارد در پروژه‌های بزرگ از این‌ها استفاده نمی‌کنیم چرا که استفاده از آن‌ها در پروژه‌های بزرگ بسیار مشکل است و باید به جای آن‌ها از ساختارهای متنوع دیگری از قبیل درخت‌های مرتب شده، کم عمق و متوازن و کامل و پر و .. استفاده کرد. پس اجازه دهید که مهمترین درخت‌هایی را که در برنامه نویسی استفاده می‌شوند، بررسی کنیم.

همان طور که می‌دانید برای مقایسه اعداد ما از علامتهای <>= استفاده می‌کنیم و اعداد صحیح بهترین اعداد برای مقایسه هستند. در درخت‌های جست و جوی دو دویی یک خصوصیت اضافه به اسم **کلید هویت یکتا Unique identification Key** داریم که یک کلید قابل مقایسه است. در تصویر زیر ما دو گره با مقدارهای متفاوتی داریم که با مقایسه‌ی آنان می‌توانیم کوچک و بزرگ بودن یک گره را محاسبه کنیم. ولی به این نکته دقت داشته باشید که این اعداد داخل دایره‌ها، دیگر برای ما حکم مقدار ندارند و کلیدهای یکتا و شاخص هر گره محسوب می‌شوند.

خلاصه‌ی صحبت‌های بالا: در هر درخت دودویی مرتب شده، گره‌های بزرگتر در زیر درخت راست قرار دارند و گره‌های کوچکتر در زیر درخت چپ قرار دارند که این کوچکی و بزرگی بر اساس یک کلید یکتا که قابل مقایسه است استفاده می‌شود.

این درخت دو دویی مرتب شده در جست و جو به ما کمک فراوانی می‌کند و از آنجا که می‌دانیم زیر درخت‌های چپ مقدار کمتری دارند و زیر درخت‌های راست مقدار بیشتر، عمل جست و جو، مقایسه‌های کمتری خواهد داشت، چرا که هر بار مقایسه یک زیر درخت کنار گذاشته می‌شود.

برای مثال فکر کنید می‌خواهید عدد 13 را در درخت بالا پیدا کنید. ابتدا گره والد 19 را مقایسه کرده و از آنجا که 19 بزرگتر از 13 است می‌دانیم که 13 را در زیر درخت راست پیدا نمی‌کنیم. پس زیر درخت چپ را مقایسه می‌کنیم (بنابراین به راحتی یک زیر درخت از مقایسه و عمل جست و جو کنار گذاشته شد). سپس گره 11 را مقایسه می‌کنیم و از آنجا که 11 کوچکتر از 13 هست، زیر درخت سمت راست را ادامه می‌دهیم و چون 16 بزرگتر از 13 هست، زیر درخت سمت چپ را در ادامه مقایسه می‌کنیم که به 13 رسیدیم.

مقایسه گره‌هایی که برای جست و جو انجام دادیم:

```
19-11-16-13
```

درخت هر چه بزرگتر باشد این روش کارآیی خود را بیشتر نشان می‌دهد.

در قسمت بعدی به پیاده سازی کد این درخت به همراه متدهای افزودن و جست و جو و حذف می‌پردازیم.

عنوان: آشنایی با الگوی طراحی Bridge
نویسنده: وحید فرهمندیان
تاریخ: ۱۳۹۳/۱۲/۰۳ ۱۹:۱۵
آدرس: www.dotnettips.info
گروه‌ها: C#, OOP, DesignPatterns

سناریو زیر را در نظر بگیرید:
قصد دارید تا در برنامه‌ی خود ارسال پیام از طریق پیامک و ایمیل را راه اندازی کنید. هر کدام از این روش‌ها نیز برای خود راه‌های متفاوتی دارند. برای مثال ارسال پیامک از طریق وب سرویس یا یک API خارجی و غیره.
کاری را که می‌توان انجام داد، بشرح زیر نیز می‌توان بیان نمود:
ابتدا یک Interface ایجاد می‌کنیم (IBridge) و در آن متد Send را قرار می‌دهیم. این متد یک پارامتر ورودی از نوع رشته می‌گیرد و به کمک آن میتوان اقدام به ارسال پیامک یا ایمیل یا هر چیز دیگری نمود. کلاس‌هایی این واسط را پیاده سازی می‌کنند که یکی از روش‌های اجرای کار باشند (برای مثال کلاس WebService که یک روش ارسال پیامک یا ایمیل است).

```
public interface IBridge
    {
        string Send(string parameter);
    }
    public class WebService: IBridge
    {
        public string Send(string parameter)
        {
            return parameter + " sent by WebService";
        }
    }
    public class API: IBridge
    {
        public string Send(string parameter)
        {
            return parameter + " sent by API";
        }
    }
```

سپس در ادامه به مکانیزمی نیاز داریم تا بتوانیم از طریق آن پیامک یا ایمیل را ارسال کنیم. خوب می‌خواهیم ایمیل ارسال کنیم؛ اولین سوالی که مطرح می‌شود این است که چگونه ارسال کنیم؟ پس باید در مکانیزم خود زیرساختی برای پاسخ به این سوال آماده باشد.

```
public abstract class Abstraction
    {
        public IBridge Bridge;
        public abstract string SendData();
    }
    public class SendEmail : Abstraction
    {
        public override string SendData ()
        {
            return Bridge.Send("Email");
        }
    }
    public class SendSMS: Abstraction
    {
        public override string SendData ()
        {
            return Bridge.Send("SMS");
        }
    }
```

در کد فوق یک کلاس انتزاعی ایجاد کردیم و در آن یک object از نوع واسط خود قرار دادیم. این object به ما کمک می‌کند تا به طریق آن شیوه‌ی ارسال ایمیل یا پیامک را مشخص سازیم و به سوال خود پاسخ دهیم. سپس در ادامه متد SendData آورده شده است که به کمک آن اعلام می‌کنیم که قصد ارسال ایمیل یا پیامک را داریم و نهایتا هر یک از کلاس‌های ایمیل یا پیامک، این متد را برای خود پیاده سازی کرده‌اند.

قبل از ادامه اجازه دهید کمی در مورد بدنه‌ی یکی از متدهای SendData صحبت کنیم. در این متد با کمک Bridge متد Send موجود

در واسط صدا زده شده است. از آنجا که این object از نظر سطح دسترسی عمومی می‌باشد، لذا از بیرون از کلاس قابل دسترسی است. این باعث می‌شود تا قبل از فراخوانی متد SendData موجود در کلاس ایمیل یا پیامک اعلام کنیم که Bridge از چه نوعی است (به چه روشی می‌خواهیم ارسال رخ دهد).

```
Abstraction ab1 = new Email();
  ab1.Bridge = new WebService();
  Console.WriteLine(ab1.SendData ());
```

```
ab1.Bridge = new API();
  Console.WriteLine(ab1.SendData ());
```

```
Abstraction ab2 = new SMS();
   ab2.Bridge = new WebService();
   Console.WriteLine(ab2.SendData ());
```

```
ab2.Bridge = new API();
    Console.WriteLine(ab2.SendData ());
```

نهایتا در کد فوق ابتدا بیان می‌کنیم که قصد ارسال ایمیل را داریم. سپس اعلام می‌داریم که این ارسال را به کمک WebService می‌خواهیم انجام دهی. و نهایتا ارسال را انجام می‌دهیم.

به کل این الگویی که ایجاد کردیم، الگوی Bridge گفته می‌شود.

حال فکر کنید قصد ارسال MMS دارید. در اینصورت فقط کافیست یک کلاس MMS ایجاد کنید و تمام؛ بدون اینکه کدی اضافی را بنویسید یا برنامه را تغییر دهید. یا فرض کنید روش ارسال جدیدی را می‌خواهید اضافه کنید. برای مثال ارسال به روش XYZ. در اینصورت فقط کافیست یک کلاس XYZ را ایجاد کنید که IBridge را پیاده سازی می‌کند.

عنوان: آشنایی با الگوی طراحی Decorator
نویسنده: وحید فرهمندیان
تاریخ: ۱۳۹۳/۱۲/۰۴ ۱۴:۵
آدرس: www.dotnettips.info
گروه‌ها: C#, OOP, DesignPatterns

این بار مثال را با شیرینی و کیک پیش می‌بریم.
فرض کنید شما قصد پخت کیک و نان را دارید. طبیعی است که برای اینکار یک واسط را تعریف کرده و عمل «پختن» را در آن اعلام می‌کنید تا هر کلاسی که قصد پیاده سازی این واسط را داشت، «پختن» را انجام دهد. در ادامه یک کلاس بنام کیک ایجاد خواهید کرد و شروع به پخت آن می‌کنید.
خوب احتمالا الان کیک آماده‌است و می‌توانید آن‌را میل کنید! ولی یک سؤال. تکلیف شخصی که کیک با روکش کاکائو دوست دارد و شمایی که کیک با روکش میوه‌ای دوست دارید چیست؟ این را چطور در پخت اعمال کنیم؟ یا منی که نان کنجدی می‌خواهم و شمایی که نان برشته‌ی غیر کنجدی می‌خواهید چطور؟
احتمالا می‌خواهید سراغ ارث بری رفته و سناریوهای این چنینی را پیاده سازی کنید. ولی در مورد ارث بری، اگر کلاس sealed (NotInheritable) باشه چطور؟
احتمالا همین دو تا سؤال کافی‌است تا در پاسخ بگوئیم، گره‌ی کار، با الگوی Decorator باز می‌شود و همین دو تا سؤال کافی‌است تا اعلام کنیم که این الگو، از جمله الگوهای بسیار مهم و پرکاربرد است.
در ادامه سناریوی خود را با کد ذیل جلو می‌بریم:

```
public interface IBakery
    {
        string Bake();
        double GetPrice();
    }
    public class Cake: IBakery
    {
        public string Bake() { return "Cake baked"; }
        public double GetPrice() { return 2000; }
    }
    public class Bread: IBakery
    {
        public string Bake() { return "Bread baked"; }
        public double GetPrice() { return 100; }
    }
```

در کد فوق فرض کرده‌ام که شما می‌خواهید محصول خودتان را بفروشید و برای آن یک متد GetPrice نیز گذاشته‌ام. خوب در ابتدا واسطی تعریف شده و متدهای Bake و GetPrice اعلام شده‌اند. سپس کلاس‌های Cake و Bread پیاده سازی‌های خودشان را انجام دادند.
در ادامه باید مخلفاتی را که روی کیک و نان می‌توان اضافه کرد، پیاده نمود.

```
public abstract class Decorator : IBakery
    {
        private readonly IBakery _bakery;
        protected string bake = "N/A";
        protected double price = -1;

        protected Decorator(IBakery bakery) { _bakery= bakery; }
        public virtual string Bake() { return _bakery.Bake() + "/" + bake; }
        public double GetPrice() { return _bakery.GetPrice() + price; }
    }
    public class Type1 : Decorator
    {
        public Type1(IBakery bakery) : base(bakery) { bake= "Type 1"; price = 1; }
    }
    public class Type2 : Decorator
    {
        private const string bakeType = "special baked";
        public Type2(IBakery bakery) : base(bakery) { name = "Type 2"; price = 2; }
        public override string Bake() { return base.Bake() + bakeType ; }
    }
```

در کد فوق یک کلاس انتزاعی ایجاد و متدهای پختن و قیمت را پیاده سازی کردیم؛ همچنین کلاس‌های Type1 و Type2 را که من

فرض کردم کلاس‌هایی هستند برای اضافه کردن مخلفات به کیک و نان. در این کلاس‌ها در متد سازنده، یک شیء از نوع IBakery می‌گیریم که در واقع این شیء یا از نوع Cake هست یا از نوع Bread و مشخص می‌کند روی کیک می‌خواهیم مخلفاتی را اضافه کنیم یا بر روی نان. کلاس Type1 روش پخت و قیمت را از کلاس انتزاعی پیروی می‌کند، ولی کلاس Type2 روش پخت خودش را دارد.
با بررسی اجمالی در کدهای فوق مشخص می‌شود که هرگاه بخواهیم، می‌توانیم رفتارها و الحاقات جدیدی را به کلاس‌های Cake و Bread، اضافه کنیم؛ بدون آنکه کلاس اصلی آنها تغییر کند. حال شما شاید در پیاده سازی این الگو از کلاس انتزاعی Decorator هم استفاده نکنید.
با این حال شیوه‌ی استفاده از این کدها هم بصورت زیر خواهد بود:

```
Cake cc1 = new Cake();
 Console.WriteLine(cc1.Bake() + " ," + cc1.GetPrice());

 Type1 cd1 = new Type1 (cc1);
 Console.WriteLine(cd1.Bake() + " ," + cd1.GetPrice());

 Type2 cd2 = new Type2(cc1);
 Console.WriteLine(cd2.Bake() + " ," + cd2.GetPrice());
```

ابتدا یک کیک را پختیم در ادامه Type1 را به آن اضافه کردیم که این باعث می‌شود قیمتش هم زیاد شود و در نهایت Type2 را هم به کیک اضافه کردیم و حالا کیک ما آماده است.

نظرات خوانندگان

نویسنده: محمد اسکندری
تاریخ: ۱۳۹۳/۱۲/۰۵ ۱۰:۵۴

در استفاده از الگوی دکوراتور روش بهتر بهره گیری از آن بصورت سری است و نه ایجاد شیء جدید برای تایپ جدید.
مثلا:

```
Cake c = new Cake();

c = new Type1(c);
c = new SubType(c); //SubType derived from  Cake (e.g. CakeComponent like Cream)

//or: c = new Type1 (new SubType(c));

Console.WriteLine(c.Bake() + ", " + c.GetPrice());
```

نویسنده: محسن خان
تاریخ: ۱۳۹۳/۱۲/۰۵ ۱۱:۴۴

بستگی به هدف نهایی دارد. اگر هدف تولید کیک با روکش کاکائویی و روکش میوه‌ای به صورت همزمان است، نحوه‌ی تزئین آن با کیکی که فقط قرار هست روکش کاکائویی داشته باشه، فرق می‌کنه.

نویسنده: محمد اسکندری
تاریخ: ۱۳۹۳/۱۲/۰۵ ۱۲:۰

فرقی نمیکنه. اگر قرار بود فرق میکرد و نیاز به ایجاد تغییرات در کد بود که این الگوها ارائه نمی‌شدند.

```
// ساخت کیک معمولی با روکش کاکائویی
Cake c = new CommonCake();
c = new Chocolate(c);

// ساخت کیک معمولی با روکش میوه‌ای
Cake c = new CommonCake();
c = new Fruity(c);

// ساخت کیک معمولی مخلوط با روکش کاکائویی و روکش میوه‌ای به صورت همزمان
Cake c = new CommonCake();
c = new Chocolate(c);
c = new Fruity(c);

// ساخت کیک مخصوص مخلوط با روکش کاکائویی و روکش میوه‌ای به صورت همزمان
Cake c = new SpecialCake();
c = new Chocolate(c);
c = new Fruity(c);
```

برای هر c میتوان متدهای اینترفیسش را اجرا کرد.

نویسنده: محسن خان
تاریخ: ۱۳۹۳/۱۲/۰۵ ۱۲:۸

عنوان کردید «در استفاده از الگوی دکوراتور روش بهتر بهره گیری از آن بصورت **سری** است و نه ایجاد شیء **جدید** برای تایپ جدید»، بعد الان برای تهیه روکش فقط میوه‌ای از حالت سری استفاده نکردید و یک وهله جدید ایجاد شده. بحث بر سر سری بودن یا نبودن مراحل بود. بنابراین بسته به هدف، می‌تونه سری باشه یا نباشه و اگر نبود، مشکلی نداره، چون هدفش تولید یک روکش مخصوص بوده و نه ترکیبی.

نویسنده: محمد اسکندری
تاریخ: ۱۳۹۳/۱۲/۰۵ ۱۲:۲۳

فرض من این بود که کاربر نیازی به رفرنس گیری از هر آبجکت ندارد.
مثلا طبق مقاله:

```
// ساخت کیک مخصوص مخلوط با روکش کاکائویی و روکش میوه‌ای به صورت همزمان
Cake c = new SpecialCake();
Chocolate ch = new Chocolate(c);
Fruity f = new Fruity(ch);
```

همانطور که در مقاله گفته شده:

```
Cake cc1 = new Cake();
Type1 cd1 = new Type1 (cc1);
Type2 cd2 = new Type2(cc1);
```

کد فوق را میتوان اینگونه **هم** داشت:

```
// ساخت کیک مخصوص مخلوط با روکش کاکائویی و روکش میوه‌ای به صورت همزمان
Cake c = new SpecialCake();
c = new Chocolate(c);
c = new Fruity(c);
```

بدون اینکه *شیء جدید برای تایپ جدید* بسازیم.

نویسنده: محسن خان
تاریخ: ۱۳۹۳/۱۲/۰۵ ۱۲:۴۳

مهم این نیست که نام تمام متغیرها را c تعریف کردید، مهم این است که به ازای هر new یک شیء کاملا جدید ایجاد می‌شود که ریفرنس آن با ریفرنس قبلی یکی نیست.

عنوان: **آشنایی با ساختار ViewBag**
نویسنده: احمد نواصری
تاریخ: ۱۳۹۳/۱۲/۰۴ ۱۵:۲۵
آدرس: www.dotnettips.info
گروه‌ها: C#, MVC, Dynamic

در روزهای اولی که با MVC آشنا شدم، این سؤال برایم پیش می‌آمد که یک ViewBag چطور می‌تواند به صورت پویا مقادیر را داخل خودش نگهداری کند؟ بعد از جستجو مشخص شد که ViewBag در حقیقت یک شیء Dynamic است. در این نوشتار قصد داریم نحوه‌ی کار یک ViewBag را نمایش دهیم. قبل از هر چیز باید بگویم که ViewBag تنها یک شیء dynamic نیست. اگر آن را از نوع dynamic تعریف و سپس یک شی را به آن Bind کنیم، در هنگام اجرای برنامه استثنای Cannot perform runtime binding صادر می‌شود. در حقیقت باید بگویم که ViewBag علاوه بر dynamic بودن، یک شیء از کلاس ExpandoObject است. با این تعاریف کلاس حاوی ViewBag ما بصورت زیر خواهد بود:

```
public class Controller
    {
        private dynamic _viewBage = new ExpandoObject();
        public  dynamic ViewBag
        {
            get { return _viewBage; }
        }
    }
```

حال برای استفاده از این ViewBag سفارشی کافی است تا کلاسی را تعریف کنیم که از کلاس پایه Controller ما ارث بری کند:

```
public class Sample : Controller
    {
        public  void ShowViewBag()
        {
            ViewBag.Title = 11;
            Console.WriteLine(ViewBag.Title);

            ViewBag.Title = "T";
            Console.WriteLine(ViewBag.Title);

            ViewBag.Title = false;
            Console.WriteLine(ViewBag.Title);

            ViewBag.Title = Math.PI;
            Console.WriteLine(ViewBag.Title);
        }
    }
}
```

در اینجا نحوه‌ی انتساب خواص پویا به ViewBag و مقدار دهی آن‌ها را مشاهده می‌کنید. همچنین در ذیل نحوه‌ی استفاده‌ی از آن‌را در یک برنامه‌ی کنسول بررسی کرده‌ایم:

```
class Program
    {
        static void Main()
        {
            Sample s = new Sample();
            s.ShowViewBag();
            Console.ReadKey();
        }
    }
```

خروجی حاصل از تکه کد بالا به صورت ذیل است:

file:///G:/Project/ViewBag/Vie
11
T
False
3.14159265358979

نظرات خوانندگان

نویسنده: نیکی
تاریخ: ۱۳۹۳/۱۲/۰۵ ۱۰:۳۸

کلمه dynamic چه کاری انجام میده؟ به عنوان نوع هست؟

نویسنده: احمد نواصری
تاریخ: ۱۳۹۳/۱۲/۰۵ ۱۲:۱۸

کلمه dynamic به کامپایلر اعلام میکنه که شی که تعریف کردیم میتونه تغییر کنه یعنی انواع مختلفی به خودش بگیره . شما وقتی یک شی رو با var تعریف میکنید و یک عدد صحیح به اون اختصاص میدید، در دفعه بعد نمیتونید چیزی غیر از عدد صحیح به اون اختصاص بدین. اما با استفاده از dynamic این کار امکان پذیره.

```
var x = 12; //OK
x = "String" //خطای کامپایل

dynamic y = 12; //OK
y="String"; //OK
y=12.7; //OK
```

عنوان: **آشنایی با الگوی طراحی Prototype**
نویسنده: وحید فرهمندیان
تاریخ: ۱۳۹۳/۱۲/۰۴ ۱۹:۲۰
آدرس: www.dotnettips.info
گروه‌ها: C#, OOP, DesignPatterns

فرض کنید در حال پختن یک کیک هستید. ابتدا کیک را می‌پذیرد و سپس آن را تزیین می‌کنید. عملیات پختن کیک، فرآیند ثابتی است و تزیین کردن آن متفاوت. گاهی کیک را با کاکائو تزیین می‌کنید و گاهی با میوه و غیره.
پیش از اینکه سناریو را بیش از این جلو ببریم، وارد بحث کد می‌شویم. طبق سناریوی فوق، فرض کنید کلاسی بنام Prototype دارید که این کلاس هم از کلاس انتزاعی APrototype ارث برده است. در ادامه یک شیء از این کلاس می‌سازید و مقادیر مختلف آن را تنظیم کرده و کار را ادامه می‌دهید.

```
public abstract class APrototype : ICloneable     {
        public string Name { get; set; }
        public string Health { get; set; }
    }

    public class Prototype : APrototype
    {
        public override string ToString() { return string.Format("Player name: {0}, Health statuse:
{1}", Name, Health); }
    }
```

در ادامه از این کلاس نمونه‌گیری می‌کنیم:

```
Prototype p1 = new Prototype { Name = "Vahid", Health = "OK" };
Console.WriteLine(p1.ToString());
```

حالا فرض کنید به یک آبجکت دیگر نیاز دارید، ولی این آبجکت عینا مشابه p1 است؛ لذا نمونه‌گیری، از ابتدا کار مناسبی نیست.
برای اینکار کافیست کدها را بصورت زیر تغییر دهیم:

```
public abstract class APrototype : ICloneable
    {
        public string Name { get; set; }
        public string Health { get; set; }
        public abstract object Clone();
    }

    public class Prototype : APrototype
    {
        public override object Clone() { return this.MemberwiseClone() as APrototype; }
        public override string ToString() { return string.Format("Player name: {0}, Health statuse:
{1}", Name, Health); }
    }
```

در متد Clone از MemberwiseClone استفاده کرده‌ایم. خود Clone هم در داخل واسط ICloneable تعریف شده‌است و هدف از آن کپی نمودن آبجکت‌ها است. سپس کد فوق را بصورت زیر مورد استفاده قرار می‌دهیم:

```
Prototype p1 = new Prototype { Name = "Vahid", Health = "OK" };
Prototype p2 = p1.Clone() as Prototype;
Console.WriteLine(p1.ToString());
Console.WriteLine(p2.ToString());
```

با اجرای کد فوق مشاهده میشود p1 و p2 دقیقا عین هم کار می‌کنند. کل این فرآیند بیانگر الگوی Prototype می‌باشد. ولی تا اینجای کار درست است که الگو پیاده سازی شده است، ولی همچنین به نظر نقصی نیز در کد دیده می‌شود:
برای واضح نمودن نقص، یک کلاس بنام AdditionalDetails تعریف می‌کنیم. در واقع کد را بصورت زیر تغییر میدهیم:

```
public abstract class APrototype : ICloneable
    {
        public string Name { get; set; }
        public string Health { get; set; }
```

```
        public AdditionalDetails Detail { get; set; }
        public abstract object Clone();
    }
    public class AdditionalDetails { public string Height { get; set; } }

    public class Prototype : APrototype
    {
        public override object Clone() { return this.MemberwiseClone() as APrototype; }
        public override string ToString() { return string.Format("Player name: {0}, Health statuse:
{1}, Height: {2}", Name, Health, Detail.Height); }
    }
```

و از آن بصورت زیر استفاده می‌کنیم:

```
Prototype p1 = new Prototype { Name = "Vahid", Health = "OK", Detail = new AdditionalDetails { Height =
"100" } };
Prototype p2 = p1.Clone() as Prototype;
p2.Detail.Height = "200";
Console.WriteLine(p1.ToString());
Console.WriteLine(p2.ToString());
```

خروجی که نمایش داده می‌شود در بخش Height هم برای p1 و هم برای p2 عدد 200 را نمایش می‌دهد که می‌تواند اشتباه باشد. چراکه p1 دارای Height برابر با 100 است و p2 دارای Height برابر با 200. به این اتفاق ShallowCopy گفته میشود که ناشی از استفاده از MemberwiseClone است که در مورد ارجاعات با آدرس رخ می‌دهد. در این حالت بجای کپی نمودن مقدار، از کپی نمودن آدرس استفاده میشود (Ref Type چیست؟)

برای حل این مشکل باید DeepCopy انجام داد. لذا کد را بصورت زیر تغییر می‌دهیم:(DeepCopy و ShallowCopy چیست؟)

```
public abstract class APrototype : ICloneable
    {
        public string Name { get; set; }
        public string Health { get; set; }
        //This is a ref type
        public AdditionalDetails Detail { get; set; }
        public abstract APrototype ShallowClone();
        public abstract object Clone();
    }

    public class AdditionalDetails { public string Height { get; set; } }

    public class Prototype : APrototype
    {
        public override object Clone()
        {
            Prototype cloned = MemberwiseClone() as Prototype;
            //We need to deep copy each ref types in order to prevent shallow copy
            cloned.Detail = new AdditionalDetails { Height = this.Detail.Height };
            return cloned;
        }
        //Shallow copy will copy ref type's address instead of their value, so any changes in cloned
object or source object will take effect on both objects
        public override APrototype ShallowClone() { return this.MemberwiseClone() as APrototype; }
        public override string ToString() { return string.Format("Player name: {0}, Health statuse:
{1}, Height: {2}", Name, Health, Detail.Height); }
    }
```

و سپس بصورت زیر از آن استفاده نمود:

```
Prototype p1 = new Prototype { Name = "Vahid", Health = "OK", Detail = new AdditionalDetails { Height =
"100" } };
            Prototype p2 = p1.Clone() as Prototype;
            p2.Detail.Height = "200";
            Console.WriteLine("<This is Deep Copy>");
            Console.WriteLine(p1.ToString());
            Console.WriteLine(p2.ToString());

            Prototype p3 = new Prototype { Name = "Vahid", Health = "OK", Detail = new
AdditionalDetails { Height = "100" } };
            Prototype p4 = p3.ShallowClone() as Prototype;
            p4.Detail.Height = "200";
            Console.WriteLine("\n<This is Shallow Copy>");
```

```
            Console.WriteLine(p3.ToString());
            Console.WriteLine(p4.ToString());
```

لذا خروجی بصورت زیر را می‌توان مشاهده نمود:

```
<This is Deep Copy>
Player name: Vahid, Health statuse: OK, Height: 100
Player name: Vahid, Health statuse: OK, Height: 200

<This is Shallow Copy>
Player name: Vahid, Health statuse: OK, Height: 200
Player name: Vahid, Health statuse: OK, Height: 200
```

البته در این سناریو ShallowCopy باعث اشتباه شدن نتایج می‌شود. شاید شما در دامنه‌ی نیازمندیهای خود، اتفاقا به ShallowCopy نیاز داشته باشید و DeepCopy مرتفع کننده‌ی نیاز شما نباشد. لذا کاربرد هر کدام از آنها وابستگی مستقیمی به دامنه‌ی نیازمندی‌های شما دارد.

عنوان: **درخت‌ها و گراف‌ها قسمت سوم**
نویسنده: علی یگانه مقدم
تاریخ: ۱۳۹۳/۱۲/۰۵ ۱۰:۴۰
آدرس: www.dotnettips.info
گروه‌ها: C#, Data Structures, Tree

همانطور که در قسمت قبلی گفتیم، در این قسمت قرار است به پیاده سازی درخت جست و جوی دو دویی مرتب شده بپردازیم. در مطلب قبلی اشاره کردیم که ما متدهای افزودن، جستجو و حذف را قرار است به درخت اضافه کنیم و برای هر یک از این متدها توضیحاتی را ارائه خواهیم کرد. به این نکته دقت داشته باشید درختی که قصد پیاده سازی آن را داریم یک درخت متوازن نیست و ممکن است در بعضی شرایط کارآیی مطلوبی نداشته باشد.

همانند مثال‌ها و پیاده سازی‌های قبلی، دو کلاس داریم که یکی برای ساختار گره است BinaryTreeNode<T> و دیگری برای ساختار درخت اصلی BinaryTree<T>.

کلاس BinaryTreeNode که در پایین نوشته شده‌است بعدا داخل کلاس BinaryTree قرار خواهد گرفت:

```
internal class BinaryTreeNode<T> :
    IComparable<BinaryTreeNode<T>> where T : IComparable<T>
{
    // مقدار گره
    internal T value;

    // شامل گره پدر
    internal BinaryTreeNode<T> parent;

    // شامل گره سمت چپ
    internal BinaryTreeNode<T> leftChild;

    // شامل گره سمت راست
    internal BinaryTreeNode<T> rightChild;

    /// <summary>سازنده</summary>
    /// <param name="value">مقدار گره ریشه</param>
    public BinaryTreeNode(T value)
    {
        if (value == null)
        {
            // از آن جا که نال قابل مقایسه نیست اجازه افزودن را از آن سلب می‌کنیم
            throw new ArgumentNullException(
                "Cannot insert null value!");
        }

        this.value = value;
        this.parent = null;
        this.leftChild = null;
        this.rightChild = null;
    }

    public override string ToString()
    {
        return this.value.ToString();
    }

    public override int GetHashCode()
    {
        return this.value.GetHashCode();
    }

    public override bool Equals(object obj)
    {
        BinaryTreeNode<T> other = (BinaryTreeNode<T>)obj;
        return this.CompareTo(other) == 0;
    }

    public int CompareTo(BinaryTreeNode<T> other)
    {
        return this.value.CompareTo(other.value);
    }
}
```

تکلیف کدهای اولیه که کامنت دارند روشن است و قبلا چندین بار بررسی کردیم ولی کدها و متدهای جدیدتری نیز نوشته شده‌اند

که آن‌ها را بررسی می‌کنیم:
ما در مورد این درخت می‌گوییم که همه چیز آن مرتب شده است و گره‌ها به ترتیب چیده شده اند و اینکار تنها با مقایسه کردن گره‌های درخت امکان پذیر است. این مقایسه برای برنامه نویسان از طریق یک ذخیره در یک ساختمان داده خاص یا اینکه آن را به یک *نوع Type* قابل مقایسه ارسال کنند امکان پذیر است. در سی شارپ نوع قابل مقایسه با کلمه‌های کلیدی زیر امکان پذیر است:

```
T : IComparable<T>
```

در اینجا T می‌تواند هر نوع داده‌ای مانند Byte و int و ... باشد؛ ولی علامت : این محدودیت را اعمال می‌کند که کلاس باید از اینترفیس IComparable ارث بری کرده باشد. این اینترفیس برای پیاده‌سازی تنها شامل تعریف یک متد است به نام CompareTo(T) obj که عمل مقایسه داخل آن انجام می‌گردد و در صورت بزرگ بودن شیء جاری از آرگومان داده شده، نتیجه‌ی برگردانده شده، مقداری مثبت، در حالت برابر بودن، مقدار 0 و کوچکتر بودن مقدارمنفی خواهد بود. شکل تعریف این اینترفیس تقریبا چنین چیزی باید باشد:

```
public interface IComparable<T>
{
    int CompareTo(T other);
}
```

نوشتن عبارت بالا در جلوی کلاس، به ما این اطمینان را می‌بخشد که که نوع یا کلاسی که به آن پاس می‌شود، یک نوع قابل مقایسه است و از طرف دیگر چون می‌خواهیم گره‌هایمان نوعی قابل مقایسه باشند IComparable<T> را هم برای آن ارث بری می‌کنیم.
همچنین چند متد دیگر را نیز override کرده‌ایم که اصلی‌ترین آن‌ها GetHashCode و Equal است. موقعی که متد CompareTo مقدار 0 بر می‌گرداند مقدار برگشتی Equals هم باید True باشد.
... و یک نکته مفید برای خاطرسپاری اینکه موقعیکه دو شیء با یکدیگر برابر باشند، کد هش تولید شده آن‌ها نیز با هم برابر هستند. به عبارتی اشیاء یکسان کد هش یکسانی دارند. این رفتار سبب می‌شود که که بتوانید مشکلات زیادی را که در رابطه با مقایسه کردن پیش می‌آید، حل نمایید.

پیاده سازی کلاس اصلی BinarySearchTree

مهمترین نکته در کلاس زیر این مورد است که ما اصرار داشتیم، T باید از اینترفیس IComparable مشتق شده باشد. بر این حسب ما می‌توانیم با نوع داده‌هایی چون int یا string کار کنیم، چون قابل مقایسه هستند ولی نمی‌توانیم با int[] یا streamreader کار کنیم چرا که قابل مقایسه نیستند.

```
public class BinarySearchTree<T>    where T : IComparable<T>
{
    /// کلاسی که بالا تعریف کردیم
    internal class BinaryTreeNode<T> :
        IComparable<BinaryTreeNode<T>> where T : IComparable<T>
    {
        // …
    }

    /// <summary>
    /// ریشه درخت
    /// </summary>
    private BinaryTreeNode<T> root;

    /// <summary>
    /// سازنده کلاس
    /// </summary>
    public BinarySearchTree()
    {
        this.root = null;
    }

//پیاده سازی متدها مربوط به افزودن و حذف و جست و جو
}
```

در کد بالا ما کلاس اطلاعات گره را به کلاس اضافه می‌کنیم و یه سازنده و یک سری خصوصیت رابه آن اضافه کرده ایم.در این مرحله گام به گام هر یک از سه متد افزودن ، جست و جو و حذف را بررسی می‌کنیم و جزئیات آن را توضیح می‌دهیم.

افزودن یک عنصر جدید

افزودن یک عنصر جدید در این درخت مرتب شده، مشابه درخت‌های قبلی نیست و این افزودن باید طوری باشد که مرتب بودن درخت حفظ گردد. در این الگوریتم برای اضافه شدن عنصری جدید، دستور العمل چنین است: اگر درخت خالی بود عنصر را به عنوان ریشه اضافه کن؛ در غیر این صورت مراحل زیر را نجام بده:

اگر عنصر جدید کوچکتر از ریشه است، با یک تابع بازگشتی عنصر جدید را به زیر درخت چپ اضافه کن.

اگر عنصر جدید بزرگتر از ریشه است، با یک تابع بازگشتی عنصر جدید را به زیر درخت راست اضافه کن.

اگر عنصر جدید برابر ریشه هست، هیچ کاری نکن و خارج شو.

پیاده سازی الگوریتم بالا در کلاس اصلی:

```
public void Insert(T value)
{
    this.root = Insert(value, null, root);
}

/// <summary>
/// متدی برای افزودن عنصر به درخت
/// </summary>
/// <param name="value">مقدار جدید</param>
/// <param name="parentNode">والد گره جدید</param>
/// <param name="node">گره فعلی که همان ریشه است</param>
/// <returns>گره افزوده شده</returns>
private BinaryTreeNode<T> Insert(T value,
        BinaryTreeNode<T> parentNode, BinaryTreeNode<T> node)
{
    if (node == null)
    {
        node = new BinaryTreeNode<T>(value);
        node.parent = parentNode;
    }
    else
    {
        int compareTo = value.CompareTo(node.value);
        if (compareTo < 0)
        {
            node.leftChild =
                Insert(value, node, node.leftChild);
        }
        else if (compareTo > 0)
        {
            node.rightChild =
                Insert(value, node, node.rightChild);
        }
    }

    return node;
}
```

متد درج سه آرگومان دارد، یکی مقدار گره جدید است؛ دوم گره والد که با هر بار صدا زدن تابع بازگشتی، گره والد تغییر خواهد کرد و به گره‌های پایین‌تر خواهد رسید و سوم گره فعلی که با هر بار پاس شدن به تابع بازگشتی، گره ریشه‌ی آن زیر درخت است.

در مقاله قبلی اگر به یاد داشته باشید گفتیم که جستجو چگونه انجام می‌شود و برای نمونه به دنبال یک عنصر هم گشتیم و جستجوی یک عنصر در این درخت بسیار آسان است. ما این کد را بدون تابع بازگشتی و تنها با یک حلقه while پیاده خواهیم کرد. هر چند مشکلی با پیاده سازی آن به صورت بازگشتی وجود ندارد.

الگوریتم از ریشه بدین صورت آغاز می‌گردد و به ترتیب انجام می‌شود:

اگر عنصر جدید برابر با گره فعلی باشد، همان گره را بازگشت بده.

اگر عنصر جدید کوچکتر از گره فعلی است، گره سمت چپ را بردار و عملیات را از ابتدا آغاز کن (در کد زیر به ابتدای حلقه برو).

اگر عنصر جدید بزرگتر از گره فعلی است، گره سمت راست را بردار و عملیات را از ابتدا آغاز کن.

در انتها اگر الگوریتم، گره را پیدا کند، گره پیدا شده را باز می‌گرداند؛ ولی اگر گره را پیدا نکند، یا درخت خالی باشد، مقدار برگشتی نال خواهد بود.

حذف یک عنصر

حذف کردن در این درخت نسبت به درخت دودویی معمولی پیچیده‌تر است. اولین گام این عمل، جستجوی گره مدنظر است. وقتی گره‌ایی را مدنظر داشته باشیم، سه بررسی زیر انجام می‌گیرد:

اگر گره برگ هست و والد هیچ گره‌ای نیست، به راحتی گره مد نظر را حذف می‌کنیم و ارتباط گره والد با این گره را نال می‌کنیم.
اگر گره تنها یک فرزند دارد (هیچ فرقی نمی‌کند چپ یا راست) گره مدنظر حذف و فرزندش را جایگزینش می‌کنیم.
اگر گره دو فرزند دارد، کوچکترین گره در زیر درخت سمت راست را پیدا کرده و با گره مدنظر جابجا می‌کنیم. سپس یکی از دو عملیات بالا را روی گره انجام می‌دهیم.

اجازه دهید عملیات بالا را به طور عملی بررسی کنیم. در درخت زیر ما می‌خواهیم گره 11 را حذف کنیم. پس کوچکترین گره سمت راست، یعنی 13 را پیدا می‌کنیم و با گره 11 جابجا می‌کنیم.

بعد از جابجایی، یکی از دو عملیات اول بالا را روی گره 11 اعمال می‌کنیم و در این حالت گره 11 که یک گره برگ است، خیلی راحت حذف و ارتباطش را با والد، با یک نال جایگزین می‌کنیم.

```
/// عنصر مورد نظر را جست و جوی می‌کند و اگر مخالف نال بود گره برگشتی را به تابع حذف ارسال می‌کند
public void Remove(T value)
{
    BinaryTreeNode<T> nodeToDelete = Find(value);
    if (nodeToDelete != null)
    {
        Remove(nodeToDelete);
    }
}

private void Remove(BinaryTreeNode<T> node)
{
    //بررسی می‌کند که آیا دو فرزند دارد یا خیر
    // این خط باید اول همه باشد که مرحله یک و دو بعد از آن اجرا شود
    if (node.leftChild != null && node.rightChild != null)
    {
        BinaryTreeNode<T> replacement = node.rightChild;
        while (replacement.leftChild != null)
        {
            replacement = replacement.leftChild;
        }
        node.value = replacement.value;
        node = replacement;
    }

    // مرحله یک و دو اینجا بررسی میشه
    BinaryTreeNode<T> theChild = node.leftChild != null ?
            node.leftChild : node.rightChild;

    // اگر حداقل یک فرزند داشته باشد
    if (theChild != null)
    {
        theChild.parent = node.parent;

        // بررسی می‌کند گره ریشه است یا خیر
        if (node.parent == null)
        {
            root = theChild;
        }
        else
        {
            // جایگزینی عنصر با زیر درخت فرزندش
            if (node.parent.leftChild == node)
            {
                node.parent.leftChild = theChild;
```

```
            }
            else
            {
                node.parent.rightChild = theChild;
            }
        }
    }
    else
    {
        // کنترل وضعیت موقعی که عنصر ریشه است
        if (node.parent == null)
        {
            root = null;
        }
        else
        {
            // اگر گره برگ است آن را حذف کن
            if (node.parent.leftChild == node)
            {
                node.parent.leftChild = null;
            }
            else
            {
                node.parent.rightChild = null;
            }
        }
    }
}
```

در کد بالا ابتدا جستجو انجام می‌شود و اگر جواب غیر نال بود، گره برگشتی را به تابع حذف ارسال می‌کنیم. در تابع حذف اول از همه برسی می‌کنیم که آیا گره ما دو فرزند دارد یا خیر که اگر دو فرزنده بود، ابتدا گره‌ها را تعویض و سپس یکی از مراحل یک یا دو را که در بالاتر ذکر کردیم، انجام دهیم.

دو فرزندی

اگر گره ما دو فرزند داشته باشد، گره سمت راست را گرفته و از آن گره آن قدر به سمت چپ حرکت می‌کنیم تا به برگ یا گره تک فرزنده که صد در صد فرزندش سمت راست است، برسیم و سپس این دو گره را با هم تعویض می‌کنیم.

تک فرزندی

در مرحله بعد بررسی می‌کنیم که آیا گره یک فرزند دارد یا خیر؛ شرط بدین صورت است که اگر فرزند چپ داشت آن را در theChild قرار می‌دهیم، در غیر این صورت فرزند راست را قرار می‌دهیم. در خط بعدی باید چک کرد که theChild نال است یا خیر. اگر نال باشد به این معنی است که غیر از فرزند چپ، حتی فرزند راست هم نداشته، پس گره، یک برگ است ولی اگر مخالف نال باشد پس حداقل یک گره داشته است.

اگر نتیجه نال نباشد باید این گره حذف و گره فرزند ارتباطش را با والد گره حذفی برقرار کند. در صورتیکه گره حذفی ریشه باشد و والدی نداشته باشد، این نکته باید رعایت شود که گره فرزند بری متغیر root که در سطح کلاس تعریف شده است، نیز قابل شناسایی باشد.

در صورتی که خود گره ریشه نباشد و والد داشته باشد، غیر از اینکه فرزند باید با والد ارتباط داشته باشد، والد هم باید از طریق دو خاصیت فرزند چپ و راست با فرزند ارتباط برقرار کند. پس ابتدا برسی می‌کنیم که گره حذفی کدامین فرزند بوده: چپ یا راست؟ سپس فرزند گره حذفی در آن خاصیت جایگزین خواهد شد و دیگر هیچ نوع اشاره‌ای به گره حذفی نیست و از درخت حذف شده است.

بدون فرزند (برگ)

حال اگر گره ما برگ باشد مرحله دوم، کد داخل else اجرا خواهد شد و بررسی می‌کند این گره در والد فرزند چپ است یا

راست و به این ترتیب با نال کردن آن فرزند در والد ارتباط قطع شده و گره از درخت حذف می‌شود.

پیمایش درخت به روش DFS یا LVR یا In-Order

```
public void PrintTreeDFS()
{
    PrintTreeDFS(this.root);
    Console.WriteLine();
}


private void PrintTreeDFS(BinaryTreeNode<T> node)
{
    if (node != null)
    {
        PrintTreeDFS(node.leftChild);
        Console.Write(node.value + " ");
        PrintTreeDFS(node.rightChild);
    }
}
```

در مقاله بعدی درخت دودویی متوازن را که پیچیده‌تر از این درخت است و از کارآیی بهتری برخوردار هست، بررسی می‌کنیم.

عنوان: آشنایی با الگوی طراحی Fly Weight
نویسنده: وحید فرهمندیان
تاریخ: ۱۳۹۳/۱۲/۰۵ ۱۴:۲۰
آدرس: www.dotnettips.info
گروه‌ها: C#, OOP, DesignPatterns

سناریوی زیر را در نظر بگیرید:
فرض کنید از شما خواسته شده است تا یک پردازشگر متن را بنویسید. خوب در این پردازشگر با یک‌سری کاراکتر روبرو هستید که هر کاراکتر احتمالا آبجکتی از نوع کلاس خود می‌باشد؛ برای مثال آبجکت XYZ که آبجکتی از نوع کلاس A هست و برای نمایش کاراکتر A استفاده می‌شود. این آبجکت‌ها دارای دو دسته خصیصه هستند: (مطالعه بیشتر)
خصیصه‌های ثابت: یعنی همه کاراکترهای A دارای یک شکل مشخص هستند. در واقع مشخصات ذاتی آبجکت می‌باشند.

خصیصه‌های پویا: یعنی هر کاراکتر دارای فونت، سایز و رنگ خاص خود است. در واقع خصیصه‌هایی که از یک آبجکت به آبجکت دیگر متفاوت هستند .

خوب احتمالا در ساده‌ترین راه حل، به ازای تک تک کاراکترهایی که کاربر وارد می‌کند، یک آبجکت از نوع کلاس متناسب با آن ساخته می‌شود. ولی بحث مهم این است که با این همه آبجکت که هر یک مصرف خود را از حافظه دارند، می‌خواهید چکار کنید؟ احتمالا به مشکل حافظه برخورد خواهید کرد! پس باید یک سناریوی بهتر ایجاد کرد.
سناریوی پیشنهادی این است که برای هر نوع کاراکتر، یک کلاس داشته باشیم، همانند قبل(یک کلاس برای A یک کلاس برای B و غیره) و یک استخر پر از آبجکت داشته باشیم که آبجکت‌های ایجاد شده در آن ذخیره شوند.
سپس کاربر، کاراکتر A را درخواست می‌کند. ابتدا به این استخر نگاه می‌کنیم. اگر کاراکتر A موجود بود، آن را برمی‌گردانیم و اگر موجود نبود، یک آبجکت از نوع A می‌سازیم، سپس این آبجکت را در استخر ذخیره می‌کنیم و آبجکت را بر می‌گردانیم. در این صورت اگر کاربر دوباره درخواست A را کرد، دیگر نیازی به ساخت آبجکت جدید نیست و از آبجکت قبلی می‌توانیم استفاده نماییم. با این شرایط تکلیف خصایص ایستا مشخص است. ولی مشکل مهم با خصایص پویا این است که می‌توانند بین آبجکت‌ها متفاوت باشند که برای این هم یک متد در کلاس‌ها قرار می‌دهیم تا این خصایص را تنظیم نماید.
به کد زیر دقت نمایید:

```
public interface IAlphabet
    {
        void Render(string font);//Define Extrinsic and non-static states for each object
    }

    public class A : IAlphabet
    {
        public void Render(string font) { Console.WriteLine(GetType().Name + " has font of type " +
font); }
    }
    public class B : IAlphabet
    {
        public void Render(string font) { Console.WriteLine(GetType().Name + " has font of type " +
font); }
    }
```

از متد Render برای تنظیم نمودن خصایص پویا استفاده خواهد شد.
سپس در ادامه به یک موتور نیاز داریم که قبل از ساخت آبجکت، استخر را بررسی نماید:

```
public class FlyWeightFactory
    {
        private readonly Dictionary<string, IAlphabet> _dictionary = new Dictionary<string,
IAlphabet>();
        public int Count { get { return _dictionary.Count; } }
        public IAlphabet GetObject(string name)
        {
            if (!_dictionary.ContainsKey(name))
                switch (name)
                {
                    case "A":
                        _dictionary.Add(name, new A());
```

```
                        Console.WriteLine("New object created");
                        break;
                    case "B":
                        _dictionary.Add(name, new B());
                        Console.WriteLine("New object created");
                        break;
                    default:
                        throw new Exception("Factory can not create given object");
                }
            else
                Console.WriteLine("Object reused");
            return _dictionary[name];
        }
    }
```

در اینجا _dictionaries همان استخر ما می‌باشد که قرار است آبجکت‌ها در آن ذخیره شوند. Count برای نمایش تعداد آبجکت‌های موجود در استخر استفاده می‌شود (حداکثر مقدار آن چقدر خواهد بود؟). GetObject نیز همان موتور اصلی کار است که در آن ابتدای استخر بررسی می‌شود. اگر آبجکت در استخر نبود، یک نمونه‌ی جدید از آن ساخته شده، به استخر اضافه گردیده و برگردانده می‌شود.

لذا برای استفاده‌ی از این کد داریم:

```
FlyWeightFactory flyWeightFactory = new FlyWeightFactory();
 IAlphabet alphabet = flyWeightFactory.GetObject(typeof(A).Name);
 alphabet.Render("Arial");
 Console.WriteLine();
 alphabet = flyWeightFactory.GetObject(typeof(B).Name);
 alphabet.Render("Tahoma");
 Console.WriteLine();
 alphabet = flyWeightFactory.GetObject(typeof(A).Name);
 alphabet.Render("Time is New Roman");
 Console.WriteLine();
 alphabet = flyWeightFactory.GetObject(typeof(A).Name);
 alphabet.Render("B Nazanin");
 Console.WriteLine();
 Console.WriteLine("Total new alphabet count:" + flyWeightFactory.Count);
```

با اجرای این کد خروجی زیر را مشاهده خواهید نمود:

```
New object created
A has font of type Arial

New object created
B has font of type Tahoma

Object reused
A has font of type Time is New Roman

Object reused
A has font of type B Nazanin

Total new alphabet count:2
```

نکته‌ی قابل توجه این است که این الگو بصورت داخلی از الگوی Factory Method استفاده می‌کند. با توجه بیشتر به پیاده سازی Flyweight Factory شباهت هایی بین آن و Singleton Pattern می‌بینیم. کلاس‌هایی از این دست را Multiton می نامند. در Multiton نمونه‌ها بصورت زوج کلیدهایی نگهداری می‌شوند و بر اساس Key دریافت شده نمونه‌ی متناظر بازگردانده می‌شود. همچنین در Singleton تضمین می‌شود که از کلاس مربوطه فقط یک نمونه در کل Application وجود دارد. در Multiton Pattern تضمین می‌شود که برای هر Key تنها یک Instance وجود دارد.

عنوان: آشنایی با الگوی طراحی Abstract Factory
نویسنده: وحید فرهمندیان
تاریخ: ۱۳۹۳/۱۲/۰۵ ۱۹:۱۰
آدرس: www.dotnettips.info
گروه‌ها: C#, OOP, DesignPatterns

قبل از مطالعه‌ی این مطلب، حتما الگوی طراحی Factory Method را مطالعه نمایید.
همانطور که در الگوی طراحی Factory Method مشاهده شد، این الگو یک عیب دارد، آن هم این است که از کدام Creator باید استفاده شود و مستقیما در کد بایستی ذکر شود.

```
class ConcreteCreator : Creator
{
     public override IProduct FactoryMethod(string type)
    {
            switch (type)
           {
                case "A": return  new ConcreteProductA();
                case "B": return  new ConcreteProductB();
                default: throw new ArgumentException("Invalid type", "type");
           }
     }
}
```

برای حل این مشکل می‌توانیم سراغ الگوی طراحی دیگری برویم که Abstract Factory نام دارد. این الگوی طراحی 4 بخش اصلی دارد که هر کدام از این بخش‌ها را طی مثالی توضیح می‌دهم:

1. Abstract Factory: در کشور، صنعت خودروسازی داریم که خودروها را در دو دسته‌ی دیزلی و سواری تولید می‌کنند :

```
public interface IVehicleFactory     {
        IDiesel GetDiesel();
        IMotorCar GetMotorCar();
    }
```

2. Concrete Factory: دو کارخانه‌ی تولید خودرو داریم که در صنعت خودرو سازی فعالیت دارند و عبارتند از ایران خودرو و سایپا که هر کدام خودروهای خود را تولید می‌کنند. ولی هر خودرویی که تولید می‌کنند یا دیزلی است یا سواری. شرکت ایران خودرو، خودروی آرنا را بعنوان دیزلی تولید می‌کند و پژو 206 را بعنوان سواری. همچنین شرکت سایپا خودروی فوتون را بعنوان خودروی دیزلی تولید می‌کند و خودروی پراید را بعنوان خودروی سواری.

```
public class IranKhodro : IVehicleFactory
    {
        public IDiesel GetDiesel() { return new Arena(); }
        public IMotorCar GetMotorCar() { return new Peugeot206(); }
    }
    public class Saipa : IVehicleFactory
    {
        public IDiesel GetDiesel() { return new Foton(); }
        public IMotorCar GetMotorCar() { return new Peride(); }
    }
```

3. Abstract Product: خودروهای تولیدی همانطور که گفته شد یا دیزلی هستند یا سواری که هر کدام از این خودروها ویژگی‌های خاص خود را دارند (در این مثال هر دو دسته خودرو برای خود نام دارند)

```
public interface IDiesel { string GetName();}
    public interface IMotorCar { string GetName();}
```

4. Concrete Product: در بین این خودروها، خودروی پژو 206 و پراید یک خودروی سواری هستند و خودروی فوتون و آرنا، خودروهای دیزلی.

```
public class Foton : IDiesel { public string GetName() { return "This is Foton"; } }
    public class Arena : IDiesel { public string GetName() { return "This is Arena"; } }
    public class Peugeot206 : IMotorCar { public string GetName() { return "This is Peugeot206"; } }
```

```
    public class Peride : IMotorCar { public string GetName() { return "This is Peride"; } }
```

حال که 4 دسته اصلی این الگوی طراحی را آموختیم می‌توان از آن بصورت زیر استفاده نمود:

```
IVehicleFactory factory = new IranKhodro();
 Console.WriteLine("***" + factory.GetType().Name + "***");
 IDiesel diesel = factory.GetDiesel();
 Console.WriteLine(diesel.GetName());
 IMotorCar motorCar = factory.GetMotorCar();
 Console.WriteLine(motorCar.GetName());

 factory = new Saipa();
 Console.WriteLine("***" + factory.GetType().Name + "***");
 diesel = factory.GetDiesel();
 Console.WriteLine(diesel.GetName());
 motorCar = factory.GetMotorCar();
 Console.WriteLine(motorCar.GetName());
```

همانطور که در کد فوق مشاهده میشود، ایراد موجود در الگوی Factory Method اینجا از بین رفته است و برای ساخت آبجکت‌های مختلف از Innterfaceها یا Abstract Classها استفاده می‌کنیم.

کلا Abstract Factory مزایای زیر را دارد:

پیاده سازی و نامگذاری Product در Factory مربوطه متمرکز می‌شود و بدین ترتیب Client به نام و نحوه پیاده سازی Typeهای مختلف Product وابستگی نخواهد داشت.

به راحتی می‌توان Concrete Factory مورد استفاده در برنامه را تغییر داد، بدون اینکه تاثیری در عملکرد سایر بخش‌ها داشته باشد.

در مواردی که بیش از یک محصول برای هر خانواده وجود داشته باشد، استفاده از Abstract Factory تضمین می‌کند که Productهای هر خانواده همه در کنار هم قرار دارند و با هم فعال و غیر فعال می‌شوند. (یا همه، یا هیچکدام)

بزرگترین عیبی که این الگوی طراحی دارد این است که با اضافه شدن فقط یک Product تازه، Abstract Factory باید تغییر کند که این مساله منجر به تغییر همه Concrete Factoryها می‌شود.

نهایتا اینکه در استفاده از این الگوی طراحی به این تکنیک‌ها توجه داشته باشید:

Factoryها معمولا Singleton هستند. زیرا هر Application **بطور معمول** فقط به یک instance از هر Concrete Factory نیاز دارد.

انتخاب Concrete Factory مناسب معمولا توسط پارامترهایی انجام می‌شود.

نمودار کلاسی این الگو نیز بصورت زیر میباشد:

و کلام آخر در مورد این الگو:

Abstract Factory یک interface یا کلاس abstract است که signature متدهای ساخت Objectها در آن تعریف شده است و Concrete Factoryها آن‌ها را implement می‌نمایند.

در Abstract Factory Pattern همه Productهای هم خانواده در Concrete Factory مربوط به آن خانواده پیاده سازی و مجتمع می‌گردند.

در کدهای برنامه تنها با Abstract Factory و Abstract Productها سر و کار داریم و به هیچ وجه درگیر این مساله که کدام یک از Concrete Classها در برنامه مورد استفاده قرار می‌گیرند، نمی‌شویم.

عنوان: **درخت‌ها و گراف قسمت چهارم**
نویسنده: علی یگانه مقدم
تاریخ: ۱۳۹۳/۱۲/۰۶ ۱:۰
آدرس: www.dotnettips.info
گروه‌ها: C#, Data Structures, Tree

در قسمت قبلی مبحث پیاده سازی ساختمان (ساختار) درخت‌های جستجوی دودویی را به پایان رساندیم. در این قسمت قرار است بر روی درخت متوازن بحث کنیم و آن را پیاده سازی نماییم.

درخت متوازن

همانطور که دیدید، عملیات جستجو روی درخت جستجوی دو دویی به مراتب راحت و آسان‌تر است؛ ولی با این حال این درخت در عملیاتی چون درج و حذف، یک نقص فنی دارد و آن هم این است که نمی‌تواند عمق خود را کنترل کند و همینطور به سمت عمق‌های بیشتر و بزرگتر حرکت می‌کند. مثلا ساختار ترتیبی زیر را برای مقداری‌های 1 و 2 و 3 و 4 و 5و 6 در نظر بگیرید:

در این حالت دیگر درخت مانند یک درخت رفتار نمی‌کند و بیشتر شبیه لیست پیوندی است و عملیات جستجو همینطور کندتر و کندتر می‌شود و دیگر مثل سابق نخواهد بود، پیچیدگی برنامه بیشتر خواهد شد و از Log(n) به n می‌رسد. از آنجا که دوست داریم برای عملیات‌های رایجی چون درج و جستجو و حذف، همین پیچیدگی لگاریتمی را حفظ کنیم، از ساختاری جدیدتر بهره خواهیم برد.

درخت دودویی متوازن: درختی است که در آن هیچ برگی، عمقش از هیچ برگی بیشتر نیست.

درخت دودویی متوازن کامل: درختی که تفاوتش در تعداد گره‌های چپ یا راست است و حداقل یک فرزند دارند.

درخت دودویی متوازن حتی اگر کامل هم نباشد، در عملیات پایه‌ای چون افزودن، حذف و جستجو در بدترین حالت هم با پیچیدگی لگاریتمی تعداد گام‌ها همراه است. برای اینکه این درخت با به هم ریختگی توازنش روبرو نشود، باید حین انجام عملیات پایه، جایگاه تعداد المان‌های آن بررسی و اصلاح شود که به این عملیات چرخش یا دوران Rotation می‌گویند. انجام این عملیات بستگی دارد که پیاده سازی این درخت به چه شکلی باشد و به چه صورتی پیاده سازی شده باشد. از پیاده سازی‌های این درخت می‌توان به درخت سرخ-سیاه Black Red Tree ، ای وی ال AVL Tree ، اسپلی Splay و ... اشاره کرد.

با توجه به موارد بالا میتوانیم به نتایج زیر برسیم که چرا این درخت در هر حالت، پیچیدگی زمانی خودش را در لگاریتم n حفظ

می‌کند:
المان‌ها و عناصرش را مرتب شده نگه می‌دارد.
خودش را متوازن نگه داشته و اجازه نمی‌دهد عمقش بیشتر از لگاریتم n شود.

نمونه ای از درخت جستجوی دو دویی

همان درخت ولی به صورت متوازن با پیاده سازی AVL

درخت‌های متوازن هم می‌متوانند **دو دویی یا باینری** باشند و هم **غیر باینری non-Binary**

درخت‌های دو دویی در انجام عملیات بسیار سریع هستند و در بالا به تعدادی از انواع پیاده سازی‌های آن اشاره کردیم.

درخت‌های غیر باینری نیز مرتب و متوازن بوده، ولی می‌توانند بیش از یک کلید داشته باشند و همچنین بیشتر از دو فرزند. این درخت‌ها نیز عملیات خود را بسیار سریع انجام می‌دهند. از نمونه‌های این درخت می‌توان به `B Tree,B+ Tree,Interval Tree` اشاره کرد.

از آنجا که پیاده‌سازی این نوع درخت کمی دشوار و پیچیده و طولانی است و همچنین پیاده سازی‌های مختلفی دارد؛ تعدادی از منابع موجود را در زیر معرفی می‌کنیم:

در خود دات نت در فضای نام `system.collection.generic` کلاس `TreeSet` یک نوع پیاده سازی از این درخت است که این پیاده سازی از نوع درخت سرخ سیاه می‌باشد و جالب است بدانید، در طی بیست گام می‌تواند در یک میلیون آیتم به جستجو بپردازد ولی خبر بد اینکه استفاده مستقیم از این کلاس ممکن نیست چرا که این کلاس به صورت داخلی `internal` برای استفاده‌ی خود کتابخانه طراحی شده است ولی خبر خوب اینکه کلاس `sortedDictionary` از این کلاس بهره برده است و به صورت عمومی در دسترس ما قرار گرفته است. همچنین کلاس `SortedSet` هم از دات نت 4 نیز در دسترس است.

کتابخانه‌های خارجی جهت استفاده در دات نت که به پیاده‌سازی درخت‌های متوازن پرداخته‌اند:

پیاده سازی `Splay Tree` با سی شارپ

پیاده سازی `AVL` به صورت بهینه و آسان برای استفاده

پیاده سازی درخت `AVL` با کارایی بالا

پیاده سازی درخت `AVL` به همراه نمایش گرافیکی آن

درخت سرخ-سیاه

کتابخانه ای متن باز برای درخت سرخ-سیاه

درخت متوازن به همراه جست و جو و حذف و پیمایش ها

غیر دودویی‌ها

پیاده سازی B Tree
پیاده سازی Interval Tree
پیاده سازی Interval Tree در سی ++

عنوان: **استفاده از قالب ویژوال استودیو 2013 برای برنامه‌های ویندوز فرم**
نویسنده: احمد نواصری
تاریخ: ۱۳۹۳/۱۲/۰۶ ۱۱:۲۵
آدرس: www.dotnettips.info
گروه‌ها: C#, Windows forms, Visual Studio, Graphics

در این نوشتار قصد داریم تا Theme ویژوال استودیو 2013 را برای برنامه‌های ویندوز شبیه سازی کنیم. در مرحله اول یک پروژه از نوع ClassLibrary می‌سازیم و پس از آن یک کلاس که از کلاس ToolStripProfessionalRenderer ارث بری کند را ایجاد می‌کنیم. در اینجا ما نام کلاس را BlueMenuStrip انتخاب می‌کنیم. از این کلاس برای تغییر رنگ منوها استفاده می‌شود. سپس متد OnRenderMenuItemBackground آن‌را Override می‌کنیم.

```
using System.Drawing;
using System.Windows.Forms;

namespace Navasser.Theme.VisualStudio
{
    public class BlueMenuStrip:ToolStripProfessionalRenderer
    {
        protected override void OnRenderMenuItemBackground(ToolStripItemRenderEventArgs e)
        {
            var borderColor = ColorTranslator.FromHtml("#E5C365");//Menu Item Border
            var selectedMenuBackColor = ColorTranslator.FromHtml("#FDF4BF");//Menu Item Background
            var menuOpenedBackColor = ColorTranslator.FromHtml("#EAF0FF");
            var borderPen = new Pen(borderColor);

            if (e.Item.Selected)//اگر آیتمی از منوها انتخاب شد
            {
                var selectedMenuBrush = new SolidBrush(selectedMenuBackColor);
                var selectedItemBounds = new Rectangle(Point.Empty, e.Item.Size);//اقدام به پر کردن یک
مستطیل به اندازه ابعاد آیتم انتخاب شده می‌کند
                e.Graphics.FillRectangle(selectedMenuBrush,selectedItemBounds);
                e.Graphics.DrawRectangle(borderPen,0,0,selectedItemBounds.Width-
1,selectedItemBounds.Height-1);// بوردر آیتم را رسم میکند
                e.Item.BackColor = menuOpenedBackColor;
            }
            else
            {
                base.OnRenderMenuItemBackground(e);
                e.ToolStrip.BackColor = ColorTranslator.FromHtml("#EAF0FF");
            }
        }
    }
}
```

تکه کد بالا فقط برای تغییر رنگ زمینه‌ی منوها بکار می‌رود. اما برای تغییر رنگ ToolStripها یک کلاس جدید ایجاد می‌کنیم که از کلاس ToolStripProfessionalRenderer ارث بری کرده باشد و متدهای OnRenderToolStripBackground و OnRenderButtonBackground آن‌را Override میکنیم.

```
using System.Drawing;
using System.Windows.Forms;

namespace Navasser.Theme.VisualStudio
{
    public class BlueToolStrip : ToolStripProfessionalRenderer
    {
        protected override void OnRenderToolStripBackground(ToolStripRenderEventArgs e)
        {
            var toolStripBackColor = ColorTranslator.FromHtml("#D6DBE9");
            var toolStripBrush = new SolidBrush(toolStripBackColor);
            e.Graphics.FillRectangle(toolStripBrush,0,0,e.AffectedBounds.Width +
10,e.AffectedBounds.Height);
        }

        protected override void OnRenderButtonBackground(ToolStripItemRenderEventArgs e)
        {
            var borderColor = ColorTranslator.FromHtml("#E5C365");//For border
            var selectedToolItemBackColor = ColorTranslator.FromHtml("#FDF4BF");
            var selectedItemBrush = new SolidBrush(selectedToolItemBackColor);
            var pressedItemBackColor = ColorTranslator.FromHtml("#FFF29D");
            var pressedItemBrush = new SolidBrush(pressedItemBackColor);
```

```
            var borderPen = new Pen(borderColor);

            if (e.Item.Selected)
            {
                e.Graphics.FillRectangle(selectedItemBrush,0,0,e.Item.Width,e.Item.Height);
                e.Graphics.DrawRectangle(borderPen,0,0,e.Item.Width-1,e.Item.Height-1);
            }

            if (e.Item.Pressed)
            {
                e.Graphics.FillRectangle(pressedItemBrush, 0, 0, e.Item.Width, e.Item.Height);
                e.Graphics.DrawRectangle(borderPen, 0, 0, e.Item.Width - 1, e.Item.Height - 1);
            }
        }
    }
}
```

سپس Dll ایجاد شده را در برنامه خود Reference دهید و از Themeهای ایجاد شده استفاده نمایید. نتیجه‌ی کدهای بالا به شکل زیر است:

سند
کاربر
ویرایش
پشتیبان
اختیارات
کمک
ویرایش درخواست
ویرایش بار تحویلی از شرکتهای تولیدی
جستجوی اقلام موجود
حذف درخواست

سند
کاربر
ویرایش
پشتیبان
اختیارات
کمک

سند
کاربر
ویرایش
پشتیبان
اختیارات
کمک
ویرایش درخواست
ویرایش بار تحویلی از شرکتهای تولیدی
جستجوی اقلام موجود
حذف درخواست

سند
کاربر
ویرایش
پشتیبان
اختیارات
کمک

همچنین می‌توانید برای انتخاب رنگ‌های دلخواه خودتان از ابزار ColorSchemer Studio استفاده کنید.

نظرات خوانندگان

نویسنده: شهریار
تاریخ: ۱۳۹۳/۱۲/۰۸ ۱۵:۶

با سلام و تشکر

من کدهای بالا را استفاده کردم ولی شبیه به رنگهای عکس اول نشد. اگر ممکنه راهنمایی بفرمایین

نویسنده: احمد نواصری
تاریخ: ۱۳۹۳/۱۲/۰۹ ۱۱:۳۲

سلام دوست عزیز. کاشکی یه عکس از فرمتون قرار میدادین تا من بهتر متوجه مشکل میشدم.

نویسنده: وحید نصیری
تاریخ: ۱۳۹۳/۱۲/۱۱ ۲۳:۴

پروژه‌ی مثال بحث جاری برای امتحان: WinFormsThemes.zip

نویسنده: احمد نواصری
تاریخ: ۱۳۹۳/۱۲/۱۲ ۲۰:۳۱

این هم یک نمونه پروژه دیگه : VS-2013-Theme.zip

عنوان: آشنایی با الگوی طراحی Template Method
نویسنده: وحید فرهمندیان
تاریخ: ۱۳۹۳/۱۲/۰۹ ۱۸:۲۵
آدرس: www.dotnettips.info
گروه‌ها: C#, OOP, DesignPatterns

سناریویی وجود دارد که در آن شما می‌خواهید تنها یک کار را انجام دهید، ولی برای انجام آن n روش وجود دارد. برای مثال قصد مرتب سازی دارید و برای اینکار روش‌های مختلفی وجود دارند. برای حل این مساله پیشتر از الگوی طراحی استراتژی استفاده نمودیم.(مطالعه بیشتر در مورد الگوی طراحی استراتژی)

حال به سناریویی برخورد کردیم که بصورت زیر است:

می‌خواهیم یک کار را انجام دهیم ولی برای انجام این کار تنها برخی بخش‌های کار با هم متفاوت هستند. برای مثال قصد تولید گزارش و چاپ آن را داریم. در این سناریو خواندن اطلاعات و پردازش آن‌ها رخدادهایی ثابت هستند. ولی اگر بخواهیم گزارش را چاپ کنیم به مشکل می‌خوریم؛ چرا که چاپ گزارش به فرمت اکسل، فرمت و روش خود را دارد و چاپ به فرمت PDF شرایط خود را دارد.

در این سناریو دیگر الگوی طراحی استراتژی جواب نخواهد داد و نیاز داریم با یک الگوی طراحی جدید آشنا بشویم. این الگوی طراحی Template Method نام دارد.

در این الگو یک کلاس انتزاعی داریم به صورت زیر:

```
public abstract class DataExporter
    {
        public void ReadData()
        {
            Console.WriteLine("Data is reading from SQL Server Database");
        }

        public void ProcessData()
        {
            Console.WriteLine("Data is processing...!");
        }

        public abstract void PrintData();

        public void GetReport()
        {
            ReadData();
            ProcessData();
            PrintData();
        }
    }
```

این کلاس abstract، یک متد بنام GetReport دارد که نحوه‌ی انجام کار را مشخص می‌کند. متدهای ReadData و ProcessData نشان می‌دهند که انجام این دو عمل همیشه ثابت هستند (منظور در این سناریو همیشه ثابت هستند). متد PrintData همانطور که مشاهده می‌شود بصورت انتزاعی تعریف شده است، چرا که چاپ عملی است که در هر فرمت دارای خروجی متفاوتی می‌باشد.

لذا در ادامه داریم:

```
public class ExcelExporter : DataExporter
    {
        public override void PrintData()
        {
            Console.WriteLine("Data exported to Microsoft Excel!");
        }
    }

    public class PDFExporter : DataExporter
    {
        public override void PrintData()
        {
            Console.WriteLine("Data exported to PDF!");
        }
    }
```

کلاس ExcelExporter برای چاپ به فرمت اکسل می‌باشد. همانطور که مشاهده می‌شود این کلاس از کلاس انتزاعی DataExporter ارث بری کرده است. این بدین معنا است که کلاس ExcelExporter کارهای ReadData و ProcessData را از کلاس

پدر خود می‌گیرد و در ادامه نحوه‌ی چاپ مختص به خود را پیاده می‌کند. همین توضیحات در مورد PDFExporter نیز صادق است.
حال برای استفاده‌ی از این کدها داریم:

```
DataExporter dataExporter = new ExcelExporter();
            dataExporter.GetReport();
            Console.WriteLine("****************************");
            dataExporter = new PDFExporter();
            dataExporter.GetReport();
```

شما شاید بخواهید متدهای ReadData و ExportData و ProcessData را با سطح دسترسی متفاوتی از public تعریف نمایید که در این مقاله به این دلیل که خارج از بحث بود به آنها اشاره نشد و بصورت پیش فرض public در نظر گرفته شد.

عنوان: **نکات کار با استثناءها در دات نت**
نویسنده: سیروان عفیفی
تاریخ: ۱۳۹۳/۱۲/۲۹ ۲۰:۵
آدرس: www.dotnettips.info
گروه‌ها: C#, Design patterns, .NET, Exception, exception handling

استثناء چیست؟

واژه‌ی استثناء یا exception کوتاه شده‌ی عبارت exceptional event است. در واقع exception یک نوع رویداد است که در طول اجرای برنامه رخ می‌دهد و در نتیجه، جریان عادی برنامه را مختل می‌کند. زمانیکه خطایی درون یک متد رخ دهد، یک شیء (exception object) حاوی اطلاعاتی درباره‌ی خطا ایجاد خواهد شد. به فرآیند ایجاد یک exception object و تحویل دادن آن به سیستم runtime، اصطلاحاً throwing an exception یا صدور استثناء گفته می‌شود که در ادامه به آن خواهیم پرداخت.

بعد از اینکه یک متد استثناءایی را صادر می‌کند، سیستم runtime سعی در یافتن روشی برای مدیریت آن خواهد کرد.

خوب اکنون که با مفهوم استثناء آشنا شدید اجازه دهید دو سناریو را با هم بررسی کنیم.

- سناریوی اول:

فرض کنید یک فایل XML از پیش تعریف شده (برای مثال یک لیست از محصولات) قرار است در کنار برنامه‌ی شما باشد و باید این لیست را درون برنامه‌ی خود نمایش دهید. در این حالت برای خواندن این فایل انتظار دارید که فایل وجود داشته باشد. اگر این فایل وجود نداشته باشد برنامه‌ی شما با اشکال روبرو خواهد شد.

- سناریوی دوم:

فرض کنید یک فایل XML از آخرین محصولات مشاهده شده توسط کاربران را به صورت cache در برنامه‌تان دارید. در این حالت در اولین بار اجرای برنامه توسط کاربر انتظار داریم که این فایل موجود نباشد و اگر فایل وجود نداشته باشد به سادگی می‌توانیم فایل مربوط را ایجاده کرده و محصولاتی را که توسط کاربر مشاهده شده، درون این فایل اضافه کنیم.

در واقع استثناءها بستگی به حالت‌های مختلفی دارد. در مثال اول وجود فایل حیاتی است ولی در حالت دوم بدون وجود فایل نیز برنامه می‌تواند به کار خود ادامه داده و فایل مورد نظر را از نو ایجاد کند.

استثناها مربوط به زمانی هستند که این احتمال وجود داشته باشد که برنامه طبق انتظار پیش نرود.

برای حالت اول کد زیر را داریم:

```
public IEnumerable<Product> GetProducts()
{
    using (var stream = File.Read(Path.Combine(Environment.CurrentDirectory, "products.xml")))
    {
        var serializer = new XmlSerializer();
        return (IEnumerable<Product>)serializer.Deserialize(stream);
    }
}
```

همانطور که عنوان شد در حالت اول انتظار داریم که فایلی بر روی دیسک موجود باشد. در نتیجه نیازی نیست هیچ استثناءایی را مدیریت کنیم (زیرا در واقع اگر فایل موجود نباشد هیچ روشی برای ایجاد آن نداریم).

در مثال دوم می‌دانیم که ممکن است فایل از قبل موجود نباشد. بنابراین می‌توانیم موجود بودن فایل را با یک شرط بررسی کنیم:

```
public IEnumerable<Product> GetCachedProducts()
{
    var fullPath = Path.Combine(Environment.CurrentDirectory, "ProductCache.xml");
    if (!File.Exists(fullPath))
        return new Product[0];

    using (var stream = File.Read(fullPath))
    {
        var serializer = new XmlSerializer();
        return (IEnumerable<Product>)serializer.Deserialize(stream);
    }
}
```

چه زمانی باید استثناءها را مدیریت کنیم؟

زمانیکه بتوان متدهایی که خروجی مورد انتظار را بر می‌گردانند ایجاد کرد.

اجازه دهید دوباره از مثال‌های فوق استفاده کنیم:

```
IEnumerable<Product> GetProducts()
```

همانطور که از نام آن پیداست این متد باید همیشه لیستی از محصولات را برگرداند. اگر می‌توانید اینکار را با استفاده از catch کردن یک استثنا انجام دهید در غیر اینصورت نباید درون متد اینکار را انجام داد.

```
IEnumerable<Product> GetCachedProducts()
```

در متد فوق می‌توانستیم از FileNotFoundException برای فایل موردنظر استفاده کنیم؛ اما مطمئن بودیم که فایل در ابتدا وجود ندارد.
در واقع استثناها حالت‌هایی هستند که غیرقابل پیش‌بینی هستند. این حالت‌ها می‌توانند یک خطای منطقی از طرف برنامه‌نویس و یا چیزی خارج کنترل برنامه‌نویس باشند (مانند خطاهای سیستم‌عامل، شبکه، دیسک). یعنی در بیشتر مواقع این نوع خطاها را نمی‌توان مدیریت کرد.

اگر می‌خواهید استثناءها را catch کرده و آنها را لاگ کنید در بالاترین لایه اینکار را انجام دهید.

چه استثناءهایی باید مدیریت شوند و کدام‌ها خیر؟

مدیریت صحیح استثناءها می‌تواند خیلی مفید باشد. همانطور که عنوان شد یک استثناء زمانی رخ می‌دهد که یک حالت استثناء در برنامه اتفاق بیفتد. این مورد را بخاطر داشته باشید، زیرا به شما یادآوری می‌کند که در همه جا نیازی به استفاده از try/catch نیست. در اینجا ذکر این نکته خیلی مهم است:
تنها استثناءهایی را catch کنید که بتوانید برای آن راه‌حلی ارائه دهید.
به عنوان مثال اگر در لایه‌ی دسترسی به داده، خطایی رخ دهد و استثناءی SqlException صادر شود، می‌توانیم آن را catch کرده و درون یک استثناء عمومی‌تر قرار دهیم:

```
public class UserRepository : IUserRepository
{
    public IList<User> Search(string value)
    {
        try
        {
             return CreateConnectionAndACommandAndReturnAList("WHERE value=@value",
Parameter.New("value", value));
        }
        catch (SqlException err)
        {
            var msg = String.Format("Ohh no!  Failed to search after users with '{0}' as search
string", value);
            throw new DataSourceException(msg, err);
        }
    }
}
```

همانطور که در کد فوق مشاهده می‌کنید به محض صدور استثنای SqlException آن را درون قسمت catch به صورت یک استثنای عمومی‌تر همراه با افزودن یک سری اطلاعات جدید صادر می‌کنیم. اما همانطور که عنوان شد کار لاگ کردن استثناءها را بهتر است در لایه‌های بالاتر انجام دهیم.
اگر مطمئن نیستید که تمام استثناءها توسط شما مدیریت شده‌اند، می‌توانید در حالت‌های زیر، دیگر استثناءها را مدیریت کنید:
ASP.NET: می‌توانید Aplication_Error را پیاده‌سازی کنید. در اینجا فرصت خواهید داشت تا تمامی خطاهای مدیریت نشده را هندل کنید.
WinForms: استفاده از رویدادهای Application.ThreadException و AppDomain.CurrentDomain.UnhandledException
WCF: پیاده‌سازی اینترفیس IErrorHandler
ASMX: ایجاد یک Soap Extension سفارشی
ASP.NET WebAPI

چه زمان‌هایی باید یک استثناء صادر شود؟

صادر کردن یک استثناء به تنهایی کار ساده‌ایی است. تنها کافی است throw را همراه شیء exception object (exception) فراخوانی کنیم. اما سوال اینجاست که چه زمانی باید یک استثناء را صادر کنیم؟ چه داده‌هایی را باید به استثناء اضافه کنیم؟ در ادامه به این سوالات خواهیم پرداخت.

همانطور که عنوان گردید استثناءها زمانی باید صادر شوند که یک استثناء اتفاق بیفتد.

اعتبارسنجی آرگومان‌ها

ساده‌ترین مثال، آرگومان‌های مورد انتظار یک متد است:

```
public void PrintName(string name)
{
     Console.WriteLine(name);
}
```

در حالت فوق انتظار داریم مقداری برای پارامتر name تعیین شود. متد فوق با آرگومان null نیز به خوبی کار خواهد کرد؛ یعنی مقدار خروجی یک خط خالی خواهد بود. از لحاظ کدنویسی متد فوق به خوبی کار خود را انجام می‌دهد اما خروجی مورد انتظار کاربر نمایش داده نمی‌شود. در این حالت نمی‌توانیم تشخیص دهیم مشکل از کجا ناشی می‌شود.

مشکل فوق را می‌توانیم با صدور استثنای ArgumentNullException رفع کنیم:

```
public void PrintName(string name)
{
    if (name == null) throw new ArgumentNullException("name");

     Console.WriteLine(name);
}
```

خوب، name باید دارای طول ثابت و همچنین ممکن است حاوی عدد و حروف باشد:

```
public void PrintName(string name)
{
    if (name == null) throw new ArgumentNullException("name");
    if (name.Length < 5 || name.Length > 10) throw new ArgumentOutOfRangeException("name", name, "Name
must be between 5 or 10 characters long");
    if (name.Any(x => !char.IsAlphaNumeric(x)) throw new ArgumentOutOfRangeException("name", name, "May
only contain alpha numerics");

     Console.WriteLine(name);
}
```

برای حالت فوق و همچنین جلوگیری از تکرار کدهای داخل متد PrintName می‌توانید یک متد Validator برای کلاسی با نام Person ایجاد کنید.

حالت دیگر صدور استثناء، زمانی است که متدی خروجی مورد انتظارمان را نتواند تحویل دهد. یک مثال بحث‌برانگیز متدی با امضای زیر است:

```
public User GetUser(int id)
{
}
```

کاملاً مشخص است که متدی همانند متد فوق زمانیکه کاربری را پیدا نکند، مقدار null را برمی‌گرداند. اما این روش درستی است؟ خیر؛ زیرا همانطور که از نام این متد پیداست باید یک کاربر به عنوان خروجی برگردانده شود.

با استفاده از بررسی null کدهایی شبیه به این را در همه جا خواهیم داشت:

```
var user = datasource.GetUser(userId);
if (user == null)
    throw new InvalidOperationException("Failed to find user: " + userId);
// actual logic here
```

به این چنین کدهایی معمولاً The null cancer گفته می‌شود (سرطان نال!) زیرا اجازه داده‌ایم متد، خروجی null را بازگشت دهد. به جای کد فوق می‌توانیم از این روش استفاده کنیم:

```
public User GetUser(int id)
{
    if (id <= 0) throw new ArgumentOutOfRangeException("id", id, "Valid ids are from 1 and above. Do you have a parsing error somewhere?");

    var user = db.Execute<User>("WHERE Id = ?", id);
    if (user == null)
        throw new EntityNotFoundException("Failed to find user with id " + id);

    return user;
}
```

نکته‌ایی که باید به آن توجه کنید این است که در هنگام صدور یک استثناء اطلاعات کافی را نیز به آن پاس دهید. به عنوان مثال در EntityNotFoundException مثال فوق پاس دادن "Failed to find user with id " + id کار دیباگ را برای مصرف کننده، راحتر خواهد کرد.

خطاهای متداول حین کار با استثناءها

صدور مجدد استثناء و از بین بردن stacktrace

کد زیر را در نظر بگیرید:

```
try
{
    FutileAttemptToResist();
}
catch (BorgException err)
{
     _myDearLog.Error("I'm in da cube! Ohh no!", err);
    throw err;
}
```

مشکل کد فوق قسمت throw err است. این خط کد، محتویات stacktrace را از بین برده و استثناء را مجدداً برای شما ایجاد خواهد کرد. در این حالت هرگز نمی‌توانیم تشخیص دهیم که منبع خطا از کجا آمده است. در این حالت پیشنهاد می‌شود که تنها از throw استفاده شود. در این حالت استثناء اصلی مجدداً صادر گردیده و مانع حذف شدن محتویات stacktrace خواهد شد(+).

اضافه نکردن اطلاعات استثناء اصلی به استثناء جدید

یکی دیگر از خطاهای رایج اضافه نکردن استثناء اصلی حین صدور استثناء جدید است:

```
try
{
    GreaseTinMan();
}
catch (InvalidOperationException err)
{
    throw new TooScaredLion("The Lion was not in the m00d", err); //<---- استثناء اصلی بهتر است به استثناء جدید پاس داده شود
}
```

ارائه ندادن context information

در هنگام صدور یک استثناء بهتر است اطلاعات دقیقی را به آن ارسال کنیم تا دیباگ کردن آن به راحتی انجام شود. به عنوان مثال کد زیر را در نظر داشته باشید:

```
try
{
```

```
   socket.Connect("somethingawful.com", 80);
}
catch (SocketException err)
{
    throw new InvalidOperationException("Socket failed", err);
}
```

هنگامی که کد فوق با خطا مواجه شود نمی‌توان تنها با متن Socket failed تشخیص داد که مشکل از چه چیزی است. بنابراین پیشنهاد می‌شود اطلاعات کامل و در صورت امکان به صورت دقیق را به استثناء ارسال کنید. به عنوان مثال در کد زیر سعی شده است تا حد امکان context information کاملی برای استثناء ارائه شود:

```
void IncreaseStatusForUser(int userId, int newStatus)
{
    try
    {
         var user  = _repository.Get(userId);
         if (user == null)
             throw new UpdateException(string.Format("Failed to find user #{0} when trying to increase
status to {1}", userId, newStatus));

         user.Status = newStatus;
         _repository.Save(user);
    }
   catch (DataSourceException err)
   {
       var errMsg = string.Format("Failed to find modify user #{0} when trying to increase status to
{1}", userId, newStatus);
        throw new UpdateException(errMsg, err);
   }
```

نحوه‌ی طراحی استثناءها

برای ایجاد یک استثناء سفارشی می‌توانید از کلاس Exception ارث‌بری کنید و چهار سازنده‌ی آن را اضافه کنید:

```
public NewException()
public NewException(string description )
public NewException(string description, Exception inner)
protected or private NewException(SerializationInfo info, StreamingContext context)
```

سازنده اول به عنوان default constructor شناخته می‌شود. اما پیشنهاد می‌شود که از آن استفاده نکنید، زیرا یک استثناء بدون context information از ارزش کمی برخوردار خواهد بود.

سازنده‌ی دوم برای تعیین description بوده و همانطور که عنوان شد ارائه دادن context information از اهمیت بالایی برخوردار است. به عنوان مثال فرض کنید استثناء KeyNotFoundException که توسط کلاس Dictionary صادر شده است را دریافت کرده‌اید. این استثناء زمانی صادر خواهد شد که بخواهید به عنصری که درون دیکشنری پیدا نشده است دسترسی داشته باشید. در این حالت پیام زیر را دریافت خواهید کرد:

```
“The given key was not present in the dictionary.”
```

حالا فرض کنید اگر پیام به صورت زیر باشد چقدر باعث خوانایی و عیب‌یابی ساده‌تر خطا خواهد شد:

```
“The key ‘abrakadabra’ was not present in the dictionary.”
```

در نتیجه تا حد امکان سعی کنید که context information شما کاملتر باشد.

سازنده‌ی سوم شبیه به سازنده‌ی قبلی عمل می‌کند با این تفاوت که توسط پارامتر دوم می‌توانیم یک استثناء دیگر را catch کرده یک استثناء جدید صادر کنیم.

سازنده‌ی سوم زمانی مورد استفاده قرار می‌گیرد که بخواهید از Serialization پشتیبانی کنید (به عنوان مثال ذخیره‌ی استثناءها درون فایل و...)

خوب، برای یک استثناء سفارشی حداقل باید کدهای زیر را داشته باشیم:

```
public class SampleException : Exception
{
    public SampleException(string description)
        : base(description)
    {
        if (description == null) throw new ArgumentNullException("description");
    }

    public SampleException(string description, Exception inner)
        : base(description, inner)
    {
        if (description == null) throw new ArgumentNullException("description");
        if (inner == null) throw new ArgumentNullException("inner");
    }

    public SampleException(SerializationInfo info, StreamingContext context)
        : base(info, context)
    {
    }
}
```

اجباری کردن ارائه‌ی Context information:

برای اجباری کردن context information کافی است یک فیلد اجباری درون سازنده تعریف کنیم. برای مثال اگر بخواهیم کاربر HTTP status code را برای استثناء ارائه دهد باید سازنده‌ها را اینگونه تعریف کنیم:

```
public class HttpException : Exception
{
    System.Net.HttpStatusCode _statusCode;

    public HttpException(System.Net.HttpStatusCode statusCode, string description)
        : base(description)
    {
        if (description == null) throw new ArgumentNullException("description");
        _statusCode = statusCode;
    }

    public HttpException(System.Net.HttpStatusCode statusCode, string description, Exception inner)
        : base(description, inner)
    {
        if (description == null) throw new ArgumentNullException("description");
        if (inner == null) throw new ArgumentNullException("inner");
        _statusCode = statusCode;
    }

    public HttpException(SerializationInfo info, StreamingContext context)
        : base(info, context)
    {
    }

    public System.Net.HttpStatusCode StatusCode { get; private set; }

}
```

همچنین بهتر است پراپرتی Message را برای نمایش پیام مناسب بازنویسی کنید:

```
public override string Message
{
        get { return base.Message + "\r\nStatus code: " + StatusCode; }
}
```

مورد دیگری که باید در کد فوق مد نظر داشت این است که status code قابلیت سریالایز شدن را ندارد. بنابراین باید متد GetObjectData را برای سریالایز کردن بازنویسی کنیم:

```
public class HttpException : Exception
{
    // [...]

    public HttpException(SerializationInfo info, StreamingContext context)
        : base(info, context)
    {
```

```
        // this is new
        StatusCode = (HttpStatusCode) info.GetInt32("HttpStatusCode");
    }

    public HttpStatusCode StatusCode { get; private set; }

    public override string Message
    {
        get { return base.Message + "\r\nStatus code: " + StatusCode; }
    }

    // this is new
    public override void GetObjectData(SerializationInfo info, StreamingContext context)
    {
        base.GetObjectData(info, context);
        info.AddValue("HttpStatusCode", (int) StatusCode);
    }
}
```

در اینحالت فیلدهای اضافی در طول فرآیند Serialization به خوبی سریالایز خواهند شد.

در حین صدور استثناءها همیشه باید در نظر داشته باشیم که چه نوع context information را می‌توان ارائه داد، این مورد در یافتن راه‌حل خیلی کمک خواهد کرد.

طراحی پیام‌های مناسب

پیام‌های exception مختص به توسعه‌دهندگان است نه کاربران نهایی.
نوشتن این نوع پیام‌ها برای برنامه‌نویس کار خسته‌کننده‌ایی است. برای مثال دو مورد زیر را در نظر داشته باشید:

```
throw new Exception("Unknown FaileType");
throw new Exception("Unecpected workingDirectory");
```

این نوع پیام‌ها حتی اگر از لحاظ نوشتاری مشکلی نداشته باشند یافتن راه‌حل را خیلی سخت خواهند کرد. اگر در زمان برنامه‌نویسی با این نوع خطاها روبرو شوید ممکن است با استفاده از debugger ورودی نامعتبر را پیدا کنید. اما در یک برنامه و خارج از محیط برنامه‌نویسی، یافتن علت بروز خطا خیلی سخت خواهد بود.
توسعه‌دهندگانی که exception message را در اولویت قرار می‌دهند، معتقد هستند که از لحاظ تجربه‌ی کاربری پیام‌ها تا حد امکان باید فاقد اطلاعات فنی باشد. همچنین همانطور که پیش‌تر عنوان گردید این نوع پیام‌ها همیشه باید در بالاترین سطح نمایش داده شوند نه در لایه‌های زیرین. همچنین پیام‌هایی مانند Unknown FaileType نه برای کاربر نهایی، بلکه برای برنامه‌نویس نیز ارزش چندانی ندارد زیرا فاقد اطلاعات کافی برای یافتن مشکل است.
در طراحی پیام‌ها باید موارد زیر را در نظر داشته باشیم:
- امنیت:
یکی از مواردی که از اهمیت بالایی برخوردار است مسئله امنیت است از این جهت که پیام‌ها باید فاقد مقادیر runtime باشند. زیرا ممکن است اطلاعاتی را در خصوص نحوه‌ی عملکرد سیستم آشکار سازند.
- زبان:
همانطور که عنوان گردید پیام‌های استثناء برای کاربران نهایی نیستند، زیرا کاربران نهایی ممکن است اشخاص فنی نباشند، یا ممکن است زبان آنها انگلیسی نباشد. اگر مخاطبین شما آلمانی باشند چطور؟ آیا تمامی پیام‌ها را با زبان آلمانی خواهید نوشت؟ اگر هم اینکار را انجام دهید تکلیف استثناءهایی که توسط Base Class Library و دیگر کتابخانه‌های thirt-party صادر می‌شوند چیست؟ اینها انگلیسی هستند.

در تمامی حالت‌هایی که عنوان شد فرض بر این است که شما در حال نوشتن این نوع پیام‌ها برای یک سیستم خاص هستید. اما اگر هدف نوشتن یک کتابخانه باشد چطور؟ در این حالت نمی‌دانید که کتابخانه‌ی شما در کجا استفاده می‌شود.
اگر هدف نوشتن یک کتابخانه نباشد این نوع پیام‌هایی که برای کاربران نهایی باشند، وابستگی‌ها را در سیستم افزایش خواهند داد، زیرا در این حالت پیام‌ها به یک رابط کاربری خاص گره خواهند خورد.

خب اگر پیام‌ها برای کاربران نهایی نیستند، پس برای کسانی مورد استفاده قرار خواهند گرفت؟ در واقع این نوع پیام می‌تواند به

عنوان یک documentation برای سیستم شما باشند.

فرض کنید در حال استفاده از یک کتابخانه جدید هستید به نظر شما کدام یک از پیام‌های زیر مناسب هستند:

```
"Unecpected workingDirectory"
```

یا:

```
"You tried to provide a working directory string that doesn't represent a working directory. It's not your fault, because it wasn't possible to design the FileStore class in such a way that this is a statically typed pre-condition, but please supply a valid path to an existing directory.

"The invalid value was: "fllobdedy"."
```

یافتن مشکل در پیام اول خیلی سخت خواهد بود زیرا فاقد اطلاعات کافی برای یافتن مشکل است. اما پیام دوم مشکل را به صورت کامل توضیح داده است. در حالت اول شما قطعاً نیاز خواهید داشت تا از دیباگر برای یافتن مشکل استفاده کنید. اما در حالت دوم پیام به خوبی شما را برای یافتن راه‌حل راهنمایی می‌کند.

همیشه برای نوشتن پیام‌های مناسب سعی کنید از لحاظ نوشتاری متن شما مشکلی نداشته باشد، اطلاعات کافی را درون پیام اضافه کنید و تا حد امکان نحوه‌ی رفع مشکل را توضیح دهید

عنوان: **تبدیل بلوک‌های یونیکد در زیرنویس برای نمایش در تلویزیون‌ها و پلیرها**
نویسنده: علی یگانه مقدم
تاریخ: ۱۳۹۴/۰۱/۰۱ ۱۰:۳۵
آدرس: www.dotnettips.info
گروه‌ها: C#, Regular expressions, Subtitle, Unicode

مقدمه

موقعی که سینمای ناطق کار خود را آغاز کرد، بسیاری از مردم از آن استقبال کردند و بسیاری از سینماگران که این استقبال را دیدند، رفته رفته به سمت سینمای ناطق کشیده شدند. ولی در این بین یک مشکلی ایجاد شده بود؛ اینکه ناشنوایان دیگر مانند قدیم یعنی دوران صامت نمی‌توانستند فیلم‌ها را تماشا کنند، پس نیاز بود این مشکل به نحوی رفع شود. از اینجا بود که ایده‌ی زیرنویس شکل گرفت و این مشکل را رفع نمود. بعدها فیلم‌ها انتقال دهنده‌ی فرهنگ و پیوند دهنده‌ی مردم با فرهنگ‌های مختلف شدند ولی تفاوت در زبان باعث می‌شد که این امر به خوبی صورت نگیرد. به همین علت زیرنویس، وظیفه‌ی دیگری را هم پیدا کرد و آن رساندن پیام فیلم با زبان خود مخاطب بود. امروزه تهیه‌ی زیرنویس‌ها توسط بسیاری از افراد که با زبان انگلیسی (آشنایی با یک زبان میانی برای ترجمه زیرنویس) آشنایی دارند رواج پیدا کرده و روزانه نزدیک به صد زیرنویس یا گاها بیشتر با زبان‌های مختلف بر روی اینترنت قرار می‌گیرند. بزرگترین سایتی که در حال حاضر با شهرت جهانی در این زمینه فعالیت دارد سایت subscene.com است.

آشنایی با انواع زیرنویس‌ها

زیرنویس‌ها فرمت‌های مختلفی دارند مانند srt,sub idx,smi و ... ولی در حال حاضر معروف‌ترین و معتبرترین فرمت در بین همه‌ی فرمت‌ها Subrip با پسوند SRT می‌باشد که قالب متنی به صورت زیر دارد:

```
203
00:16:38,731 --> 00:16:41,325
<i>Happy Christmas, your arse
I pray God it's our last</i>
```

که باعث میشود حجم بسیار کمی در حد چند کیلوبایت داشته باشد.

بررسی مشکل ما با زیرنویس در تلویزیون‌ها

یکی از مشکلاتی که ما در اجرای زیرنویس‌ها بر روی تلویزیون‌ها داریم این است که حروف فارسی را به خوبی نمی‌شناسند و در هنگام نمایش با مشکل مواجه می‌شوند که البته در اکثر مواقع با تبدیل زیرنویس از ANSI به Unicode یا UTF-8 مشکل حل می‌شود. ولی در بعضی مواقع تلویزیون یا پلیرها از پشتیبانی زبان فارسی سرباز می‌زنند و زیرنویس را به شکل زیر نمایش می‌دهند.

سلام = م ا ل س

به این جهت ما از یک برنامه به اسم srttouni استفاده می‌کنیم که با استفاده یک روش جایگزینی و معکوس سازی، مشکل ما را حل می‌کند. ولی باز هم این برنامه مشکلاتی دارد و از آنجا که برنامه نویس این برنامه که واقعا کمال تشکر را از ایشان، دارم مشخص نیست، مجبور شدم به جای گزارش، خودم این مشکلات را حل کنم. *مشکلات این برنامه :*

عدم حذف تگ‌ها ، گاها برنامه نویس‌ها از تگ هایی چون Bold,italic,underline,color استفاده می‌کنند که معدود برنامه‌هایی آن را پشتیبانی کرده و تلویزیون و پلیرها هم که اصلا پشتیبانی نمی‌کنند و باعث میشود که متن روی تلویزیون مثل کد html ظاهر شود بعضی جملات دوبار روی صفحه ظاهر می‌شوند.

تنها یک فایل را در هر زمان تبدیل می‌کند. مثلا اگر یک سریال چند قسمته داشته باشید، برای هر قسمت باید زیرنویس را انتخاب کرده و تبدیل کنید، در صورتی که میتوان دستور داد تمام زیرنویس‌های داخل دایرکتوری را تبدیل کرد یا چند زیرنویس را برای این منظور انتخاب کرد.

نحوه‌ی خواندن زیرنویس با کدنویسی

با تشکر از دوست عزیز ما در این صفحه می‌توان گفت یک کد تقریبا خوب و جامعی را برای خواندن این قالب داریم. بار دیگر نگاهی به قالب یک دیالوگ در زیرنویس می‌اندازیم و آن را بررسی می‌کنیم:

```
203
00:16:38,731 --> 00:16:41,325
<i>Happy Christmas, your arse
I pray God it's our last</i>
```

اولین خط شامل شماره‌ی خط است که از یک آغاز می‌گردد تا به تعداد دیالوگ‌ها، خط دوم، زمان آغاز و پایان دیالوگ مورد نظر است، موقعی که دیالوگ روی صفحه ظاهر میشود تا موقعی که دیالوگ از روی صفحه محو شود که به ترتیب بر اساس ساعت:دقیقه:ثانیه و میلی ثانیه می‌باشد. خطوط بعدی هم متن دیالوگ است است و بعد از پایان متن دیالوگ یک خط خالی زیر آن قرار می‌گیرد تا نشان دهد این دیالوگ به پایان رسیده است. اگر همین خط خالی حذف گردد برنامه‌هایی چون Media player classic خطهای زیری را جز متن دیالوگ قبلی به حساب می‌آورند و شماره خط و زمان بندی دیالوگ بعدی به عنوان متن روی صفحه ظاهر می‌گردند و بعضی playerها هم قاطی کرده و کلا زیرنویس را نمی‌خوانند یا اون خط رو نشون نمیدن مثل Kmplayer و هر کدام رفتار خاص خودشان را بروز می‌دهند.

کد زیر در کلاس SubRipServices وظیفه‌ی خواندن محتوای فایل srt را بر اساس عبارتی که دادیم دارد:

```
private readonly static Regex regex_srt = new
Regex(@"(?<sequence>\d+)\r\n(?<start>\d{2}\:\d{2}\:\d{2},\d{3}) --\> " +
            @"(?<end>\d{2}\:\d{2}\:\d{2},\d{3})\r\n(?<text>[\s\S]*?)\r\n\r\n", RegexOptions.Compiled);

 public string ToUnicode(string lines)
        {

        string subtitle= regex_srt.Replace(lines,delegate(Match m)
             {
                 string text = m.Groups["text"].Value;
                 //1.remove tags
                 text = CleanScriptTags(text);

                 //2.replace letters
                 PersianReshape reshaper = new PersianReshape();
                 text = reshaper.reshape(text);
                 string[] splitedlines = text.Split(new string[] { Environment.NewLine },
StringSplitOptions.None);
                 text = "";
                 foreach (string line in splitedlines)
                 {
                     //3.reverse tags
                     text += ReverseText(reshaper.reshape(line))+Environment.NewLine ;
                 }
                 return
                     string.Format("{0}\r\n{1} --> {2}\r\n", m.Groups["sequence"],
m.Groups["start"].Value,
                         m.Groups["end"]) + text + Environment.NewLine+Environment.NewLine ;
             }
            );

            return subtitle;
        }
```

در اولین خط ما یک Regular Expersion یا یک عبارت با قاعده تعریف کردیم که در اینجا میتوانید با خصوصیات آن آشنا شوید. ما برای این کلاس یک الگو ایجاد کردیم و بر حسب این الگو، متن یک زیرنویس را خواهد گشت و خطوطی را که با این تعریف جور در می‌آیند و معتبر هستند، برای ما باز می‌گرداند.

عبارتهایی که به صورت <name>? تعریف شده‌اند در واقع یک نامگذاری برای هر قسمت از الگوی ما هستند تا بعدا این امکان برای ما فراهم شود که خطوط برگشتی را تجزیه کنیم که مثلا فقط قسمت متن را دریافت کنیم، یا فقط قسمت زمان شروع یا پایان را دریافت کنیم و ...

متد tounicode یک آرگومان متنی دارد (lines) که شامل محتویات فایل زیرنویس است. متد Replace در شی regex_srt با هر بار پیدا کردن یک متن بر اساس الگو در رشته lines دلیگیتی را فرا می‌خواند که در اولین پارامتر آن که از نوع matchEvaluator است، شامل اطلاعات متنی است که بر اساس الگو، یافت شده است. خروجی آن از نوع string می‌باشد که با متن پیدا شده بر اساس الگو جابجا خواهد کرد و در نهایت بعد از چندین بار اجرا شدن، کل متن‌های تعویض شده، به داخل متغیر subtitle ارسال خواهند شد.

کاری که ما در اینجا می‌کنیم این است که هر دیالوگ داخل زیرنویس را بر اساس الگو، یافته و متن آن را تغییر داده و متن جدید را جایگزین متن قبلی می‌کنیم. اگر زیرنویس ما 800 دیالوگ داشته باشد این دلیگیت 800 مرتبه اجرا خواهد شد.

از آنجا که ما تنها می‌خواهیم متن زیرنویس را تغییر دهیم، در اولین خط فرامین این دلیگیت تعریف شده، متن مورد نظر را بر اساس همان گروه‌هایی که تعریف کرده‌ایم دریافت می‌کنیم و در متغیر text قرار می‌دهیم:

```
m.Groups["text"].Value
```

در مرحله‌ی بعدی ما اولین مشکلمان (حذف تگ‌ها) را با تابعی به اسم CleanScriptTags برطرف میکنیم که کد آن به شرح زیر است:

```
private static readonly Regex regex_tags = new Regex("<.*?>", RegexOptions.Compiled);
 private  string CleanScriptTags(string html)
        {
            return regex_tags.Replace(html, string.Empty);
        }
```

کد بالا از یک regular Expression دیگر جهت پیدا کردن تگ‌ها استفاده می‌کند و به جای آن‌ها عبارت "" را جایگزین می‌کند. این کد قبلا در سایت جاری در این صفحه توضیح داده شده است. خروجی این تابع را مجددا در text قرار می‌دهیم و به مرحله‌ی دوم، یعنی تعویض کاراکترها می‌رویم:

```
PersianReshape reshaper = new PersianReshape();
                text = reshaper.reshape(text);
                string[] splitedlines = text.Split(new string[] { Environment.NewLine },
StringSplitOptions.None);
                text = "";
                foreach (string line in splitedlines)
                {
                    //3.reverse tags
                    text += ReverseText(reshaper.reshape(line))+Environment.NewLine ;
                }
```

برای اینکه دقیقا متوجه شویم قرار است چکاری انجام شود بیاید دو گروه یا بلوک مختلف در یونیکد را بررسی کنیم. هر بلوک کد در یونیکد شامل محدوده‌ای از کد پوینت هاست که نامی منحصرفرد برای خود دارد و هیچ کدام از کدپوینت‌ها در هر بلوک یا گروه، اشتراکی با بقیه‌ی بلوک‌ها ندارد. سایت codetable از آن دست سایت‌هایی است که اطلاعات خوبی در مورد کدهای یونیکد دارد. در قسمت Unicode Groups دو گروه برای زبان عربی وجود دارند که در جدول این گروه، هر سطر آن یکی از کدها را به صورت دسیمال، هگزا دسیمال و نام و نماد آن، نمایش می‌دهد. ^

^ ,

```
Arabic Presentation Forms-A
```

^

```
,
```

^

```
Arabic Presentation Forms-B
```

بلوک اول طبق گفته‌ی ویکی پدیا دسته‌ی متنوعی از حروف مورد نیاز برای زبان فارسی، اردو، پاکستانی و تعدادی از زبان‌های آسیای مرکزی است.

بلوک دوم شامل نمادها و نشانه‌های زبان عربی است و در حال حاضر برای کد کردن استفاده نمی‌شوند و دلیل حضور آن برای سازگاری با سیستم‌های قدیمی است.

اگر خوب به مشکلی که در بالا برای زیرنویس‌ها اشاره کردیم دقت کنید، گفتیم حروف از هم جدا نشان داده می‌شوند و اگر به بلوک دوم در لینک‌های داده شده نگاه کنید می‌بینید که حروف متصل را داراست. یعنی برای حرف س 4 حرف یا کدپوینت داراست : ﺳ برای کلماتی مثل *سبد* ، ﺴ برای کلماتی مثل *شانس* ، ﺳ برای کلماتی مثل *بسیار* ، ولی خود س برای کلمات غیر متصل مثل *ناس* ، البته بعضی حروف یک یا دو حالت می‌طلبند مثل **د** ، **ر** که فقط دو حالت ﺪ **و د** ، ﺮ **و ر** را دارند یا مثل آ که یک حالت دارد.

من قبلا یک کلاس به نام lettersTable ایجاد کرده بودم (و دیگر نوشتن آن را ادامه ندادم) که برای هر حرف، یک آیتم در شیءایی از نوع dictionary ساخته بودم و هر کدپوینت بلوک اول را در آن کلید و کد متقابلش را در بلوک دوم، به صورت مقدار ذخیره کرده بودم (گفتیم که هر نماد در بلوک اول، برابر با 4 نماد در بلوک دوم است؛ ولی ما در دیکشنری تنها مقدار اول را ذخیره می‌کنیم. زیرا کد بقیه نمادها دقیقا پشت سر یکدیگر قرار گرفته‌اند که می‌توان با یک جمع ساده از عدد 0 تا 3، به مقدار هر

کدام از نمادها رسید. البته ناگفته نماند بعضی نمادها 2 عدد بودند که این هم باید بررسی شود). برای همین هر کاراکتر را با کاراکتر قبل و بعد می‌گرفتم و بررسی می‌کردم و از یک جدول دیکشنری دیگر هم به اسم specialchars هم استفاده کردم تا آن کاراکترهایی که تنها دو نماد یا یک نماد را دارند، بررسی کنم و این کاراکترها همان کاراکترهایی بودند که اگر قبل یک حرف هم بیایند، حرف بعدی به آن‌ها نمی‌چسبد. برای درک بهتر، این عبارت مثال زیر را برای حرف س در نظر بگیرید:

مستطیل = چون بین هر دو طرف س حر وجود دارد قطعا باید شکل س به صورت ـسـ انتخاب شود ، حالا مثال زیر را در نظر بگیرید:

دست = دـسـت که اشتباه است و باید باشد دست یعنی شکل سـ باید صدا زده شود، پس این مورد هم باید لحاظ شود.

نمونه‌ای از کد این کلاس:

```
Dictionary<int ,int>  letters=new Dictionary<int, int>();

   //0=0x0 ,1=1x0 ,2=0x1 ,3=1x1
        private void FillPrimaryTable()
        {
            //آ
            letters.Add(1570, 65153);
            //ا
            letters.Add(1575, 65166);
            //أ
            letters.Add(1571, 65155);
            //ب
            letters.Add(1576, 65167);
            //ت
            letters.Add(1578, 65173);
            //ث
            letters.Add(1579, 65177);
            //ج
            letters.Add(1580, 65181);
.....
}

Dictionary<int,byte> specialchars=new Dictionary<int, byte>();

  private void SetSpecialChars()
        {
            //آ
            specialchars.Add(1570, 0);
            //ا
            specialchars.Add(1575, 0);
            //د2
            specialchars.Add(1583, 1);
            //ذ2
            specialchars.Add(1584, 1);
            //ر2
            specialchars.Add(1585, 1);
            //ز2
            specialchars.Add(1586, 1);
            //ژ
            specialchars.Add(1688, 1);
            //و2
            specialchars.Add(1608, 1);
            //أ
            specialchars.Add(1571, 1);

        }
```

کلاس بالا تنها برای ذخیره‌ی کدپوینت‌ها بود، ولی یک کلاس دیگر هم به اسم lettersCrawler نوشته بودم که متد آن وظیفه‌ی تبدیل را به عهده داشت.

در آن متد هر بار یک حرف را انتخاب می‌کرد و حرف قبلی و بعدی آن را ارسال می‌کرد تا تابع CalculateIncrease آن را محاسبه کرده و کاراکتر نهایی را باز گرداند و به متغیر finalText اضافه می‌کرد. ولی در حین نوشتن، زمانی را به یاد آوردم که اندروید به تازگی آمده بود و هنوز در آن زمان از زبان فارسی پشتیبانی نمی‌کرد و حروف برنامه‌هایی که می‌نوشتیم به صورت جدا از هم بود و همین مشکل را داشت که ما این مشکل را با استفاده از یک کلاس جاوا که دوست عزیزی آن را در اینجا به اشتراک گذاشته بود، حل می‌کردیم. پس به این صورت بود که از ادامه‌ی نوشتن کلاس انصراف دادم و از یک کلاس دقیق‌تر و آماده استفاده کردم.

در واقع این کلاس همین کار بالا را با روشی بهتر انجام می‌دهد. همه‌ی نمادها به طور دقیق‌تری کنترل می‌شوند حتی تنوین‌ها و دیگر علائم، همه نمادها با کدهای متناظر در یک آرایه ذخیره شده‌اند که ما در بالا از نوع Dictionary استفاده کرده بودیم.

تنها کاری که نیاز بود، باید این کد به سی شارپ تبدیل میشد و از آنجایی که این دو زبان خیلی شبیه به هم هستند، حدود ده دقیقه‌ای برای ویرایش کد وقت برد که می‌توانید کلاس نهایی را از اینجا دریافت کنید.
پس خط زیر در متد ToUnicode کار تبدیل اصلی را صورت می‌دهد:

```
PersianReshape reshaper = new PersianReshape();
                text = reshaper.reshape(text);
```

بنابراین مرحله‌ی دوم انجام شد. این تبدیل در بسیاری از سیستم‌ها همانند اندروید کافی است؛ ولی ما گفتیم که تلویزیون یا پلیر به غیر از جدا جدا نشان دادن حروف، آن‌ها را معکوس هم نشان می‌دهند. پس باید در مرحله‌ی بعد آن‌ها را معکوس کنیم که اینکار با خط زیر و صدا زدن تابع ReverseText انجام میگیرد

```
//3.reverse tags
                text = ReverseText(text);
```

از آنجا که یک دیالوگ ممکن است چند خطی باشد، این معکوس سازی برای ما دردسر می‌شد و ترتیب خطوط هم معکوس می‌شد. پس ما با استفاده از کد زیر هر یک خط را شکسته و هر کدام را جداگانه معکوس می‌کنیم و سپس به یکدیگر می‌چسبانیم:

```
string[] splitedlines = text.Split(new string[] { Environment.NewLine }, StringSplitOptions.None);
                text = "";
                foreach (string line in splitedlines)
                {
                    //3.reverse tags
                    text += ReverseText(reshaper.reshape(line))+Environment.NewLine ;
                }
```

همه‌ی ما معکوس سازی یک رشته را بلدیم، یکی از روش‌ها این است که رشته را خانه به خانه از آخر به اول با یک for بخوانیم یا اینکه رشته را به آرایه‌ای از کارکاکترها، تبدیل کنیم و سپس با Array.Reverse آن را معکوس کرده و خانه به خانه به سمت جلو بخوانیم و خیلی از روش‌های دیگر. ولی این معکوس سازی‌ها برای ما یک عیب هم دارد و این هست که این معکوس سازی روی نمادهایی چون . یا ! و غیره که در ابتدا و انتهای رشته آمده‌اند و حروف انگلیسی، نباید اتفاق بیفتند. پس می‌بینیم که تابع معکوس سازی هم باز باید ویژه‌تر باشد. ابتدا قسمت‌های ابتدا و انتها را جدا کرده و از آن حذف می‌کنیم. سپس رشته را معکوس می‌کنیم. ولی ممکن هست و احتمال دارد که بین حروف فارسی هم حروف انگلیسی یا اعداد به کار رود که آن‌ها هم معکوس می‌شوند. برای همین بعد از معکوس سازی یکبار هم باید آن‌ها را با یک عبارت با قاعده یافته و سپس هر کدام را جداگانه معکوس کرده و سپس مثل روش بالا Replace کنیم و رشته‌های جدا شده را به ابتدا و انتهای آن، سر جای قبلیشان می‌چسبانیم.
این دو تابع برای معکوس کردن عادی یک رشته به کار می‌روند:

```
private string Reverse(string text)
        {
            return Reverse(text,0,text.Length);
        }

        private string Reverse(string text,int start,int end)
        {
            if (end < start)
                return text;
            string reverseText = "";

            for (int i = end-1; i >=start; i--)
            {
                reverseText += text[i];
            }
            return reverseText;
        }
```

ولی این تابع ReverseText جمعی از عملیات معکوس سازی ویژه‌ی ماست؛ مرحله اول، مرحله دریافت و ذخیره‌ی حروف خاص در ابتدای رشته به اسم پیشوند prefix است:

```
private string ReverseText(string text)
        {
```

```
            char[] chararray = text.ToCharArray();
            string reverseText = "";
            bool prefixcomp = false;
            bool postfixcomp = false;
            string prefix = "";
            string postfix = "";

            #region get prefix symbols
            for (int i = 0; i < chararray.Length; i++)
            {
                if (!prefixcomp)
                {
                    char ch =(char) chararray.GetValue(i) ;
                    if (ch< 130)
                    {
                        prefix += chararray.GetValue(i);
                    }
                    else
                    {
                        prefixcomp = true;
                        break;
                    }
                }
            }
            #endregion
}
```

مرحله‌ی دوم هم دریافت و ذخیره‌ی حروف خاص در انتهای رشته به اسم پسوند postfix است که به این تابع اضافه می‌کنیم:

```
#region get postfix symbols
            for (int i = chararray.Length - 1; i >-1 ; i--)
            {
                if (!postfixcomp && prefix.Length!=text.Length)
                {
                    char ch = (char)chararray.GetValue(i);
                    if (ch < 130)
                    {
                        postfix += chararray.GetValue(i);
                    }
                    else
                    {
                        postfixcomp = true;
                        break;
                    }
                }
            }
            #endregion
```

مرحله‌ی سوم عملیات معکوس سازی روی رشته است و سپس با استفاده از یک Regular Expression حروف انگلیسی و اعداد بین حروف فارسی را یافته و یک معکوس سازی هم روی آن‌ها انجام می‌دهیم تا به حالت اولشان برگردند. کل عملیات معکوس سازی در اینجا به پایان می‌رسد:

```
#region reverse text

            reverseText = Reverse(text, prefix.Length, text.Length-postfix.Length);


            reverseText = unTagetdLettersRegex.Replace(reverseText, delegate(Match m)
            {
                return Reverse(m.Value);
            });
            #endregion
```

تعریف عبارت با قاعده‌ی بالا به اسم unTargetedLetters:

```
private static readonly Regex unTagetdLettersRegex = new Regex(@"[A-Za-z0-9]+", RegexOptions.Compiled);
```

آخر سر هم رشته را به‌علاوه پیشوند و پسوند جدا شده بر می‌گردانیم:

```
return prefix+ reverseText+postfix;
```

کد کامل تابع بدین شکل در می‌آید:

```
private static readonly Regex unTagetdLettersRegex = new Regex(@"[A-Za-z0-9]+", RegexOptions.Compiled);
private string ReverseText(string text)
        {
            char[] chararray = text.ToCharArray();
            string reverseText = "";
            bool prefixcomp = false;
            bool postfixcomp = false;
            string prefix = "";
            string postfix = "";

            #region get prefix symbols
            for (int i = 0; i < chararray.Length; i++)
            {
                if (!prefixcomp)
                {
                    char ch =(char) chararray.GetValue(i) ;
                    if (ch< 130)
                    {
                        prefix += chararray.GetValue(i);
                    }
                    else
                    {
                        prefixcomp = true;
                        break;
                    }
                }
            }
            #endregion

            #region get postfix symbols
            for (int i = chararray.Length - 1; i >-1 ; i--)
            {
                if (!postfixcomp && prefix.Length!=text.Length)
                {
                    char ch = (char)chararray.GetValue(i);
                    if (ch < 130)
                    {
                        postfix += chararray.GetValue(i);
                    }
                    else
                    {
                        postfixcomp = true;
                        break;
                    }
                }
            }
            #endregion

            #region reverse text

            reverseText = Reverse(text, prefix.Length, text.Length-postfix.Length);


            reverseText = unTagetdLettersRegex.Replace(reverseText, delegate(Match m)
            {
                return Reverse(m.Value);
            });
            #endregion



            return prefix+ reverseText+postfix;
        }
```

در نهایت، خط آخر دلیگت همه چیز را طبق فرمت یک دیالوگ srt چینش کرده و بر می‌گردانیم.

```
return
                      string.Format("{0}\r\n{1} --> {2}\r\n", m.Groups["sequence"],
m.Groups["start"].Value,
                        m.Groups["end"]) + text + Environment.NewLine+Environment.NewLine ;
```

رشته subtitle را به صورت srt ذخیره کرده و انکودینگ را هم Unicode انتخاب کنید و تمام.

نمایی از برنامه‌ی نهایی

اجرای زیرنویس تبدیل شده روی کامپیوتر

روی پلیر یا تلویزیون

نکته‌ی نهایی: هنگام تست زیرنویس روی فیلم متوجه شدم پلیر خطوط بلند را که در صفحه‌ی نمایش جا نمی‌شود، می‌شکند و به دو خط تقسیم می‌کند. ولی نکته‌ی خنده دار اینجا بود که خط اول را پایین می‌اندازد و خط دوم را بالا. برای همین این تکه کد را نوشتم و به طور جداگانه در گیت هاب هم قرار داده‌ام.

این تکه کد را هم بعد از

```
//1.remove tags
                text = CleanScriptTags(text);
```

به برنامه اضافه می‌کنیم:

```
text =StringUtils.ConvertToMultiLine(text);
```

از این پس خطوط به طولی بین 30 کاراکتر تا چهل کاراکتر شکسته خواهند شد و مشکل خطوط بلند هم نخواهیم داشت.
کد متد ConvertToMultiline:

```
namespace Utils
{
```

```
    public static class StringUtils
    {
        public static string ConvertToMultiLine(String text, int min = 30, int max = 40)
        {
            if (text.Trim() == "")
                return text;

            string[] words = text.Split(new string[] { " " }, StringSplitOptions.None);

            string text1 = "";
            string text2 = "";
            foreach (string w in words)
            {
                if (text1.Length < min)
                {
                    if (text1.Length == 0)
                    {
                        text1 = w;
                        continue;
                    }

                    if (w.Length + text1.Length <= max)
                        text1 += " " + w;
                }
                else
                    text2 += w + " ";

            }
            text1 = text1.Trim();
            text2 = text2.Trim();
            if (text2.Length > 0)
            {
                text1 += Environment.NewLine + ConvertToMultiLine(text2, min, max);
            }
            return text1;
        }

    }
}
```

آرگومان‌های min و max که به طور پیش فرض 30 و 40 هستند، سعی می‌کنند که هر خط را در نهایت به طور حدودی بین 30 تا 40 کاراکتر نگه دارند.

نکته پایانی : خوشحال میشم دوستان در این پروژه مشارکت داشته باشند و اگر جایی نیاز به اصلاح، بهبود یا ایجاد امکانی جدید دارد کمک حال باشند و سعی کنند تا آنجا که می‌شود برنامه را روی net frame work 2. نگه دارند و بالاتر نبرند. چون استفاده کننده‌های این برنامه کاربران عادی و گاها با دانش پایین هستند و خیلی از آن‌ها هنوز از ویندوز xp استفاده می‌کنند تا در اجرای برنامه خیلی دچار مشکل نشده و راحت برای بسیاری از آن‌ها اجرا شود.

برنامه مورد نظر را به طور کامل می‌توانید از اینجا یا اینجا به صورت فایل نهایی و هم سورس دریافت کنید.

نظرات خوانندگان

نویسنده: وحید نصیری
تاریخ: ۱۳۹۴/۰۱/۰۱ ۱۲:۶

- در فایل‌های PDF هم این چرخاندن حروف برای نمایش صحیح متون فارسی باید انجام شود. در مطلب « استخراج متن از فایل‌های PDF توسط iTextSharp » در انتهای بحث آن، کلاسی بر اساس API ویندوز البته، برای اصلاح این جایگذاری ارائه شده‌است. شاید در این پروژه هم کاربرد داشته باشد. البته در این حالت پروژه تنها در ویندوز قابل اجرا خواهد بود. یا نمونه‌ی دیگر آن فایل bidi.js موزیلا است که در پروژه‌ی PDF آن استفاده شده‌است.
- در یک سری پلیرها به نظر وجود BOM برای خواندن زیرنویس فارسی اجباری است؛ وگرنه فایل را یونیکد تشخیص نمی‌دهند.
- در حین ذخیره سازی از Encoding.Unicode استفاده کرده‌اید (UTF 16 هست در دات نت). شاید Encoding.UTF8 را هم آزمایش کنید، مفید باشد. حجم UTF 16 نسبت به UTF 8 نزدیک به دو برابر است و شاید بعضی پخش کننده‌ها با آن مشکل داشته باشند.
- به روز رسانی نرم افزار و firmware دستگاه هم در بسیاری از اوقات مفید است؛ خصوصا برای رفع مشکلات یونیکد آن‌ها.

نویسنده: علی یگانه مقدم
تاریخ: ۱۳۹۴/۰۱/۰۱ ۱۵:۸

در مورد انکودینگ طبق گفته شما اون رو به UTF-8 تغییر دادم و دستگاه هم نمایش داد. برنامه رو هم به روز کردم و گستره شکستن جمله رو هم از 40 کاراکتر تا 50 کاراکتر تغییر دادم . چون فکر کنم قبلی جملات رو خیلی کوتاه می‌کرد.
در مورد به روزآوری firmware هم بهتر هست که کاربرها اصلا این کار رو نکنن یا بعد از تحقیق در مورد آپدیت جدید تصمیم بگیرن. چون بسیاری از دستگاه‌ها به خصوص سامسونگ که خودم پلیر BD-d5900 رو دارم بعد از به روز آوری دچار مشکل میشن که این مشکل ویژگی cinavia هست که باعث میشه دستگاه بعضی از فیلم‌ها که شامل این فناوری هستن رو تشخیص بده که کپی هستند. بدین صورت که بعد از 15 تا 20 دقیقه از تماشای فیلم صدا قطع میشه و یک پیام روی صفحه نمایش داده میشه.

به غیر از اون سامسونگ در آپدیت‌ها جدیدش روش‌های مقابله با sammy Go و روت کردن دستگاه رو هم گنجانده که از نصب اون جلوگیری کنه
کلا هیچ خیری در آپدیت این نوع دستگاه وجود نداره، ما هم به امید خواندن بهتر بعضی از کدکها آپدیت کردیم ولی تنها چیزی که گیرمان آمد همین بود و آخرین آپدیتش هم همین بود. حالا یه فکری هم باید برای حل این مشکل کرد حالا با داونگرید یا تغییرکد منطقه.