عنوان: **آشنایی با Saltarelle کامپایلر قدرتمند #C به جاوااسکریپت**
نویسنده: رضا بازرگان
تاریخ: ۱۳۹۳/۰۸/۱۶ ۱۹:۵۵
آدرس: www.dotnettips.info
گروه‌ها: C#, JavaScript, Javascript extensions, Compiler, Saltarelle

شاید ساده‌ترین تعریف برای Saltarelle این باشد که «کامپایلریست که کدهای #C را به جاوا اسکریپت تبدیل می‌کند». محاسن زیادی را می‌توان برای اینگونه کامپایلرها نام برد؛ مخصوصا در پروژه‌های سازمانی که نگهداری از کدهای جاوا اسکریپت بسیار سخت و گاهی خارج از توان است و این شاید مهمترین عامل ظهور ابزارهای جدید از قبیل Typescript باشد.

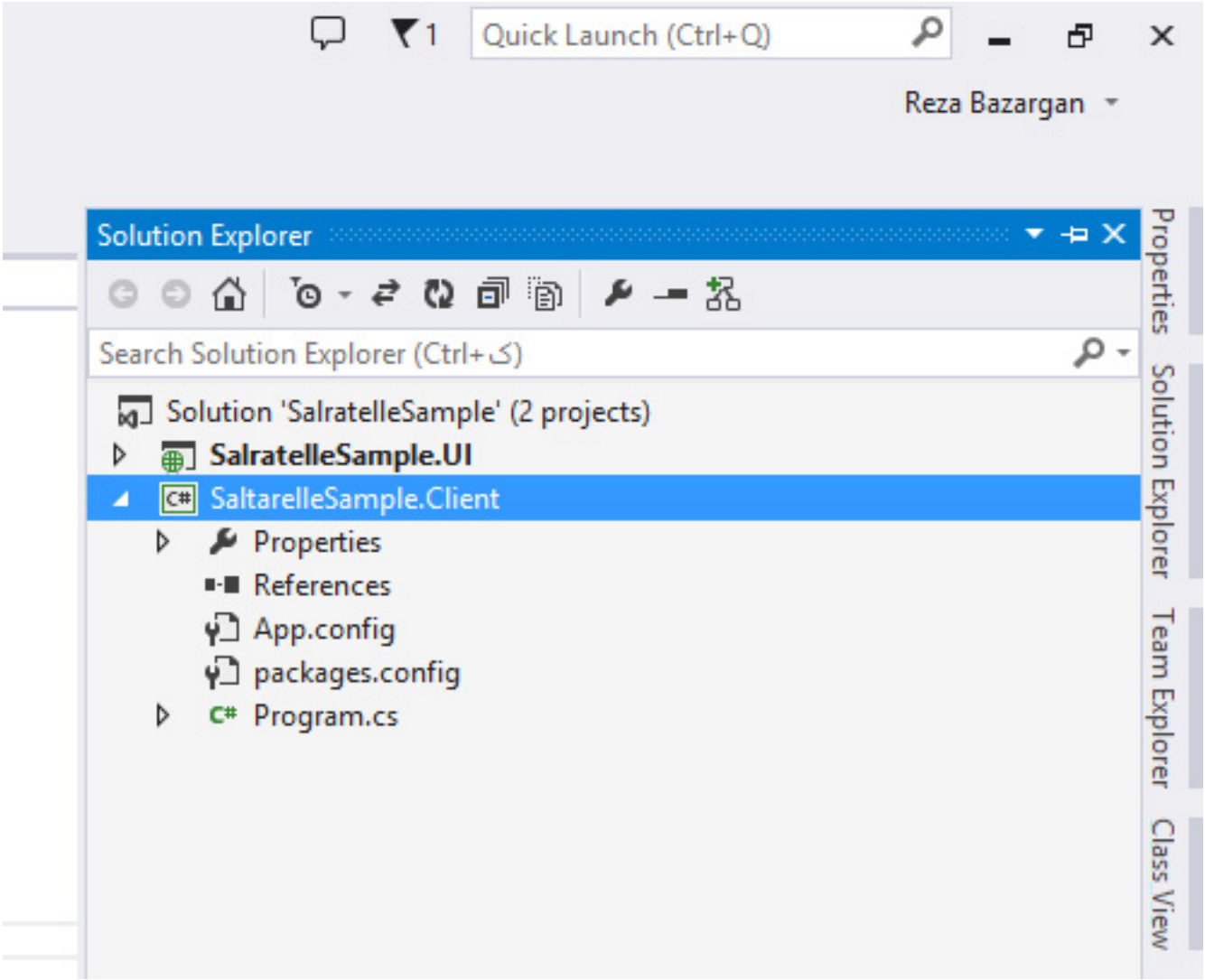
در هر صورت اگر حوصله و وقت کافی برای تجهیز تیم نرم افزاری، به دانش یک زبان جدید مانند Typescript نباشد، استفاده از توان و دانش تیم تولید، از زبان #C ساده‌ترین راه حل است و اگر ابزاری مطمئن برای استفاده از حداکثر قدرت JavaScript همراه با امکانات نگهداری و توسعه کدها وجود داشته باشد، بی شک Saltarelle یکی از بهترین‌های آنهاست.

قبلا کامپایلر هایی از این دست مانند #Script وجود داشتند، اما فاقد همه امکانات #C بوده وعملا قدرت کامل #C در کد نویسی وجود نداشت. اما با توجه به ادعای توسعه دهندگان این کامپایلر سورس باز در استفاده‌ی حداکثری از کلیه ویژگی‌های #C 5 و با وجود Library های متعدد می‌توان Saltarelle را عملا یک کامپایلر موفق در این زمینه دانست.

برای استفاده از Saltarelle در یک برنامه وب ساده باید یک پروژه Console Application به Solution اضافه کرد و پکیج Saltarelle.Compiler را از nuget نصب نمایید. بعد از نصب این پکیج، کلیه Reference ها از پروژه حذف می‌شوند و هر بار Build توسط کامپایلر Saltarelle انجام می‌شود. البته با اولین Build ، مقداری Error را خواهید دید که برای از بین بردنشان نیاز است پکیج Saltarelle.Runtime را نیز در این پروژه نصب نمایید:

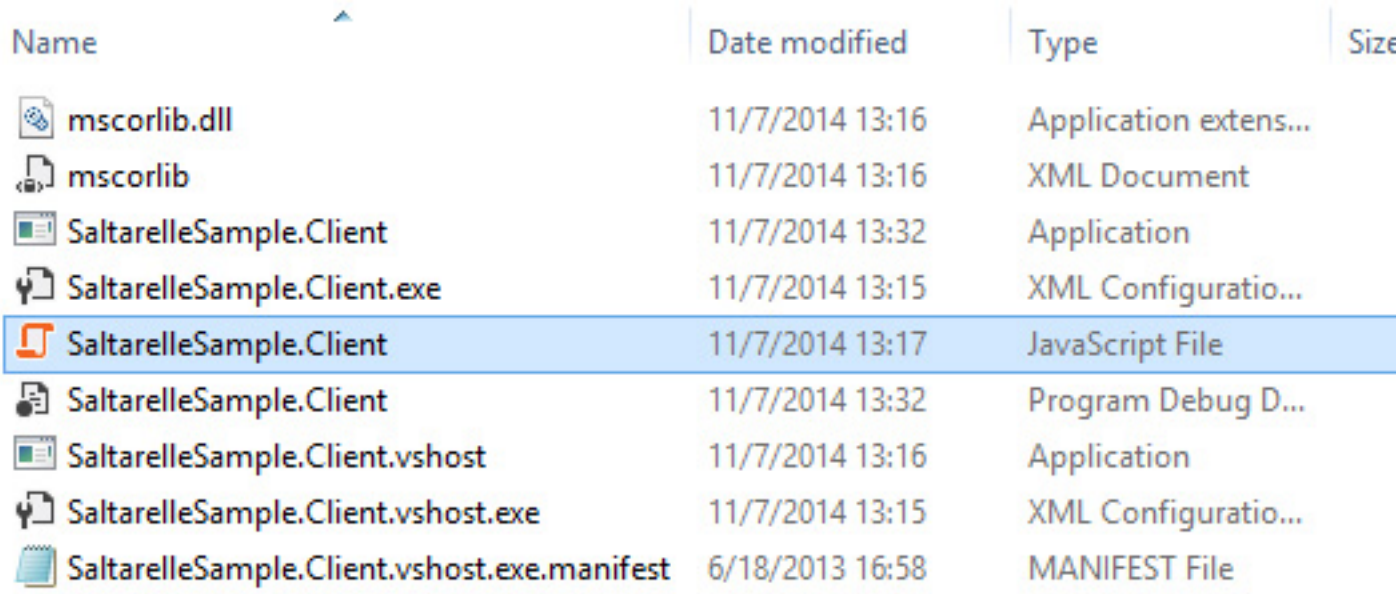
```
PM> Install-Package Saltarelle.Compiler
PM> Install-Package Saltarelle.Runtime
```

در صورتیکه کماکان Build نهایی با Error همرا بود، یکبار این پروژه را Unload و سپس مجددا Load نمایید



UI یک پروژه وب MVC است و Client یک Console Application که پکیج‌های مورد نیاز Saltarelle روی آن نصب شده است.

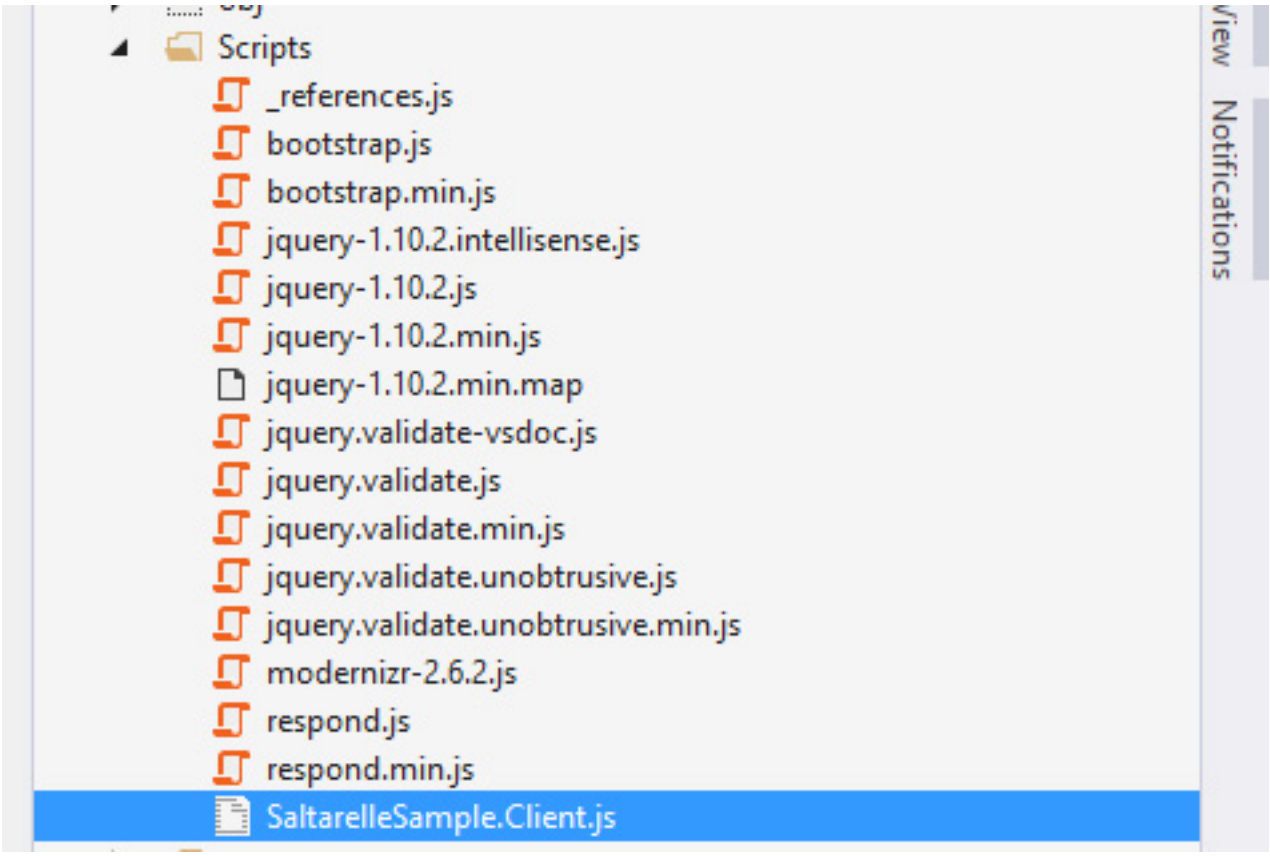
در صورتیکه پروژه را Build نماییم و نگاهی به پوشه‌ی Debug بیاندازیم، یک فایل JavaScript همنام پروژه وجود دارد:



برای اینکه بعد از هر بار Build ، فایل اسکریپت به پوشه‌ی مربوطه در پروژه UI منتقل شود کافیست کد زیر را در Post Build پروژه Client بنویسیم:

```
copy "$(TargetDir)$(TargetName).js" "$(SolutionDir)SalratelleSample.UI\Scripts"
```

اکنون پس از هر بار Build ، فایل اسکریپت مورد نظر در پوشه‌ی Scripts پروژه UI آپدیت می‌شود:



در ادامه کافیست فایل اسکریپت را به layout اضافه کنیم.

```
<script src="~/Scripts/SaltarelleSample.Client.js"></script>
```

در پوشه‌ی Saltarelle.Runtime در پکیج‌های نصب شده، یک فایل اسکریپت به نام mscorlib.min.js نیز وجود دارد که حاوی اسکریپت‌های مورد نیاز Saltarelle در هنگام اجراست. آن را به پوشه اسکریپت‌های پروژه UI کپی نمایید و سپس به Layout اضافه کنید.

```
<script src="~/Scripts/mscorlib.min.js"></script>
<script src="~/Scripts/SaltarelleSample.Client.js"></script>
```

حال نوبت به اضافه نمودن libraryهای مورد نیازمان است. برای دسترسی به آبجکت هایی از قبیل document, window, element و غیره در جاوااسکریپت می‌توان پکیج Saltarelle.Web را در پروژه‌ی Client نصب نمود و برای دسترسی به اشیاء و فرمانهای jQuery، پکیج Salratelle.jQuery را نصب نمایید.

```
> Install-Package Saltarelle.Web
> Install-Package Saltarelle.jQuery
```

به این libraryها imported library می‌گویند. در واقع، در زمان کامپایل، برای این libraryها فایل اسکریپتی تولید نمی‌شود و فقط آبجکت‌های #C هستند که که هنگام کامپایل تبدیل به کدهای ساده اسکریپت می‌شوند که اگر اسکریپت مربوط به آنها به صفحه اضافه نشده باشد، اجرای اسکریپت با خطا مواجه می‌شود.

به طور ساده‌تر وقتی از jQuery library استفاده می‌کنید هیچ فایل اسکریپت اضافه‌ای تولید نمی‌شود، اما باید اسکریپت jQuery به صفحه شما اضافه شده باشد.

```
<script src="~/Scripts/jquery-1.10.2.min.js"></script>
```

مثال ما یک اپلیکیشن ساده برای خواندن فیدهای همین سایت است. ابتدا کدهای سمت سرور را در پروژه UI می نویسیم.

کلاس‌های مورد نیاز ما برای این فید ریدر:

```
public class Feed
    {
        public string FeedId { get; set; }
        public string Title { get; set; }
        public string Address { get; set; }

    }
    public class Item
    {
        public string Title { get; set; }
        public string Link { get; set; }
        public string Description { get; set; }
    }
```

و یک کلاس برای مدیریت منطق برنامه

```
public class SiteManager
    {
        private static List<Feed> _feeds;
        public static List<Feed> Feeds
        {
            get
```

```
            {
                if (_feeds == null)
                    _feeds = CreateSites();
                return _feeds;
            }
        }
        private static List<Feed> CreateSites()
        {
            return new List<Feed>() {
                new Feed(){
                    FeedId = "1",
                    Title = "آخرین تغییرات سایت",
                    Address = "http://www.dotnettips.info/rss.xml"
                },
                 new Feed(){
                    FeedId = "2",
                    Title = "مطالب سایت",
                    Address = "http://www.dotnettips.info/feeds/posts"
                },
                 new Feed(){
                    FeedId = "3",
                    Title = "نظرات سایت",
                    Address = "http://www.dotnettips.info/feeds/comments"
                },
                 new Feed(){
                    FeedId = "4",
                    Title = "خلاصه اشتراک ها",
                    Address = "http://www.dotnettips.info/feed/news"
                },
            };
        }

        public static IEnumerable<Item> GetNews(string id)
        {
            XDocument feedXML = XDocument.Load(Feeds.Find(s=> s.FeedId == id).Address);
            var feeds = from feed in feedXML.Descendants("item")
                        select new Item
                        {
                            Title = feed.Element("title").Value,
                            Link = feed.Element("link").Value,
                            Description = feed.Element("description").Value
                        };
            return feeds;
        }

    }
```

کلاس SiteManager فقط یک لیست از فیدها دارد و متدی که با گرفتن شناسه‌ی فید ، یک لیست از آیتم‌های موجود در آن فید ایجاد می‌کند.

حال دو ApiController برای دریافت داده‌ها ایجاد می‌کنیم

```
public class FeedController : ApiController
{
        // GET api/<controller>
        public IEnumerable<Feed> Get()
        {
            return SiteManager.Feeds;
        }
    }

public class ItemsController : ApiController
    {
        // GET api/<controller>/5
        public IEnumerable<Item> Get(string id)
        {
            return SiteManager.GetNews(id);
        }
    }
```

در View پیش‌فرض که Index از کنترلر Home است، یک Html ساده برای فرم صفحه اضافه می‌کنیم

```
<div>
    <div>
        <h2>Feeds</h2>
        <ul id="Feeds">

        </ul>
    </div>
    <div>
        <h2>Items</h2>
        <p id="FeedItems">
        </p>
    </div>

</div>
```

در المنت Feeds لیست فیدها را قرار می‌دهیم و در FeedItems آیتم‌های مربوط به هر فید. حال به سراغ کدهای سمت کلاینت می‌رویم و به جای جاوا اسکریپت از Saltarelle استفاده می‌کنیم.

کلاس Program را از پروژه Client باز می‌کنیم و متد Main را به شکل زیر تغییر می‌دهیم:

```
static void Main()
        {
            jQuery.OnDocumentReady(() => {
                FillFeeds();
            });
        }
```

بعد از کامپایل شدن، کد #C شارپ بالا به صورت زیر در می‌آید:

```
$SaltarelleSample_Client_$Program.$main = function() {
$(function() {
$SaltarelleSample_Client_$Program.$fillFeeds();
});
};
$SaltarelleSample_Client_$Program.$main();
```

و این همان متد معروف jQuery است که Saltarelle.jQuery برایمان ایجاد کرده است.

متد FillFeeds را به شکل زیر پیاده سازی می‌کنیم

```
private static void FillFeeds()
        {
            jQuery.Ajax(new jQueryAjaxOptions()
            {
                Url = "/api/feed",
                Type = "GET",
                Success = (d,t,r) => {

                    // Fill
                    var ul = jQuery.Select("#Feeds");
                    jQuery.Each((List<Feed>)d, (idx,i) => {
                        var li = jQuery.Select("<li>").Text(i.Title).CSS("cursor", "pointer");
                        li.Click(eve => {
                            FillData(i.FeedId);
                        });
                        ul.Append(li);
                    });
                }
            });
        }
```

آبجکت jQuery، متدی به نام Ajax دارد که یک شی از کلاس jQueryAjaxOptions را به عنوان پارامتر می‌پذیرد. این کلاس کلیه خصوصیات متد Ajax در jQuery را پیاده سازی می‌کند. نکته شیرین آن توانایی نوشتن lambda برای Delegate هاست.

خاصیت Success یک Delegate است که 3 پارامتر ورودی را می‌پذیرد.

```
public delegate void AjaxRequestCallback(object data, string textStatus, jQueryXmlHttpRequest request);
```

data همان مقداریست که api باز می‌گرداند که یک لیست از Feed هاست. برای زیبایی کار، من یک کلاس Feed در پروژه Client اضافه می‌کنم که خصوصیاتی مشترک با کلاس اصلی سمت سرور دارد و مقدار برگشتی Ajax را به آن تبدیل می‌کنم.

کلاس Feed و Item

```
[PreserveMemberCase()]
    public class Feed
    {
        //[ScriptName("FeedId")]
        public string FeedId;

        //[ScriptName("Title")]
        public string Title;

        //[ScriptName("Address")]
        public string Address;

    }

    [PreserveMemberCase()]
    public class Item
    {
        // [ScriptName("Title")]
        public string Title;

        // [ScriptName("Link")]
        public string Link;

        // [ScriptName("Description")]
        public string Description;
    }
```

Attrubuteهای زیادی در Saltarelle وجود دارند و از آنجایی که کامپایلر اسم فیلدها را camelCase کامپایل می‌کند من برای جلوگیری از آن از PreserveMemberCase بر روی هر کلاس استفاده کردم. می‌توانید اسم هر فیلد را سفارشی کامپایل نمایید.

```
jQuery.Each((List<Feed>)d, (idx,i) => {
                        var li = jQuery.Select("<li>").Text(i.Title).CSS("cursor", "pointer");
                        li.Click(eve => {
                            FillData(i.FeedId);
                        });
                        ul.Append(li);
                    });
```

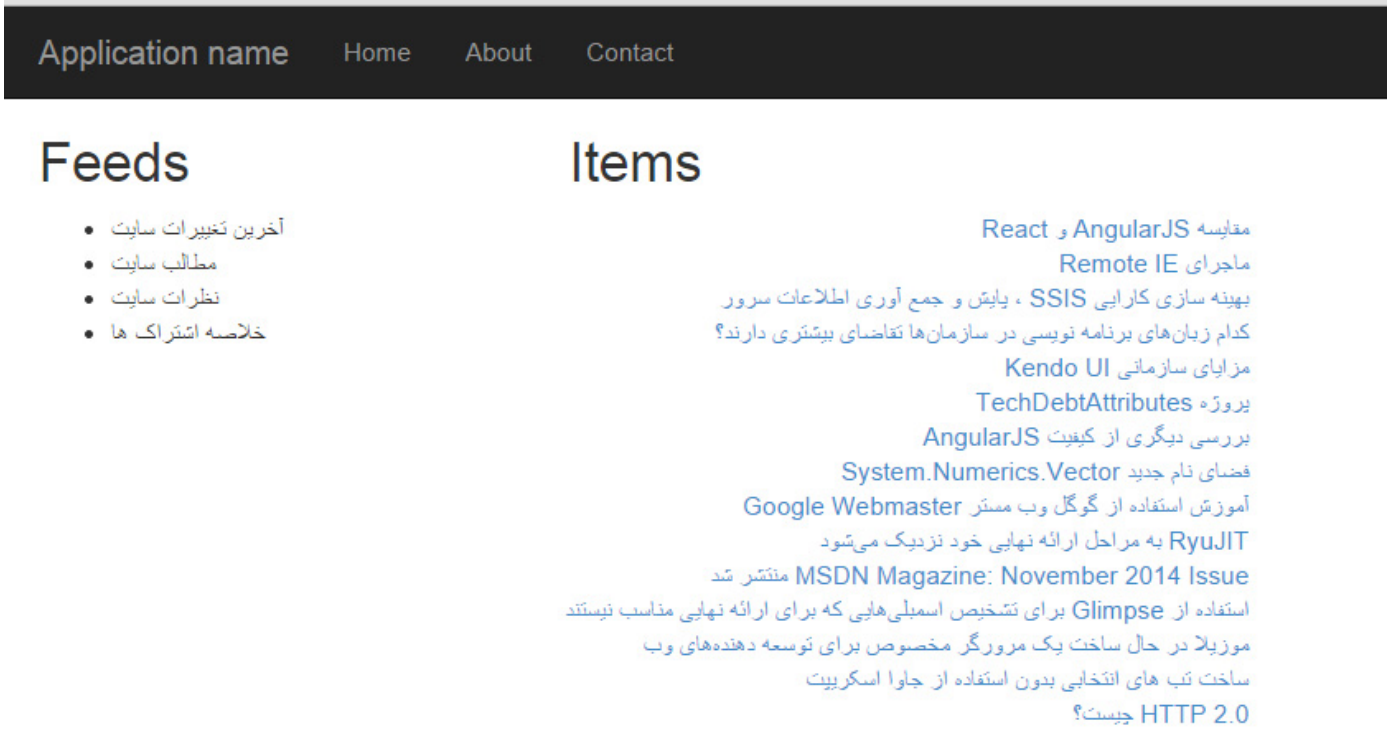
به ازای هر آیتمی که در شیء بازگشتی وجود دارد، با استفاد از متد each در jQuery یک li ایجاد می‌کنیم. همان طور که می‌بینید کلیه خواص، به شکل Fluent قابل اضافه شدن می‌باشد. سپس برای li یک رویداد کلیک که در صورت وقوع، متد FillData را با شناسه فید کلیک شده فراخوانی می‌کند و در آخر li را به المنت ul اضافه می‌کنیم.

برای هر کلیک هم مانند مثال بالا api را با شناسه‌ی فید مربوطه فراخوانی کرده و به ازای هر آیتم، یک سطر ایجاد می‌کنیم.

```
private static void FillData(string p)
        {
            jQuery.Ajax(new jQueryAjaxOptions()
            {
                Url = "/api/items/" + p,
                Type = "GET",
                Success = (d, t, r) => {
                    var content = jQuery.Select("#FeedItems");
                    content.Html("");
                    foreach (var item in (List<Item>)d)
                    {
```

```
                    var row = jQuery.Select("<div>").AddClass("row").CSS("direction", "rtl");
                    var link = jQuery.Select("<a>").Attribute("href", item.Link).Text(item.Title);
                    row.Append(link);
                    content.Append(row);
                }
            }
        });
    }
```

خروجی برنامه به شکل زیر است:



در این مثال ما از Saltarelle.jQuery برای استفاده از jQuery.js استفاده نمودیم. libraryهای متعددی برای Saltarelle از قبیل linq,angular,knockout,jQueryUI,nodeJs ایجاد شده و همچنین قابلیت‌های زیادی برای نوشتن imported libraryهای سفارشی نیز وجود دارد.

مطمئنا استفاده از چنین کامپایلرهایی راه حلی سریع برای رهایی از مشکلات متعدد کد نویسی با جاوا اسکریپت در نرم افزارهای بزرگ مقیاس است. اما مقایسه آنها با ابزارهایی از قبیل typescript احتیاج به زمان و تجربه کافی در این زمینه دارد.

فایل پروژه ضمیمه

نظرات خوانندگان

نویسنده: یاسر مرادی
تاریخ: ۱۳۹۳/۰۸/۱۶ ۲۳:۴۶

با تکمیل شدن Roslyn و آسانتر شدن امور، روز به روز شاهد تعداد بیشتری از چنین مبدل هایی خواهیم بود، اما به صورت بنیادی هرگونه مبدل کد CSharp به JavaScript یا هر زبان دیگری محکوم به شکست است، و آنچه که به صورت بنیادی مشکل ندارد، تبدیل IL به سایر زبانها است، چرا که فرض کنید شما یک DLL تقریبا ساده مانند Humanizer را که برای کار با رشته‌ها و ... به کار می‌رود را در مثالتان استفاده کنید، در این صورت دیگر برنامه شما کار نخواهد کرد، حتی اگر در حد یک Pluralize کردن باشد، اما اگر تبدیل IL به JavaScript باشد، هر رفتاری را که با DLL شما داشته باشد، همان رفتار را با Humanizer خواهد داشت، برای همین است که امروزه تبدیلگرهای قدرتمند از IL استفاده می‌کنند، مانند JSIL، و SharpKit که اوایل از تبدیل CSharp به JavaScript استفاده می‌کرد و هم اکنون تازه به این نتیجه رسیده که آب در هاون می‌کوبیده و الآن با استفاده از Cecil، به تبدیل IL به JavaScript روی آورده است.
همچنین تبدیل گر مربوطه، باید یکسری کتابخانه جاوا اسکریپتی پایه که امکانات پایه و اولیه .NET را ارائه دهد داشته باشد، که باز دقیقا تبدیلگرهای حرفه ای همین رفتار را دارند، چون همه‌ی امکانات .NET در JavaScript موجود نیست که صرف تبدیل کد کافی باشد و لااقل بعضی امکانات پایه باید ارائه شوند، مثلا برای ایجاد کردن اعداد تصادفی معادل کلاس Random در .NET

نویسنده: رضا بازرگان
تاریخ: ۱۳۹۳/۰۸/۱۸ ۱۹:۳۳

با تشکر از نوشتارتون ذکر دو نکته را لازم می‌دونم.
اول اینکه هدف از این مطلب الزام به استفاده یا عدم استفاده از این نوع کامپایلرها نیست و فقط برای آشنایی با این گونه ابزارها بوده که در حال حاضر در بسیاری از نرم افزارهای اینترپراز در حال استفاده اند.
و دوم اینکه تفاوت ساختاری و ماهیتی سی شارپ و جاوا اسکریپت اونقدر واضح هست که نباید از این دو انتظار یکسان داشت. و مهمترین عامل به وجود اومدن چنین کامپایلرهایی استفاده از سینتکس سی شارپ بوده و نه قدرت دات نت فریم ورک. بنا براین فکر می‌کنم لزومی به وجود مبدل هایی از زبان میانی وجود ندارد و همچنین واضح است کلاس هایی از قبیل Random و غیره که نه توانایی زبان سی شارپ بلکه امکانات درونی دات نت فریم ورک است برای همچین ابزاری بی معناست.
و فکر می‌کنم برای چنین کامپایلری لازم نیست جاوا اسکریپت همه امکانات سی شارپ را داشته باشد. و اینکه سی شارپ بتواند قسمت زیادی از امکانات جاوا اسکریپت را در اختیار برنامه ساز قرار دهد کافیست.
باز هم تشکر می‌کنم