برنامه نویس

اولین مجله الکترونیکی جامعه برنامه نویسان

سال اول - شماره چهارم - خرداد ۱۳۸۹

نگاهی به ASP.NET 4.0

ویژگی های جدید Visual C++ 2010

آشنایی با مشاهیر دنیای نرم افزار

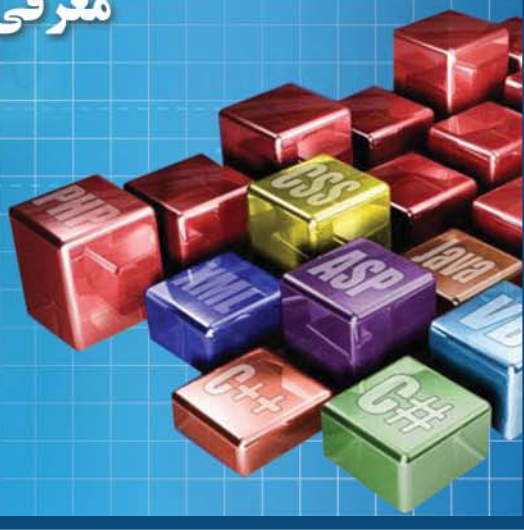
جورچین نقطه به نقطه با استفاده از HTML5.0 و jQuery

بهینه سازی پردازش رشته ها در VB6

سیستم رتبه بندی ستاره ای با استفاده از CSS

نگاهی اجمالی بر مقابله با مهندسی معکوس در .NET

معرفی کتاب



مجری انحصاری جذب آگهی مجله الکترونیکی برنامه نویس

نشریه الکترونیکی جامعه برنامه نویسان

شماره چهارم/ سال اول/ بهار ۱۳۸۹

و با تشکر از تمام کسانی که ما را در این شماره یاری کردند.

صاحب امتیاز:
www.barnamenevis.org

مدیر مسئول:
علی کشاورز

سردبیر:
مهدی عسگری

مدیر اجرایی:
حمیدرضا متقیان

طراح و صفحه آرا:
فهیمه زنجانی‌نژاد

همکاران این شماره:
مهدی موسوی - بهروز راد
محمد شمس جاوی
مهدی عسگری- عماد فیاض
سیاوش شهرویان

سازمان آگهی ها:
شرکت پیشرو ارتباط پارمیدا
تلفن:
۰۲۱-۲۲۳۲۷۳۰۰
۰۲۱-۲۲۳۲۷۳۰۱

وب سایت:
http://adv.barnamenevis.org

سخن سردبیر

مهدی عسگری

سلام

بالاخره شمارهٔ ۴ مجله برنامه‌نویس هم منتشر شد!

از این که با وجود مشکلات فراوان در انتشار مجله، همچنان همراه ما هستید، ممنون و خوشحالیم.

پس از مشورت و بحث‌هایی که بین اعضای اصلی مجله صورت گرفت، تصمیم گرفتیم از این به بعد مجله را به صورت فصل‌نامه منتشر کنیم که شمارهٔ فعلی مربوط به بهار ۱۳۸۹ است.

مجلهٔ برنامه‌نویس از دل سایت برنامه‌نویس و به منظور کمک به پیشرفت بیشتر اعضای سایت (و جامعهٔ برنامه‌نویسان فارسی زبان) به وجود آمد. این حرکت به صورت خودجوش شروع به کار کرده و تمام کسانی که تا به حال با مجله همکاری داشته‌اند به صورت داوطلبانه بوده است.

هر ماه مطالب زیادی برای مجله ارسال می‌شود که متأسفانه حجم زیادی از آن‌ها مشکلات فنی و ویرایشی زیادی دارند و در واقع مطالبی که در مجله منتشر می‌شود قسمت اندکی از کل مقالاتی است که برای ما ارسال می‌شود؛ ولی ما با اکثر نویسندگان این مقالات در ارتباط هستیم تا با رفع مشکلات فنی/ادبی موجود، مقالاتشان را منتشر کنیم و از شما خوانندهٔ عزیز نیز دعوت می‌کنیم که در صورت تمایل، مقاله‌های تألیفی یا ترجمه شدهٔ خود را برای چاپ در مجله برای ما ارسال کنید.

از دیگر حرکت‌های نویی که در سایت شروع کرده‌ایم، طرحی به نام «گفتگوهای فنی» است که در آن با انجام گفتگوهایی بین چند تن از برنامه‌نویسان خبره و باتجربه، مکان خوبی برای یادگیری مطالب مختلف برنامه‌نویسی به وجود آورده‌ایم و امیدواریم این گفتگوها را در آینده به صورت صوتی (pod cast) نیز منتشر کنیم تا گامی دیگر در راستای ارتقای دانش برنامه‌نویسی مخاطبانمان برداریم.

در پایان از تمام کسانی که کمک کردند این شماره منتشر شود، تشکر می‌کنم و امید است تا با یاری شما خوانندگان عزیز در هر شماره شاهد بهبود کیفی و کمی مطالب باشیم. در صورتی که در مورد مجله پیشنهاد یا انتقادی دارید، مایل به همکاری با مجله هستید با ایمیل مجله در آدرس magazine@barnamenevis.org تماس بگیرید.

نویسنده: K. Scott Allen

نگاهی به ASP.NET 4.0

mehdi_mousavi@hotmail.com

مترجم: مهدی موسوی

نگاهی به ASP.NET 4.0

سال دیگر هنگامی که Visual Studio 2010 و .NET 4.0 عرضه شوند، ما توسعه‌دهندگان ASP.NET دو framework آماده و کارآزموده برای درست کردن نرم افزارهای تحت وب در اختیار خواهیم داشت: ASP.NET Web Form و ASP.NET MVC. هر دو بر اساس هسته اجرایی ASP.NET ساخته شده‌اند و هر دو در حال جذب قابلیت‌های جدیدی برای شروع دهه بعدی هستند.

در این مقاله فرصت بررسی تمامی قابلیت‌های جدیدی که به ASP.NET افزوده شده است وجود ندارد، زیرا تغییرات به وجود آمده در هر یک از آن‌ها، بزرگ و بعضاً بنیادی هستند.

بنابراین تنها به قابلیت‌هایی در مورد هر دو framework خواهیم پرداخت که به گمانم از اهمیت بیشتری برخوردار است.

قابلیت‌های جدید ASP.NET Web Forms

هنگامی که مایکروسافت نسخه 4 را عرضه کند، هشت سال از سن ASP.NET Web Form ها سپری شده است و تیم مربوطه، کماکان در حال پالایش framework و ارتقاء آن است. در مقاله آخر خود، به برخی از این پیشرفت‌ها مانند کلاس‌های جدیدی که استفاده از قابلیت مسیریابی URL را تسهیل می‌کند و هم اکنون بخشی از خدمات اصلی ASP.NET محسوب می‌شود، اشاره کردم. همچنین، در مورد Property های MetaKeyword و MetaDescription در کلاس پایه Page که کنترل metatag های فرم را آسان می‌کند، سخن گفتم. این تغییرات البته جزیی هستند. تغییرات کلیدی در Web Form ها، پاسخی به انتقادات وارده بر این framework بوده است. بسیاری از توسعه دهندگان مایل بودند تا کنترل بیشتری روی HTML تولید شده توسط فرم‌ها و کنترل‌ها داشته باشند. در نسخه 4، روی بسیاری از کنترل‌های سمت سرور ASP.NET کار شده است تا HTML ای تولید شود که بتوان سبک نمایشی آن را به سادگی توسط CSS کنترل نموده و در عین حال، با قواعد عرفی وب نیز هم خوانی داشته باشد. همچنین، Property های جدیدی به کلاس‌های پایه افزوده شده است تا کنترل بیشتری روی شناسه های تولید شده سمت Client به توسعه دهنده بدهد. اکنون به این تغییرات اشاره خواهم کرد.

HTML مناسب برای CSS

یکی از کنترل هایی که تعیین سبک دشوار آن رسوای عام و خاص است، کنترل menu در ASP.NET است. ترجمه این کنترل به HTML باعث ایجاد جداول تو در تویی می‌شود که خصائص cellspacing، cellpadding و border را با خود بهمراه دارد. برای بدتر شدن شرایط، این کنترل اطلاعات css را در درون cell های جداول درونی نگاه داشته و یک بلاک in-line style در بالای صفحه ایجاد می‌کند. به عنوان نمونه، به تعریف یک منوی ساده توجه کنید:

```
<asp:Menu runat="server" ID="_menu">
    <Items>
        <asp:MenuItem Text="Home" NavigateUrl="~/Default.aspx" />
        <asp:MenuItem Text="Shop" NavigateUrl="~/Shop.aspx" />
    </Items>
</asp:Menu>
```

در ASP.NET 3.5، این منوی ساده چنین HTML ای تولید می‌کند (که برخی از خصائص آن برای خوانایی بیشتر حذف شده است):

```
<table class="..." cellpadding="0" cellspacing="0" border="0">
    <tr id="_menun0">
        <td>
            <table cellpadding="0" cellspacing="0"
                border="0" width="100%">
                <tr>
                    <td style="...">
                        <a class="..." href="Default.aspx">Home</a>
                    </td>
                </tr>
            </table>
        </td>
    </tr>
</table>
```

در ASP.NET 4.0، مایکروسافت کنترل منو مزبور را برای تولید semantic markup بررسی کرد. همین کنترل، در ASP.NET 4.0 کد HTML زیر را تولید می‌کند:

```
<div id="_menu">
    <ul class="level1">
        <li><a class="level1"
               href="Default.aspx"
               target="">Home
            </a>
        </li>
    </ul>
</div>
```

تولید چنین کد CSS-Friendlyای در نسخه‌های قبلی ASP.NET نیز میسر بود، اگر از Control Adapter ای برای ارائه منطق rendering دیگری برای کنترل استفاده می‌کردید، اما هم اکنون، کد تولید شده به طور پیش فرض، CSS-Friendly است. اگر در حال حاضر، style sheet ها و scriptهای client-side ای دارید که برای ASP.NET 3.5 نوشته شده‌اند، می‌توانید خصیصه controlRenderingCompatibilityVersion از بخش pages در web.config را برابر 3.5 قرار دهید تا کنترل مزبور همان جداول تو در تو را تولید کند. مقدار پیش فرض این خصیصه، 4.0 است. دقت کنید که Menu Control نسخه 4 کماکان Script Block ای را بالای صفحه تولید می‌کند، اما می‌توانید تولید این بلاک را با Set کردن IncludeStyleBlock کنترل به false، متوقف کنید.

کنترل‌های بسیار دیگری در نسخه 4.0 نیز CSS-friendly شده‌اند. به عنوان مثال، Validation Control ها مانند RangeValidator و RequiredFieldValidator دیگر inline-style تولید نمی‌کنند. همچنین کنترل های FormView، Login و Wizard نیز دیگر خودشان را درون یک table tag قرار نمی‌دهند (به شرطی که خصیصه RenderOuterTable را false تعیین کنید). مابقی کنترل ها نیز تغییر کرده‌اند. تنها به عنوان نمونه، شما می‌توانید RadioButtonList و CheckBoxList را مجبور کنید تا محتوای خود را در List Element ها قرار دهند و این امر با تعیین مقدار OrderedList یا UnorderedList برای خصیصه RepeatLaout میسر شده است که باعث می‌شود تا این دو کنترل، به ترتیب المنت‌های ol و li تولید کنند.

تولید Client ID ها

اگر تا به حال کد client-side ای برای دستکاری DOM نوشته باشید، احتمالا از این مساله مطلعید که ASP.NET بر اساس نیاز خود، خصیصه id در client-side را تغییر می‌دهد. برای این که ASP.NET از غیر تکراری بودن شناسه‌های تولید شده در صفحه

اطمینان حاصل کند، ClentID ها را با به هم چسباندن شناسه Control و دیگر اطلاعات اضافی تشکیل می‌دهد. شما به این ID های تولید شده سمت سرور، توسط خصیصه ClientID کنترل دسترسی دارید.

به عنوان نمونه اگر کنترل داخل یک naming container باشد (کنترلی که INamingContainer را پیاده‌سازی کرده است، مانند User Control ها و Master Page ها)، در آن صورت ASP.NET این شناسه را با قرار دادن شناسه naming container در ابتدای شناسه کنترل ایجاد می‌کند. برای کنترل‌های Data-Bound ای که بلاک های HTML متوالی تولید می‌کنند، ASP.NET پیشوندی حاوی شماره‌ای متوالی تولید می‌کند. اگر به Source صفحات ASP.NET رجوع کنید، احتمالاً شناسه‌هایی از قبیل ctl00_content_ctl20_ctl00_loginlink خواهید دید. این مقادیر تولید شده، نوشتن کدهای client-side در صفحات Web Form را دشوار می‌کند.

در Web Forms 4.0، خصیصه جدیدی تحت عنوان ClientIDMode به هر کنترلی افزوده شده است. شما می‌توانید از این خصیصه برای نفوذ در الگوریتم تولید شناسه‌های مزبور استفاده کنید. تعیین این خصیصه به صورت Static، از ASP.NET می‌خواهد تا از شناسه Control به عنوان ClientID استفاده کند بدون آن‌که رشته‌ای به آن چسبانده یا پیشوندی برایش قائل شود. به عنوان نمونه، CheckBoxList در کد زیر ol tag ای تولید می‌کند که شناسه آن checklist است، بدون توجه به این‌که کنترل در کجای صفحه قرار گرفته است:

```
<asp:CheckBoxList runat="server" RepeatLayout="OrderedList"
                  ID="checklist" ClientIDMode="Static">
    <asp:ListItem>Candy</asp:ListItem>
    <asp:ListItem>Flowers</asp:ListItem>
</asp:CheckBoxList>
```

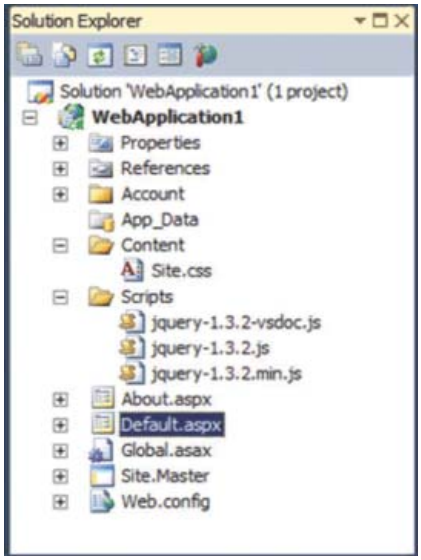
هنگامی که ClientIDMode را، Static تعیین می‌کنید، باید از غیر تکراری بودن شناسه ها سمت Client اطمینان حاصل کنید. اگر شناسه‌هایی با مقادیر تکراری در صفحه موجود باشند، script هایی که توسط ID مزبور به دنبال المنت مورد نظر هستند را از کار خواهید انداخت. سه مقدار دیگر برای ClientIDMode در دسترس است. مقدار Predictable برای کنترل‌هایی مفید است که IDataBoundListControl را پیاده‌سازی کرده‌اند، مانند GridView و ListView. از مقدار Predictable به‌همراه خصیصه ClientIDRowSuffix در این کنترل‌ها استفاده کنید تا شناسه‌های غیر تکراری توسط پسوند خاصی تولید کنید. برای مثال، ListView زیر به لیستی از اشیاء Employee بایند شده است. هر شی حاوی خصیصه‌های EmployeeID و IsSalaried است. ترکیب ClientIDMode و ClientIDRowSuffix از کنترل CheckBox می‌خواهد تا ClientIDهایی مانند employeeList_IsSalaried_10 تولید کند که در اینجا 10 بیانگر شناسه یک کارمند است:

```
<asp:ListView runat="server" ID="employeeList"
                    ClientIDMode="Predictable"
                    ClientIDRowSuffix="EmployeeID">
        <ItemTemplate>
            <asp:CheckBox runat="server" ID="IsSalaried"
                          Checked=<%# Eval("IsSalaried") %> />
        </ItemTemplate>
</asp:ListView>
```

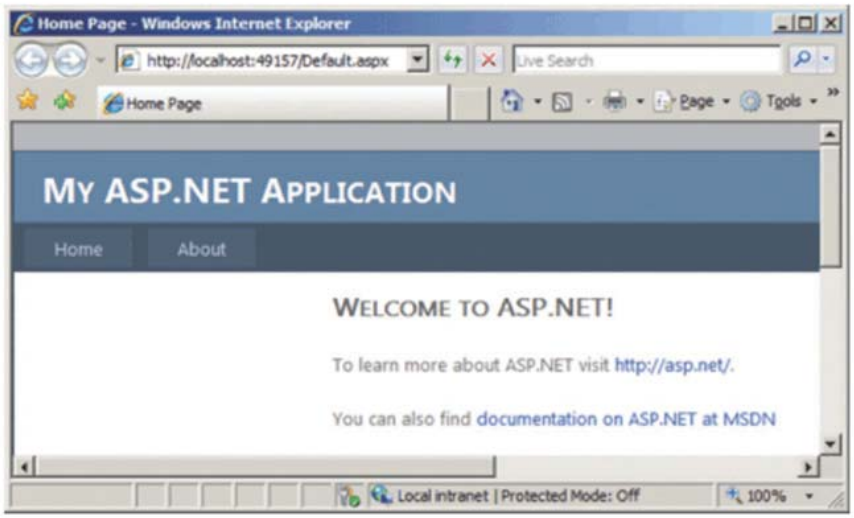
مقدار پیش فرض دیگر برای ClientIDMode، Inherit است. تمامی کنترل های صفحه از مقدار Inherit برای این خصیصه بصورت پیش فرض استفاده می‌کنند. این بدان معناست که کنترل از همان ClientIDMode پدر باید استفاده کند. در نمونه کد قبلی، CheckBox مقدار Predictable را از ListView به ارث برده است. آخرین گزینه ممکن برای این خصیصه، AutoID است. این گزینه از ASP.NET می‌خواهد تا از همان الگوریتمی که در نسخه 3.5 برای تولید شناسه‌ها استفاده می‌کرده، سود برد. مقدار پیش فرض این خصیصه در صفحه AutoID است. از آنجایی که کلیه کنترل‌های موجود در صفحه از Inherit به عنوان مقدار پیش فرض این خصیصه استفاده می‌کنند، انتقال برنامه‌های موجود به ASP.NET 4.0، باعث تغییر الگوریتم مورد استفاده نخواهد شد، مگر آن که ClientIDMode را تغییر دهید. این خصیصه را همچنین می‌توان در بخش pages فایل web.config تعیین کرد تا پیش فرض دیگری برای تمامی صفحات برنامه در نظر گرفته شود.

Project Template جدید

در Visual Studio 2008، Template پروژه‌های Web application و Web site، به طور پیش فرض یک صفحه Default.aspx، یک فایل Web.config و شاخه App_Data را ایجاد می‌کند. این Template های آغازین ساده بوده و نیازمند برخی کارهای اضافی است تا بتوان با استفاده از آن‌ها شروع به نوشتن برنامه‌هایی حقیقی کرد. Template های مزبور در Visual Studio 2010، زیربنای مورد نیاز برای ساخت برنامه‌هایی با روش‌های امروزی را فراهم می‌کنند. تصویری از یک برنامه جدید ایجاد شده توسط این Template ها را در شکل روبه‌رو می‌بینید:



دقت کنید که چگونه این برنامه جدید حاوی master page ای به صورت پیش فرض است (Site.master). کلیه فایل‌های aspx ای که در پروژه جدید وجود دارند، content page هایی هستند که از کنترل ContentPlaceHolder برای ضمیمه کردن محتوای خود در ساختار تعریف شده master page استفاده می‌کنند. همین‌طور دقت کنید که پروژه جدید حاوی style sheet ای در شاخه Content است (Site.css). این فایل با استفاده از link tag در Master Page قرار گرفته و حاوی چند style از پیش تعیین شده برای کنترل نحوه نمایش body، heading، چهارچوب اولیه صفحه و ... است. پروژه جدید همچنین حاوی شاخه Scripts است که آخرین نسخه از کتابخانه jQuery که یک Open Source JavaScript Framework بوده و توسط مایکروسافت پشتیبانی می‌شود و به عنوان بخشی از عملیات نصب Visual Studio 2010 در نظر گرفته شده، همراه آن است.



این Template جدید برای ایجاد پروژه‌ها با استفاده از Master Page ها و Style Sheet های خود به توسعه‌دهندگان برای شروع در جهتی مناسب کمک می‌کند. نسخه در حال اجرای این برنامه جدید در شکل روبه رو نمایش داده شده است.

Visual Studio 2010 همچنین حاوی Template های "Empty" برای Web Site ها و Web Application ها است. این Template ها هنگام استفاده، حاوی فایل‌ها و شاخه‌های فوق‌الذکر نیستند، بنابراین شما می‌توانید خودتان از ابتدا شروع کنید.

خبر خوب دیگر در مورد پروژه های ASP.NET 4.0 آن است که فایل های web.config تقریباً خالی هستند. بیشتر پیکربندی‌هایی که عادت کرده بودیم در فایل‌های web.config در ASP.NET 3.5 مشاهده کنیم، هم اکنون بخشی از machine.config شده‌اند که در شاخه نصب شده از framework نسخه 4 قرار گرفته‌اند. این موضوع شامل پیکربندی‌های کنترل‌های موجود در System.Web.Extensions، Http Handler/Module های پیکربندی شده برای پشتیبانی از JavaScript Proxy ها در Web Service ها و بخش system.webserver برای سایت‌هایی که تحت IIS7 اجرا می‌شوند، می‌باشد.

قسمت‌های جدید ASP.NET MVC

Visual Studio 2010 برای ما نسخه دوم ASP.NET MVC را به ارمغان می‌آورد. اگرچه این framework هنوز نوپا است، اما نظر توسعه‌دهندگان بسیاری را که به دنبال framework ای آزمون‌پذیر بودند، به خود جلب کرده است. نسخه دوم این frame-work قصد دارد تا بر تولید محصول بهتر توسط توسعه‌دهنده تمرکز کرده و زیرساخت‌هایی را بیفزاید که امکان تعامل با پروژه‌هایی در مقیاس بزرگ را نیز فراهم کند.

Area ها

یکی از روش های ساخت برنامه های ASP.NET Web Form بسیار بزرگ، تقسیم پروژه به زیر پروژه‌ها است (روشی که توسط P&P Web Client Composite Library ترویج داده شده است). پیاده‌سازی این روش در ASP.NET MVC 1.0 کاری دشوار است زیرا با برخی از ایده‌های MVC در تضاد است. MVC 2.0 به طور رسمی از این سناریو با استفاده از مفهوم Area پشتیبانی می‌کند. یک area به شما امکان بخش‌بندی MVC Application را در سطح پروژه های Web Application، یا در سطح شاخه‌های موجود در یک پروژه می‌دهد. این مفهوم جدید به تقسیم بندی منطقی بخش‌های متفاوت یک application برای نگهداری بهتر، کمک می‌کند.

در یک MVC Web Application، بخش Parent Area (پدر)، پروژه MVC ای است که شامل فایل global.asax و web.config است. این بخش می‌تواند حاوی بخش‌های متعارفی از محتوا مانند style sheet ها، کتابخانه‌های JavaScript و Master Page ها باشد. بخش Child Area (فرزند) نیز پروژه‌های MVC است، اما از آنجایی که این پروژه‌ها به صورت فیزیکی تحت پروژه پدر در runtime قرار گرفته‌اند، بخش پدر و فرزند، به عنوان برنامه‌ای واحد دیده می‌شوند.

به عنوان مثال، نرم افزار انبارداری بزرگی را تصور کنید. علاوه بر Parent Area، برنامه انبارداری ممکن است به بخش‌های سفارش، توزیع، گزارش و مدیریت تقسیم شود. هر بخش می‌تواند در MVC Web Project مجزایی قرار گیرد و هر پروژه باید خود را توسط کلاسی که از کلاس abstract ای تحت عنوان AreaRegistration مشتق شده است، ثبت کند. در کد زیر، ما خصیصه AreaName را به منظور برگرداندن نامی مناسب برای بخش گزارش‌ها و متد RegisterArea را برای تعریف مسیرهای جدید در reporting area، override کرده‌ایم:

```
public class ReportingAreaRegistration:
                        AreaRegistration
{
    public override string AreaName
    {
        get { return "Reporting"; }
    }

    public override void
       RegisterArea(
          AreaRegistrationContext context
                    )
    {
        context.MapRoute(
            // route name
            "ReportingDefault",
            // url pattern
            "reporting/{controller}/{action}",
            // route defaults
            new { controller = "Home",
                  action = "Index" },
            // namespaces
            new string[]{"Reporting.Controllers"}
        );
    }
}
```

دقت کنید که ما آرایه ای از string های حاوی namespace ها هنگام جستجوی کنترلر reporting area به تابع مورد نظر فرستاده‌ایم. این مساله امکان داشتن کنترلرهایی با نام یکسان (به عنوان مثال، می‌توانیم کلاس‌های HomeController متفاوتی در Application داشته باشیم) را فراهم می‌کند.

اعتبار سنجی آسان با DataAnnotation ها

DefaultModelBinder در ASP.NET MVC موظف است تا داده‌ها را از محیط درخواست به خصیصه‌ی Model انتقال دهد. به عنوان مثال هنگامی که model binder، model object ای با خصیصه Title می‌بیند، برای پیدا کردن متغیری با نام یکسان (Title) در فرم، QueryString و متغیرهای سرور جستجو می‌کند. اما model binder فراتر از تبدیل data type های ساده هیچ گونه اعتبار سنجی انجام نمی‌دهد. اگر نیاز دارید تا خصیصه Title از model object شما حداکثر حاوی 50 کاراکتر یا کمتر باشد، شما باید این شرط را هنگام اجرای con-troller action بررسی کرده، یک custom model binder پیاده‌سازی نموده یا اینترفیس IDataErrorInfo را روی model خود پیاده‌سازی کنید.

در DefaultModelBinder، ASP.NET MVC 2.0 به خصیصه DataAnnotation روی model object ها نگاه می‌کند. این خصیصه، به شما امکان می‌دهد تا منطق اعتبار سنجی خود را روی مدل، اعمال کنید. به عنوان مثال، کلاس Movie را در نظر بگیرید:

```
public class Movie
{
    [Required(ErrorMessage="The movie
                must have a title.")]
    [StringLength(50, ErrorMessage="
       The movie title is too long.")]
    public string Title { get; set; }
}
```

خصیصه های موجود model binder، Title را از اجباری بودن فیلد مطلع کرده و حداکثر طول این فیلد را 50 کاراکتر اعلام می‌کند. هنگامی که اعتبارسنجی با خطا مواجه می‌شود، MVC Framework به طور خودکار می‌تواند پیام ErrorMessage را در مرورگر نشان دهد. از دیگر خصیصه‌های اعتبارسنجی می‌توان به خصیصه‌ای برای آزمایش یک بازه و خصیصه‌ای برای آزمایش Regular Expression ها، اشاره کرد.

هنگام نوشتن این مقاله، MVC runtime فقط اعتبار سنجی را سمت سرور انجام می‌دهد. انتظار می‌رود که تیم MVC منطق اعتبار سنجی سمت Client را از خصیصه‌های اعتبارسنجی در نسخه نهایی MVC 2.0 تولید کند. اعتبارسنجی بدین طریق و با استفاده از این خصیصه‌ها سمت Client و سرور، گامی بزرگ در جهت پشتیبانی برنامه محسوب می‌شود.

Templated Helpers

Template Helper ها در ASP.NET MVC 2.0 نیز از خصائص DataAnnotation استفاده می‌کنند. اما جای آن که از این خصائص برای تعیین منطق اعتبارسنجی سود برند، آن‌ها از این خصائص برای جنبه نمایشی یک مدل استفاده می‌کنند.

این Template Helperها با متدهای جدید Html Helper تحت عناوین DisplayFor و EditorFor آغاز می‌شوند. متدهای کمکی مزبور، Template هایی با استفاده از مدل داده شده بر اساس Type مدل را می‌یابند. به عنوان مثال اجازه دهید تا به کلاس Movie و Property ای جدیدی که به آن افزوده‌ایم، نگاهی داشته باشیم:

```
public class Movie
{
    // ...

    [DataType(DataType.Date)]
    public DateTime ReleaseDate
     { get; set; }
}
```

در این سناریو هر فیلم تاریخ انتشار خود را به همراه داد اما برای هیچ کس اهمیت ندارد که ساعت انتشار فیلم در روز چه هنگام بوده است. ما فقط نیاز داریم که اطلاعاتی در مورد تاریخ انتشار فیلم نمایش دهیم و نه ساعت آن. توجه کنید که این نیت خود را با استفاده از خصیصه DataType بیان کرده ایم.

برای نمایش صحیح تاریخ انتشار، نیاز به یک Display Tem-plate داریم که در واقع Partial View ای با دنباله ascx می‌باشد که در شاخه DisplayTemplates قرار گرفته است.

شاخه DisplayTemplates به خودی خود می‌تواند در زیر شاخه viewی کنترلر قرار گرفته (که در این‌صورت این Template فقط روی view های آن کنترلر اعمال می‌شود) یا در شاخه Shared View قرار گیرد (که در این حالت، template در همه بخش‌ها در دسترس است). در چنین شرایطی نام Template باید Date. ascx باشد و کد آن بدین شکل است:

```
<%@ Control Language="C#" Inherits="
   System.Web.Mvc.ViewUserControl"
%>
<%= Html.Encode(String.Format(
                "{0:d}", Model))
 %>
```

برای آن که MVC Framework از این Template استفاده کند، باید از متد کمکی DisplayFor هنگام Render کردن خصیصه ReleaseDate استفاده کنیم. کد زیر متعلق به Tem-plate دیگری با نام Movie.ascx است

```
<%@ Control Language="C#"
  Inherits="
  System.Web.Mvc.ViewUserControl<Movie>"
 %>

 <fieldset>
   <legend>Fields</legend>
     <p>
       Title:
       <%= Html.LabelFor(m => m.Title)%>
       <%= Html.DisplayFor(m => m.Title)%>
     </p>
     <p>
       <%= Html.LabelFor(m => m.ReleaseDate)%>
       <%= Html.DisplayFor(m => m.ReleaseDate)%>
     </p>
 </fieldset>
```

به strongly-typed بودن متدهای کمکی LabelFor و Dis-playFor توجه کنید. این مساله به انتشار تغییرات هنگام refac-tor شدن مدل، کمک خواهد کرد.

برای استفاده از Movie.ascx Template به منظور نمایش فیلم در هر بخشی از برنامه، کافی است تا مجددا از متد کمکی DisplayFor استفاده کنیم. کد زیر متعلق به View ای است که برای کلاس Movie بصورت strongly-typed نوشته شده است:

```
<asp:Content ID="detailContent"
          ContentPlaceHolderID="MainContent"
          runat="server">
        Movie:
        <%= Html.DisplayFor(m => m) %>

    </p>
</asp:Content>
```

متود DisplayFor برای استفاده از همان مدل به کار برده شده در صفحه، strongly-typed است بنابراین تایپ پارامتر m در lambda expression، Movie است.

این متد به طور خودکار از Movie.ascx Template برای نمایش یک فیلم استفاده خواهد کرد و این Template نیز به نوبه خود از DisplayFor برای پیدا کردن Tem-plate Date.ascx استفاده می‌کند.

اگر از خصیصه DataType برای ReleaseDate در کلاس Movie استفاده نمی‌کردیم، متد DisplayFor از Date. ascx Template استفاده نمی‌کرد و تاریخ و ساعت انتشار فیلم تواماً نمایش داده می‌شد، اما خصیصه DataType کمک کرد تا Framework را برای یافتن Template صحیح یاری کنیم. استفاده از Template های تو در توی strongly-typed و data type annotation ها قدرتمند است و به توسعه سریع‌تر برنامه‌ها کمک خواهد کرد.

K.Scott Allen عضو هیات علمی Pluralsight و موسس OdeToCode است.

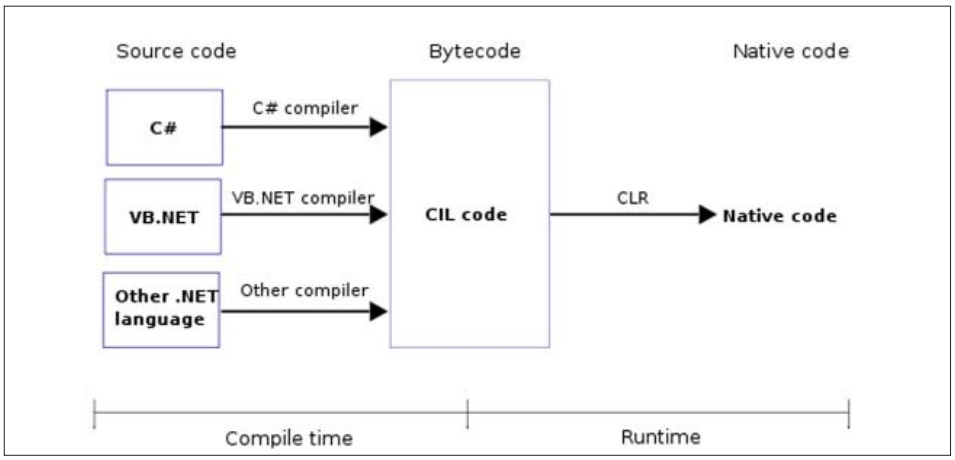
شما می‌توانید به آدرس الکترونیکی scott@OdeToCode. com با وی تماس گرفته یا وبلاگ وی را در http://odeto-code.com/blogs/scott مطالعه نمایید.

از متخصصین فنی، آقایان Phill Haack و Matthew Osborn برای مرور این مقاله نیز سپاسگزارم.

نویسنده: مهدی عسگری

ویژگی‌های جدید Visual C++ 2010

mehdi.asgari@yahoo.com



در این مقاله قصد دارم ویژگی‌های جدیدی را که به زبان، رانتایم و محیط توسعهٔ Visual C++ 2010 اضافه شده‌اند بررسی کنم. Visual C++ 2010 به همراه Visual Studio 2010 و .NET Framework 4.0 در آپریل 2010 عرضه شد.

نسخه‌های تجاری آن جزئی از Visual Studio 2010 هستند اما نسخه‌ی رایگان آن (Express) را می‌توان به تنهایی نصب کرد.

1- ویژگی‌های جدید در IDE، پروژه‌ها و سیستم Build

Intellisense و IDE در نسخه‌ی جدید، از نظر کارایی نسبت به گذشته بهبود بسیاری پیدا کرده‌اند.

(در واقع کاملاً از صفر بازطراحی شده‌اند) در نسخه‌های قبلی در فولدر حاوی فایل‌های پروژه، فایلی با پسوند .NCB وجود داشت که معمولاً حجیم‌ترین فایل موجود در آن فولدر بود.

این فایل دربردارنده‌ی اطلاعات Intellisense و Auto Completion پروژه است و هیچ وقت حجمش کم نمی‌شود!

و گاه‌به‌گاه نیز فایل خراب می‌شد که باید از صفر ایجاد می‌شد.

در نسخه‌های قبل با کوچک‌ترین تغییری در فایل‌های هدر، کل پروژه باید دوباره parse می‌شد و در این زمان معمولاً محیط یخ می‌زد! و قابل استفاده نبود.

در نسخه‌ی جدید فایل‌ها در بکگراند parse می‌شوند و فقط translation unit فعلی (فایل کد و تمام هدرهایی که آن به طور مستقیم یا غیر مستقیم از آن‌ها استفاده می‌کند) parse می‌شود؛ ضمناً اطلاعات در فایل جدیدی با فرمت .SDF (فایل دیتابیس SQL Server) ذخیره می‌شوند (در ضمن فولدر جدیدی در فولدر پروژه وجود دارد با نام iPCH، که حاوی اطلاعات پشتیبان برای Intellisense و فایل دیتابیس است).

این مسائل با هم باعث می‌شوند که کلاً کار با این برنامه سریع‌تر شود.

Auto completion برای هدرها

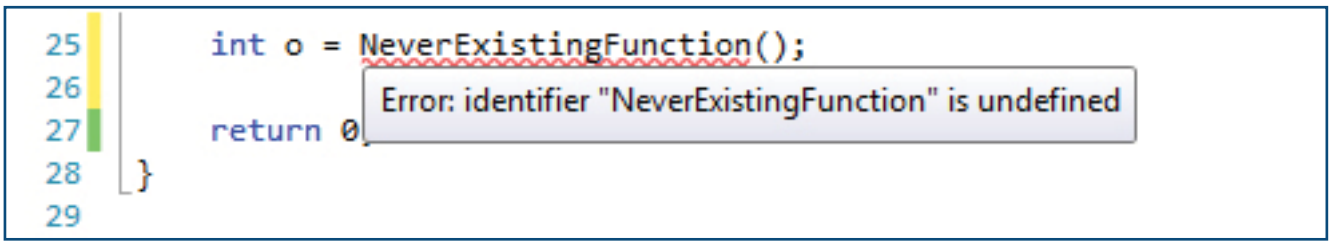
بعد از تایپ کردن عبارت< #include در بالای فایل، منوی جدید باز شده و لیست تمام هدرها را نشان می‌دهد.

سلسله مراتب فراخوانی توابع

توسط این ویژگی می‌توانید تمام توابعی که یک تابع خاص را فراخوانی کرده‌اند یا توسط آن تابع فراخوانی شده‌اند، مشاهده کنید. (کلیک راست روی تابع و انتخاب View Call Hierarchy)

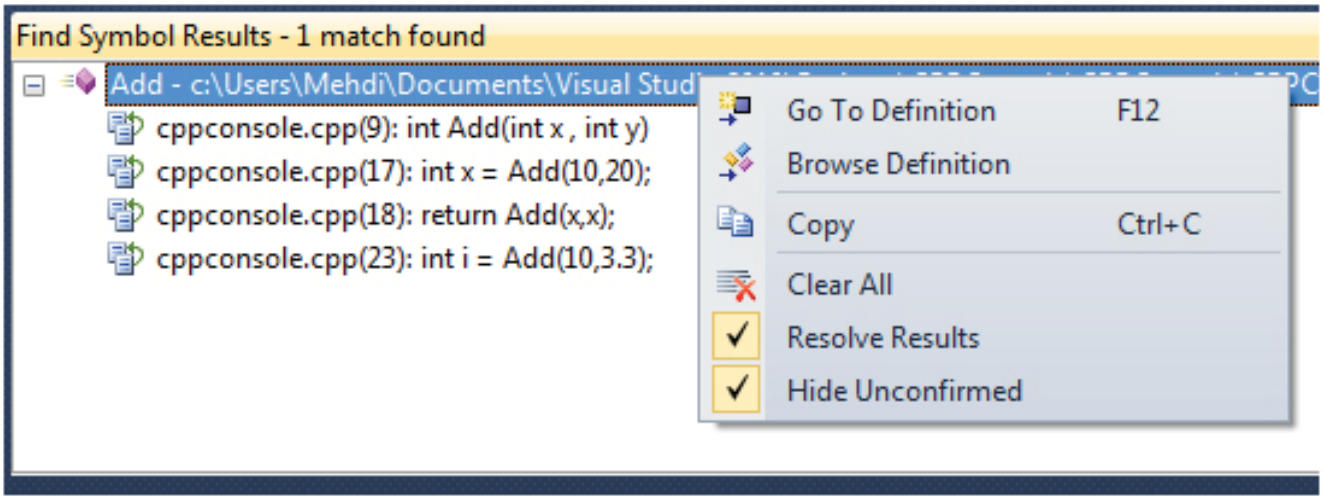
خطوط قرمز زیر کلمات

این ویژگی باعث می‌شود که خطاهای سینتکسی و معنایی برنامه را بدون نیاز به کامپایل کردن ببینیم. خطوط و کلماتی (متغیر، عبارت ، ...) که به خطایی مربوط می‌شوند با یک خط قرمز در زیرشان مشخص می‌شوند که با بردن ماوس به روی آن‌ها، متن خطا درون یک tooltip نشان داده می‌شود (در ضمن این خطا در پنجرهٔ Error List نیز لیست می‌شود)



Find All References

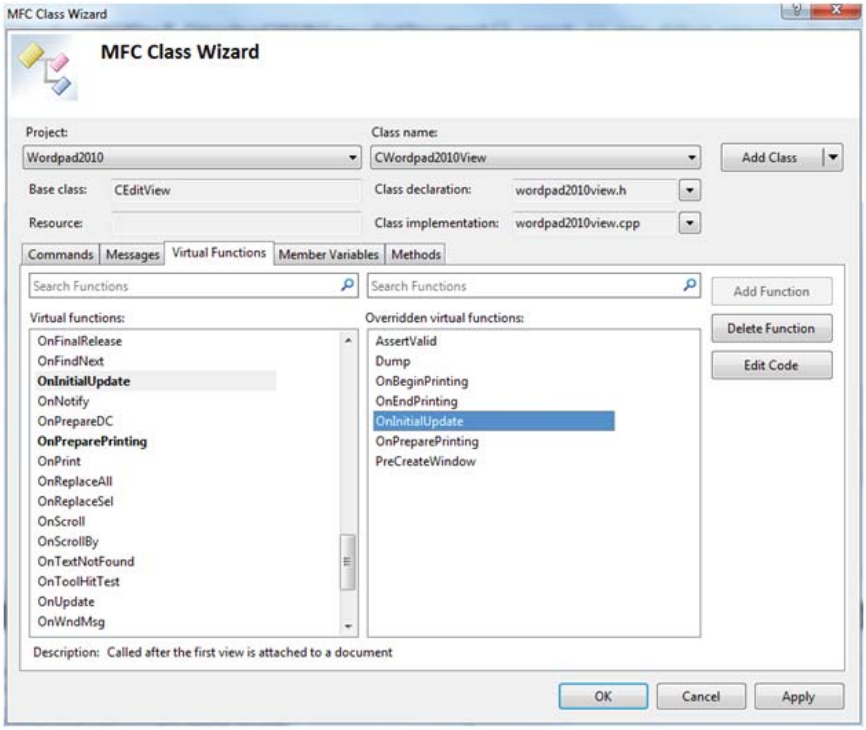
در نسخه‌های گذشته، این ویژگی فقط نتایج بررسی و تایید شده توسط کامپایلر را نشان می‌داد. به عنوان مثال اگر به دنبال متد M از کلاس C می‌بودید، فقط referenceهایی که شامل استفاده از M در کانتکست C بودند نشان داده می‌شدند. در نسخه‌ی جدید دو نوع جستجو وجود دارد: جستجوی سریع‌تر که تمام عبارات تطبیقی را (بدون توجه به کانتکست) نشان می‌دهد (البته باز هم نسبت به نتایج Find in Files دقیق‌تر است.) و جستجوی دقیق‌تر که فقط نتایج بررسی شده توسط کامپایلر را برمی‌گرداند.



Class Wizard

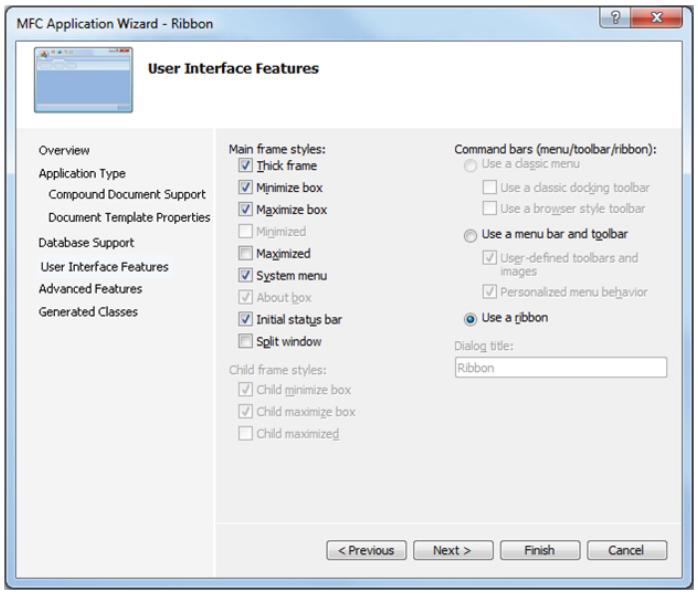
برنامه‌نویسان قدیمی‌تر این ویژگی دوست داشتنی VC6 که از Visual Studio 2002 حذف شد را به یاد دارند. حالا دوباره در نسخه‌ی 2010 این امکان بازگشته‌است.

نسخه‌ی جدید همانند نسخه‌ی موجود در VC6 است؛ البته بهبودهایی مانند جستجو در توابع و اعضا و دستورات دارد.



Ribbon Designe

در Visual C++ 2008، ریبون‌ها را باید با کد ایجاد می‌کردید. اما در VC10 یک طراح ویژوالی در خدمت شماست.



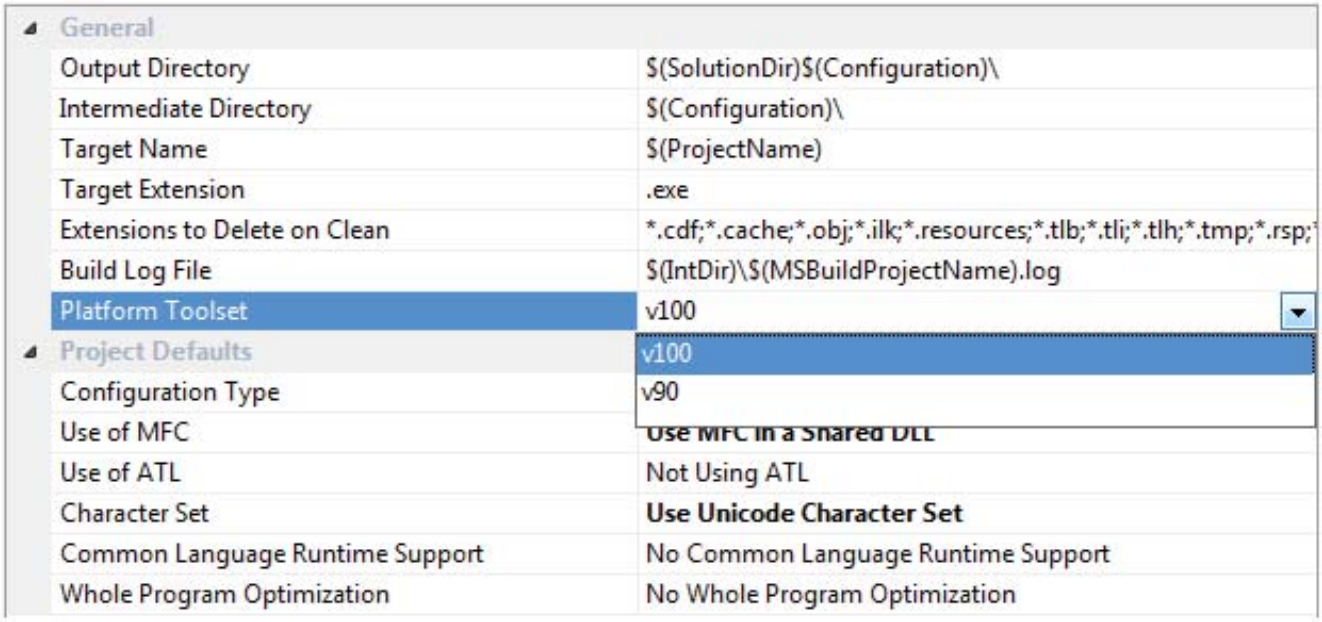
سیستم MS-Build

در VC9 (Visual C++ 2008)، پسوند فایل‌های پروژه، .vcproj بود؛ در نسخه‌ی جدید این پسوند به .vcxproj تغییر یافته‌است. البته این تغییر فقط یک تغییر پسوندی نیست، زیرا سیستم build نیز از VC-Build به MS-Build (که در C# و VB.NET نیز استفاده می‌شد) تغییر یافته‌است. برای اطلاعات بیشتر در مورد MS-Build به MSDN مراجعه کنید.

ضمناً یک فایل .vcxproj.filters نیز اضافه شده که حاوی اسامی فایل‌های موجود در Solution Explorer است (قبلاً این اطلاعات در خود فایل پروژه قرار داشتند)

Multi Targeting

در VS2008 برای زبان‌های تحت دات نت (VB, C#, C++/CLI) این ویژگی معرفی شد که به شما اجازه می‌داد برنامه‌تان را برای نسخه‌های مختلف فریم ورک ایجاد کنید. در VC10 این ویژگی برای نسخه‌ی نیتیو VC++ نیز اضافه شده‌است. البته باید ابزار و فایل‌های مربوط به نسخه‌های قدیمی را روی سیستم‌تان داشته باشید تا بتوانید از این ویژگی استفاده کنید. به عنوان مثال VS2008 SP1 و VC10 روی سیستم من نصب هستند:



می‌توانید نسخه‌های قبلی (2008,2005,2003,2002) را نیز هدف قرار دهید؛ از نظر تئوری می‌شود این کار را برای VC6 نیز انجام داد اما مایکروسافت این امکان را به ما نمی‌دهد.

2- ویژگی‌های جدید در زبان، رانتایم و کامپایلر

VC++ 2010 چندین ویژگی از نسخه‌ی استاندارد بعدی زبان C++ (یعنی C++0x) را پیاده‌سازی کرده‌است که باعث غنی‌تر و قدرتمندتر شدن این زبان شده و در موارد زیادی برنامه‌نویسی با این زبان را راحت‌تر می‌کنند.

کلمه‌ی کلیدی auto

احتمالاً در درس برنامه‌سازی پیشرفته در دانشگاه (یا در کتاب‌های آموزشی این زبان) کلمهٔ کلیدی auto را دیده‌بودید و این سوال برایتان مطرح بوده که چرا هیچ‌گاه از این کلمه در دنیای واقعی استفاده نمی‌شود. کمیته‌ی مسئول طراحی C++ با هوشمندی، به جای اضافه کردن کلمه‌ی کلیدی جدیدی به زبان، معنای کلمه‌ی auto را عوض کرده‌اند و احتمالاً از این به بعد این کلمه پرکاربردترین کلمه‌ی کلیدی C++ شود. کارکرد این کلمه مانند var در C# یا let در F# است و به برنامه‌نویس اجازه می‌دهد که هنگام تعریف متغیر، نوع آن را تعریف نکند، تا خود کامپایلر از مقدار سمت راست انتساب داده شده، نوع را حدس بزند (به این مکانیسم Type Inference یا استنتاج نوع گفته می‌شود)

مثال:

```
11      auto i = 10;
12      auto s = "Hello World";
13      auto f = 4.45;
14      auto v = vector<map<int,string>>();
15      auto t = tuple<float,string>();
16
17      retur
```

std::tr1::tuple<float, std::string, std::tr1::_Nil, std::tr1::_Nil,

نوع متغیرهای تعریف شده (که کامپایلر بر اساس کاربردشان حدس زده):

i : int
*s : const char
f : double
<<v : vector<map<int,string
<t : tuple<float,string

مسلماً این مکانیسم کار برنامه‌نویسان را خیلی راحت می‌کند. به عنوان مثال دو تعریف زیر را که هر دو عملکرد یکسان ولی تعریف متفاوتی دارند ببینید تا به ارزش این مکانیسم و کارکرد آن در خوانایی و کاهش بیشتر کد پی ببرید:

```
11      auto a = map<tuple<int,float,int>,map<int,vector<double>>>();
12
13      map<tuple<int,float,int>,map<int,vector<double>>> b =
14          map<tuple<int,float,int>,map<int,vector<double>>>();
15
```

توابع لامبدا:

این توابع همان توابع بی نام هستند که قبلاً در C# به عنوان anonymous methods و lambda expressions آن‌ها را دیده‌ایم. (زبان‌های بسیاری از این ویژگی پشتیبانی می‌کنند. مثلاً در پایتون کلمهٔ کلیدی lambda برای این کار استفاده می‌شود)

مثال:

```
auto add = [](int x, int y) { return x + y; };
cout << add(1,2) << endl;
                  int add(int x, int y)
return 0;
```

البته درون این توابع مثل توابع دیگر می‌تواند شامل چند خط و دیگر ساختارهای برنامه‌نویسی باشد و نیز می‌توان از متغیرهای تعریف شده در بیرون تابع لامبدا نیزاستفاده کرد.

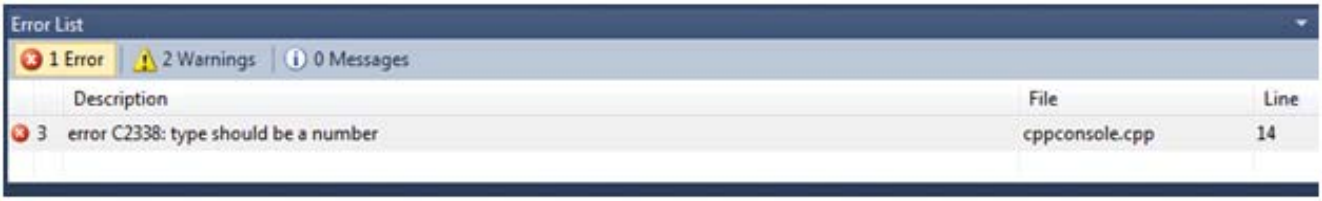
static_assert

بر خلاف دیگر assert ها که نتیجه‌شان موقع اجرای برنامه مشخص می‌شود، حاصل تست این assert در زمان کامپایل معلوم می‌شود و در صورت false شدن، عمل کامپایل متوقف شده و پیام خطا نشان داده می‌شود.

در مثال زیر تابع ما فقط روی انواع عددی صحیح عمل می‌کند وگرنه پیغام خطای مشخص شده توسط ما را نشان می‌دهد:

```
11  template <typename T>
12  bool CompareNumbers(T v1, T v2)
13  {
14      static_assert(is_integral<T>::value, "type should be a number");
15      return v1 > v2;
16  }
17
```

در صورتی که تابع را با آرگومان‌های غیر عدد صحیح (مثلا float) فراخوانی کنیم، کامپایل شکست خورده و پیغام زیر در پنجرهٔ Error List مشاهده می‌شود:



decltype

معمولاً از این کلمهٔ کلیدی همراه با auto استفاده می‌شود و کاربرد زیادی در generic programming دارد. این کلمهٔ کلیدی، نوع یک عبارت را بر می‌گرداند. به عنوان مثال:

```
double d = 42.0;      // decltype(i) yields double
const int&& f();      // decltype(f()) yields const int&&
struct foo {int i;}; // decltype(f.i) yields int (f being an object of type foo)
```

به‌عنوان مثال می‌توان توسط این ویژگی قدرتمند، تابعی با template تعریف کرد که نوع بازگشتی آن بستگی به آرگومان‌های templateش دارد؛ یا تابعی با template تعریف کرد که تابع دیگری را فراخوانی می‌کند و سپس مقداری از نوع نوع برگشتی تابع فراخوانی شده، برمی‌گرداند.

R-value references

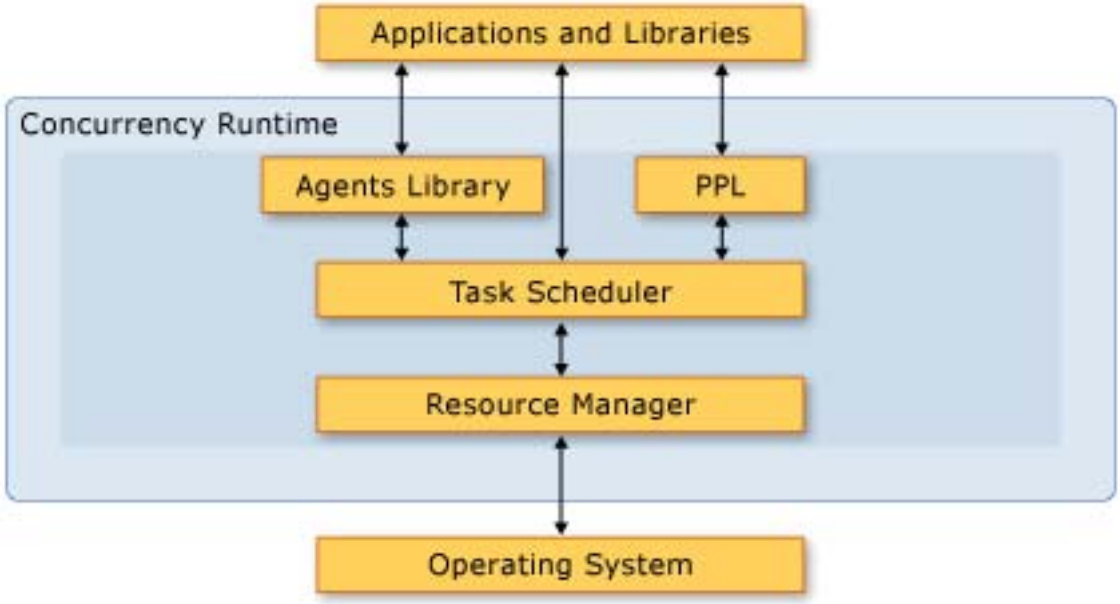
توضیح این ویژگی، خود یک مقاله‌ی مجزا می‌طلبد. برای اطلاعات بیشتر به این مقاله مراجعه کنید:

http://blogs.msdn.com/vcblog/archive/2009/02/03/rvalue-references-c-0x-features-in-vc10-part-2.aspx

nullptr

این کلمهٔ کلیدی مشخص می‌کند که یک هندل شی یا اشاره گر، به چیزی اشاره نمی‌کند. این کلمه هم در کد C++/CLI و هم در کد نیتیو معنی دارد. کلمهٔ nullptr__ که مخصوص مایکروسافت است، فقط در کد نیتیو استفاده می‌شود.

برنامه نویسی موازی



Concurrency Runtime یک رانتایم جدید است که جزئی از رانتایم C (CRT) بوده و از برنامه‌های موازی در C++ پشتیبانی می‌کند. این رانتایم شامل مولفه‌های زیر است:

Parallel Patterns Library: شامل کانتینترهای همه منظوره (مثل concurrent_vector) و الگوریتم‌هایی (مثل parallel_for_each) برای برنامه‌نویسی موازی است. در ضمن واحد موازی‌سازی (مانند .NET 4.0)، تسک است. هدرهای مربوطه: concurrent_queue.h، ppl.h و concurrent_vector.h

Asynchronous Agents Library که برنامه‌نویسی مبتنی بر اکتورها و مدل تبادل پیام را پشتیبانی می‌کند (مثل Erlang). هدرهای مربوطه: agents.h

Task Scheduler: این مولفه وظیفه‌ی زمان‌بندی و هماهنگی بین تسک‌ها را در زمان اجرا بر عهده دارد. این مولفه از الگوریتم work stealing (که با Cilk به شهرت رسید و الان کاربرد تقریباً همگانی پیدا کرده) استفاده می‌کند. هدرهای مربوطه: concrt.h

Resource Manager: این مولفه وظیفه‌ی مدیریت منابعی مثل پردازنده‌ها و حافظه را بر عهده دارد. این مولفه با Task Scheduler ارتباط دارد. هدرهای مربوطه: concrtm.h

(برای استفاده از این رانتایم، از فضای نام Concurrency استفاده کنید)

کتابخانهٔ SafeInt

از این کتابخانه برای انجام عملیات محاسباتی امن (که به عنوان مثال از سرریز اعداد جلوگیری به‌عمل می‌آورند) استفاده می‌شود.

این، همه‌ی آن چیزی نبود که در نسخه‌ی جدید Visual C++ عرضه شده، ولی من سعی کردم ویژگی‌های مهم‌تر را معرفی کنم. برای مشاهده‌ی ویژگی‌های دیگر که در اینجا لیست نشده‌اند به آدرس زیر مراجعه کنید:

http://msdn.microsoft.com/en-us/library/dd465215.aspx

منابع:

MSDN
http://mariusbancila.ro/blog
http://blogs.msdn.com/vcblog

Alan Curtis Kay

آشنایی با مشاهیر دنیای نرم افزار

نویسنده: مهدی عسگری

طبق روال ، در این شماره نیز می‌خواهم شما را با یکی دیگر از مشاهیر و بزرگان دنیای نرم افزار و برنامه نویسی آشنا کنم.

Alan Curtis Kay که اغلب با نام Alan Kay نام برده می‌شود، یکی از دانشمندان به نام علوم کامپیوتر است که در سال 1940 در امریکا متولد شد.
معمولاً این شخص را به خاطر کارهای بنیادی وی در زمینهٔ «طراحی رابط‌های کاربری گرافیکی و پنجره‌ها» و نیز «شی‌گرایی» می‌شناسیم. وی قبلاً جزو محققان برجستهٔ آزمایشگاه‌های HP و نیز استاد دانشگاه‌های MIT و کیوتو بود، ولی اکنون رییس موسسه تحقیقاتی Viewpoints Research و نیز استاد دانشگاه UCLA است.
آیا جملهٔ مشهور «بهترین راه برای پیشگویی آینده ، ساختن آن است» را شنیده‌اید ؟
این جمله از Alan Kay است!
وی در جوانی یک گیتاریست سبک جاز بود؛ بعدها به دانشگاه رفته و در رشته های ریاضیات و زیست شناسی مولکولی در مقطع کارشناسی تحصیل کرد. وی کارشناسی ارشد (مهندسی برق) و Ph.D (علوم کامپیوتر) را در کالج مهندسی دانشگاه Utah گذراند.
درگیرشدن وی با پروژه‌های تحقیقاتی ARPA باعث شد که حرفهٔ موسیقی و گیتار را ترک کند.
در سال 1970 به مرکز تحقیقاتی شرکت زیراکس (PARC) پیوست.
در آن‌جا روی ایجاد ورک استیشن‌های متصل به شبکه (توسط زبان Smalltalk) کار کرد (ابداعی که بعد ها توسط Apple در کامپیوترهای Lisa و Macintosh استفاده شد).
Kay به نوعی پدر برنامه نویسی شی گرا محسوب می شود (حتی اسم «شی‌گرایی» را نیز وی برای اولین بار استفاده کرد).
ضمنا مفهوم Dynabook (کامپیوترهای کوچک و قابل حمل که با باطری کار کرده و رابط گرافیکی دارند) توسط وی ابداع شد، چیزی که ما امروزه با نام لپ تاپ یا Tablet می‌شناسیم (در واقع پلتفرم OLPC یا «یک لپ تاپ برای هر بچه» نشات گرفته از ایدهٔ Dyna-book است، زیرا کاربران اصلی مدنظر Kay کودکان بودند).
ضمناً مفهوم رابط کاربر گرافیکی با پنجره‌هایی که می‌توانند روی همدیگر قرار بگیرند (همپوشانی داشته باشند) برای اولین بار توسط ایشان ابداع شد.
وی پس از 10 سال کار در Xerox PARC به مدت 3 سال محقق اصلی Atari بود.
او مدتی نیز در Apple و نیز Walt Disney و بعدها HP به عنوان محقق برجسته فعالیت کرد.
در سال 2001 موسسه تحقیقاتی Viewpoints Research را تأسیس کرد که یک مؤسسه غیر انتفاعی برای کودکان، آموزش و توسعه نرم افزارهای پیشرفته است.
وی در سال 1995 همراه با چند نفر دیگر برنامه چند رسانه‌ای و پویای Squeak را بنا نهاد (و هنوز نیز روی این پروژه کار می‌کند)
Kay اعتقاد دارد که هنوز انقلابی در برنامه‌نویسی به وجود نیامده است و تمام ایده‌های خوب هنوز پیاده‌سازی نشده‌اند و نظر خود را در کنفرانس OOPSLA 97 و نیز سخنرانی‌اش برای دریافت جایزهٔ ACM Turing Award تحت عنوان «انقلاب کامپیوتر هنوز اتفاق نیفتاده است» بیان کرد.

برخی از جایزه‌هایی که وی دریافت کرده است عبارتند از:

- جایزهٔ Udk-01 در برلین در سال 2001 به خاطر کارهای بنیادی وی بر روی GUI
- جایزهٔ ACM Turing به خاطر کار بر روی برنامه‌نویسی شی گرا در سال 2003
- دکترای افتخاری از انستیتو سلطنتی فن آوری استکهلم در سال 2002

• دکترای افخاری از انستیتو فن آوری Georgia در سال 2005

• دکترای افتخاری از کالج کلمبیای شیکاگو در سال 2005

• دکترای افتخاری از دانشگاه واترلو در سال 2008

• پروفسور افتخاری دانشگاه هنرها در برلین

• عضو افتخاری
- آکادمی هنر ها و علوم امریکا
- آکادمی ملی مهندسی امریکا
- جامعهٔ سلطنتی هنر
- موزهٔ تاریخ کامپیوتر
- ACM

• جایزهٔ NEC Computers & Communication Foundation Prize

وی همچنان کیبرد و گیتار می‌نوازد. Alan از 3 سالگی قادر به خواندن بود و دربارهٔ این مورد می‌گوید:

«تا قبل از رفتن مدرسه، صدها کتاب خوانده بودم. از همان ابتدا می‌دانستم که آن ها به من دروغ می‌گویند زیرا قبلاً با نقطه نظرهای دیگری آشنا شده بودم. مدرسه اساساً یک نقطه نظر دارد: دید معلم و یا دید کتاب‌های درسی. آن‌ها ایدهٔ «چندین نقطه نظر متفاوت» را دوست ندارند، و این برای من یک نبرد بود!»

اگر اشخاص خاصی هستند که مایلید با آن ها بیشتر آشنا شوید، با من تماس بگیرید تا در مورد آن‌ها نیز مطلبی بنویسم.

مهدی عسگری mehdi.asgari@yahoo.com

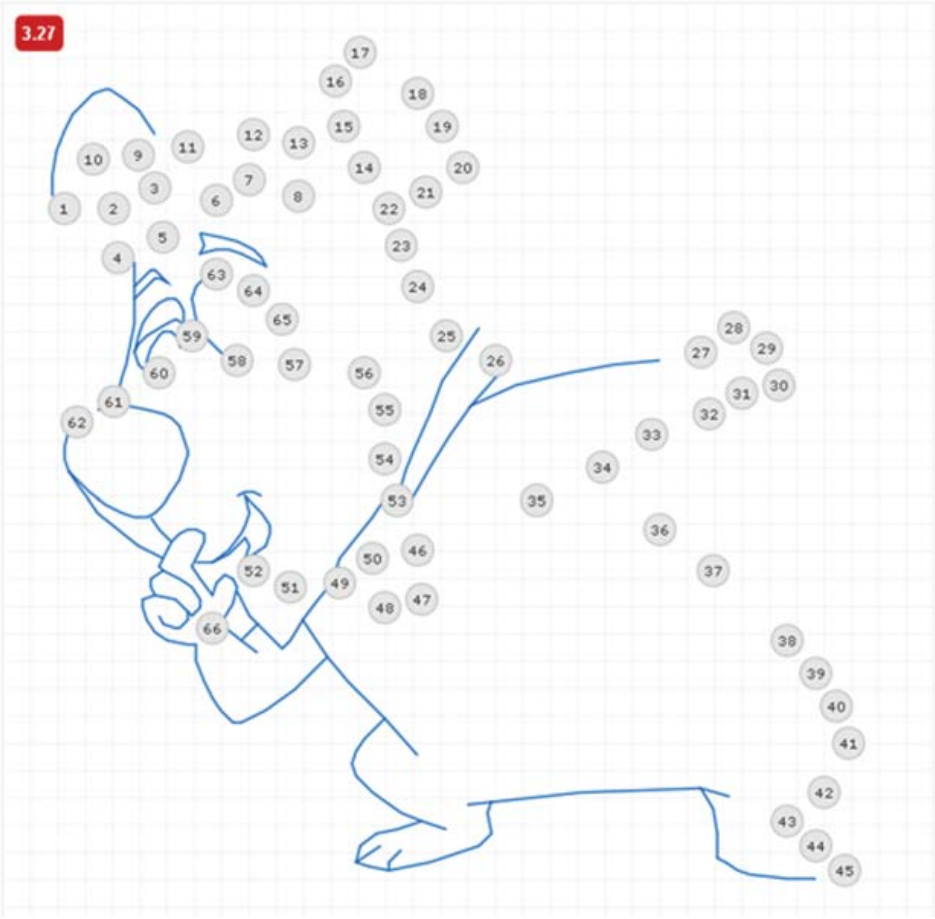
نویسنده: مهدی موسوی

جور چین نقطه به نقطه با استفاده از HTML 5.0و jQuery

mehdi_mousavi@hotmail.com

ایده اصلی این مقاله هنگامی به ذهنم خطور کرد که در اینترنت در حال جستجوی تصویری برای رنگ آمیزی بودم (البته نه برای خودم، بلکه برای پسر 28 ماهه‌ام). در این مقاله، به HTML 5 و برخی از توانایی‌های آن اشاره گذرایی خواهم کرد، اما این مقاله هرگز منبعی برای یادگیری HTML 5 نیست. اگر می‌خواهید از چند و چون HTML 5 مطلع شوید و این‌که چه امکاناتی در اختیارتان قرار می‌دهد، لطفا به سایت « در HTML 5 غوطه‌ور شوید» رجوع کنید.

تصویر زیر، نمایی از آنچه در این مقاله تولید خواهیم کرد را نشان می‌دهد:



هدف آن است که کاربر، Mouse خود را روی دایره‌های خاکستری رنگ (که از این پس با نام «نقاط فعال» از آن‌ها یاد خواهیم کرد)، با شروع از دایره شماره یک و به صورت صعودی حرکت داده تا نقاط به ترتیب به یکدیگر متصل شده و شکل فوق کامل شود. عددی نیز در بالای سمت چپ صفحه قرار دارد که بیانگر میزان زمان سپری شده بر حسب ثانیه بوده و به محض تکمیل شدن شکل، متوقف خواهد شد. ساخت چنین برنامه‌ای، متشکل از مراحل مختلفی است که در ذیل به آن‌ها اشاره خواهم کرد.

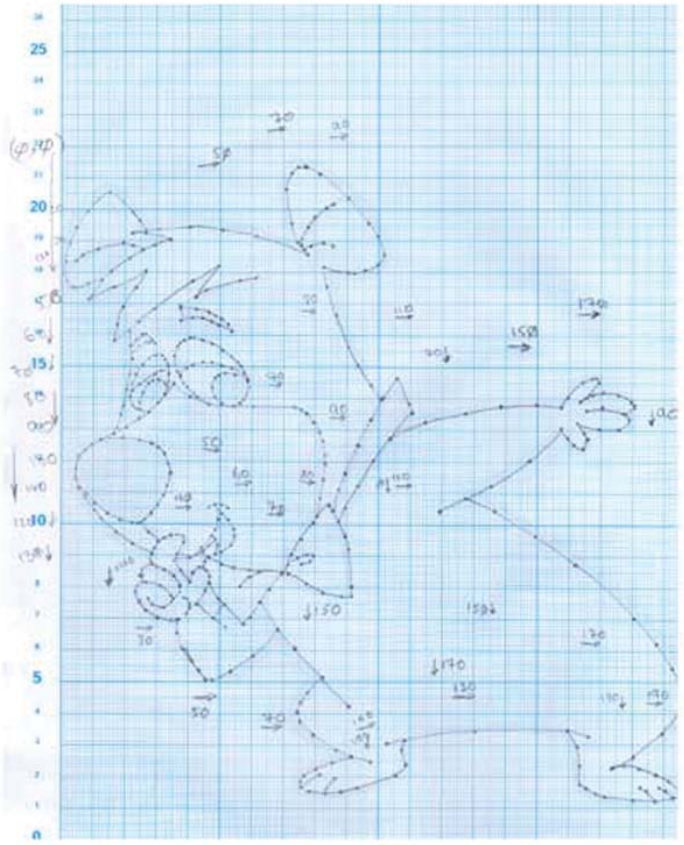
گام اول – یافتن تصویری برای رنگ‌آمیزی

به منظور یافتن تصویری برای رنگ‌آمیزی، اینترنت را جستجو کردم و در نهایت «خرس بوبو» که شخصیتی در کارتون «یوگی و دوستان» بود را از این صفحه انتخاب کردم. لطفا در استفاده از تصاویر این سایت دقت کنید، زیرا همگی آنها مشمول قانون حق کپی هستند.

گام دوم – چاپ تصویر روی کاغذ میلیمتری

هدف از این گام، تبدیل تصویر بوبو از Raster به Vector است. این تبدیل به روش‌های متفاوت امکان‌پذیر است. همانطور که توسط دیگران نیز ذکر شده است، نرم افزار Adobe Illustrator قادر است تا این تبدیل را انجام دهد. اما از آنجا که نرم افزار تبدیل رایگانی برای این منظور نیافتم، تصمیم گرفتم تا تصویر را بر روی کاغذ میلی‌متری کشیده و کار را دستی انجام دهم.
اگر به کاغذ میلیمتری دسترسی ندارید، نگران نباشید. می‌توانید از نرم افزار GridOmatic برای ایجاد چنین کاغذی استفاده کنید.

گام سوم – پشتیبانی از Canvas در Internet Explorer



اگر چه نسل IE6 توسط پسرش IE7 و دختر بزرگش IE8 هنوز حفظ شده است، اما حقیقتا بر این باورم که شرکت مایکروسافت باید به نیازهای جامعه در مرورگر خود بها دهد زیرا در غیر این‌صورت، بخش عمده‌ای از سهام خود را در بازار از دست خواهد داد. همانطور که تاکنون متوجه شده‌اید، IE7 و IE8 از عنصر Canvas در HTML5 پشتیبانی نمی‌کنند. با این وجود ما قصد نداریم کاربران این مرورگر را نادیده بگیریم، پس باید به‌گونه‌ای این مشکل را حل کنیم.
برای حل این مشکل، وب را جستجو کردم و در نهایت از وجود ExplorerCanvas مطلع شدم. این کد جاوا اسکریپتی، قابلیت Canvas را برای Internet Explorer به ارمغان می‌آورد (همانطور که مطلع هستید، IE سالیان متمادی است که از سیستمی کاملاً متفاوت تحت عنوان VRML برای مقاصد گرافیکی پشتیبانی می‌کند. Explorer-Canvas از همین توانایی، برای آوردن بسیاری از قابلیت‌های Canvas در HTML5 به IE استفاده کرده است).
برای استفاده از ExplorerCanvas که نرم افزاری Open Source است، کافی است با استفاده از Tag زیر، این کد را در صفحه قرار دهید:

```
<!--[if IE]>
<script src="excanvas.js"></script>
<![endif]-->
```

گام چهارم – HTML

مطلب خاصی در مورد این کد HTML وجود ندارد. این کد HTML فقط حاوی یک عنصر Canvas در ابعاد 680x680 پیکسل و دو برچسب div که به شناسه timer و کلاس filled مزین شده است، می‌باشد.

```
<canvas id="graph" width="680" height="680">
</canvas>
<div id="timer"></div>
<div class="filled"></div>
```

در حقیقت کلاس filled و عنصر مربوطه باید حذف شوند. حقیقت آن است که هنگام طراحی برنامه، مساله رنگ آمیزی چشمان، بینی و لبِ بوبو را مد نظر قرار نداده بودم، در نتیجه در اتمام کار متوجه شدم که قادر به رنگ آمیزی این سه عضو از بدن او نیستم. بنابراین تصمیم گرفتم تا با حقه‌ای این کار را انجام دهم. اما اگر شما از قبل برای این منظور برنامه‌ریزی کرده باشید، دیگر نیازی نیست تا این کار را انجام دهید و می‌توانید از متد fill برای پر کردن یک مسیر بسته استفاده کنید.

گام پنجم – CSS

```
.circle {
    background: url('circle.png');
    width: 24px;
    height: 24px;
    text-align: center;
    font-size: .8em;
    line-height: 24px;
    display: block;
    position: absolute;
    cursor: pointer;
    color: #333;
    z-index: 100;
}
```

```
.filled {
    background: url('filled.png');
    position: absolute;
    width: 172px;
    height: 251px;
    display: none;
}



#timer {
    position: absolute;
    font-family: Arial;
    font-weight: bold;
    font-size: 1em;
    background: #c00;
    color: #fff;
    padding: 5px;
    text-align: center;
    -moz-border-radius: 5px;
    -webkit-border-radius: 5px;
    font-variant: small-caps;
}
```

بسیار خوب! Style Sheet مورد استفاده در این مقاله، ساده است. این CSS حاوی کلاس circle است که وظیفه تعیین سبک نمایشی «نقاط فعال» (دایره های خاکستری رنگ شماره دار در شکل اول) را به عهده دارد.

Class Selector دیگری تحت عنوان filled برای استفاده از حقه‌ای که بدان اشاره کردم در نظر گرفته شده است. همچنین ID Selector ای تحت عنوان graph جایگاه Canvas را روی صفحه تعیین می‌کند.

در نهایت، CSS Selector ای با نام timer تعریف شده است که وظیفه تعیین سبک نمایشی ثانیه شمار نمایان در بالای سمت چپ صفحه بر دوش آن نهاده شده است.

گام ششم – JavaScript

ابتدا باید مختصات نقاط تشکیل دهنده خرس بوبو را از تصویر اولیه جدا کرده و آن‌ها را Vectorize می‌کردم. بدین منظور، تصمیم گرفتم آرایه‌ای از نقاط موجود در صفحه که برای نمایش تصویر بوبو به آن‌ها نیاز بود ایجاد کنم. مختصات هر نقطه توسط زوج (x, y) تعیین می‌شود.

نقطه‌ای که حاوی مختصات (-1, -1) است بیانگر اتمام یک پاره

خط بوده، و نشان می‌دهد که باید قلم را از روی صفحه برداشته، و به نقطه دیگری از صفحه انتقال دهیم. سپس از نقطه دیگری با مختصات (x, y) شروع کرده و این روند را تا کشیده شدن کامل شکل ادامه می‌دهیم.

```
var dots = [193, 207, 188, 208, 185, 205, 183, 204, -1, -1, 188, 208, 181, 208, 172, 207, 169,
206, 166, 204, 164, 203, 164, 197, 163, 191, 160, 186, 167, 188, -1, -1, 160, 186, 131, 187,
103, 190];
```

برای نمونه، آرایه فوق را در نظر بگیرید. این آرایه نشان می‌دهد که قصد کشیدن پاره خطی از مختصات (193, 207) به (188, 208) به (185, 205) به (183, 204) را داریم. نقطه بعدی (-1, -1) است. اکنون چه باید کرد؟ منظور از این مختصات آنست که کشیدن پاره خط اول پایان یافته، پس باید قلم را از روی کاغذ بلند کرده و آن را به مختصات بعدی یعنی (188, 208) منتقل کنیم. سپس قلم را در مختصات بعدی روی کاغذ قرار می دهیم و شروع به ترسیم شکل می‌کنیم. به عبارت دیگر، خطی از (188, 208) به (181, 208) و الی آخر ترسیم می‌کنیم.

سپس تصمیم گرفتم تا پنجره‌ای مشبک به رنگ خاکستری کمرنگ در پشت صفحه قرار دهم که منجر به نوشتن تابع displayGrid شد:

```
function displayGrid(ctx) {
    var gridSize = 20, width = 900;
    for (var ypos = 0; ypos < width; ypos += gridSize) {
        ctx.moveTo(0, ypos);
        ctx.lineTo(width, ypos);
    }
    for (var xpos = 0; xpos < width; xpos += gridSize) {
        ctx.moveTo(xpos, 0);
        ctx.lineTo(xpos, width);
    }

    ctx.strokeStyle = "#eee";
    ctx.lineWidth = .7;
    ctx.stroke();
}
```

سپس نیاز بود نقاط فعال را بصورت تصادفی انتخاب کنم، تا اگر کاربر صفحه را Refresh کند، نقاط فعال خاکستری رنگ تغییر کنند. از طرف دیگر، هر دایره باید به‌گونه‌ای انتخاب شود که هیچ دو دایره‌ای با یکدیگر تلاقی نداشته باشند.

```
function addPoint(index, x1, y1) {
            for (var i = 0; i < points.length; i++) {
                var x2 = points[i].x, y2 = points[i].y;
                var d1 = Math.sqrt(Math.pow(y2 - y1, 2) + Math.pow(x2 - x1, 2));
                var d2 = radius * 2 + 2;
                if (d2 > d1) {
                    return false;
                }
            }

            points.push({ 'x': x1, 'y': y1, 'index': index });
            return true;
        }
```

وظیفه تابع فوق آن است که تشخیص دهد آیا یک نقطه شرایط تبدیل شدن به یک «نقطه فعال» را دارد یا خیر و انتظار سه پارامتر در ورودی خود را دارد: index، x1 و y1.

index نشان‌دهنده اندیس نقطه در آرایه dots است، در حالی‌که پارامترهای (x1, y1) بیانگر مرکز دایره مورد نظر است. این تابع، از آرایه points برای اطمینان از عدم تلاقی دو «نقطه فعال» استفاده می‌کند. اگر این تابع از امکان تبدیل یک نقطه، به نقطه فعال اطمینان حاصل کند، اندیس نقطه و مختصات آن را در آرایه points حفظ خواهد کرد و true را بعنوان مقدار بازگشتی به تابع فراخواننده باز می‌گرداند. در غیر این‌صورت، مقدار false برگردانده خواهد شد. اکنون باید ببینیم که از پاره خطها و نقاط موجود کدامیک را برای محو کردن و تبدیل آن به «نقاط فعال» انتخاب کنیم. این وظیفه‌ای است که بر دوش getRandomPoints گذاشته شده است:

```
function getRandomPoints(totalPoints)
{
   var indexes = new Array();
   for (var i = 0; i < totalPoints; i++)
   {
    var done = false;
    while (!done)
    {
      var index =
      Math.floor(Math.random()*dots.length);
      if (index % 2 != 0)
      {
        index++;
        if (index > dots.length)
        {
           continue;
        }
      }
      if (!contains(indexes, index))
      {
       indexes.push(index);
       done = true;
      }
    }
   }
 return indexes.sort(function(a, b)
  {return a - b;}
  );
}
```

که در آن totalPoints نشان‌دهنده تعداد نقاط مورد استفاده است. دقت کنید که این عدد بیانگر تعداد نقاط فعال نیست. در نهایت ما باید از انتخاب n نقطه انتخاب شده اطمینان حاصل کنیم، در حالی‌که شرایط زیر نیز برقرار باشد:

۱. حتی الامکان نقاط نزدیک به یکدیگر انتخاب شوند.

۲.هیچ دو نقطه‌ای با دیگری تلاقی نداشته باشد.

شرط اول توسط متغیری با نام hops در کد اعمال شده است در حالیکه شرط دوم در تابع addPoint ای که قبلا به آن اشاره کردیم، محقق شده است.

باقی کد ساده است؛ تنها باید از رخداد mouseover «نقاط فعال» مطلع شویم و اطمینان حاصل کنیم که کاربر Mouse را بصورت ترتیبی روی «نقاط فعال» حرکت می دهد و با انتخاب هر «نقطه فعال» آن را با حرکتی از پیش تعیین شده از صفحه خارج کنیم.

فهمیدن Source Code این مقاله ساده است، و برای راحتی شما در یک فایل zip قرار گرفته است که می توانید آن‌را از اینجا دریافت کنید. در پایان، از همسرم که مرا در Vectorize کردن نقاط تصویر اولیه یاری کرد و بدون کمک وی نوشتن این مقاله میسر نبود، متشکرم.

منبع:
http://blog.mehdi.biz/2010/02/having-fun-with-jquery-and-html-5.html

این مقاله اولین بار حدود 3 ماه پیش در معرض دید عموم قرار گرفت و به‌عنوان یکی از مفیدترین مطالب آموزشی ماه از دید w3Avenue شناخته شد.

مترجم: سیاوش شهرویان

بهینه‌سازی پردازش رشته‌ها در VB6

در ویژوال بیسیک، پردازش رشته، اگر از روشی اشتباه انجام شود به کندی صورت می‌گیرد. شما می‌توانید با بکارگیری چند قانون ساده، کارایی عملیات رشته‌ای را به میزان قابل توجهی افزایش دهید.

بخش اول

رشته‌های سریع‌تر با VB6

ویژوال بیسیک 6 دامنه وسیعی از توابع مفید پردازش رشته را، مانند Asc ، Len ، Right ، Mid ، Left و InStr. ارائه می‌دهد. این توابع راهکار قدرتمندی برای کار با رشته‌ها فراهم می‌کنند. متأسفانه بسیاری از توابع کار با رشته‌ها از نظر سرعت بهینه‌سازی نشده‌اند. به همین دلیل است که برنامه‌های VB6 کندتر از آنچه که باید اجرا می‌شوند.

شما می‌توانید با برنامه‌نویسی هوشمندانه بر خیلی از محدودیت‌های سرعت فائق آیید. این مقاله، تعدادی ترفند مفید را برای افزایش سرعت برنامه‌ها نشان می‌دهد؛ ترفندهایی که در آنها تنها از دستورات VB استفاده شده، بدون نیاز به هیچ فایل و فراخوانی هیچ تابع APIای.

توابع و عملگرهای vb6 استفاده شده در این مقاله:

```
Asc, AscW, Chr$, ChrW$, Error$,
Format$, Hex$, InStr, InStrB,
LCase$, Left$, Len, LenB, LTrim$,
Mid$, Oct$, Replace, Right$,
RTrim$, Space$, Str$, StrComp,
String$, Trim$, UCase$, ByRef,
.$ ,ByVal, Like
```

چه کسی باید این مقاله را بخواند؟

این نکات بر اساس ویژوال بیسیک 6.0 و رشته‌های با طول متغیر بنا شده. این نکات برای برنامه‌هایی که حجم زیادی از داده‌های متنی را می‌خوانند، تجزیه، یا دستکاری می‌کنند، مفید است. اگر شما در برنامه‌های‌تان این فراخوانیها را هزاران یا صدها هزار بار اجرا می‌کنید، استفاده از این تکنیک‌ها، برای افزایش کارایی برنامه مهم است. اگر فقط گه‌گاهی عمل خواندن و نوشتن در رشته‌ها را آن‌هم بیرون از حلقه‌ها انجام می‌دهید، این نکات چندان کمکی به شما نخواهند کرد. اگرچه این نکات بهترین عملکرد را در VB6 دارند، اما بعضی از آنها عمومی بوده و میتوانند روی نسخه‌های قبلی و بعدی هم مورد استفاده قرار گیرند.

چرا رشته‌های VB6 بسیار کند هستند؟

شاید بتوان گفت بزرگترین مشکل این است که VB از داده‌های رشته‌ای در زمان انجام برخی عملیات، کپی تهیه می‌کند. حتی زمانی که شما رشته‌ای را فقط می‌خوانید (بدون قصد هیچ گونه تغییری)، در نهایت می‌توانید بسادگی کپی‌های زیادی را ایجاد کنید. اگر بخش عظیمی از برنامه شما به پردازش رشته‌ها اختصاص پیدا کرده باشد، عمل کپی‌برداری از لحاظ زمانی هزینه‌بر است. دلیل بعدی این است که برخی از توابع پر کاربرد به طور سرراست و مستقیم پیاده‌سازی نشده‌اند. این توابع ممکن است کاری بیش از آنچه که شما لازم دارید انجام دهند، خوشبختانه اغلب اوقات می‌توانید، توابع پیشرفته‌تری را به سادگی و سرعت جایگزین کنید.

بهینه‌سازی رشته‌های خالی (empty)

آیا عبارت «» اغلب در کدهای شما به چشم می‌خورد؟ مراقب باشید! زمان زیادی از CPU برای یک چنین رشته‌هایی تلف می‌شود. آزمایش و تخصیص رشته‌های خالی محل خوبی برای بهینه‌سازی است.

بررسی خالی بودن رشته

اغلب لازم است تا خالی بودن رشته‌ای را بررسی کنیم. روش‌های معمول برای این کار از قرار زیر هستند:

```
If Text$ = "" Then
If Text$ <> "" Then
```

اما، VB دستورات معادل زیر را بسیار سریع‌تر اجرا می‌کند:

```
If LenB(Text$) = 0 Then
If LenB(Text$) <> 0 Then
```

اساساً ریسک این جایگزینی مجانی است، کد شما مثل قبلاً اجرا می‌شود، اما سریع‌تر. تابع LenB در پیاده سازی VB سریع است. LenB معادل تابع Len، اما بر حسب بایت است. درواقع خودِ تابع Len از روی LenB/2 پیاده‌سازی می‌شود. این باعث می‌شود که LenB سریع‌تر از Len عمل کند. بنابراین باید هر جا امکانش هست از این تابع استفاده کنید. VB3 و VB.NET فاقد تابع LenB هستند و شما بایست در آن‌ها از تابع Len استفاده کنید.
توجه داشته باشید که ما از عملگر <> استفاده می‌کنیم و نه < ، عملگر <> بسادگی عدم تساوی را بررسی می‌کند، درحالی‌که عملگر > ارزش بیشتر را بررسی می‌کند. چنانکه حاصل Len/ LenB هیچگاه منفی نخواهد شد. با خیال راحت می‌توانیم امتحان کنیم.

تخصیص یک رشته خالی به یک متغیر

این راه معمول برای خالی کردن یک متغیر رشته‌ای است:

```
Text$ = ""
```

چه اصرافی! اول از همه این‌که، هر وقت شما رشته "" را بکار می‌برید، شش بایت از حافظه RAM را اشغال می‌کنید. به معادل جایگزین آن توجه کنید:

```
Text$ = vbNullString
```

خب، این چیست؟ vbNullString، یکی از ثابت های خاص VB است که معنی یک رشته تهی (Null) را میدهد. لفظ "" معنی رشته خالی (empty) را میدهد. تفاوت مهمی بین این دو وجود دارد. یک رشته خالی، یک رشته حقیقی است. اما یک رشته تهی، هیچ است. این تنها یک صفر است. اگر با زبان C آشنایی دارید، vbNullString معادل همان NULL در C است.
برای بسیاری از مقاصد، vbNullString معادل همان "" در VB است. تنها تفاوت مهم این است که vbNullString سریعتر تخصیص می‌یابد، سریعتر پردازش می‌شود، و حافظه کمتری را مصرف می‌کند.
اگر شما از API ها و ابزارهایی غیر از API های خود VB استفاده کردید. قبل از توزیع و پخش برنامه‌تان یک بار فراخوانی‌ها را با vbNullString امتحان کنید. تابعی که شما فراخوانی می کنید ممکن است یک رشته تهی (NULL) را بررسی نکند؛ در این‌صورت ممکن است برنامه شما crash کند. توابعِ غیر از خود VB بایست قبل از پردازش یک مقدار رشته‌ای، تهی بودن آن را بررسی کنند. شاید تابعی که شما فراخوانی می کنید این کار را انجام ندهد؛ در همچین مواردی از "" استفاده کنید. معمولاً API ها از vb-NullString پشتیبانی می‌کنند و حتی می‌توانند با وجود آن بهتر عمل کنند.

لطفا هرگز از نوع variant استفاده نکنید

این نکته ساده‌ای ست، اما اغلب اوقات نادیده گرفته می‌شود. تمام متغیرها، پارامترها و توابع، باید نوع داده تعریف شده داشته باشند. اگر یک داده، رشته‌ای است، پس نوع داده باید string تعریف شده باشد. اگر نوع داده را مشخص نکنید، از یک variant استفاده میکنید. نوع داده variant، کاربرد خودش را دارد، نه این‌که برای پردازش رشته استفاده شود. استفاده از یک variant در اکثر موارد به معنی از دست دادن کارایی است.
پس همین حالا دستورات Option Explicit را اضافه کنید و همه متغیرها را با نوع داده مناسب تعریف کنید. توابع‌تان را بازبینی کنید، و مطمئن شوید که نوع داده برگشتی‌شان تعریف شده است.

$هایی که باعث سرعت بخشیدن به برنامه شما می‌شوند

اگر از توابع زیر برای کار با رشته‌ها استفاده می‌کنید، هیچ کدام از آن‌ها بهینه نیستند.

```
Left(), Mid(), Right(), Chr(),
,()ChrW
UCase(), LCase(), LTrim(), RTrim(),
,()Trim
Space(), String(), Format(), Hex(),
,()Oct
Str(), Error
```

اینها همان توابع ترسناک variant هستند؛ یک variant می‌گیرند و یک variant برمی‌گردانند. اگر در حال پردازش داده‌های Variant هستید، استفاده از این توابع مشکلی ندارد. این مورد در برنامه‌نویسی بانک‌های اطلاعاتی، که ممکن است ورودی شما شامل مقادیر Null شود، وجود دارد. پس آن همه استفاده از variant در پردازش رشته‌ها برای چیست؟ همگی زائد و اضافی هستند. اگر با رشته‌ها سر و کار دارید، variant را فراموش کنید؛ درعوض از نسخه رشته‌ای این توابع استفاده کنید:

```
Left$(), Mid$(), Right$(), Chr$(),
,()$ChrW
UCase$(), LCase$(), LTrim$(),
,()$RTrim$(), Trim
Space$(), String$(),  Format$(),
,()$Hex$(), Oct
$Str$(), Error
```

متغیرهای $دار، چیزهای خوبی نیستند

در رابطه با متغیرهایی با علامت $ چطور؟ آیا این هم مثل نام توابع می‌باشد؟ آیا به بهینه شدن دستورات کمک می‌کند؟

خیر، علامت $ فقط در توابع فوق‌الذکر مفید خواهد بود. در این مقاله ما نیز از علامت $ برای معرفی متغیرهای رشته‌ای، مثل Text$، استفاده کرده‌ایم. این کار فقط به منظور حفظ خوانایی کد انجام شد، و یک نکته بهینه‌سازی نمی‌باشد. ما علامت $ را برای متغیرهای رشته‌ای توصیه نمی‌کنیم. در برنامه‌هایتان بایست متغیرها (و توابع) را با یک نوع داده واقعی تعریف کنید. مانند این: Dim Text As String. علامت دلار به عنوان یک شاخص نوع، منسوخ شده است.

استفاده از replace، آری یا خیر؟

نکته زیر ممکن است بدیهی باشد، اما برای ما این‌طور نبود. اگر نمی‌خواهید چیزی را جایگزین کنید، هیچ دلیلی ندارد که تابع Replace را فراخوانی کنید. تابع Replace به کندی اجرا می‌شود. این تابع همیشه یک کپی از رشته ورودی می‌گیرد، حتی اگر جایگزینی‌ای انجام نگیرد. کپی گرفتن هم عمل کندی است. اگر احتمال می‌دهید که جایگزینی انجام نشود، ابتدا بررسی کنید که چیزی برای جایگزینی وجود داشته باشد (مثلاً با استفاده از تابع InStr یا InStrB)

```
If InStr(Text$, ToBeReplaced$) <> 0 Then
   Text$ = Replace(Text$, ToBeReplaced$, "xyz")
End If
```

اگر احتمال می‌دهید یا مطمئن هستید که جایگزینی انجام می‌شود، آن‌گاه نیازی به فراخوانی InStr نیست. چون در این‌صورت فقط یک سربار اضافه می‌کند. لزومی ندارد که برای فراخوانی این تابع، علامت $ را به انتهای آن بیافزایید. این یک استثنا برای قاعده $ است که در بالا ذکر شد.

استفاده از ثابت‌ها

ثابت‌های رشته‌ای موجود

بجای فراخوانی توابع Chr$()/ChrW$() برای مقادیر عددی زیر، از ثابت‌های رشته‌ای از پیش تعیین شده برای این کار استفاده کنید. این ثابت‌ها، شما را از فراخوانی تابع رهایی می‌بخشند.

```
vbNullChar      0
vbBack          8
vbTab           9
vbLf           10
vbVerticalTab  11
vbFormFeed     12
vbCr           13
vbCrLf         13+10
vbNewline      13+10
34             ""
```

بنا به دلایلی، vbNewline ذره‌ای از vbCrLf سریع‌تر است.

آخرین نمونه ("") در واقع یک ثابت نیست، اما کاراکتری ویژه است. برای نشان دادن یک نقل قول می‌توانید از این کاراکتر در هر جایی از یک رشته استفاده کنید. معادل آن(Chr(34 است، که در بعضی از نسخه‌های اولیه بیسیک که نحو «» در آن‌ها وجود نداشت، ضروری بود.

همچنین برای اجتناب از فراخوانی توابع Chr$()/ChrW$() می‌توانید مقادیر ثابت دیگری را تعریف کنید. اگر کاراکتر در محدوده 0 تا 31 اسکی قرار دارد، قبل از استفاده باید آن‌ها را به شکل متغیر تعریف کنید، و سپس کاراکتر صحیح را به آن نسبت دهید:

```
Dim BEL As String
BEL = ChrW$(7)  ' The BEL character, or ^G
```

برای سایر کاراکترها می‌توانید بسادگی از یک ثابت استفاده کنید:

```
Const Percentage = "%"
```

توابع غیر ضروری Asc/AscW

پر واضح است که فراخوانی Asc/AscW روی یک ثابت رشته‌ای کاملاً غیرمنطقی است؛ چرا که مقدار بازگشتی هرگز تغییر نمی‌کند. بجای ("A"Asc از مقدار 65 استفاده کنید. حتی بهتر است یک ثابت همانند زیر تعریف کنید:

```
Const ascA = 65
```

و حالا برای خوانایی بیشتر، بجای عدد 65 از این استفاده کنید. در واقع، VB.NET عبارت «Asc«A را بهتر کامپایل می‌کند. اما چون ما در مورد VB6 بحث می‌کنیم، باید این ثابت را تعریف کنیم.

ثابت‌های رشته‌ای خودتان

اگر در پروژه شما، یک رشته در جاهای مختلف عیناً تکرار شده است، در فایل اجرایی نیز چندین بار تکرار می‌شوند. این مورد مربوط به VB6 می شود (VB.NET رشته‌های تکراری را در حین کامپایل پیوند می زند).

شما می توانید با تعریف رشته‌ها به صورت ثابت و ارجاع دادن ثابت‌ها در هر جایی‌که نیاز دارید، بهینه‌سازی کنید. بدین طریق، هر

ثابت را فقط یک بار ذخیره می‌کنید. گذشته از این، اگر شما به فکر بومی‌سازی برنامه‌تان باشید، لیست مفیدی از ثابت‌های رشته‌ای دارید که می‌توانید به مترجم بدهید. اما یک استثنای ناخوشایند وجود دارد؛ و آن این‌که نمی‌توان با تعریف یک ثابت از روی ثابت‌های دیگر، فضایی را صرفه‌جویی کرد.

```
Const MSG1 = "Hello, "
Const MSG2 = "world!"
Const MSG3 = MSG1 & MSG2
```

در واقع، در این حالت، متن مشابه را دو بار در فایل اجرایی خواهید داشت. MSG2 ،MSG1 و MSG3، هر سه ذخیره خواهند شد. و این چیزی نیست که می‌خواستید به آن برسید! اگر می‌خواهید فضا را صرفه‌جویی کنید، MSG1 و MSG2 را در زمان اجرا الحاق کنید. برای سرعت بیشتر، آن را در یک متغیر برای استفاده مجدد ذخیره کنید. همچنین توجه داشته باشید که نکته فوق تنها شامل رشته‌ها می شود. ثابت‌های عددی نیز محاسبه و ذخیره می‌شوند، اما ثابت‌های رشته‌ای فضای بیشتری را برای ذخیره شدن می‌طلبند (6 بایت سربار + 2 بایت به ازاء هر کاراکتر).

String literal analysis یک تحلیلگر پروژه است که قابلیت گزارش رشته‌های تکراری را دارد.

ذخیره‌سازی رشته‌ها در فایل‌های .res

وقتی پروژه را برای تولید فایل اجرایی کامپایل می‌کنید، VB عبارات رشته‌ای را به صورت Unicode ذخیره می‌کند، که نیازمند 2 بایت فضا به ازاء هر کاراکتر است. اگر می‌خواهید رشته‌های‌تان را به صورت 1 بایت برای هر کاراکتر ذخیره کنید از فایل‌های منبع (resource files) استفاده کنید، اگر میزان داده‌های رشته‌ای زیاد باشد، این کار ممکن است حجم فایل اجرایی را به میزان قابل توجهی کاهش دهد. توجه داشته باشید که باید رشته‌ها را به صورت custom resource ذخیره کنید (فرمت باینری)، نه در یک resource معمولی از نوع string table Unicode. برای اضافه کردن یک فایل متنی به صورت custom resource در پنجره Resource Editor روی دکمه Add Custom Resource کلیک کنید.

فایل‌های منبع، برای ذخیره‌سازی رشته‌های طولانی، رشته‌های چند خطی و رشته‌هایی برای بومی‌سازی، نیز به سهولت قابل استفاده هستند. هشدار خطا: اگر رشته‌ها را به صورت یک منبع سفارشی ذخیره می‌کنید، مطمئن شوید که رشته‌هایی از مجموعه کاراکترهای اسکی (0-127) هستند. همچنین مطمئن شوید که همه کاربران از کدپیجی استفاده کنند که شما استفاده می‌کنید. در غیر این‌صورت ممکن است متن ها در مکان‌های مختلف، به شکل مختلف ظاهر شوند. به عنوان مثال، یک کاربر یونانی ممکن است بجای کاراکتر Ä کاراکتر Δ را ببیند. راه حل ساده برای این مشکل، ذخیره‌سازی رشته‌ها به صورت Unicode است.

مقایسه رشته‌ها

مقایسه رشته‌ها در مقابل هم، ممکن است بیشتر از آن‌چه که انتظار داشتید به طول انجامد. در ادامه به چند ترفند اشاره می‌کنیم:

مقایسه سمت چپ‌ترین کاراکتر: در اینجا دو راه غیر بهینه برای انشعاب روی کاراکتر اول یک رشته وجود دارد.

```
' Case 1
If Left$(Text$, 1) = "A" Then

' Case 2
Select Case Left$(Text$, 1)
   Case "A"
   Case "B"
End Select
```

و دو راه سریع‌تر برای این کار:

```
' Case 1
If LenB(Text$) <> 0 Then
   If AscW(Text$) = 65 Then

' Case 2
If LenB(Text$) <> 0 Then
   Select Case AscW(Text$)
      Case 65  ' AscW("A")=65
      Case 66
   End Select
```

فراخوانی(AscW) سریع‌تر از آن است که اول($ Left) را فراخوانی کنید، و سپس نتیجه را با رشته‌ی دیگر مقایسه کنید. البته توجه داشته باشید که بکارگیری تابع(AscW) روی یک رشته خالی یا تهی، یک خطای زمان اجرا را نتیجه می‌دهد؛ به همین دلیل شما باید ابتدا(LenB) را فراخوانی کنید، تا احتمال آن را از بین ببرید. فقط زمانی که مطمئن هستید رشته مورد نظر حداقل شامل یک کاراکتر است می‌توانید از(LenB) استفاده نکنید.

استفاده از ساختار Select Case یک امتیاز اضافی محسوب می‌شود. داشتن اعداد یکتا در Caseها به مراتب زمان کمتری نسبت به مقایسه رشته می‌برد.

مقایسه یک کاراکتر در میانه یک رشته

همانند ترفند قبلی، این روش بررسی یک کاراکتر در وسط یک رشته است.

```
If AscW(Mid$(Text$, index, 1)) = 65
Then
```

به خاطر داشته باشید که index باید کوچکتر یا مساوی Len$(Text) باشد، در غیر این‌صورت با خطای زمان اجرا مواجه خواهید شد.

نکته: اگر احتمالا تابع(Mid$) مقدار بزرگی را برمی‌گرداند، پارامتر سوم در(1 , ,)Mid$ برای بهینه سازی ضروری است. بدون این پارامتر، Mid$ می‌تواند زمان زیادی را صرف کپی طولانی و غیر ضروری از Text$ کند. بخش سوم از این مقاله به طور عمیق‌تر وارد این حوزه می‌شود.

مقایسه در قالب باینری

هر گاه که بخواهید می‌توانید از مقایسه باینری استفاده کنید، که از قابلیت‌های پیش‌فرض VB است. مقایسه Text بسیار کند انجام می‌شود. این دستورات سرعت برنامه‌تان را کاهش می‌دهند:

```
Option Compare Text
StrComp(, , vbTextCompare)
InStr(, , , vbTextCompare)
```

اگر یک(StrComp) می‌خواهید که حساس به حروف کوچک و بزرگ نباشد، از تابع(LCase$) استفاده کنید، مخصوصاً اگر فقط در یک پارامتر به آن نیاز باشد:

```
StrComp(Text1$, "abc", vbTextCompare)
' Slower
StrComp(LCase$(Text1$), "abc", vbBinaryCompare)
' Faster
```

در مثال زیر، دو بار فراخوانی LCase$ باعث از بین بردن کارایی بدست آمده در مثال قبل می‌شود.

```
StrComp(LCase$(Text1$), LCase$(Text2$),
        vbBinaryCompare)
```

به یاد داشته باشید که StrComp(,,vbTextCompare) بیش از یک مقایسه معمولی است. این تابع درواقع برای مرتب‌سازی ساخته شده، نه برای مقایسه برابری. در بسیاری از موارد، همچین مقایسه متنی وابسته به زبان (locale-dependant)، یک کار اضافی محسوب می‌شود، و حتی می‌تواند منجر به تولید خطاهای ظریفی شود.

بررسی وجود با InStrB

InStr تابع خوبی برای پیدا کردن یک رشته در درون رشته‌ای دیگر است. معمولاً شما از همین فرم ساده InStr، یعنی نسخه یونیکد، استفاده می‌کنید. اگر فقط می‌خواهید وجود یک رشته در رشته‌ای دیگر را بررسی کنید و زبان آن رشته برای‌تان اهمیت ندارد؛ استفاده از نسخه بایتی – یعنی غیر یونیکد - این تابع (InStrB) یک بهینه‌سازی محسوب می‌شود. در آن صورت، می‌توانید از دستور زیر استفاده کنید:

```
If InStrB(Text$, SearchFor$) <> 0
Then
```

از آن‌جایی که شما فقط مقدار بازگشتی را با صفر مقایسه می‌کنید، نیازی نیست نگران تبدیل اندیس مبتنی بر بایت به اندیس مبتنی بر کاراکتر باشید. اما این، همه داستان نیست. شما باید از نکات زیر مطلع باشید:

- InStrB به طور صحیح روی اندیس‌هایی بر حسب بایت عمل می‌کند، مقدار بازگشتی تابع و همچنین مقدار پارامتر شروع (اولین پارامتر عددی، نه آن چیزی که در بالا نوشته شده)، هر دو بر حسب بایت هستند، و نه بر حسب کاراکتر. هر کاراکتر 2 بایت است. از معادله byteindex = (characterindex * 2) – 1 برای تبدیل شاخص‌ها استفاده کنید. مثلاً اگر در کاراکتر سوم مطابقت داشته باشد، اندیس بایت آن 5 می‌شود.

- InStrB یک تابع با داده‌های بایتی است و استفاده از آن روی ورودی کاراکتر خطرناک است. اگر رشته‌ها شامل کاراکترهایی با مقادیر خارج از محدوده 1-255 باشند، مراقب باشید، زیرا احتمالش هست که InStrB برای این کار شما مناسب نباشد. از آنجایی که InStrB، مبنای جستجو را بایت قرار می‌دهد، می‌تواند تطابق بین کاراکترها را برگرداند:

```
Text$ = ChrW$(&H1234) & ChrW$(&H5678)
' Bytes 34 12 78 56 hex
SearchFor$ = ChrW$(&H7812)
' Bytes    12 78
```

در اینجا، InStrB عدد 2 را که نقطه شروع کاراکترهای متوالی 78 12 است را بر می‌گرداند. در حالی‌که با هیچکدام از کاراکترهای ورودی مطابقت ندارد. این احتمالاً آن چیزی نیست که شما در زمان کار با رشته‌ها می‌خواهید. توجه داشته باشید که حتی اگر رشته‌های شما در مجموعه کاراکترهای اسکی باشند، باز کاراکتر null مشکل‌ساز است (برای مثال InStrB("A" & vbNull-

(Char, vbNullChar) عدد 2 را برمی‌گرداند، نه عدد 3 که انتظارش را داشتید).

- در استفاده از پارامتر vbTextCompare عوارض جانبی وجود دارد. اما vbBinaryCompare برای درک ساده‌تر است.
این یعنی چه؟ تنها زمانی از InStrB برای بهینه‌سازی استفاده کنید که به طور یقین می‌دانید چطور کار می‌کند.

Like

عملگر Like اصلاً سریع نیست. معادله‌هایی برای آن در نظر بگیرید. یک قانون کلی برای این کار نداریم. باید خودتان تفاوت کارایی این عملگر را با معادلی که می‌نویسید، بسنجید. هرچند در اینجا یک راه حل برای این کار ذکر کرده‌ایم. اگر به دنبال یک رشته خاص در رشته‌ای دیگر می‌گردید، این دستور را بکار ببرید:
بجای این دستور:

```
If Text$ Like "*abc*" Then
```

از این دستور استفاده کنید:

```
If InStr(Text$, "abc") <> 0 Then
```

همچنین، اگر به آنچه که انجام می‌دهید آگاهی دارید، می توانید از InStrB استفاده کنید.

پارامترهای رشته‌ای

طرز عمل پارامترهای رشته‌ای با پارامترهای عددی متفاوت است. در پارامترهای رشته‌ای، نحوه ارسال پارامتر، تفاوت فاحشی در کارایی برنامه ایجاد می‌کند.

ارسال رشته‌ها با ارجاع (ByRef)

چطور باید پارامترها را برای صدا زدن از داخل پروژه تعریف کرد؟
ByVal برای پارامترهای رشته‌ای کند است. ByVal در هر فراخوانی یک کپی از رشته تهیه می‌کند. جنبه خوب پارامترهای ByVal این است که در مقابل دستکاری و تغییرات مصون هستند؛ یعنی تغییرات به فراخواننده تابع ارسال نمی‌شود.
ByRef سریع‌تر است، زیرا از رشته‌ها کپی نمی‌گیرد. به همین دلیل شما باید مراقب باشید که اگر قصد ندارید مقداری را به فراخواننده برگردانید، نباید تصادفاً در این پارامتر چیزی بنویسید.

استفاده از ByRef بجای مقدار بازگشتی تابع

حقه بهینه‌سازی دیگری نیز برای برگرداندن یک مقدار رشته‌ای از تابع وجود دارد. برگرداندن یک رشته به عنوان مقدار بازگشتی تابع یک عمل طبیعی است. اما برگرداندن یک رشته در یک پارامتر ByRef سریعتر است. این ترفند زیرکانه هم برای برگرداندن مقدار تابع و هم برای Property Get صدق می‌کند. در اینجا روش معمول این کار(روش کند) نشان داده شده:

```
' Slow:
Property Get Name() As String
   Name = m_sName
End Property
' Slow:
Function Name() As String
   Name = m_sName
End Function
```

اگر می‌خواهید فراخوانی‌های زیادی انجام دهید، این راه سریع‌تر است:

```
' Fast:
Sub GetName(ByRef Name_out As String)
   Name_out = m_sName
End Sub
```

اغلب تصور می‌شود که برگرداندن مقادیر از طریق پارامترها، سبک بدی از برنامه‌نویسی است. به طور عادی رویه‌ها (Procedures) نباید بواسطه تغییر در پارامترهای ByRef شان، عوارض جانبی ایجاد کنند. به هر حال اگر سرعت بیشتری می‌خواهید، باید بعضی وقت‌ها شیوه‌های برنامه‌نویسی مقبول را کنار بگذارید، تا کمی در عملکرد CPU بهینه‌سازی کنید. بنابراین بهینه‌سازی هدفمند می‌تواند توجیهی برای از دست دادن سبک و سیاق باشد. می‌توانید از ByRef استفاده کنید، اما بگویید که چرا دارید این کار را می‌کنید. مثلاً می‌توانید همه پارامترهای خروجی را با رشته out علامت‌گذاری کنید، یا یک توضیحی بنویسید که نشان دهد ByRef به منظور سرعت بخشیدن استفاده شده است.

استفاده از ByVal برای فراخوانی‌های out-of-process

در یک مورد، ByRef کندتر از ByVal عمل می‌کند. و آن زمانی است که یک پارامتر ByRef را به یک سرور خارج از پروسه (out-of-process) ارسال می‌کنید. در اینجا متغیر دو بار جابجا می‌شود. یک بار در زمان ورود به متد، و یک بار در حین بازگشت. این به معنی آن است که از فراخوانی ByVal برای رابطه‌ای عمومی

سرور استفاده کنید.

یافتن نقاط گلوگاه‌ها

اغلب بهبود کارایی که از طریق بهینه‌سازی رشته‌ها بدست می‌آید، ممکن است در واقع به علت پاره‌ای از تغییرات در برخی مکان‌های کلیدی از کد شما باشد. حال سؤال اینجاست که این مکان‌ها کجا هستند و چطور می‌توان آن‌ها را یافت؟

یک مکان حساس در کد، هر دفعه هزاران یا صدها هزار بار اجرا می‌شود. ممکن است در یک حلقه یا در یک الگوریتم بازگشتی باشد. ممکن است شامل یک مشت رویه یا خطوط مشخصی از آن‌ها باشد.

این امکان همواره وجود ندارد که بتوانیم با نگاه کردن به کدها و پیدا کردن حلقه‌ها و بازگشتی‌ها، گلوگاه‌ها را تشخیص بدهیم. یک code profiler مثل VB Watch می‌تواند برای یافتن نقاط حساس کد، مفید باشد. این برنامه تعداد دفعات اجرای کدها را در زمان اجرای برنامه‌تان ثبت می‌کند. زمانی که آماده شد، به شما می‌گوید که کدام رویه‌ها یا خطوط بیشترین بار اجرا شدند، و کدام‌ها بیشترین زمان را برای اجرا گرفتند. این محل‌ها بهترین کاندیدها برای بهینه‌سازی دستی هستند. برای سایر بخش‌های کد، می‌توانید به برخی شیوه‌های بهینه‌سازی خودکار تکیه کنید. مثل سپردن کدها به دست برنامه‌های تحلیل‌گر، که به طور خودکار از طریق جایگزینی فراخوانی‌های بی‌فایده با نمونه بهتر، این کار را انجام می‌دهند.

شیوه‌های بهینه‌سازی پیشرفته رشته‌ها

بخش دوم این مقاله، به معرفی توابع کند و سریع ویژوال بیسیک، فراخوانی API به صورت Unicode، و مدیریت قوی رشته‌های حجیم می‌پردازد.

منبع:

http://www.aivosto.com/vbtips/stringopt.html

Book intro

معرفی کتاب

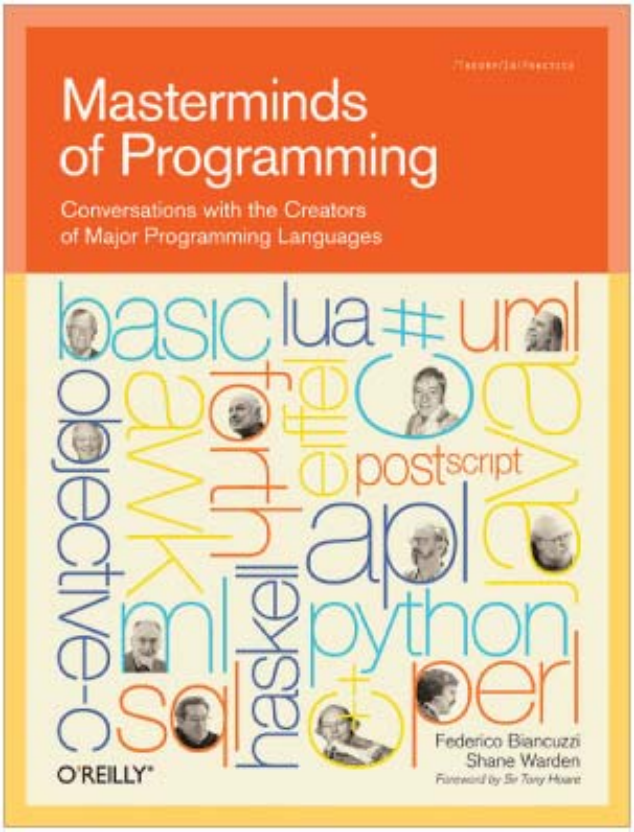
نویسنده: مهدی عسگری

به دلیل طولانی‌تر شدن زمان انتشار مجله، از این به بعد در هر شماره سعی خواهم کرد کتاب‌های بیشتر (و متفاوت‌تری) را معرفی کنم. در این شماره دو دسته کتاب معرفی خواهم کرد.

الف- موفقیت

یکی از روش‌های تسریع در موفقیت، مطالعه سرگذشت انسان‌های موفق و حرفه‌ای است؛ چرا که با مطالعه زندگی این افراد می‌توان الگوهای مشترک زیادی در بین آن‌ها پیدا کرد و در صورت الگو قرار دادن صحیح آن‌ها و به کار بردن این الگوها، نتایج مشابهی را (در مقیاس خود) انتظار داشت.

در این شماره قصد دارم 3 کتاب را که به معرفی و مصاحبه با افراد بزرگ و موفق در حرفه‌ی برنامه‌نویسی (و کلا دنیای نرم افزار) پرداخته اند، معرفی کنم.



نویسندگان: Federico Biancuzzi و Shane Warden (پیش مقدمه: Tony Hoare)

انتشارات O'Reilly

سال چاپ: 2009

تعداد صفحات: 496

این کتاب در 17 فصل، با نویسنده‌های 17 زبان برنامه‌نویسی مصاحبه می‌کند. در هر فصل به تفصیل با خالق زبان مصاحبه شده و در مورد تمام جوانب آن (تاریخچه، عوامل موثر بر طراحی، آیندهٔ زبان، ویژگی‌های خاص آن زبان و ... بحث می‌شود)

فصل‌ها به ترتیب عبارتند از :

1- C++ Bjarne Stroustrup

2- Python Guido von Rossum

3- APL Adin D. Falkoff

4- FORTH Charles D. Moore

5- BASIC Thomas E. Kurtz

6- AWK Alfred Aho, Peter Weinberger, Brian Kernighan

7- Lua LuizHenrique de Figueiredo, Roberto Ierusalimschy

8- Haskell Simon Peyton Jones, Paul Hudak, Philip Wadler, John Hughes

9- ML Robin Milner

10- SQL Don Chamberlin

11- Objective-C Brad Cox, Tom Love

12- Java James Gosling

13- C# Anders Hejlsberg

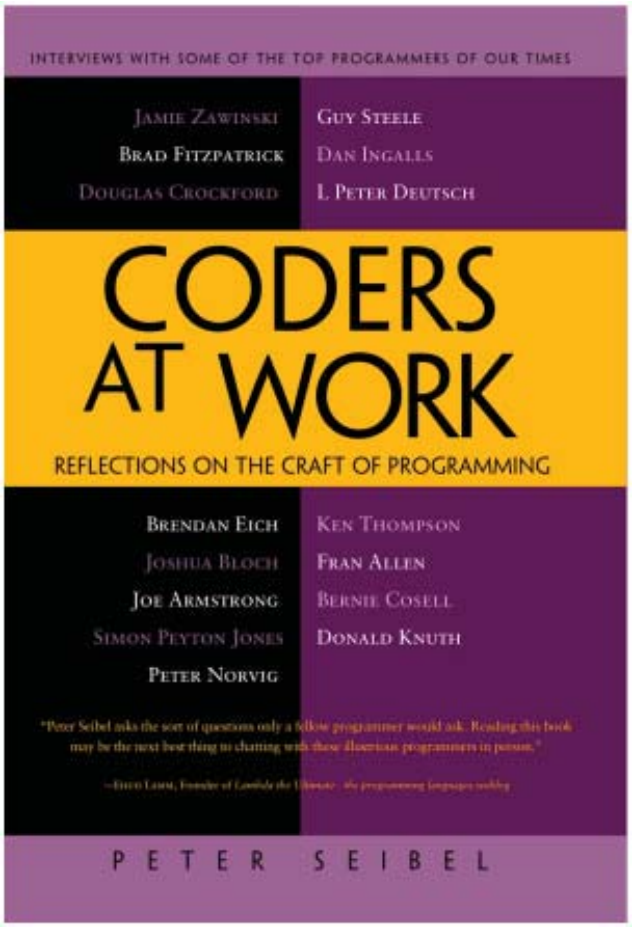
14- UML Ivar Jacobson, James Rumbaugh, Grady Booch

15- Perl Larry Wall

16- Postscript Charles Geschke, John Warnock

17- Eiffel Bertrand Meyer

۲- Coders at Work



نویسنده: Peter Seibel
انتشارات: APress
سال چاپ: 2009
تعداد صفحه‌ها: 632

این کتاب نیز مانند کتاب قبلی در قالب مصاحبه است، منتها مصاحبه‌ها با 15 برنامه‌نویس برتر جهان است. انتخاب این 15 نفر از بین 284 اسمی است که از نظر سنجی‌ای در سایت www.codersatwork.com به‌دست آمده‌است. Apress قبلاً نیز کتابی در این قالب منتشر کرده‌بود: Founders at Work که در شمارهٔ 2 مجله‌ی برنامه‌نویس در همین ستون معرفی شد. موضوع مصاحبه‌ها علاوه بر خود فرد، دیدگاه‌های وی در مورد یادگیری برنامه‌نویسی، آینده‌ی برنامه‌نویسی، توصیه به برنامه‌نویسان جوان‌تر و ... است.

اسامی برنامه‌نویسان مصاحبه‌شده در کتاب (به‌ترتیب):

1- Jamie Zawinski: از برنامه‌نویسان Netscape و XEmacs . Peter Norvig وی را از بهترین برنامه‌نویسانی که به عمرش دیده معرفی می‌کند.

2-Brad Fitzpatrick: جوان‌ترین برنامه‌نویس این کتاب! خالق LiveJournal، memcached،MogileFS و Perlbal.

3- Douglas Crockford: خالق JSON.

4- Brench Eich: خالق JavaScript؛ وی و Jamie Zawinski در سال 1998، NetScape را متقاعد کردند تا مرورگر خود را اپن سورس کند که این امر منجر به ایجاد Mozilla.org شد. اخیرا وی علاوه بر نقش داشتن در تصمیم‌گیری‌های مهم در Mozilla، کارهای برنامه‌نویسی مثل برنامه‌نویسی TraceMonkey را نیز انجام می‌دهد.

5- Joshua Bloch: وی اکنون معمار جاوا در گوگل است. قبلا در Sun مرحوم! جزو برنامه‌نویسان برجسته‌ای بود که طراحی و پیاده‌سازی Java Collections Framework بر عهده‌اش بود. ضمناً وی نویسنده‌ی کتاب معروف Effective Java است.

6- Joe Armstrong: خالق Erlang (که در شماره‌ی دوم مجله معرفی شد)

7- Simon Peyton Jones: از برنامه‌نویسان و خالقان اصلی زبان Haskell و نیز برنامه‌نویس اصلی GHC (کامپایلر استاندارد Haskell). وی اکنون در مرکز تحقیقات مایکروسافت در کمبریج کار می‌کند.

8- Peter Norvig: از کارکنان سابق ناسا؛ اکنون در گوگل به عنوان یک محقق مشغول به کار است. ترجمهٔ کتاب Artificial Intelligence: A Modern Approach که حاصل کار مشترک وی و Stuart Russel است در اغلب دانشگاه‌های دنیا به عنوان منبع اصلی هوش مصنوعی تدریس می‌شود. ترجمه‌ی مقاله‌ی معروف وی به نام «خودآموز برنامه نویسی در 10 سال» در شماره‌ی اول مجله‌ی برنامه‌نویس چاپ شد.

9- Guy Steele: خالق Scheme و از افراد مهم در دنیای Common LISP. جزو افراد موثر در استانداردسازی ECMAScript، C، Fortran، Common LISP و Scheme و نیز نویسندهٔ Java Specification. ضمنا وی نقش مهمی در ایجاد Emacs داشت. وی در حال حاضر روی زبان Fortress که زبانی برای انجام محاسبات علمی سریع است کار می‌کند. (در شماره‌ی یک مجله در ستون آشنایی با بزرگان با وی آشنا شدیم)

10- Dan Ingalls: به قول Seibel اگر Alan Kay پدر Smalltalk باشد، باید Ingalls را مادر این زبان بدانیم زیرا کار سخت پیاده‌سازی و به دنیا آوردن این زبان کار این شخص بود. (اولین نسخه را با BASIC پیاده سازی کرده بود!) در حال حاضر وی نقش مهمی در پیاده سازی Squeak (نسخهٔ اپن سورس Smalltalk) دارد. در سال 2002 جایزهٔ Dr. Dobb's Excellence in Programming را که هر ساله به تاثیرگذارترین شخص در حوزه‌ی نرم افزار اعطا می‌شود، به وی تقدیم شد.

11- L Peter Deutsch: وی در XerocPARC روی Interlisp کار می‌کرد (که در سال 1992 جایزهٔ ACM Software System به این خاطر به وی اعطا شد). در ضمن وی خالق Ghostscript نیز هست.

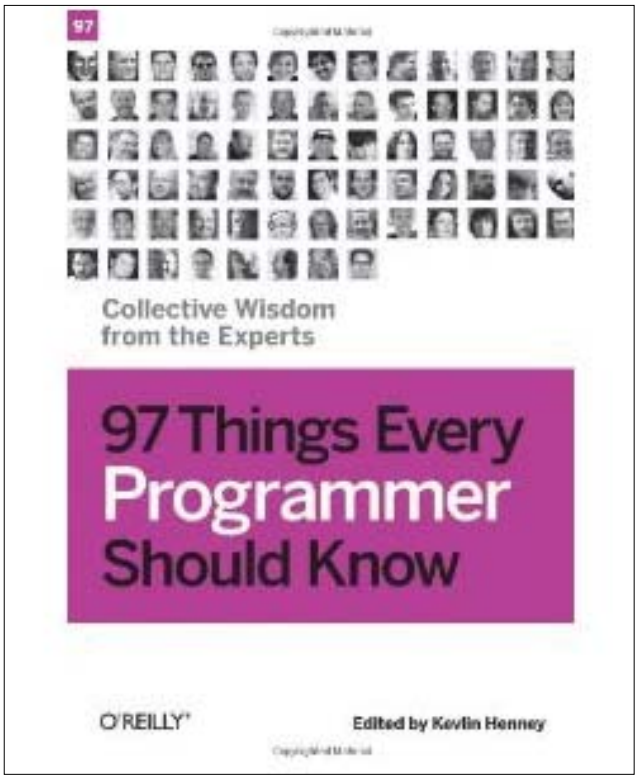
12-Ken Thompson: از خالقین یونیکس و نیز خالق زبان B (که بعدها زبان C بر اساس این زبان ایجاد شد). در زمانی که در آزمایشگاه‌های Bell روی پروژهٔ سیستم عامل Plan 9 کار می‌کرد، انکودینگ UTF-8 را نیز ابداع کرد. در ضمن در سال 1983 به خاطر کار بر روی یونیکس به طور مشترک با Denis Ritchie جایزهٔ Turing Award را دریافت کرد.

13- Fran Allen: وی بیشتر در زمینه‌ی کامپایلرها و زبان‌های برنامه‌نویسی کارهای تحقیقاتی کرده‌است که در سال 2002 جایزهٔ Turing Award را به خاطر تحقیقاتش دریافت کرد (و اولین زنی در تاریخ بود که مفتخر به دریافت این جایزه شد یا به نظر من به این جایزه افتخار بیشتری داد)

14- Bernie Cosell: از برنامه‌نویسان اصلی PDP-I

15- Donald Knuth: نویسنده‌ی سری کتاب‌های The Art of Computer Programming. استفاده از نشانه‌گذاری O بزرگ در الگوریتم‌ها توسط وی محبوب و همگانی شد. ضمناً وی روش تجزیهٔ LR را ایجاد کرد. وی بر خلاف Dijkstra از استفاده از عبارات goto دفاع کرد. وی سیستم TeX را نیز ابداع کرد. وی جایزه‌های Turing Award (سال 1974) و مدال ملی علوم (1979) را دریافت کرده‌است.

Things Every Programmer Should Know 3-97



نویسنده: این کتاب 97 نویسنده داردا و ویرایش آن توسط Kevlin Henney انجام شده‌است.

انتشارات: O'Reilly

سال چاپ: 2010

تعداد صفحه‌ها: 230

این کتاب سومین کتاب در سری Things 97 است. دو کتاب قبلی برای Architect ها و Project Managerها بودند. کتاب‌ها قبل از چاپ شدن در یک ویکی قابل مشاهده برای عموم جمع شده و سپس بعد از ویرایش ادبی تبدیل به کتاب می‌شوند. هر فصل حاوی توصیه‌ی یک برنامه‌نویس خوب به بقیه‌ی برنامه‌نویسان است و معمولاً 2 صفحه است. معمولاً عناوین فصل‌ها به خوبی گویای توصیه‌ی مربوطه هستند. به عنوان مثال:

Apply Functional Programming Principles

Ask, "What Would the User Do?" (You Are Not the (User

Choose Your Tools with Care

A Comment on Comments

Domain-Specific Languages

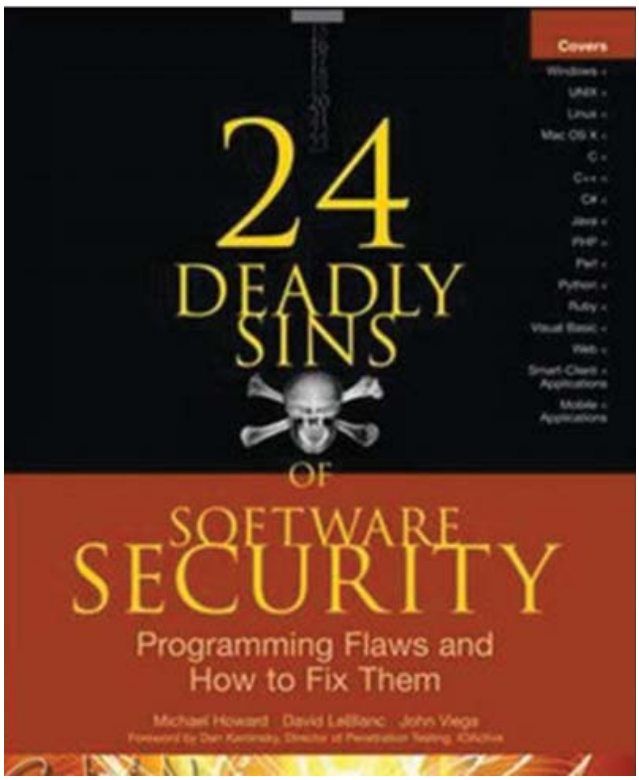
Don't Just Learn the Language, Understand its Culture

Know Your IDE

Put Everything under Version Control

Your Customers Do Not Mean What They Say

ب- امنیت نرم افزار



کتابی که برای این شماره در نظر گرفته‌ام 24 Deadly Sins of Software Security نام دارد؛ نویسندگان این کتاب Michael Howard ، David LeBlanc و John Viega نام دارند؛ اگر در حوزه‌ی امنیت ویندوز مدتی جستجو و مطالعه کرده‌باشید حتما دو نویسنده‌ی اول را با سه کتاب Writing Secure Code می‌شناسید.

کتاب قبلی این سه نویسنده 19 Deadly Sins of Software Security است که در سال 2005 منتشر شد.

کتاب 394 صفحه داشته و توسط McGraw Hill در سال 2010 منتشر شده‌است. همانطور که از عنوان کتاب پیداست، 24 گناه یا خطای مرگبار! برنامه‌نویسی در این کتاب معرفی می‌شوند. ابتدا مفهوم معرفی شده سپس پیاده سازی و نمونه کد حاوی خطا در زبان‌ها و محیط‌های مختلف (ویندوز، لینوکس، مکینتاش. C/ C++، C#، Java، PHP، Perl، Python، Ruby، Visual Basic ...) عرضه شده و روش‌های پیدا کردن و جلوگیری از آن معرفی می‌شوند. کتاب چهار بخش دارد: گناهان برنامه‌های وب، گناهان پیاده‌سازی، گناهان رمزنگاری و گناهان شبکه.

فصل‌ها (به ترتیب):

1- SQL Injection

2 - آسیب‌پذیری‌های مرتبط با وب سرور (XSS، Response Splitting,XSRF)

3- آسیب‌پذیری‌های مربوط به کلاینت‌های وب (XSS)

4- استفاده از Magic URL ها، کوکی‌های قابل پیش‌بینی و فیلدهای فرم مخفی

5- سرریز بافر

6- مشکلات Format String

7- سرریز اعداد

8- فجایع C++

9- دریافت استثناها

10- تزریق دستور

11- عدم مدیریت صحیح خطاها

12- نشت اطلاعات

13- Race Conditions

14- Poor Usability

15- Not Updating Easily

16- اجرای کد با مجوز زیاده از حد

17- عدم محافظت از داده‌های ذخیره شده

18- گناهان کد قابل حمل (برنامه‌های آفیس، اسکریپت‌های vbscript ...)

19- استفاده از سیستم‌های مبتنی بر پسورد ضعیف

20- اعداد رندوم ضعیف

21- استفادهٔ ناصحیح از رمزنگاری

22- عدم محافظت از ترافیک شبکه

23- استفادهٔ ناصحیح از PKI (مخصوصاً SSL)

24- اعتماد به Network Name Resolution

نویسنده: محمد شمس جاوی

نگاهی اجمالی بر مقابله با مهندسی معکوس کدهای .Net

http://www.mshams.ir

همانطور که می‌دانید، ساختار فایل‌های اجرایی ویندوز از فرمت Microsoft Portable Executable یا PE تبعیت می‌کند. اما این مسئله در مورد فایل‌های کامپایل شده تحت فریم ورک دات نت کمی متفاوت است.
برای روشن شدن این مسئله، مهم‌ترین طبقه‌بندی در روش کامپایل فایل‌های اجرایی سیستم عامل ویندوز را تشریح می‌کنیم:

فایل‌های PE در سیستم عامل ویندوز به دو شیوه (معماری) کلی کامپایل می‌شوند:
۱. کد مدیریت شده (Managed Code)
۲. کد مدیریت نشده (Unmanaged Code) یا کد ماشین (Machine Code)

کد مدیریت شده (Managed Code)

به کدهای کامپایل شده در زبان‌های برنامه‌نویسی مانند Visual Studio.Net languages: C#, J#, VB.NET, VC++. Net, Java, ActionScript و غیره اطلاق گشته و مهم‌ترین ویژگی‌های ساختاری این کدها به شرح زیر هستند:

- این کدها، تحت مدیریت و محافظت یک ماشین مجازی (Virtual Machine) در سیستم عامل اجرا می‌شوند و در واقع سیستم عامل مسئول اجرای ماشین مجازی بوده و ماشین مجازی نیز مسئولیت اجرای کدهای تحت مدیریت خود در سیستم عامل را به عهده دارد. منظور از ماشین مجازی می‌تواند Microsoft .Net Framework که مهم‌ترین و مطرح‌ترین ماشین مجازی موجود در سیستم عامل ویندوز است بوده و یا به فریم ورک‌های دیگری مانند Java Virtual Machine یا Flash نیز اشاره نمود. در واقع کدهای جاوا و اسکریپت‌های زبان ActionScript در محیط فلش، تحت مدیریت فریم ورک‌های خود یعنی JVM و FP اجرا می‌گردند.
- فایل‌های اجرایی آن‌ها حاوی کد ماشین نیستند، چرا که پس از کامپایل برنامه‌های نوشته شده در محیط‌های مدیریت شده، فایل اجرایی (نهایی) به زبانی به نام Intermediate Language IL یا به اصطلاح زبان میانی تبدیل می‌شوند.

- به دلیل وجود واسطه‌ای به نام فریم ورک، کاملا مستقل از سخت افزار (مستقل از سکو) بوده و قابل حمل (Portable) هستند. چرا که مسئولیت اجرای آن‌ها با فریم ورک بوده و به هیچ وجه در تعامل مستقیم با سخت افزار سیستم نیستند. در واقع میزان تعامل و حدود دسترسی آن‌ها به منابع سیستم، منوط به امکانات ارائه شده از طرف بستر آن‌ها (فریم ورک) جهت کار با منابع است.

کد مدیریت نشده (Unmanaged Code)

به این کدها اصطلاحاً کد ماشین (Machine Code) یا کد محلی (Native Code) گفته شده و در واقع همان کدهای کامپایل شده در زبان‌های غیر فریم ورک مانند VC++, Delphi, VB6.0 و غیره هستند. مهم‌ترین ویژگی‌های ساختاری این کدها به شرح زیر هستند:

- پس از کامپایل، مستقیماً به کد ماشین (زبان ماشین) تبدیل می‌شوند. در سیستم عامل ویندوز کدهای ماشین بر مبنای معماری X86 اینتل طراحی شده‌اند.
- در هنگام اجرا، مستقیماً و بدون واسطه، توسط سیستم عامل و بر روی CPU اجرا می‌شوند.
- در هنگام اجرا، مستقیماً و بدون واسطه، توسط سیستم عامل و بر روی CPU اجرا می‌شوند.

ماشین مجازی Microsoft .NET Framework

ماشین مجازی .Net حاوی کتابخانه بسیار بزرگی از کدهای از پیش تهیه شده و آماده دات نت است که به منظور تسریع روند برنامه‌نویسی، به صورت اجزایی به نام Namespace در

اختیار برنامه‌نویسان قرار می‌گیرند. مهم‌ترین بخش‌های معماری این ماشین مجازی بزرگ MSIL، CLR و JIT هستند که مختصراً تشریح می‌گردند:

Common Language Runtime CLR

در واقع هسته اصلی فریم ورک دات نت محسوب می‌شود و تمامی اعمال مهم مدیریتی مانند:

- مدیریت‌حافظه (Memory manage-ment)
-مدیریت نخها (Thread management)
-مدیریت استثناها (Exception handling)
-جمع آوری زباله‌ها (Garbage collection) یا به عبارتی همان آزادسازی حافظه‌های اشغال شده
-امنیت (Security) .....

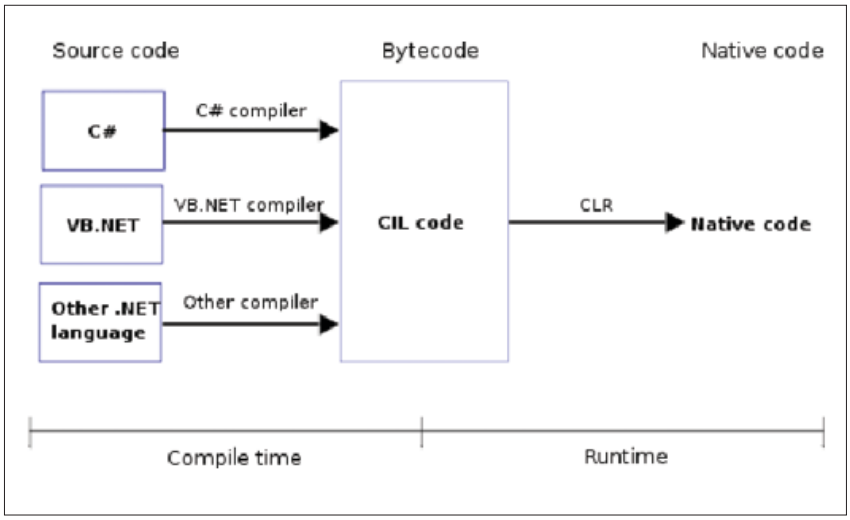
را به عهده دارد. با توجه به شکل 1 مشاهده می‌کنید که پس از تبدیل کدهای دات نت به کد میانی، این CLR است که وظیفه مدیریت کد میانی جهت اجرا را بر عهده می‌گیرد.

لازم به ذکر است که در حال حاضر پروژه‌ای موسوم به Mono به شبیه‌سازی CLR و یا به عبارتی شبیه‌سازی بستر دات نت در سیستم عامل‌های مبتنی بر Unix و Linux می‌پردازد. با استفاده از امکانات Mono می توان فایل‌های مدیریت شده .Net را بر روی سکوهای لینوکس نیز اجرا نمود. (البته محدودیت‌هایی نیز وجود دارد)

Common Intermediate Language CIL, IL, MSIL

کد زبان میانی یا IL پایین‌ترین سطح کدهای موجود در معماری دات نت است. به نوعی همان کد ماشین، اما در فریم ورک دات نت محسوب می‌شود.

مهم‌ترین نقطه ضعف کدهای این زبان، قابل فهم بودن آن‌ها توسط انسان است. در واقع یک برنامه‌نویس با تجربه با مطالعه کدهای کامپایل شده به IL و استفاده از Metadata ها، تا حدودی می‌تواند منطق و روند اجرای برنامه را کشف نماید. به دلیل همین نقطه ضعف عمده، به منظور محافظت از کدهای دات نت در برابر مهندسی معکوس، از انواع ابزارهای محافظ مانند Obfuscator ها استفاده می‌گردد.



(شکل 1) CLR در معماری دات نت

Just-in-time compiler JIT

کامپایلر بلادرنگ یا JIT کامپایلر، بخشی از CLR محسوب شده که وظیفه کامپایل کدهای زبان میانی (Intermediate Lan-guage) به کدهای زبان ماشین (Machine Code) را بر عهده دارد. برای درک بهتر این موضوع به روند و پروسه اجرای کدها و برنامه‌های دات نت توجه نمایید:

۱. برنامه‌نویس، برنامه خود را در محیط Visual Studio.Net و در یکی از زبان‌های آن مانند C، #VB.Net یا ... می‌نویسد.

۲. برنامه مذکور توسط کامپایلر زبان مربوطه، به کد زبان میانی (IL) کامپایل می شود. به عنوان مثال اگر برنامه با #C نوشته شده باشد، توسط کامپایلر آن به نام CSC.EXE و اگر با VB.Net نوشته شده باشد توسط کامپایلر VBC.EXE به کد میانی کامپایل می‌شود.

۳. کاربر اقدام به اجرای برنامه کامپایل شده (فایل exe) خود می‌نماید.

۴. در این مرحله CLR کار را به دست گرفته و پس از خواندن محتوای فایل مربوطه از دیسک سخت و بارگذاری آن در حافظه، ابتدا تمام پیش‌نیازهای مربوط به آن مانند کتابخانه‌های اشتراکی و غیره را در حافظه بارگذاری نموده و سپس توسط JIT Compiler کد زبان میانی را به کد ماشین (Machine Code) تبدیل و ترجمه می‌نماید.

این عمل ترجمه معمولاً تنها یک بار صورت گرفته و پس از آن به دلیل این‌که کل کد میانی به یک باره به کد ماشین تبدیل شده است، در مراحل بعدی اجرای برنامه با سرعت بیشتری انجام می‌گردد.

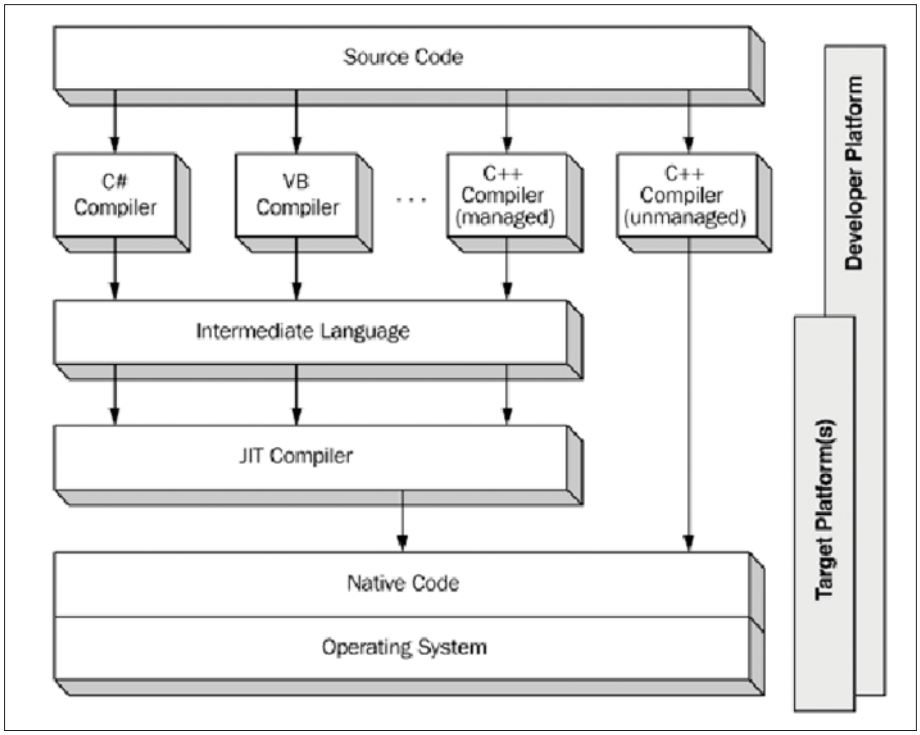
تفاوت کدهای مدیریت شده و مدیریت نشده از نظر ساختار PE

حال که مفاهیم CLR و IL تشریح شدند، به وجه تمایز کدهای مدیریت شده و مدیریت نشده از نظر ساختار PE می‌پردازیم.
مهم‌ترین تفاوت فایل‌های PE که به صورت کدهای مدیریت شده کامپایل و ترجمه شده‌اند، وجود بخشی اضافه به نام ".Net Directory" در سکشن‌های آن‌ها است. سکشن ".Net Directory" دارای سه بخش اصلی و بسیار مهم با نام‌های زیر است:

۱. CLR Header که هدر تشریح کننده عملیات CLR است.
۲. IL code که شامل کد اجرایی برنامه به زبان میانی است.
۳. Metadata که اطلاعاتی اضافه در باره چگونگی تفسیر و اجرای کدهای اجرایی زبان میانی موجود در بخش قبلی (IL code) می‌باشد. در واقع مهم‌ترین نقطه ضعف این معماری از همین قسمت ناشی می‌گردد.

وجود Metadata در فایل‌های کامپایل شده تحت فریم ورک برای اجرای کدهای دات نت الزامی است، اما مشکل این است که با بررسی Metadata به همراه کدهای IL به راحتی می‌توان عملیات انجام شده در IL را کشف نمود.

معنی این کار این است که با استفاده از دانش مهندسی معکوس و اتکا بر Metadata های موجود در فایل های دات نت، می‌توان سورس کد اولیه برنامه را محاسبه و تعیین نمود.
به عنوان مثال در شکل زیر، برنامه نوشته شده در زبان C # و کد معادل (تبدیل شده) آن در زبان IL را مشاهده می‌کنید. با توجه به قابل خواندن بودن کد IL و با استفاده از اطلاعات موجود در Metadata می توان سورس کد اولیه C# را از آن استخراج نمود.



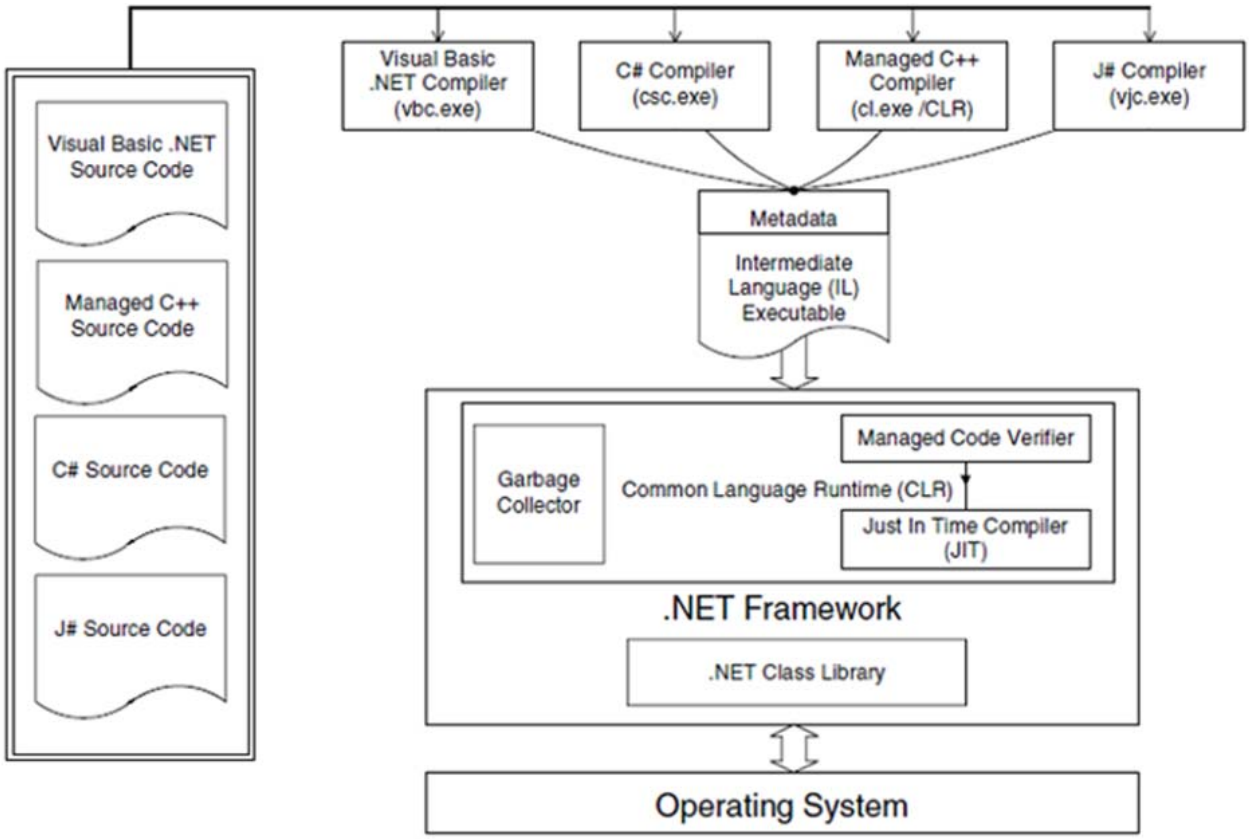
(شکل 2) موقعیت کامپایلر JIT در معماری دات نت

```
public void Stop()                                          } C# Source
{ this.flag = false; }

.method public hidebysig instance void Stop() cil managed   } IL Code
{  .maxstack 8
   L_0000: ldarg.0
   L_0001: ldc.i4.0
   L_0002: volatile.
   L_0004: stfld bool modreq([mscorlib]System.Runtime.Compiler_
         Services.IsVolatile) GTS.TSA::flag
   L_0009: ret  }
```

(شکل 3) برنامه نوشته شده در C# و کد معادل آن در IL

شکل بعدی، معماریNet. و نحوه و چگونگی برخورد آن با فایل‌ها در زمان اجرا (Runtime) را به خوبی نشان می‌دهد.



(شکل 4) معماری dotNet Framework و طریقه تعامل در آن

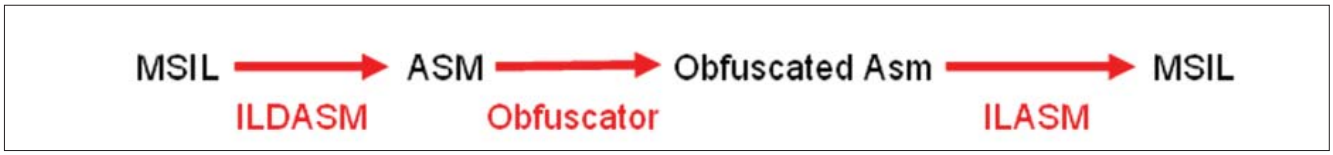
ایمن‌سازی کدهای Net.

در این بخش به تشریح سه ایده کلی مورد استفاده برای مقابله با مهندسی معکوس کدهای دات نت می‌پردازیم:

1. روش های مبتنی بر Obfuscator ها
2. روش های مبتنی بر Compiler ها
3. روش های مبتنی بر Linker ها

Obfuscator (پیچیده‌سازها)

مبنای کار آن‌ها، تبدیل کد IL برنامه (.Net Assembly) به یک کد IL معادل ولی پیچیده و غیر قابل فهم است. در واقع عملیات انجام شده در این ابزارها، در ساده‌ترین شکل ممکن به صورت زیر است:



(شکل 5) مراحل کار Obfuscator

الگوریتم این روال را به این شکل می‌توان تعریف نمود:

[1]. ابتدا کد زبان میانی (IL) را با استفاده از ابزار ILDASM به کد اسمبلی تبدیل می‌نماییم. این برنامه جزء ابزارهای استاندارد ویژوال استودیو بوده و در تمامی نسخه‌ها و نگارش‌های آن موجود است.

[2]. با انجام عملیات پیچیده‌سازی، کد اسمبلی به دست آمده را به یک کد اسمبلی معادل ولی غیر قابل فهم تبدیل می‌نماییم. لازم به ذکر است که در اینجا منظور از کد معادل، کدی است که دقیقاً همان کار را انجام می‌دهد، اما از نظر اندازه یا محتوا می‌تواند کمی متفاوت باشد.
و در واقع همین طور بوده و معمولاً پس از پایان فرایند پیچیده سازی، حجم کد معادل به دست آمده کمی بیشتر از حجم کد اولیه است. (به دلیل اضافه شدن برخی روتین های پیچیده سازی).

[3]. با استفاده از ابزار ILASM کد اسمبلی معادل را به کد زبان میانی (IL) تبدیل می‌نماییم. این برنامه نیز جزء ابزارهای استاندارد ویژوال استودیو دات نت بوده و در تمامی نسخه‌ها و نگارشهای آن موجود است.

طی این فرآیند برنامه کامپایل شده دات نت (فایل اجرایی)، به یک فایل اجرایی معادل (از نظر عملیاتی و کاری که انجام خواهد داد) ولی با محتوایی متفاوت تبدیل می‌گردد.

ابزارهای مورد استفاده

از معروف ترین ابزارهای پیچیده‌سازی کدهای دات نت به موارد زیر می‌توان اشاره نمود:

شرکت Xenocode Inc:
ابزار Xenocode Postbuild for .NET

شرکت Remotesoft Inc:
ابزار Salamander .NET Obfuscator
ابزار Salamander .NET Protector
ابزار Salamander .NET Linker and Mini-Deployment Tool

شرکت Wise Owl:
ابزار Demeanor for .NET, Enterprise Edition

شرکت Jungle Creatures, Inc:
ابزار Decompiler.NET
ابزار Deploy.NET

شرکت PreEmptive Solutions:
ابزار Dotfuscator for .NET Professional Edition

شرکت PV Logiciels:
ابزار dotNet Protector

Compiler (مبدل‌های IL به Native)

مبنای این روش، صرف نظر کردن از تمام ویژگی‌ها و قابلیت‌های مدیریتی CLR در زمان اجرا، و نهایتاً تبدیل کد مدیریت شده (Managed Code) به کد مدیریت نشده (Unmanaged Code) است.
در این روش با استفاده از کامپایلرهای خاص تبدیل IL به Native، فایل اجرایی برنامه دات نت به یک فایل اجرایی حاوی کد ماشین تبدیل می‌گردد. مهم‌ترین نکته‌ای که در اینجا می‌بایست مورد توجه قرار بگیرد، خارج شدن برنامه حاصل، از حالت مستقل از سکو (platform-independent) به وابسته به سکو است، چرا که پس از تبدیل فایل اجرایی به Native، دیگر وجود فریم ورک در سیستم عامل اهمیتی نداشته و برنامه جهت اجرا، مستقیماً توسط سیستم عامل، بر روی CPU اجرا خواهد شد.

Linker (برنامه‌های چسباننده فریم ورک به فایل برنامه)

وظیفه این لینکرها، اتصال یا لینک ایستای تمامی بخش‌های مورد نیاز از CLR جهت اجرای برنامه، به فایل اجرایی است. با این‌کار بخش‌هایی از کتابخانه CLR که در زمان اجرای برنامه مورد استفاده قرار خواهند گرفت، به صورت ایستا به فایل اجرایی برنامه متصل شده و یک فایل اجرایی مستقل (Stand alone) را تشکیل می‌دهند.
پس از این عمل، می‌توان بر روی کل فایل اجرایی مستقل بدست آمده، به یکباره عملیات Obfuscation یا پیچیده‌سازی را انجام داد.
البته باید توجه داشت که پس از تشکیل فایل اجرایی مستقل، حجم آن در حدود 15 تا 30 مگابایت افزایش یافته (به دلیل الصاق CLR به آن) و از سرعت اجرای آن نیز مقداری کاسته خواهد شد.

روتین‌های مورد استفاده در پیچیده‌سازها (Obfuscators)

Obfuscator ها به عنوان بهترین و کاراترین ابزارهای محافظت از کدهای دات نت، دارای روش‌های متنوع و متفاوتی برای پیچیده‌سازی

کد برنامه هستند که مهم‌ترین انواع آن‌ها را به پنج دسته اصلی می‌توان تقسیم نمود:

Decompiled executable:

```
private static void Converse(string name1, string name2)
{
   Friendly friendly = new Friendly(name1);
   Friendly friendly2 = new Friendly(name2);
   friendly.SayHello();
   friendly2.SayHello();
   friendly.SayGoodbye(friendly2.Name);
   friendly2.SayGoodbye(friendly.Name);
}
```

```
public void SayGoodbye(string othername)
{
   Console.WriteLine("Goodbye {0}", othername);
}
```

(شکل 6) کد Decompile شده برنامه قبل از تغییر نام موجودیتها

Decompiled obfuscated executable

```
private static void a(string A_0, string A_1)
{
   a a = new a(A_0);
   a a2 = new a(A_1);
   a.b();
   a2.b();
   a.b(a2.a());
   a2.b(a.a());
}
```

```
public void b(string A_0)
{
   Console.WriteLine("Goodbye {0}", A_0);
}
```

(شکل 7) کد Decompile شده،برنامه بعد از تغییر نام موجودیت‌ها

اضافه کردن کدهای خنثی (Dead-Code Insertion)

در این روش با افزودن کدها و دستورات بی‌تاثیری مانند دستور Nop، یا استفاده متوالی از دستورات Push و Pop، یا اعمال شرطها و حلقه‌های خنثی به کد اولیه، سعی در پیچیده سازی کد اولیه می‌نماییم. بدیهی است که تشخیص بی اثر بودن برخی کدها در میان انبوه کدهای فایل اولیه، عمل مهندسی معکوس را دشوار می‌سازد.

تغییر و پیچیده‌سازی روند اجرا (Control Flow Obfuscation)

با استفاده از انواع دستورات پرش مثل Jump و Goto روند اجرای برنامه را پیچیده کرده تا دنبال کردن سیر خطی روند اجرا (Trace) دشوار گردد.

تغییر نام موجودیت‌ها (Entity Renaming)

از آن‌جا که نام تمامی موجودیت‌هایی مانند Namespace ها، انواع داده‌ای (Data Types)، روتین‌ها، متدها و صفات (Properties) در فایل اجرایی اولیه قابل تشخیص هستند، نام آن‌ها را تغییر می‌دهند.

به عنوان مثال دو تصویر بالا، کد برنامه قبل و بعد از تغییر نام موجودیت ها را نشان می‌دهند:

کدگذاری رشته‌ها (String Encryption)

با استفاده از انواع متدهای کدگذاری، رشته‌های موجود در برنامه را به رمز در آورده تا در برنامه نهایی به صورت plain قابل دیدن نباشند.

حقه‌های مقابله با دیکامپایلرها (Decompiler Cracher Tricks)

شامل انواع روش های مورد استفاده جهت ایجاد خلل در کار Decompiler ها هستند. برخی از این روش‌ها حالت عمومی داشته و علیه اکثر Decompiler ها مورد استفاده قرار می‌گیرند، برخی دیگر نیز فقط برای نسخه‌های خاص و معروف نوشته می‌شوند. از جمله این روش‌ها به موارد زیر می‌توان اشاره نمود:

- خاتمه دادن به پروسس Decompiler با جستجوی نام پروسس آن
- تزریق کد به حافظه پروسس
- بستن Handle های حیاتی و مهم پروسس‌ها

منابع:

[1]. The .NET File Format -- Daniel Pistelli 2006

[2]. How-To-Select an Obfuscation Tool for .NET -- Mike Gunderloy 2005

[3]. Compiling MSIL to Native Code -- MSDN Library 2008

[4]. NET Internals and Code Injection -- Daniel Pistelli 2006.

[5]. Rebel.NET Official Guide -- Daniel Pistelli 2006

[6]. NGen Revs Up Your Performance with Powerful Features -- Reid Wilkes 2006

[7]. Native Image Generator (Ngen.exe) -- MSDN Library 2008

نویسنده: بهروز راد

کوئری‌های کامپایل شده در LINQ to SQL

behrouz.rad@gmail.com

اگر از Stored Procedureها استفاده می‌کنید، مطمئناً با مزایای استفاده از آن‌ها نیز آشنایی دارید. یکی از مهم‌ترین مزایای Stored Procedureها، ایجاد نسخه‌ای Cache شده از آن‌ها در SQL Server و تولید یک Execution Plan است، که موجب می‌شود تا مجموعه عملیاتی که SQL Server برای آماده‌سازی و اجرای یک SP طی می‌کند فقط یک بار انجام شود.
حال ببینیم در LINQ to SQL چه اتفاقی رخ می‌دهد:
در هنگام اجرای یک دستور LINQ در حالتی که از LINQ to SQL استفاده می‌کنید، تعدادی عملیات در پشت صحنه برای آماده‌سازی و ارسال آن دستور به SQL Server انجام می‌گیرد. کلاس‌ها و متدهای زیادی در این بین دخیل هستند و فراخوانی‌های بسیاری بین متدهای مختلف صورت می‌گیرد. این مجموعه عملیات زمان‌بَر است و موجب می‌شود تا سربار زیادی به سیستم عامل تحمیل شود؛ در نتیجه باعث افت محسوس سرعت برنامه می‌شود.
یک نمونه استفاده‌ی متداول از LINQ to SQL را در ذیل ملاحظه می‌کنید:

```
testDbDataContext dc = new testDbDataContext();
var result = from o in dc.Orders
        whrere o.ShipCity == "Esfahan"
        select o;

foreach (var o in result)
{
 Console.WriteLine("Order ID: " + o.OrderID +
   Environment.NewLine + "Ship Name: " + o.ShipName +
   Environment.NewLine + "Ship City: " + o.ShipCity);
 Console.WriteLine("------------------------------");
}
Console.Read();
```

در دات‌نت برای جلوگیری از ایجاد سربار گفته شده، و ایجاد یک Execution Plan؛ از تکنیکی با عنوان «Compiled Queries» (کوئری های کامپایل شده) استفاده می‌شود.

مراحل ایجاد یک کوئری کامپایل شده به ترتیب ذیل است:

۱) ابتدا باید ورودی‌های دستور LINQ را مشخص نمود. در کدهای فوق، «dc.Orders» و «Esfahan» به عنوان ورودی به دستور LINQ پاس داده شده‌اند؛ پس دو ورودی داریم. ورودی اول DataContext است که اجزای اصلی آن موجودیت‌ها – همانند Orders

هستند. ورودی دوم متغیری رشته‌ای است که به عنوان ورودی فیلد ShipCity در نظر گرفته شده است.

۲) خروجی یک دستور LINQ to SQL، نسخه‌ی Generic اینترفیس IQueryable است. پارامتر T این اینترفیس، نام موجودیتی را مشخص می‌کند که به عنوان خروجی دستور LINQ to SQL خواهد بود. در مثال ما، نام موجودیت Order است. همان‌طور که می‌دانید، این موجودیت‌ها توسط ORMای که همراه با VS 2008 وجود دارد، ایجاد می‌شوند.

۳) از آنجایی که مراحل آماده‌سازی دستور LINQ to SQL برای تمامی درخواست‌ها یکسان است، برای کارایی بهتر می‌توان متدی که ایجاد می کنیم را از نوع static Shared در VB.NET در نظر گرفت.
با توجه به سه نکته‌ی فوق، حالت اولیه‌ی متدی که ایجاد می‌شود به شکل ذیل خواهد بود:

```
public static IQueryable<Order> qt(
        testDbDataContext dc, string shipCity)
{
   return(from o in dc.Orders
          where o.ShipCity == shipCity
          select o);
}
```

همان‌طور که ملاحظه می‌کنید، متدی از نوع static با دو ورودی DataContext و string، و با خروجی IQueryable<Order> ایجاد شده است.

۴) در مرحله‌ی بعد، متد را با استفاده از عبارت لامبدا و دیلیگیت Func بازنویسی می‌کنیم. دلیل این تبدیل را در مرحله‌ی بعد متوجه خواهید شد:

```
public static readonly Func<testDbDataContext,
   string, IQueryable<Order>> GetCompiledQuery=

  CompiledQuery.Compile<testDbDataContext, string,
                        IQueryable<Order>>(

  (dc,shipCity)=> (from o in dc.Orders
                   where o.ShipCity == shipCity
                   select o);
```

۵) به آخرین و حساس‌ترین مرحله می‌رسیم. در این مرحله قصد داریم تا حالت کامپایل شده‌ی کوئری را ایجاد کنیم. بدین منظور باید از متد Compile کلاس CompiledQuery استفاده کنیم. این متد، چهار Overload دارد که از 1 تا 4 پارامتر ورودی را می‌پذیرند، و 1 خروجی دارند. ما نیز دو پارامتر ورودی و یک خروجی داریم. پس از نسخه‌ای از این متد با امضای ذیل استفاده می‌کنیم:

```
System.Data.Linq.CompiledQuery.Compile
                  <TArg0, TArg1, TResult>
(System.Linq.Expressions.Expression<System.Func
                        <TArg0,TArg1,TResult>>)
```

همان‌طور که در امضای فوق ملاحظه می‌کنید، متد Compile، متدی با دو پارامتر ورودی و یک پارامتر خروجی را می‌پذیرد. به همین دلیل در مرحله‌ی 4، برای کوتاه‌تر شدن کدها و راحتی کار، متد را با استفاده از عبارت لامبدا و دیلیگیت Func بازنویسی کردیم. البته می توان از حالت‌های دیگری نیز همانند تعریف کلاسیک دیلیگیت که در C# 1.0 استفاده می‌شد، یا Anonymous Methods که در C# 2.0 معرفی شد، استفاده کنیم.
متد اصلاح شده به شکل ذیل خواهد بود:

```
public static readonly Func<testDbDataContext,
          string, IQueryable<Order>> getQuery=
  (dc,shipCity)=> (from o in dc.Orders
                   where o.ShipCity == shipCity
                   select o);
```

و در نهایت، فراخوانی متد به شکل ذیل است:

```
var result = GetCompiledQuery(new testDbDataContext(),
                                        "Esfahan");
```

موفق باشید.

نویسنده: عماد فیض

ساخت سیستم رتبه بندی ستاره ای با CSS

emad.feiz@gmail.com

مطمئناً تاکنون سیستم‌های رتبه بندی که جهت تعیین رتبه برای هر مطالب در سایت‌ها در نظر گرفته می‌شوند را دیده‌اید، این سیستم‌ها علاوه بر زیبایی، کمک می‌کنند تا ارزیابی بهتری از مطالب یک سایت انجام شود. در ساخت این نوع سیستم از اشکال مختلفی می‌توان استفاده کرد، اما نمونه‌ای از آن که بیشتر رواج یافته است، سیستم ستاره‌ای می‌باشد که با کلیک بر روی ستاره مورد نظر، رتبه داده شده به سرور منتقل و ثبت می‌گردد. البته در این مقاله تنها قصد دارم نحوهٔ ساخت آن را به کمک CSS آموزش دهم و پیاده سازی کدهای سمت سرور به عهدهٔ خود شما خواهد بود.

در این مقاله از تصویر زیر که در فایل ضمیمهٔ مقاله موجود است، استفاده می‌کنیم:

از ستاره زرد رنگ برای نمایش رتبه به عنوان رتبهٔ اولیه، از رنگ سبز برای زمانی که موس بر روی ستاره ها قرار می‌گیرد و از رنگ سفید یا ستاره خالی برای نمایش خانه‌های فاقد رتبه استفاده می‌کنیم. قابل ذکر است که تصویر ما دارای ابعاد 30*90 پیکسل می‌باشد که ابعاد آن برای تعیین اندازه تگ‌ها اهمیت دارد.

در ساخت این سیستم از تگ های ul و li به عنوان عناصر اصلی html استفاده می‌کنیم، به شکلی که هر تگ li معادل یکی از ستاره‌های ما خواهد شد و تگ ul نیز به عنوان دربرگیرندهٔ ستاره‌ها. برای شروع کار کد html زیر را در نظر بگیرید:

```
<ul class="rating">
    <li>1</li>
    <li>2</li>
    <li>3</li>
    <li>4</li>
    <li>5</li>
</ul>
```

تگ های ul و li را به صورت زیر استایل دهی می‌کنیم:

```
ul.rating
{
    list-style: none;
    width: 150px;
    height: 30px;
    padding: 0;
    background: url("star_rating.gif") left top;
}
ul.rating li
{
    float: left;
    width: 30px;
    height: 30px;
}
```

در ابتدا عرض کردم که اندازهٔ کل تصویر ما (شکل 1) 90*30 پیکسل است، به این معنی که طول و عرض هر یک از ستاره‌ها 30 پیکسل می باشد. با توجه به این‌که پنج ستاره در سیستم رتبه‌بندی داریم، عرض و ارتفاع تگ ul به ترتیب برابر 150 و 30 پیکسل در نظر گرفته شده است، همچنین تصویر ستاره‌ها را در زمینهٔ تگ ul قرار می دهیم. همان‌طور که گفته شد هر تگ li معادل یکی از ستاره‌های ما خواهد بود، بنابراین برای کنار هم قرار گرفتن هر 5 ستاره خاصیت float:left را برای li ها در نظر می‌گیریم. عرض و ارتفاع تگ‌های li هم مطابق ابعاد هر ستاره برابر 30 پیکسل در نظر گرفته شده است. توجه داشته باشید، به دلیل این‌که ارتفاع هر ستاره 30 پیکسل است و تصویر زمینه را به صورت left top در تگ ul موقعیت‌دهی کردیم، تنها قسمت بالایی تصویر یعنی ستارهٔ خالی نمایش داده می‌شود.

کار ما تا این لحظه به صورت زیر خواهد بود:

همانطور که در شکل شماره 2 مشاهده می کنید، اعداد موجود در هر li بر روی ستاره‌ها قرار گرفته‌اند که برای حل این مشکل کافی است خاصیت text-indent را به خصوصیات li اضافه کنیم:

```
ul.rating li
{
    ....
text-indent: -999px;
}
```

در ادامه برای ایجاد جلوهٔ rollover، کلاس کاذب hover را برای تگ‌های li استایل‌دهی می‌کنیم. با توجه به این‌که می‌خواهیم با رفتن موس بر روی هر ستاره رنگ آن به سبز تغییر پیدا کند، بار دیگر تصویر سه ستاره را در زمینهٔ تگ li قرار داده و آن را به گونه‌ای موقعیت‌دهی می‌کنیم که ستارهٔ وسطی نمایش یابد:

```
ul.rating li:hover
{
    background: url("star_rating.gif")
                left center;
    width: 150px;
    margin-left: -120px;
    cursor: pointer;
}
```

دقت کنید که تصویر زمینه در راستای محور عمودی به صورت center موقعیت‌دهی شده است که باعث می‌شود تصویر در وسط li ها قرار گیرد. مقدار width نیز برای هر li برابر 150 پیکسل در نظر گرفته شده است. علت انتخاب این مقدار این است که در واقع ما می‌خواهیم هنگام قرارگیری موس بر روی هر ستاره، آن ستاره و ستاره‌های قبل از آن همگی به رنگ سبز درآیند، به طور مثال اگر کاربری موس را بر روی ستارهٔ چهارم قرار داد، سه ستارهٔ قبلی نیز باید سبز شوند. بنابراین باید با رفتن موس بر روی هر ستاره یا همان تگ li، عرض آن تگ برابر 150 پیکسل (همان عرض تگ ul) شود. در واقع تمام ناحیه ul توسط این تگ پوشانده شده و ستارهای سبز رنگ بر روی ستاره‌های خالی که در زمینه ul قرار داشتند، قرار می‌گیرند.

شرایط ذکر شده در صورتی اتفاق می‌افتد که کاربر موس را تنها بر روی ستارهٔ اول قرار دهد، در این صورت تگ li به اندازه عرض ul بزرگ شده و تمام ناحیه را می‌پوشاند. اما به طور مثال اگر موس بر روی ستارهٔ سوم قرار گیرد، تغییر عرض li از همان ستارهٔ سوم به بعد صورت می‌گیرد و تصویری مطابق شکل زیر را خواهیم داشت:

با توجه به این‌که عرض هر ستاره 30 پیکسل است، لازم است تگ li در هنگام رفتن موس بر روی آن به اندازهٔ 120 پیکسل به سمت چپ کشیده شود تا جلوه مورد نظر ما ایجاد گردد، به همین منظور خاصیت margin-left را برابر -120 پیکسل مقداردهی کردیم.

با اِعمال تغییرات ذکر شده مجدداً اگر موس را بر روی ستاره سوم قرار دهید، شکلی مطابق شکل 4 خواهیم داشت:

همانطور که در شکل 4 مشاهده می‌کنید دو ستاره در سمت چپ اضافه است که برای حل این مشکل باید خاصیت overflow را در تگ ul برابر hidden قرار دهیم تا قسمت‌های خارج از ul نشان داده نشوند:

```
ul.rating
{
    ....
    overflow: hidden;
    background: url("star_rating.gif")
                left top;
}
```

حال اگر کار خود را تست کنید دیگر مشکلی از بابت قسمت‌های اضافی وجود ندارد و زمانی که موس را از چپ به راست بر روی ستاره‌ها حرکت می‌دهید بدون مشکل ستاره‌ها به رنگ سبز در می‌آیند، اما اگر موس را از سمت راست به چپ حرکت دهید هیچ تغییری ایجاد نمی‌شود. علت این است که وقتی موس بر روی یکی از li ها قرار می‌گیرد، سایر تگ های li به زیر تگ فعلی رفته و قابل دسترس نیستند، به همین دلیل دیگر کلاس hover اتفاق نمی‌افتد و تغییری انجام نمی‌پذیرد.

برای رفع این مشکل لازم است تگی که موس بر روی آن قرار گرفته به زیر دیگر تگ‌ها رفته تا زمانی که تغییر اندازه صورت می‌گیرد، سایر تگ‌ها از دسترس خارج نشوند. برای قرارگیری عناصر در لایه‌های مختلف از z-index استفاده می‌کنیم. تغییرات را مطابق کدهای زیر اِعمال کنید:

```
ul.rating li
{
    ....
    position: relative;
    z-index: 3;
}
ul.rating li:hover
{
    ....
    z-index: 2;
}
```

بنابراین زمانی که موس بر روی یکی از ستاره ها می‌رود، آن ستاره به یک لایه پایین‌تر از دیگر ستاره‌ها منتقل شده و طول آن افزایش می‌یابد. ناگفته نماند به دلیل این‌که z-index تنها در حالات position:relative، position:fixed و position:absolute کار می کند، position:relative را نیز به خصوصیات li اضافه کردیم.

با اعمال این قسمت، شکل کلی سیستم ما ایجاد شده است و در زمان حرکت موس بر روی ستاره‌ها، تغییرات به خوبی صورت می‌گیرد. اکنون می‌خواهیم حالت اولیهٔ ستاره‌ها در زمان لود صفحه را که مبین رتبه های داده شده توسط کاربران مختلف است را ایجاد کنیم، به طور مثال می‌خواهیم در زمانی که صفحه لود شد، 2.5 ستاره از پنج ستاره فعال باشد.

برای انجام این کار یک تگ li در ابتدای دیگر li ها اضافه می‌کنیم:

```
<ul class="rating">
    <li class="current rate"></li>
    <li>1</li>
    <li>2</li>
    <li>3</li>
    <li>4</li>
    <li>5</li>
</ul>
```

و سپس خصوصیات زیر را به کلاس اختصاص داده شده به آن اضافه می‌کنیم:

```
ul.rating li.current_rate
{
    position: absolute;
    left: 0;
    top: 0;
    height: 30px;
    width: 75px;
    z-index: 1;
    background: url("star_rating.gif")
}               left bottom;
```

توجه کنید با اضافه کردن تگ جدید، در حال حاضر شش تگ li داریم، درحالی که اندازه تگ ul فقط برای پنج ستاره در نظر گرفته شده بود. بنابراین لازم است تگ جدید را به صورت absolute موقعیت دهی کنیم تا از جریان کلی صفحه خارج شود و تاثیری بر دیگر عناصر نگذارد. با توجه به اینکه تگ li اضافه شده باید درون ul موقعیت‌دهی شود، لازم است که position:relative را نیز به خصوصیات ul اضافه کنید:

```
ul.rating
{
    ....
    position: relative;
    background: url("star_rating.gif")
                left top;
}
```

در مراحل قبل با استفاده از z-index دو لایه مختلف را ایجاد

کردیم. به دلیل اینکه تگ li جدید در واقع نمایانگر حالت اولیهٔ رتبه است، باید در لایه‌ای زیر دو لایه دیگر قرار گیرد تا در زمان رفتن موس بر روی ستاره‌ها، توسط سایر li ها همپوشانی شود. به همین دلیل خاصیت z-index این تگ را برابر 1 قرار می‌دهیم. دقت کنید که بار دیگر تصویر سه ستاره در زمینه این تگ نیز قرار داده شده است، البته این بار تصویر را در راستای محور عمودی به صورت bottom موقعیت‌دهی کردیم که باعث می شود ستارهٔ زرد رنگ نمایش یابد.
خاصیت width که در واقع تعیین کننده میزان رتبهٔ اولیه ماست عرض تگ li را تعیین می کند. به طور مثال من در این‌جا مقدار 75 پیکسل را به آن اختصاص دادم که برابر 2.5 ستاره می باشد.

نکته:
در نسخه 6 و نسخه های اولیه اکسپلورر 7 کلاس کاذب hover تنها برای تگ a شناخته شده است، در واقع کلاس hover که در اینجا برای li استایل دهی شد، در ورژن‌های مذکور قابل شناسایی نیستند. برای شناساندن کلاس‌های کاذب دیگر از جمله active، focus، hover و ... فایلی با پسوند htc به همراه کدهای این مقاله قرار داده شده است. بعد از دانلود فایل مذکور این کد را به برچسب body صفحه خود اضافه کنید:

```
body
{
    behavior: url("csshover.htc");
}
```

ناگفته نماند که behavior یک خصیصه غیر استاندارد است که تنها در مرورگر اینترنت اکسپلولر شناخته می‌شود.

جامعه برنامه نویسان فارسی زبان برگزار می کند:

- دوره عملی "فروشگاه الکترونیک (eShop) با استفاده از C# و ASP.NET
- دوره آموزشی برنامه نویسی Java
- دوره آموزشی برنامه نویسی Web با استفاده از ASP.NET + پروژه سیستم مدیریت محتوا
- دوره کامل آموزشی برنامه نویسی Web با استفاده از ASP.NET و C#
- دوره آموزشی طراحی Web با استفاده از HTML, CSS, JavaScript و jQuery
- دوره آموزشی برنامه نویسی در Delphi ۲۰۱۰: از مبتدی تا حرفه ای

جهت کسب اطلاعات بیشتر به آدرس www.barnamenevis.org مراجعه نمایید.

برگزاری دوره های آموزشی برنامه نویسی