عنوان: **آموزش TypeScript #1**
نویسنده: مسعود پاکدل
تاریخ: ۱۳۹۲/۰۵/۲۳ ۸:۴۵
آدرس: www.dotnettips.info
برچسب‌ها: JavaScript, TypeScript, Client Side Framework

با گسترش روز افزون زبان برنامه نویسی Javascript و استفاده هر چه بیشتر آن در تولید برنامه‌های تحت وب این زبان به یکی از قدرت‌های بزرگ در تولید برنامه‌های مبتنی بر وب تبدیل شده است. ترکیب این زبان با Css و Html5 تقریبا هر گونه نیاز برای تهیه و توسعه برنامه‌های وب را حل کرده است. جاوا اسکریپت در ابتدا برای اسکریپت نویسی سمت کلاینت برای صفحات وب ایجاد شد و برای سال‌ها به‌عنوان ابزاری برای مدیریت کردن رویدادهای صفحات وب محدود شده بود و در نتیجه بسیاری از امکانات لازم برای برنامه‌نویسی برنامه‌های مقیاس بزرگ را به‌همراه نداشت. امروزه به قدری Javascript توسعه داده شده است که حتی در تولید برنامه‌های Native مانند Windows Store و برنامه‌های تحت Cloud نیز استفاده می‌شود. پیشرفت‌های صورت گرفته و اشاره شده در این حوزه موجب شد تا شاهد پیداش برنامه‌های مبتنی بر جاوا اسکریپت با سایزهای بی سابقه‌ای باشیم و این بیانگر این بود که تولید برنامه‌های مبتنی بر جاوا اسکریپت در مقیاس‌های بزرگ امر دشواری است و اینک TypeScript توسط غول نرم افزاری جهان پا به عرصه گذاشته که این فرآیند را آسان‌تر نماید. به کمک TypeScript می‌توان برنامه تحت JavaScript در مقیاس بزرگ تولید کرد به طوری با هر مرورگر و سیستم عاملی سازگار باشد. TypeScript از شی گرایی نیز پشتیبانی می‌کند و خروجی آن در نهایت به JavaScript کامپایل می‌شود. خیلی‌ها عقیده دارند که هدف اصلی مایکروسافت از تولید و توسعه این زبان رقابت با CoffeeScript است. CoffeeScript یک زبان متن باز است که در سال 2009 توسط Jeremy Ashkenas ایجاد شده است و سورس آن در GitHub موجود می‌باشد. در آینده، بیشتر به مباحث مربوط به CoffeeScript و آموزش آن خواهم پرداخت.

در تصویر ذیل یک مقایسه کوتاه بین CoffeeScript و TypeScript را مشاهده می‌کنید.

Feature	CoffeeScript	TypeScript
Compiles to JavaScript	●	●
Static Type Checking		●
Interfaces		●
Visual Studio Support (Web Essentials)	●	●
Intellisense		●
Loop Comprehensions	●	
Splats/RestParameters (...)	●	●
Classes	●	●
String Interpolations	●	
Proper Variable Hoisting	●	
Prevents use of ==	●	
Operator Goodness (?, < val <, etc)	●	
Write less code	●	
Stable	●	

با TypeScript چه چیزهایی به دست خواهیم آورد؟

یک نکته مهم این است که این زبان به خوبی در Visual Studio پشتیبانی می‌شود و قابلیت Intellisense نوشتن برنامه به این زبان را دلپذیرتر خواهد کرد و از طرفی دیگر به نظر من یکی از مهم‌ترین مزیت هایی که TypeScript در اختیار ما قرار می‌دهد این است که می‌توانیم به صورت Syntax آشنای شی گرایی کد نویسی کنیم و خیلی راحت‌تر کدهای خود را سازمان دهی کرده و از نوشتن کدهای تکراری اجتناب کنیم.

یکی دیگر از مزیت‌های مهم این زبان این است که این زبان از Static Typing به خوبی پشتیبانی می‌کند. این بدین معنی است که شما ابتدا باید متغیرها را تعریف کرده و نوع آن‌ها را مشخص نمایید و هم چنین در هنگام پاس دادن مقادیر به پارامترهای توابع باید حتما به نوع داده ای آن‌ها دقت داشته باشید چون کامپایلر بین انواع داده ای در TypeScript تمایز قایل است و در صورت رعایت نکردن این مورد شما با خطا مواجه خواهید شد. این تمایز قایل شدن باعث می‌شود که برنامه هایی خواناتر داشته باشیم از طرفی باعث می‌شود که خطا یابی و نوشتن تست برای برنامه راحت‌تر و تمیزتر باشد. بر خلاف JavaScript، در TypeScript(به دلیل پشتیبانی از شی گرایی) می‌توانیم علاوه بر داشتن کلاس، اینترفیس نیز داشته باشیم و در حال حاضر مزایای استفاده از اینترفیس بر کسی پوشیده نیست.

به دلیل اینکه کدهای TypeScript ابتدا کامپایل شده و بعد تبدیل به کدهای JavaScript می‌شوند در نتیجه قبل از رسیدن به مرحله اجرای پروژه، ما از خطاهای موجود در کد خود مطلع خواهیم شد.

البته این نکته را نیز فراموش نخواهیم کرد که این زبان تازه متولد شده است(سال 2012 توسط Anders Hejlsberg) و همچنان در حال توسعه است و این در حال حاضر مهم‌ترین عیب این زبان می‌تواند باشد چون هنوز به پختگی سایر زبان‌های اسکریپتی در نیامده است.

در ذیل یک مثال کوچک به زبان TypeScript و JavaScript را برای مقایسه در خوانایی و راحتی کد نویسی قرار دادم:

:TypeScript

```
class Greeter {
    greeting: string;

    constructor (message: string) {
        this.greeting = message;
    }

    greet() {
        return "Hello, " + this.greeting;
    }
}
```

بعد از کامپایل کد بالا به کدی معادل زیر در JavaScript تبدیل خواهد شد:

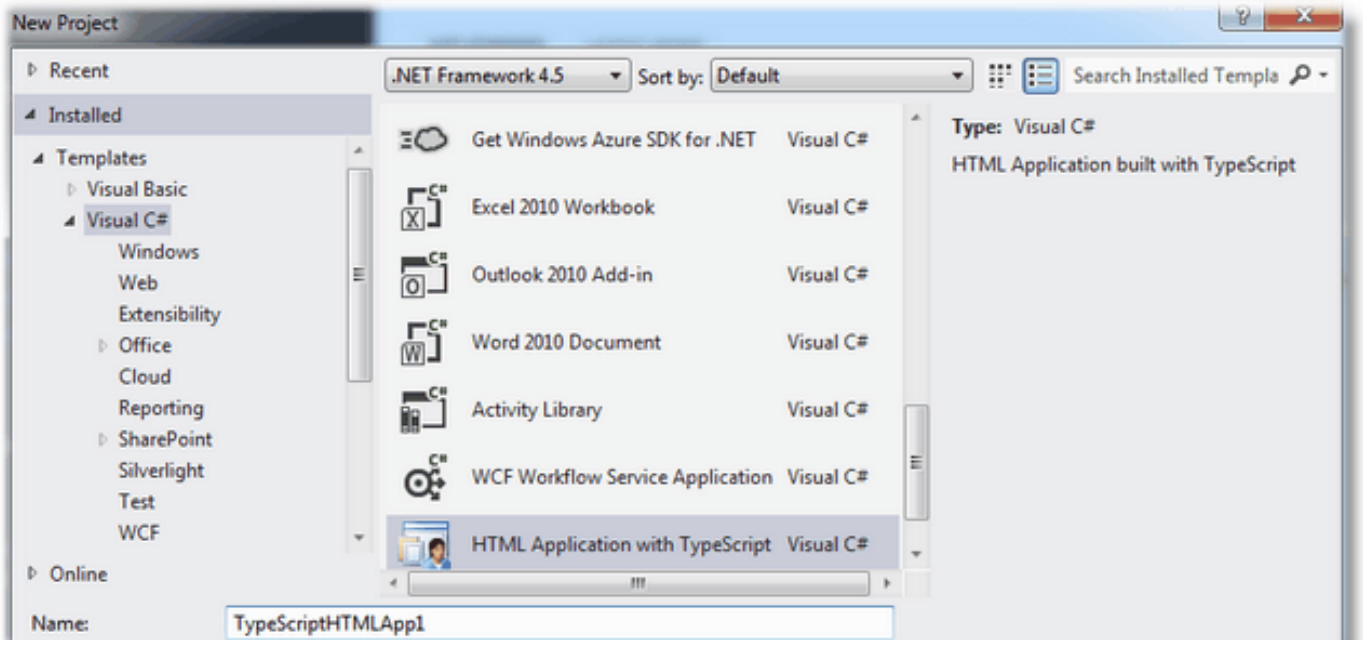
```
var Greeter = (function () {
    function Greeter(message) {
        this.greeting = message;
    }
    Greeter.prototype.greet = function () {
        return "Hello, " + this.greeting;
    };
    return Greeter;
})();
```

توضیح چند واژه در TypeScript

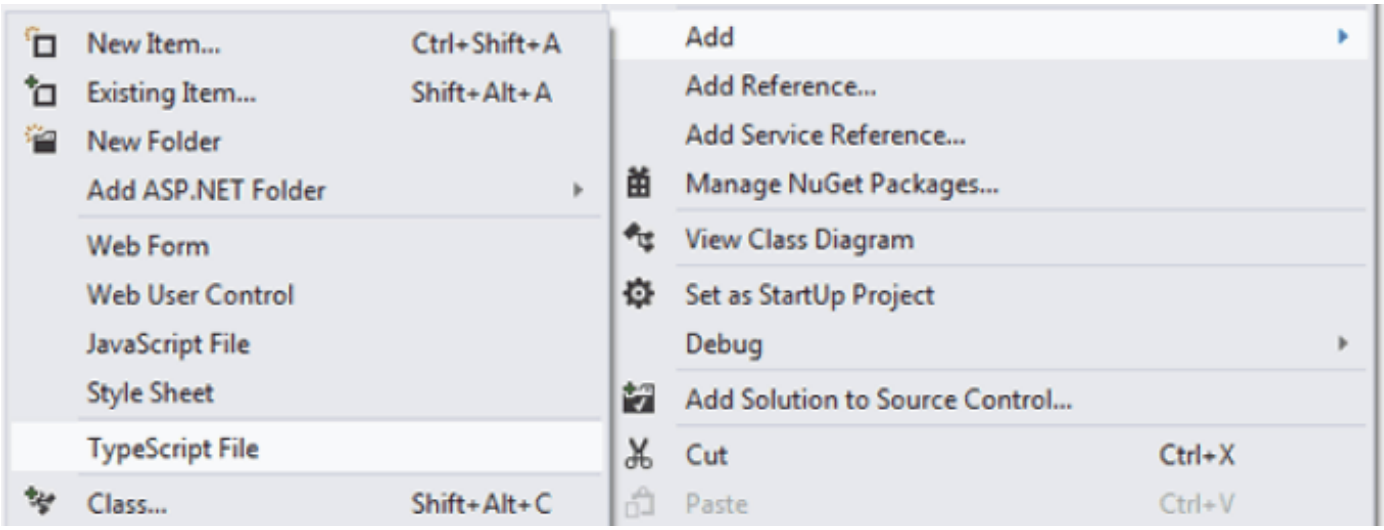
Program : یک برنامه TypeScript مجموعه ای از یک یا چند Source File است. این Source Fileها شامل کدهای پیاده سازی برنامه هستند ولی در خیلی موارد برای خوانایی بیشتر برنامه می‌توان فقط تعاریف را در این فایل‌های سورس قرار داد.
Module : ماژول در TypeScript شبیه به مفاهیم فضای نام یا namespace در دات نت است و می‌تواند شامل چندین کلاس یا اینترفیس باشد.
Class : مشابه به مفاهیم کلاس در دات نت است و دقیقا همان مفهوم را دارد. یک کلاس می‌تواند شامل چندین تابع و متغیر با سطوح دسترسی متفاوت باشد. در TypeScript مجاز به استفاده از کلمات کلیدی public و private نیز می‌باشید. یک کلاس در Typescript می‌تواند یک کلاس دیگر را توسعه دهد(ارث بری در دات نت) و چندین اینترفیس را پیاده سازی نماید.
Interface : یک اینترفیس فقط شامل تعاریف است و پیاده سازی در آن انجام نخواهد گرفت. یک اینترفیس می‌تواند چندین اینترفیس دیگر را توسعه دهد.
Function : معادل متد در دات نت است. می‌تواند پارامتر ورودی داشته باشد و در صورت نیاز یک مقدار را برگشت دهد.
Scope : دقیقا تمام مفاهیم مربوط به محدوده فضای نام و کلاس و متد در دات نت در این جا نیز صادق است.

آماده سازی Visual Studio برای شروع به کار

در ابتدا باید Template مربوطه به TypeScript را نصب کنید تا از طریف VS.Net بتوانیم به راحتی به این زبان کد نویسی کنیم. می‌توانید فایل نصب را از اینجا دانلود کنید. بعد از نصب از قسمت Templateهای موجود گزینه Html Application With TypeScript را انتخاب کنید



یا از قسمت Add در پروژه‌های وب خود نظیر MVC گزینه TypeScript File را انتخاب نمایید.



در پست بعدی کد نویسی با این زبان را آغاز خواهیم کرد.

نظرات خوانندگان

نویسنده: کامی
تاریخ: ۱۳۹۲/۰۵/۲۳ ۱۴:۲۷

باسلام
ممنون از مطالب مفیدتون
ایا می‌تونیم مثل جاوااسکریپت داخل صفحات html با استفاده از تگ script برنامه typescript بنویسیم

نویسنده: مسعود م.پاکدل
تاریخ: ۱۳۹۲/۰۵/۲۳ ۱۶:۱

بله امکان پذیر است. اما با توجه به این نکته که فلسفه وجودی TypeScript این است که در پروژه هایی با مقیاس پزرگ برای سازمان دهی کدهای سمت کلاینت مورد استفاده قرار گیرند و یکی از روش‌های سازمان دهی کدها این است که کدهای TypeScript در فایل هایی جداگانه با پسوند ts ذخیره شده تا کامپایل و تبدیل به کد JavaScript شوند (مهم‌ترین مزیت این روش این است که از نوشتن کدهای تکراری جلوگیری می‌شود). اما در صورتی که مایل به نوشتن کد به صورت Embed در تگ Script هستید باید از پروژه‌های متن بازی همچون TypeScript Compile یا ts-htaccess استفاده کنید.

نویسنده: مصطفی عسگری
تاریخ: ۱۳۹۲/۰۵/۲۴ ۱۸:۴۳

با نگاهی به زبان TypeScript متوجه میشویم که خیلی Syntax روان و آسانی دارد.
سوالی که همیشه من داشته ام این است .... چرا خود زبان JavaScript را تغییر نمیدهند؟
مسلما TypeScript و CoffeeScript برای برطرف کردن ضعف JavaScript بوجود آمده اند اما چرا خود مشکل را برطرف نمیکنند؟
میتوانستند همانند ارائه HTML5 و CSS3 نسخه جدیدی از JavaScript ارائه کنند که سختی‌های کار با JavaScript را برطرف کرده باشند!

نویسنده: مسعود م.پاکدل
تاریخ: ۱۳۹۲/۰۵/۲۵ ۱:۰

یکی از دلایل محبوبیت زبان JavaScript، راحتی در نوشتن کد با این زبان است. اگر قرار باشد این زبان یک محصول همه منظوره باشد به طور قطع دچار پیچیدگی‌های پیاده سازی شده و این همه محبوبیت به دست نمی‌آورد. هدف اولیه از تولید و توسعه زبان JavaScript، استفاده از آن در پروژه‌های سمت کلاینت بود. اما با مرور زمان و محبوبیت بیش از اندازه، توسعه گران مختلف تصمیم به توسعه این زبان گرفتند که هر محصول برای یک منظور خاص به وجود آمد. برای مثال Node.Js برای پروژه‌های RealTime استفاده می‌شود و بر مبنای منطق event-driven می‌باشد که خیلی‌ها از آن به عنوان Server side JavaScript یاد می‌کنند یا به عنوان مثال دیگر Dart محصول شرکت گوگل در سال 2011 (طراحی شده بر مبنای Scratch)و TypeScript محصول شرکت مایکروسافت در سال 2012 (طراحی شده بر مبنای JavaScript)عرضه شدند که هدف اصلی از تولید این زبان‌ها پشتیبانی از مبحث static typing و مباحث OOP برای پیاده سازی پروژه‌های در سطوحی با مقیاس بزرگ بود. JavaScript به عنوان زبان پایه باقی خواهد ماند و نسخه‌های مختلف در شکل سایر زبان‌ها و فریم ورک‌های مختلف عرضه می‌شوند تا هر کدام یک نیاز را برطرف سازند. البته در پایان این نکته را هم متذکر شوم که JavaScript هم روند با توسعه ECMAScript تغییر می‌کند. برای مثال در نسخه ECMAScript 6، امکان تعریف کلاس و ماژول در JavaScript فراهم شده است.

نویسنده: سالار
تاریخ: ۱۳۹۲/۰۵/۲۵ ۱۰:۴۰

با سلام.

- برای پروژه‌های بزرگ تحت وب که کدهای سمت کلاینت زیادی دارد استفاده از typescript را پیشنهاد میکنید؟

نویسنده: محسن خان
تاریخ: ۱۳۹۲/۰۵/۲۵ ۱۰:۵۰

Typescript - a real world story of adoption in TFS

نویسنده: مسعود م.پاکدل
تاریخ: ۱۳۹۲/۰۵/۲۵ ۱۳:۱۱

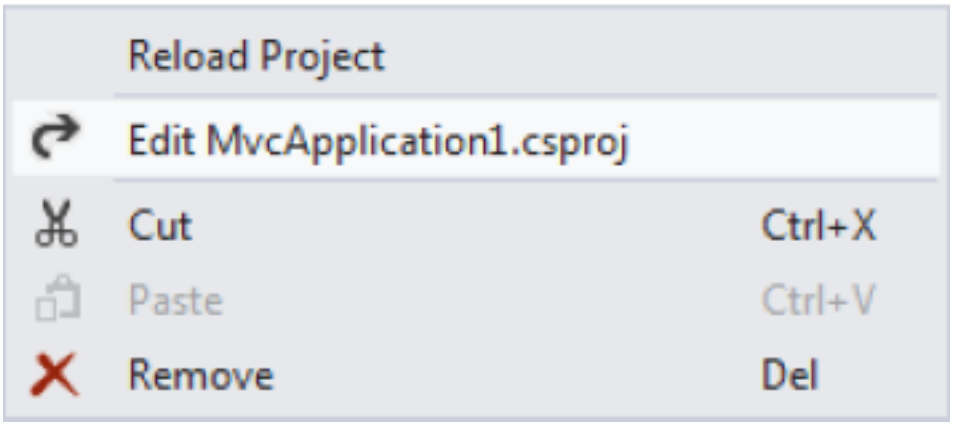
از آن جا که این زبان syntax نزدیکی به زبان‌های دات نتی دارد و به خوبی در Vs.Net پشتیبانی می‌شود نه تنها گزینه مناسبی برای توسعه در پروژه‌های وب است بلکه در توسعه پروژه‌های Windows Store App نیز می‌تواند یکی از بهترین انتخاب‌ها باشد. در ضمن این زبان به صورت **پیش فرض** از ECMA Script 3 هنگام تبدیل کدها به زبان Javascript استفاده می‌کند و تقریبا با تمام مرورگرهای قدیمی و جدید سازگار است البته به راحتی امکان تغییر این option برای سازگاری کامپایلر TypeScript با ECMA Script 5 نیز وجود دارد.

عنوان: **آموزش TypeScript #2**
نویسنده: مسعود پاکدل
تاریخ: ۱۳۹۲/۰۵/۲۶ ۸:۴۵
آدرس: www.dotnettips.info
گروه‌ها: JavaScript, TypeScript, Client Side Framework

در این پست قصد دارم به بررسی چند نکته که از پیش نیازهای کار با TypeScript است بپردازم. همان طور که در پست قبلی مشاهده شد بعد از دانلود و نصب افزونه TypeScript در VS.Net یک Template به نام Html Application With TypeScript به Installed Template اضافه خواهد شد. بعد از انتخاب این قسمت شما به راحتی می‌توانید در هر فایل با پسوند ts کدهای مورد نظر به زبان TypeScript را نوشته و بعد از build پروژه این کدها تبدیل به کدهای JavaScript خواهند شد. بعد کافیست فایل مورد نظر را با استفاده از تک Script در فایل خود رفرنس دهید. دقت کنید که پسوند فایل حتما باید js باشد(به دلیل اینکه بعد از build پروژه فایل‌های ts تبدیل به js می‌شوند).
برای مثال:

```
<!DOCTYPE html>
<html xmlns="http://www.w3.org/1999/xhtml">
<head>
    <meta charset="utf-8" />
    <title>TypeScript HTML App</title>
    <link rel="stylesheet" href="app.css" type="text/css" />
    <script src="app.js"></script>
</head>
<body>
    <h1>Number Type in TypeScript</h1>
    <div id="content"/>
</body>
</html>
```

اما اگر یک پروژه وب نظیر Asp.Net MVC داشته باشیم و می‌خواهیم یک یا چند فایل که حاوی کدهای typeScript هستند را به این پروژه اضافه کرده و از آن‌ها در صفحات وب خود استفاده کنیم باید به این صورت عمل نمود:
بعد از اضافه کردن فایل‌های مورد نیاز، پروژه مورد نظر را Unload کنید. بعد به صورت زیر فایل پروژه(csproj) رو با یک ویرایشگر متنی باز کنید:



در این مرحله باید دو قسمت اضافه شود. یک بخش ItemGroup است که هر فایلی که در پروژه شما دارای پسوند ts است باید در این جا تعریف شود. در واقع این قسمت فایل هایی را که باید کامپایل شده تا در نهایت تبدیل به فایل‌های JavaScript شوند را مشخص می‌کند.
بخش دوم target است که مراحل Build پروژه را برای این فایل‌های مشخص شده تعیین می‌کند. برای مثال:

```
  <ItemGroup>
    <TypeScriptCompile Include="$(ProjectDir)\**\*.ts" />
  </ItemGroup>
  <Target Name="BeforeBuild">
    <Exec Command="&quot;$(PROGRAMFILES)\Microsoft SDKs\TypeScript\tsc&quot;
@(TypeScriptCompile ->'&quot;%(fullpath)&quot;', ' ')" />
  </Target>
```

همان طور که می‌بینید در قسمت ItemGroup تمام فایل‌های با پسوند ts در پروژه include شده اند. در قسمت target دستور کامپایل این فایل‌ها تعیین شد. اما نکته مهم این است که TypeScript به صورت پیش فرض از ECMAScript 3 در هنگام کامپایل کدها استفاده می‌کند.(ECMAScript 3 در سال 1999 منتشر شد و تقریبا با تمام مرورگرها سازگاری دارد اما از امکانات جدید در Javascript پشتیبانی نمی‌کند). اگر قصد دارید که از ECMAScript 5 در هنگام کامپایل کدها استفاده نمایید کافیست دستور زیر را اضافه نمایید:

```
--target ES5
```

مثال:

```
  <ItemGroup>
    <TypeScriptCompile Include="$(ProjectDir)\**\*.ts" />
  </ItemGroup>
  <Target Name="BeforeBuild">
    <Exec Command="&quot;$(PROGRAMFILES)\Microsoft SDKs\TypeScript\tsc&quot;
--target ES5 @(TypeScriptCompile ->'&quot;%(fullpath)&quot;', ' ')" />
  </Target>
```

اما به این نکته دقت داشته باشید که ECMAScript 5 در سال 2009 منتشر شده است در نتیجه فقط با مرورگرهای جدید سازگار خواهد بود و ممکن است کدهای شما در مرورگرهای قدیمی با مشکل مواجه شود.
مرورگرهایی که از ECMAScript 5 پشتیبانی می‌کنند عبارتند از:
IE 9 و نسخه‌های بعد از آن؛

FireFox 4 و نسخه‌های بعد از آن؛
Opera 12 و نسخه‌های بعد از آن؛
Safari 5.1 و نسخه‌های بعد از آن؛
Chrome 7 و نسخه‌های بعد از آن.

ادامه دارد...

نظرات خوانندگان

نویسنده: آریو
تاریخ: ۱۳۹۲/۱۱/۱۵ ۱۷:۶

سلام من زمانی که فایل پروژه رو ویرایش کردم به این مشکل برخوردم . امکانش هست بگید مشکل از کجاست :

```
Error  1  The command ""C:\Program Files (x86)\Microsoft SDKs\TypeScript\0.8.0.0\tsc" -target ES5 "
exited with code 3.  c:\users\IT\documents\visual studio
2012\Projects\MvcApplication6\MvcApplication6\MvcApplication6.csproj  259  5  MvcApplication6
```

نویسنده: مسعود پاکدل
تاریخ: ۱۳۹۲/۱۱/۱۵ ۲۱:۲۹

پاسخ مورد نظر را اینجا می‌توانید مشاهده کنید

عنوان: آموزش TypeScript #3
نویسنده: مسعود پاکدل
تاریخ: ۱۳۹۲/۰۵/۲۷ ۸:۴۵
آدرس: www.dotnettips.info
برچسب‌ها: JavaScript, TypeScript, Client Side Framework

در این پست به تشریح انواع داده در زبان TypeScript و ذکر مثال در این زمینه می‌پردازیم.

تعریف متغیرها و انواع داده

در TypeScript هنگام تعریف متغیرها باید نوع داده ای آن‌ها را مشخص کنیم. در TypeScript پنج نوع داده ای وجود دارد که در زیر با ذکر مثال تعریف شده اند. مفاهیم ماژول، کلاس و تابع در پست بعدی به تشریح توضیح داده خواهند شد.

number : معادل نوع داده ای number در JavaScript است. برای ذخیره سازی اعداد صحیح و اعشاری استفاده می‌شود.
یک مثال:

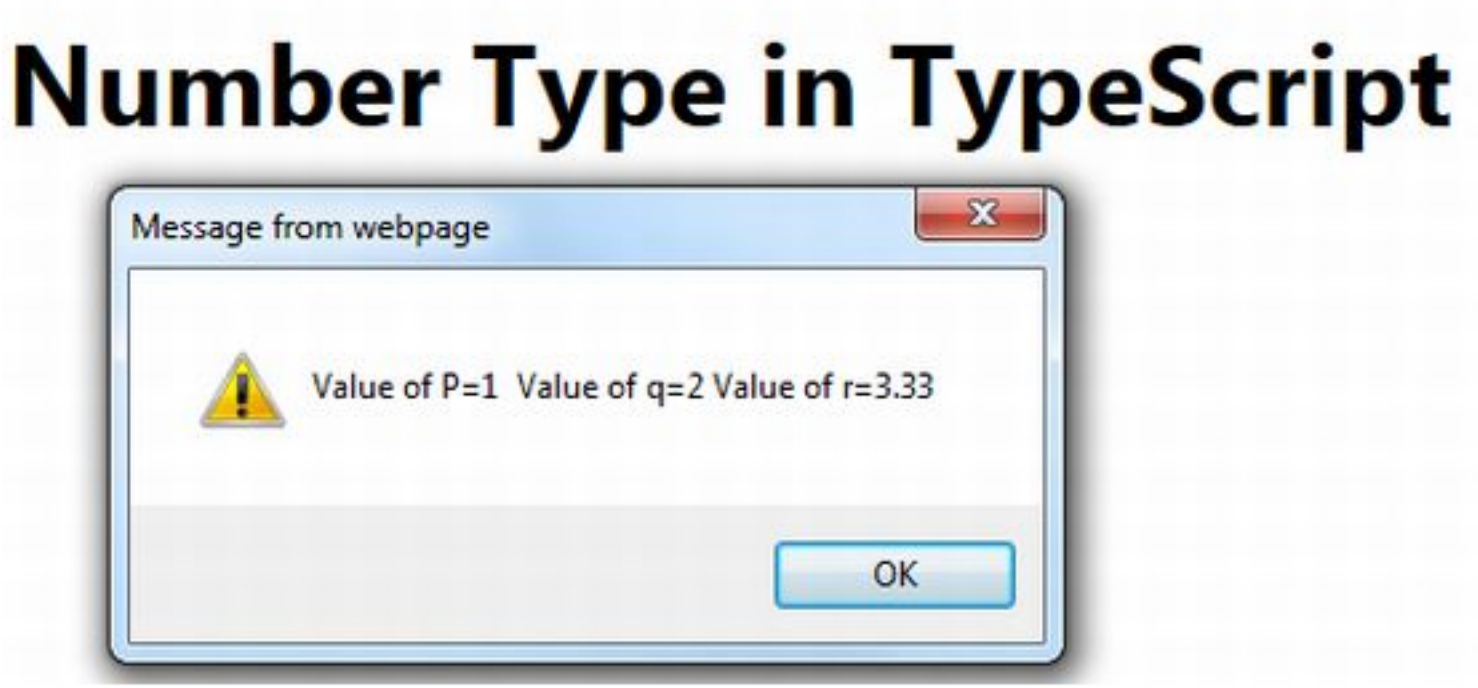
```
class NumberTypeOfTypeScript {
    MyFunction()
    {
        var p: number;
        p = 1;
        var q = 2;
        var r = 3.33;
        alert("Value of P=" + p + "  Value of q=" + q + " Value of r=" + r);
    }
}

window.onload = () =>{
    var value = new NumberTypeOfTypeScript();
    value.MyFunction();
}
```

حال باید یک فایل Html برای استفاده از این کلاس داشته باشیم. به صورت زیر:

```
<!DOCTYPE html>
<html xmlns="http://www.w3.org/1999/xhtml">
<head>
    <meta charset="utf-8" />
    <title>TypeScript HTML App</title>
    <link rel="stylesheet" href="app.css" type="text/css" />
    <script src="app.js"></script>
</head>
<body>
    <h1>Number Type in TypeScript</h1>
    <div id="content"/>
</body>
</html>
```

بعد از اجرای پروژه خروجی به صورت زیر خواهد بود:



string : معادل نوع داده ای رشته ای است و برای ذخیره سازی مجموعه ای از کاراکترها از نوع UTF-16 استفاده می‌شود.

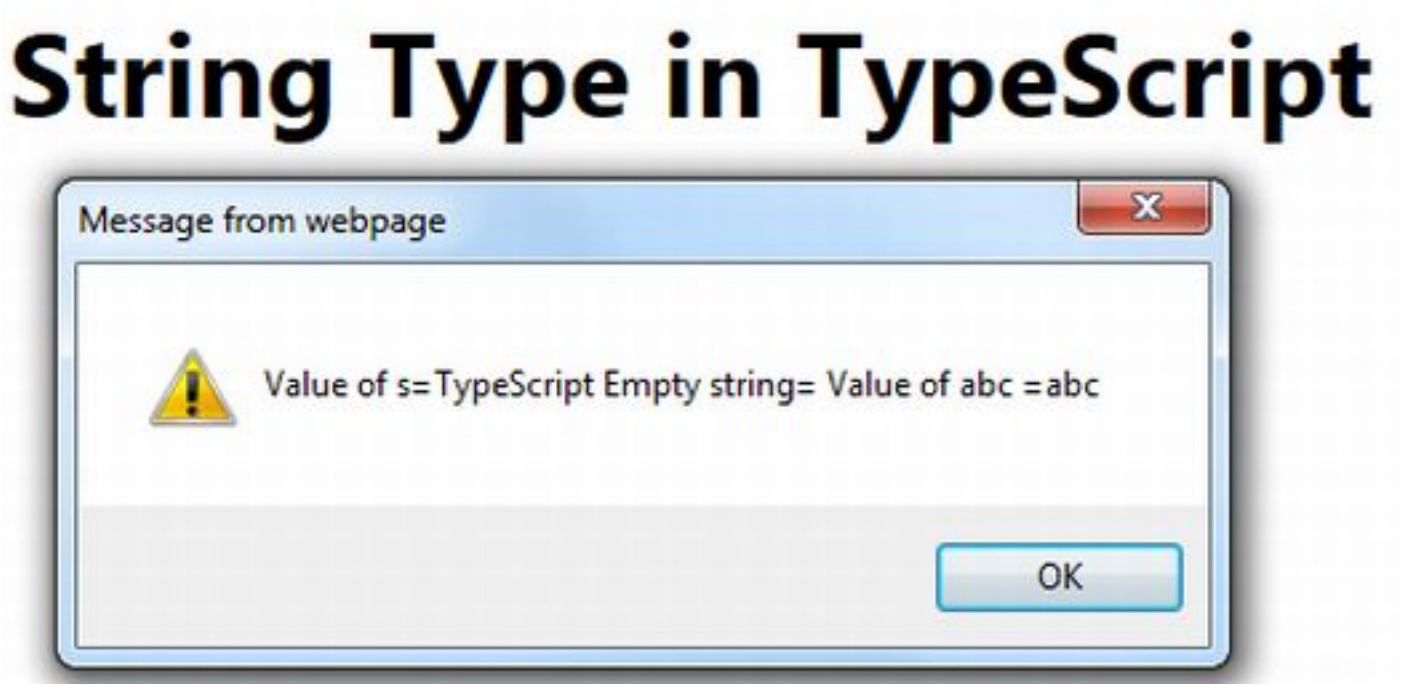
یک مثال:

```
class StringTypeOfTypeScript {
    Myfunction() {
      var  s: string;
      s="TypeScript"
      var empty = "";
      var abc = "abc";
      alert("Value of s="+ s+" Empty string="+ empty+" Value of abc ="+abc) ;
    }
}
window.onload = () =>{
    var value = new StringTypeOfTypeScript();
    value.Myfunction();
}
```

کد کامپایل شده و تبدیل آن به JavaScript:

```
var StringTypeOfTypeScript = (function () {
    function StringTypeOfTypeScript() { }
    StringTypeOfTypeScript.prototype.Myfunction = function () {
        var s;
        s = "TypeScript";
        var empty = "";
        var abc = "abc";
        alert("Value of s=" + s + " Empty string=" + empty + " Value of abc =" + abc);
    };
    return StringTypeOfTypeScript;
})();
window.onload = function () {
    var value = new StringTypeOfTypeScript();
    value.Myfunction();
};
```

خروجی به صورت زیر است:



boolean : برای ذخیره سازی مقادیر true یا false می‌باشد.

مثال:

```
class booleanTypeofTypeScript {
    MyFunction() {
        var lie: bool;
        lie = false;
        var a = 12;
        if (typeof (lie) == "boolean" && typeof (a) == "boolean") {
            alert("Both is boolean type");
        }

        if (typeof (lie) == "boolean" && typeof (a) != "boolean") {
            alert("lie is boolean type and a is not!")
        }
        else {
            alert("a is boolean type and lie is not!");
        }
    }
}

window.onload =()=> {
    var access = new booleanTypeofTypeScript();
    access.MyFunction();
 }
```

کد کامپایل شده و تبدیل آن به JavaScript:

```
var booleanTypeofTypeScript = (function () {
    function booleanTypeofTypeScript() { }
    booleanTypeofTypeScript.prototype.MyFunction = function () {
        var lie;
        lie = false;
        var a = 12;
        if(typeof (lie) == "boolean" && typeof (a) == "boolean") {
            alert("Both is boolean type");
        }
        if(typeof (lie) == "boolean" && typeof (a) != "boolean") {
            alert("lie is boolean type and a is not!");
        } else {
            alert("a is boolean type and lie is not!");
        }
    };
```

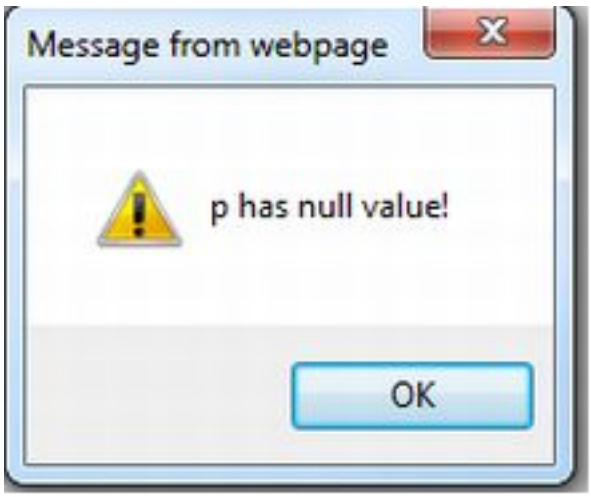
```
    return booleanTypeofTypeScript;
})();
window.onload = function () {
    var access = new booleanTypeofTypeScript();
    access.MyFunction();
};
```

null : همانند دات نت هنگامی که قصد داشته باشیم مقدار یک متغیر را null اختصاص دهیم از این کلمه کلیدی استفاده می‌کنیم.
مثال:

```
class NullTypeinTypeScript {
    MyFunction() {
        var p: number = null;
        var x = null;
        if (p== null) {
            alert("p has null value!");
        }
        else { alert("p has a value"); }
    }
}
window.onload = () =>{
    var value = new NullTypeinTypeScript();
    value.MyFunction();
}
```

کد کامپایل شده و تبدیل آن به JavaScript:

```
var NullTypeinTypeScript = (function () {
    function NullTypeinTypeScript() { }
    NullTypeinTypeScript.prototype.MyFunction = function () {
        var p = null;
        var x = null;
        if(p == null) {
            alert("p has null value!");
        } else {
            alert("p has a value");
        }
    };
    return NullTypeinTypeScript;
})();
window.onload = function () {
    var value = new NullTypeinTypeScript();
    value.MyFunction();
};
```



undefined :معادل نوع undefined در Javascript است. اگر به یک متغیر مقدار اختصاص ندهید مقدار آن undefined خواهد بود.

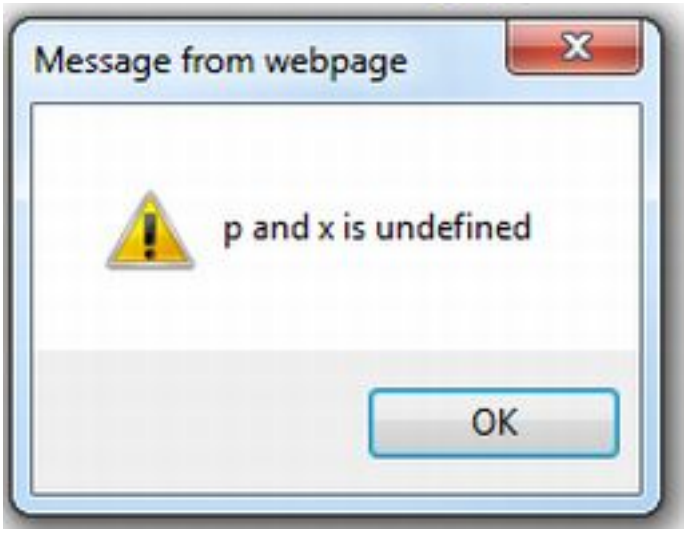
مثال:

```
class UndefinedTypeOfTypeScript {
    Myfunction() {
        var p: number;
        var x = undefined;
        if (p == undefined && x == undefined) {
            alert("p and x is undefined");
        }
        else { alert("p and c cannot undefined"); }
    }
}
window.onload = () =>{
    var value = new UndefinedTypeOfTypeScript();
    value.Myfunction();
}
```

کد کامپایل شده و تبدیل آن به JavaScript:

```
var UndefinedTypeOfTypeScript = (function () {
    function UndefinedTypeOfTypeScript() { }
    UndefinedTypeOfTypeScript.prototype.Myfunction = function () {
        var p;
        var x = undefined;
        if(p == undefined && x == undefined) {
            alert("p and x is undefined");
        } else {
            alert("p and c cannot undefined");
        }
    };
    return UndefinedTypeOfTypeScript;
})();
window.onload = function () {
    var value = new UndefinedTypeOfTypeScript();
    value.Myfunction();
};
```

خروجی این مثال نیز به صورت زیر است:



ادامه دارد...

عنوان: **آموزش TypeScript #4**
نویسنده: مسعود پاکدل
تاریخ: ۱۳۹۲/۰۵/۲۸ ۸:۴۵
آدرس: www.dotnettips.info
برچسب‌ها: JavaScript, TypeScript, Client Side Framework

در پست‌های قبل با کلیات و primitive types در زبان TypeScript آشنا شدیم:
کلیات TypeScript
انواع داده ای اولیه در TypeScript

در این پست به مفاهیم شی گرایی در این زبان می‌پردازیم.

ماژول ها:

تعریف یک ماژول: برای تعریف یک ماژول باید از کلمه کلیدی module استفاده کنید. یک ماژول معادل یک ظرف است برای نگهداری کلاس‌ها و اینترفیس‌ها و سایر ماژول ها. کلاس‌ها و اینترفیس‌ها در TypeScript می‌توانند به صورت internal یا public باشند(به صورت پیش فرض internal است؛ یعنی فقط در همان ماژول قابل استفاده و فراخوانی است). هر چیزی که در داخل یک ماژول تعریف می‌شود محدوده آن در داخل آن ماژول خواهد بود. اگر قصد توسعه یک پروژه در مقیاس بزرگ را دارید می‌توانید همانند دات نت که در آن امکان تعریف فضای نام‌های تودرتو امکان پذیر است در TypeScript نیز، ماژول‌های تودرتو تعریف کنید. برای مثال:

```
module MyModule1 {
    module  MyModule2 {
     }
}
```

اما به صورت معمول سعی می‌شود هر ماژول در یک فایل جداگانه تعریف شود. استفاده از چند ماژول در یک فایل به مرور، درک پروژه را سخت خواهد کرد و در هنگام توسعه امکان برخورد با مشکل وجود خواهد داشت. برای مثال اگر یک فایل به نام MyModule.ts داشته باشیم که یک ماژول به این نام را شامل شود بعد از کامپایل یک فایل به نام MyModule.js ایجاد خواهد شد.

کلاس ها:
برای تعریف یک کلاس می‌توانیم همانند دات نت از کلمه کلیدی class استفاده کنیم. بعد از تعریف کلاس می‌توانیم متغیرها و توابع مورد نظر را در این کلاس قرار داده و تعریف کنیم.

```
module Utilities {
   export class Logger {
      log(message: string): void{
       if(typeofwindow.console !== 'undefined') {
           window.console.log(message);
        }
      }
   }
}
```

نکته مهم و جالب قسمت بالا کلمه export است. export معادل public در دات نت است و کلاس logger را قابل دسترس در خارج ماژول Utilities خواهد کرد. اگر از export در هنگام تعریف کلاس استفاده نکنیم این کلاس فقط در سایر کلاس‌های تعریف شده در داخل همان ماژول قابل دسترس است.
تابع log که در کلاس بالا تعریف کردیم به صورت پیش فرض public یا عمومی است و نیاز به استفاده export نیست.
برای استفاده از کلاس بالا باید این کلمه کلیدی new استفاده کنیم.

```
window.onload = function() {
  varlogger = new Utilities.Logger();
  logger.log('Logger is loaded');
};
```

برای تعریف سازنده برای کلاس بالا باید از کلمه کلیدی constructor استفاده نماییم:

```
export class Logger{
constructor(private num: number) {
}
```

با کمی دقت متوجه تعریف متغیر num به صورت private خواهید شد که برخلاف انتظار ما در زبان‌های دات نتی است. بر خلاف دات نت در زبان TypeScript، دسترسی به متغیر تعریف شده در سازنده با کمک اشاره گر this در هر جای کلاس ممکن می‌باشد. در نتیجه نیازی به تعریف متغیر جدید و پاس دادن مقادیر این متغیرها به این فیلدها نمی‌باشد.
اگر به تابع log دقت کنید خواهید دید که یک پارامتر ورودی به نام message دارد که نوع آن string است. در ضمن Typescript از پارامترهای اختیاری(پارامتر با مقدار پیش فرض) نیز پشتیبانی می‌کند. مثال:

```
pad(num: number, len: number= 2, char: string= '0')
```

استفاده از پارامترهای Rest

منظور از پارامترهای Rest یعنی در هنگام فراخوانی توابع محدودیتی برای تعداد پارامترها نیست که معادل params در دات نت است. برای تعریف این گونه پارامترهاکافیست به جای params از ... استفاده نماییم.

```
function addManyNumbers(...numbers: number[]) {
  var sum = 0;
  for(var i = 0; i < numbers.length; i++) {
    sum += numbers[i];
 }
  returnsum;
}
var result = addManyNumbers(1,2,3,5,6,7,8,9);
```

تعریف توابع خصوصی

در TypeScript امکان توابع خصوصی با کلمه کلیدی private امکان پذیر است. همانند دات نت با استفاده از کلمه کلیدی private می‌توانیم کلاسی تعریف کنیم که فقط برای همان کلاس قابل دسترس باشد(به صورت پیش فرض توابع به صورت عمومی هستند).

```
module Utilities {
    Export class Logger {
     log(message: string): void{
                 if(typeofwindow.console !== 'undefined') {
                    window.console.log(this.getTimeStamp() + ' -'+ message);
                    window.console.log(this.getTimeStamp() + ' -'+ message);
                }
        }
  private getTimeStamp(): string{
      var now = newDate();
      return now.getHours() + ':'+
      now.getMinutes() + ':'+
      now.getSeconds() + ':'+
      now.getMilliseconds();
  }
 }
}
```

از آن جا که تابع getTimeStamp به صورت خصوصی تعریف شده است در نتیجه امکان استفاده از آن در خارج کلاس وجود ندارد. اگر سعی بر استفاده این تابع داشته باشیم توسط کامپایلر با یک warning مواجه خواهیم شد.

```
window.onload = function () {
    var logger = new Utilities.Logger();
    logger.getTimeStamp();
};
```

یک نکته مهم این است که کلمه private فقط برای توابع و متغیرها قابل استفاده است.

تعریف توابع static:

در TypeScript امکان تعریف توابع static وجود دارد. همانند دات نت باید از کلمه کلیدی static استفاده کنیم.

```
classFormatter {
static pad(num: number, len: number, char: string): string{
      var output = num.toString();
         while(output.length < len) {
         output = char + output;
      }
   returnoutput;
   }
  }
}
```

و استفاده از این تابع بدون وهله سازی از کلاس :

```
Formatter.pad(now.getSeconds(), 2, '0') +
```

Function Overload

همان گونه که در دات نت امکان overload کردن توابع میسر است در TypeScript هم این امکان وجود دارد.

```
static pad(num: number, len?: number, char?: string);
static pad(num: string, len?: number, char?: string);
static pad(num: any, len: number= 2, char: string= '0') {
 var output = num.toString();
 while(output.length < len) {
 output = char + output;
 }
 returnoutput;
}
```

ادامه دارد...

عنوان: **آموزش TypeScript #5**
نویسنده: مسعود پاکدل
تاریخ: ۱۳۹۲/۰۵/۳۰ ۸:۴۵
آدرس: www.dotnettips.info
برچسب‌ها: JavaScript, TypeScript, Client Side Framework

در ادامه مباحث شی گرایی در TypeScript قصد داریم به مباحث مربوط به interface و طریقه استفاده از آن بپردازیم. همانند زبان‌های دات نتی در TypeScript نیز به راحتی می‌توانید interface تعریف کنید. در یک اینترفیس اجازه پیاده سازی هیچ تابعی را ندارید و فقط باید عنوان و پارامترهای ورودی و نوع خروجی آن را تعیین کنید. برای تعریف اینترفیس از کلمه کلیدی interface به صورت زیر استفاده خواهیم کرد.

```
export interface ILogger {
  log(message: string): void;
}
```

همان طور در پست‌های قبلی مشاهده شد از کلمه کلیدی export برای عمومی کردن اینترفیس استفاده می‌کنیم. یعنی این اینترفیس از بیرون ماژول خود نیز قابل دسترسی است.
حال نیاز به کلاسی داریم که این اینترفیس را پیاده سازی کند. این پیاده سازی به صورت زیر انجام می‌گیرد:

```
export class Logger implements ILogger
{
}
```

یا:

```
export class AnnoyingLogger implements ILogger {
   log(message: string): void{
        alert(message);
    }
}
```

همانند دات نت یک کلاس می‌تواند چندین اینترفیس را پیاده سازی کند.(اصطلاحا به این روش explicit implementation یا پیاده سازی صریح می‌گویند)

```
export class MyClass implements IFirstInterface, ISecondInterface
{

}
```

*یکی از قابلیت جالب و کارآمد زبان TypeScript این است که در هنگام کار با اینترفیس‌ها حتما نیازی به پیاده سازی صریح نیست. اگر یک object تمام متغیرها و توابع مورد نیاز یک اینترفیس را پیاده سازی کند به راحتی همانند روش explicit emplementation می‌توان از آن object استفاده کرد. به این قابلیت **Duck Typing** می‌گویند. مثال:

```
IPerson {
   firstName: string;
   lastName: string;
}
class Person implements IPerson {
  constructor(public firstName: string, public lastName: string) {
  }
}

varpersonA: IPerson = newPerson('Masoud', 'Pakdel'); //expilict
varpersonB: IPerson = { firstName: 'Ashkan', lastName: 'Shahram'}; // duck typing
```

همان طور که می‌بینید object دوم به نام personB تمام متغیرها ی مورد نیاز اینترفیس IPerson را پیاده سازی کرده است در

نتییجه کامپایلر همان رفتاری را که با object اول به نام personA دارد را با آن نیز خواهد داشت.

پیاده سازی چند اینترفیس به صورت همزمان

همانند دات نت که یک کلاس فقط می‌تواند از یک کلاس ارث ببرد ولی می‌تواند n تا اینترفیس را پیاده سازی کند در TypeScript نیز چنین قوانینی وجود دارد. یعنی یک اینترفیس می‌تواند چندین اینترفیس دیگر را توسعه دهد(extend) و کلاسی که این اینترفیس را پیاده سازی می‌کند باید تمام توابع اینترفیس‌ها را پیاده سازی کند. مثال:

```
interface IMover {
 move() : void;
}

interface IShaker {
 shake() : void;
}

interface IMoverShaker extends IMover, IShaker {
}
class MoverShaker implements IMoverShaker {
 move() {
 }
 shake() {
 }
}
```

*به کلمات کلیدی extends و implements و طریقه به کار گیری آن‌ها دقت کنید.

instanceof

از instanceof زمانی استفاده می‌کنیم که قصد داشته باشیم که یک instance را با یک نوع مشخص مقایسه کنیم. اگر instance مربوطه از نوع مشخص باشد یا از این نوع ارث برده باشد مقدار true برگشت داده می‌شود در غیر این صورت مقدار false خواهد بود.

یک مثال:

```
var isLogger = logger instanceof Utilities.Logger;
var isLogger = logger instanceof Utilities.AnnoyingLogger;
var isLogger = logger instanceof Utilities.Formatter;
```

Method overriding

در TypeScript می‌توان مانند زبان‌های شی گرای دیگر Method overriding را پیاده سازی کرد. یعنی می‌توان متدهای کلاس پایه را در کلاس مشتق شده تحریف کرد. با یک مثال به شرح این مورد خواهم پرداخت.

فرض کنید یک کلاس پایه به صورت زیر داریم:

```
class BaseEmployee
{
    constructor (public fname: string,public lname: string)
    {
    }
    sayInfo()
    {
       alert('this is base class method');
    }
}
```

کلاس دیگری به نام Employee می‌سازیم که کلاس بالا را توسعه می‌دهد(یا به اصطلاح از کلاس بالا ارث می‌برد).

```
class Employee extends BaseEmployee
{
   sayInfo()
     {
        alert('this is derived class method');
     }
}
```

```
window.onload = () =>
{
    var first: BaseEmployee= new Employee();
    first.sayInfo();
    var second: BaseEmployee = new BaseEmployee();
    second.sayInfo();
}
```

نکته مهم این است که دیگر خبری از کلمه کلیدی virtual برای مشخص کردن توابعی که قصد overriding آن‌ها را داریم نیست. تمام توابع که عمومی هستند را می‌توان override کرد.

*اگر در کلاس مشتق شده قصد داشته باشیم که به توابع و فیلدهای کلاس پایه اشاره کنیم باید از کلمه کلیدی super استفاده کنیم.(معادل base در #C).

مثال:

```
class Animal {
    constructor (public name: string) {
    }
}

class Dog extends Animal {
      constructor (public name: string, public age:number)
      {
        super(name);
      }

    sayHello() {


      alert(super.name);


 }
}
```

اگر به سازنده کلاس مشتق شده دقت کنید خواهید دید که پارامتر name را به سازنده کلاس پایه پاس دادیم: کد معادل در #C به صورت زیر است:

```
public class Dog : Animal
{
      public Dog (string name, int age):base(name)
      {
      }
}
```

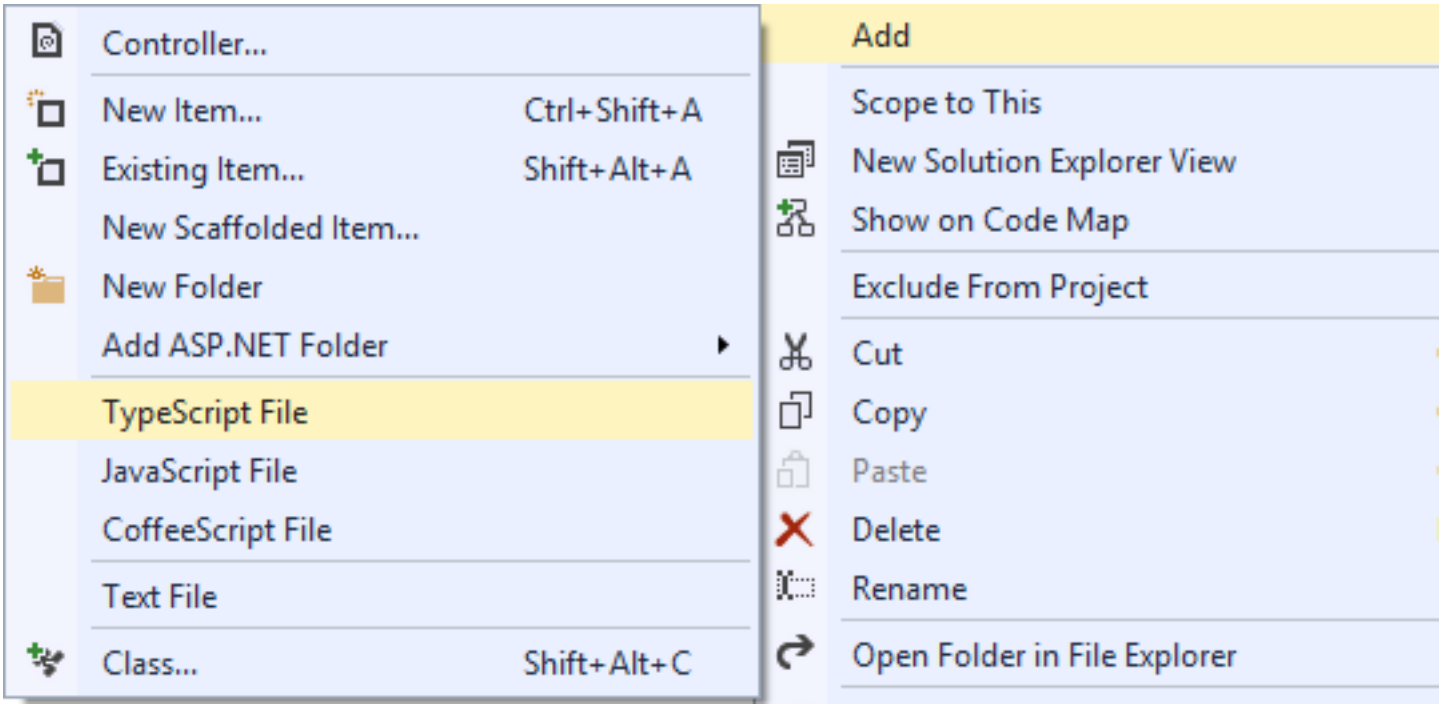
در تابع sayHello نیز با استفاده از کلمه کلیدی super به فیلد name در کلاس پایه دسترسی خواهیم داشت.

*دقت کنید که مباحث مربوط به interface و private modifier و Type safety که پیش‌تر در مورد آن‌ها بحث شد، فقط در فایل‌های TypeScript و در هنگام کد نویسی و طراحی معنی دار هستند، زیرا بعد از کامپایل فایل‌های ts این مفاهیم در Javascript پشتیبانی نمی‌شوند در نتیجه هیچ مورد استفاده هم نخواهد داشت.

ادامه دارد...

عنوان: **پیاده سازی کنترلرهای Angular با استفاده از Typescript**
نویسنده: مسعود پاکدل
تاریخ: ۱۳۹۲/۱۰/۰۸ ۲۲:۵
آدرس: www.dotnettips.info
گروه‌ها: TypeScript, AngularJS

پیشتر با ویژگی ها و نحوه کد نویسی این زبان آشنا شدید. از طرفی دیگر، نحوه تعریف کنترلرها در Angular نیز آموزش داده شد . در این پست قصد دارم طی یک مثال ساده با استفاده از زبان Typescript یک کنترلر Angular را ایجاد و سپس از آن در یک پروژه Asp.Net MVC استفاده نمایم. از آن جا که به صورت پیش فرض در VS.Net امکانات TypeScript نصب نشده است، برای شروع ابتدا TypeScript را از اینجا دانلود نمایید. بعد از نصب یک پروژه Asp.Net MVC ایجاد نمایید و سپس با استفاده از nuget فایل‌های مربوط به AngularJs را نصب نمایید. در این پست به تفصیل این مورد بررسی شده است (عملیات BundleConfig فایل‌های مورد نیاز به عهده خودتان). در پوشه scripts یک فولدر به نام app ساخته، سپس یک فایل TypeScript به نام ProductController.ts ایجاد کنید. (بعد از نصب TypeScript گزینه TypeScript File مشاهده خواهد شد)



در فایل ProductController.ts کدهای زیر را کپی نمایید:

```
module Product {
    export interface Scope {
        message: string;
    }

    export class Controller {
        constructor($scope: Scope) {
            $scope.message = "Hello from Masoud";
        }
    }
}
```

توضیح کدها بالا :

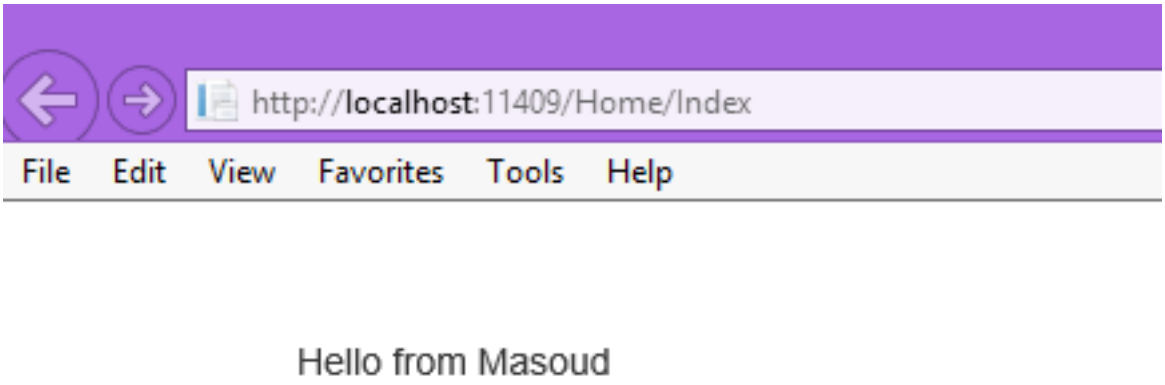
ابتدا یک ماژول به نام Product ایجاد می‌کنیم. سپس یک اینترفیس برای پیاده سازی آبجکت Scope که جهت مقید سازی عناصر DOM به آبجکت‌های کنترلر مورد استفاده قرار می‌گیرد، ایجاد می‌کنیم. در داخل این اینترفیس متغیری به نام message از نوع string داریم. قصد داریم این متغیر را به یک عنصر مقید کنیم. حال یک کلاس به نام کنترلر ایجاد می‌کنیم که در تابع سازنده آن تزریق

وابستگی برای $scope از نوع اینترفیس Scope تعیین شده است. در نتیجه در بدنه سازنده می‌توانیم به متغیر message مقدار مورد نظر را نسبت دهیم .

کلمه کلیدی export برای تعریف عمومی کلاس استفاده شده است .
یک View ایجاد و کدهای زیر را در آن کپی کنید :

```
<script type="text/javascript" src="~/scripts/app/ProductController.js"></script>
<div ng-app>
    <div ng-controller="Product.Controller">
        <p>{{message}}</p>
    </div>
</div>
```

اولین نکته در تگ script است که فراخوانی فایل TypeScript باید با پسوند js. انجام گیرد. به دلیل اینکه فایل‌های TypeScript بعد از کامپایل تبدیل به فایل‌های JavaScript خواهند شد؛ در نتیجه پسوند آن نیز js. است. دومین نکته در فراخوانی کنترلر مورد نظر است که از ترکیب نام ماژول و نام کلاس است. بعد از اجرای پروژه خروجی به صورت زیر خواهد بود :



نظرات خوانندگان

نویسنده: ایلیا اکبری فرد
تاریخ: ۱۳۹۲/۱۰/۲۵ ۱۵:۲۶

با سلام.
در کنترلر ، چگونه $watch مربوط به شی $scope را بوسیله TypeScript میتوان فراخوانی کرد؟

نویسنده: مسعود پاکدل
تاریخ: ۱۳۹۲/۱۰/۲۷ ۱۳:۷

ابتدا کنترلر خود را به صورت زیر تعریف کنید:

```
class MyController {
        thescope: any;
        static $inject = ['$scope'];
```

```
constructor($scope) {
              this.thescope = $scope;
```

```
this.thescope.$watch('watchtext', function(newValue, oldValue) {
              this.thescope.counter = scope.counter + 1;
              this.thescope.lastvalue = oldValue;
              this.thescope.currentvalue = newValue;});
```

```
}
    }
```

حال برای استفاده از کنترلر بالا به صورت زیر عمل نمایید:

```
module myApp.ctrl {
    myApp.controller("MyController", function($scope) {
        return new MyController($scope);
    }
}
```

نویسنده: محمد آزاد
تاریخ: ۱۳۹۴/۰۱/۲۹ ۱:۲۲

من از ورژن angular 1.3.15 استفاده میکنم و در صفحه یک چنین اروری میده Error: [ng:areq]
http://errors.angularjs.org/1.3.15/ng/areq?p0=Product.Controller&p1=not%20a%20function%2C%20got%20undefined
مثل اینکه نوع تعریف controller در این ورژن متفاوت با قبله.چطور میشه کد typescript رو اصلاح کرد تا مشکل حل بشه؟

عنوان: **واکشی اطلاعات سرویس Web Api با استفاده از TypeScript و AngularJs**
نویسنده: مسعود پاکدل
تاریخ: ۱۳۹۲/۱۰/۰۹ ۸:۴۵
آدرس: www.dotnettips.info
گروه‌ها: TypeScript, AngularJS, Web API

در پست‌های قبلی با AngularJs ، TypeScript و Web Api آشنا شدید. در این پست قصد دارم از ترکیب این موارد برای پیاده سازی عملیات واکشی اطلاعات سرویس Web Api در قالب یک پروژه استفاده نمایم. برای شروع ابتدا یک پروژه Asp.Net MVC ایجاد کنید.
در قسمت مدل ابتدا یک کلاس پایه برای مدل ایجاد خواهیم کرد:

```
public abstract class Entity
    {
        public Guid Id { get; set; }
    }
```

حال کلاسی به نام Book ایجاد می‌کنیم:

```
public class Book : EntityBase
    {
        public string Name { get; set; }
        public decimal Author { get; set; }
    }
```

در پوشه مدل یک کلاسی به نام BookRepository ایجاد کنید و کدهای زیر را در آن کپی نمایید(به جای پیاده سازی بر روی بانک اطلاعاتی، عملیات بر روی لیست درون حافظه انجام می‌گیرد):

```
public class BookRepository
    {
        private readonly ConcurrentDictionary<Guid, Book> result = new ConcurrentDictionary<Guid,
Book>();

        public IQueryable<Book> GetAll()
        {
            return result.Values.AsQueryable();
        }

        public Book Add(Book entity)
        {
            if (entity.Id == Guid.Empty) entity.Id = Guid.NewGuid();

            if (result.ContainsKey(entity.Id)) return null;

            if (!result.TryAdd(entity.Id, entity)) return null;

            return entity;
        }
    }
```

نوبت به کلاس کنترلر می‌رسد. یک کنترلر Api به نام BooksController ایجاد کنید و سپس کدهای زیر را در آن کپی نمایید:

```
public class BooksController : ApiController
    {
        public static BookRepository repository = new BookRepository();

public BooksController()
        {
            repository.Add(new Book
            {
                Id=Guid.NewGuid(),
                Name="C#",
                Author="Masoud Pakdel"
            });

            repository.Add(new Book
```

```
            {
                Id = Guid.NewGuid(),
                Name = "F#",
                Author = "Masoud Pakdel"
            });

            repository.Add(new Book
            {
                Id = Guid.NewGuid(),
                Name = "TypeScript",
                Author = "Masoud Pakdel"
            });
        }

        public IEnumerable<Book> Get()
        {
            return repository.GetAll().ToArray();
        }
    }
```

در این کنترلر، اکشنی به نام Get داریم که در آن اطلاعات کتاب‌ها از Repository مربوطه برگشت داده خواهد شد. در سازنده این کنترلر ابتدا سه کتاب به صورت پیش فرض اضافه می‌شود و انتظار داریم که بعد از اجرای برنامه، لیست مورد نظر را مشاهده نماییم.

حال نوبت به عملیات سمت کلاینت میرسد. برای استفاده از قابلیت‌های TypeScript و AngularJs در Vs.Net از این مقاله کمک بگیرید. بعد از آماده سازی در فولدر script، پوشه ای به نام app می‌سازیم و یک فایل TypeScript به نام BookModel در آن ایجاد می‌کنیم:

```
module Model {
    export class Book{
        Id: string;
        Name: string;
        Author: string;
    }
}
```

واضح است که ماژولی به نام Model داریم که در آن کلاسی به نام Book ایجاد شده است. برای انتقال اطلاعات از طریق سرویس $http در Angular نیاز به سریالایز کردن این کلاس به فرمت Json خواهیم داشت. قصد داریم View مورد نظر را به صورت زیر ایجاد نماییم:

```
<div ng-controller="Books.Controller">
        <table class="table table-striped table-hover" style="width: 500px;">
            <thead>
                <tr>
                    <th>Name</th>
                    <th>Author</th>
                </tr>
            </thead>
            <tbody>
                <tr ng-repeat="book in books">
                    <td>{{book.Name}}</td>
                    <td>{{book.Author}}</td>
                </tr>
            </tbody>
        </table>
    </div>
```

توضیح کدهای بالا:
ابتدا یک کنترلری که به نام Controller که در ماژولی به نام Book تعریف شده است باید ایجاد شود. اطلاعات تمام کتب ثبت شده باید از سرویس مورد نظر دریافت و با یک ng-repeat در جدول نمایش داده خواهند شود.
در پوشه app یک فایل TypeScript دیگر برای تعریف برخی نیازمندی‌ها به نام AngularModule ایجاد می‌کنیم که کد آن به صورت زیر خواهد بود:

```
declare module AngularModule {
    export interface HttpPromise {
        success(callback: Function) : HttpPromise;
    }
    export interface Http {
        get(url: string): HttpPromise;
    }
}
```

در این ماژول دو اینترفیس تعریف شده است. اولی به نام HttpPromise است که تابعی به نام success دارد. این تابع باید بعد از موفقیت آمیز بودن عملیات فراخوانی شود. ورودی آن از نوع Function است. بعنی اجازه تعریف یک تابع را به عنوان ورودی برای این توابع دارید.
در اینترفیس Http نیز تابعی به نام get تعریف شده است که برای دریافت اطلاعات از سرویس api، مورد استفاده قرار خواهد گرفت. از آن جا که تعریف توابع در اینترفیس فاقد بدنه است در نتیجه این جا فقط امضای توابع مشخص خواهد شد. پیاده سازی توابع به عهده کنترلرها خواهد بود:
مرحله بعد مربوط است به تعریف کنترلری به نام BookController تا اینترفیس بالا را پیاده سازی نماید. کدهای آن به صورت زیر خواهد بود:

```
/// <reference path='AngularModule.ts' />
/// <reference path='BookModel.ts' />

module Books {
    export interface Scope {
        books: Model.Book[];
    }

    export class Controller {
        private httpService: any;

        constructor($scope: Scope, $http: any) {
            this.httpService = $http;

            this.getAllBooks(function (data) {
                $scope.books = data;
            });
            var controller = this;
    }

        getAllBooks(successCallback: Function): void {
            this.httpService.get('/api/books').success(function (data, status) {
                successCallback(data);
            });
        }
    }
}
```

توضیح کدهای بالا:
برای دسترسی به تعاریف انجام شده در سایر ماژول‌ها باید ارجاعی به فایل تعاریف ماژول‌های مورد نظر داشته باشیم. در غیر این صورت هنگام استفاده از این ماژول‌ها با خطای کامپایلری روبرو خواهیم شد. عملیات ارجاع به صورت زیر است:

```
/// <reference path='AngularModule.ts' />
/// <reference path='BookModel.ts' />
```

در پست قبلی توضیح داده شد که برای مقید سازی عناصر بهتر است یک اینترفیس به نام Scope تعریف کنیم تا بتوانیم متغیرهای مورد نظر برای مقید سازی را در آن تعریف نماییم در این جا تعریف آن به صورت زیر است:

```
export interface Scope {
        books: Model.Book[];
    }
```

در این جا فقط نیاز به لیستی از کتاب‌ها داریم تا بتوان در جدول مورد نظر در View آنرا پیمایش کرد. تابعی به نام getAllBooks در کنترلر مورد نظر نوشته شده است که ورودی آن یک تابع خواهد بود که باید بعد از واکشی اطلاعات از سرویس، فراخوانی شود.
اگر به کدهای بالا دقت کنید می‌بینید که در ابتدا سازنده کنترلر،سرویس $http موجود در Angular به متغیری به نام httpService نسبت داده می‌شود. با فراخوانی تابع get و ارسال آدرس سرویس که با توجه به مقدار مسیر یابی پیش فرض کلاس WebApiConfig باید با api شروع شود به راحتی اطلاعات مورد نظر به دست خواهد آمد. بعد از واکشی در صورت موفقیت آمیز بودن عملیات تابع success اجرا می‌شود که نتیجه آن انتساب مقدار به دست آمده به متغیر books تعریف شده در $scope می‌باشد.

در نهایت خروجی به صورت زیر خواهد بود:

Name	Author
C#	Masoud Pakdel
TypeScript	Masoud Pakdel
F#	Masoud Pakdel

سورس پیاده سازی مثال بالا در Visual Studio 2013

نظرات خوانندگان

نویسنده: sadegh hp
تاریخ: ۱۳۹۲/۱۲/۲۳ ۱۱:۳۳

چجوری میشه با jasmine یک تست برای متدی که $http.post رو در یک سرویس انگولار پیاده کرده نوشت؟ تست متدهای async در انگولار چجوریه ؟

نویسنده: مسعود پاکدل
تاریخ: ۱۳۹۲/۱۲/۲۳ ۱۳:۱

angularJs کتابخانه ای برای mock آبجکت‌ها خود تهیه کرده است.(angular-mock) . از آن جا که در angular مبحث تزریق وابستگی بسیار زیبا پیاده سازی شده است با استفاده از این کتابخانه می‌توانید آبجکت‌های متناظر را mock کنید. برای مثال:

```
describe('myApp', function() {
var scope;

 beforeEach(angular.mock.module('myApp'));
 beforeEach(angular.mock.inject(function($rootScope) {
    scope = $rootScope.$new();
});
it('...')
});
```

هم چنین برای تست سرویس $http و شبیه سازی عملیات request و response در انگولار سرویس $httpBackend تعبیه شده است که یک پیاده سازی Fake از $http است که در تست‌ها می‌توان از آن استفاده کرد. برای مثال:

```
describe('Remote tests', function() {
    var $httpBackend, $rootScope, myService;
       beforeEach(inject(
function(_$httpBackend_, _$rootScope_, _myService_) {
      $httpBackend = _$httpBackend_;
      $rootScope = _$rootScope_;
      myService = _myService_;
}));
it('should make a request to the backend', function() {
   $httpBackend.expect('GET', '/v1/api/current_user')
     .respond(200, {userId: 123});
     myService.getCurrentUser();

     $httpBackend.flush();
});
});
```

دستور $httpBackend.expect برای ایجاد درخواست مورد نظر استفاده می‌شود که نوع verb را به عنوان آرگومان اول دریافت می‌کند. respond نیز مقدار بازگشتی مورد انتظار از سرویس مورد نظر را بر میگرداند. می‌توانید از دستورات زیر برای سایر حالات استفاده کنید:

- $httpBackend.expectGet
- $httpBackend.expectPut
- $httpBackend.expectPost
- $httpBackend.expectDelete
- $httpBackend.expectJson
- $httpBackend.expectHead
- $httpBackend.expectPatch

Flush کردن سرویس $httpBackend در پایان تست نیز برای همین مبحث async اجرا شدن سرویس‌های $http backend است.

نویسنده: صادق اچ پی
تاریخ: ۱۳۹۲/۱۲/۲۵ ۹:۴۸

ممنون از پاسخ شما.
اما سوال بعد اینکه چرا اصلا باید بیرون از سرویس http رو ساخت؟ فرض کنید که ما دسترسی به محتوی متود درون سرویس نداریم و فقط میخواهیم اون رو صدا کنیم و ببینیم که متود درون سرویس درست کار میکنه یا نه! بدون اینکه بدونیم چجوری داخل متود پیاده سازی شده که در این مورد یک http.post یا get هست.

نویسنده: مسعود پاکدل
تاریخ: ۱۳۹۲/۱۲/۲۵ ۱۰:۴۳

$httpBackend یک پیاده سازی fake از $http است، در نتیجه می‌توانید در هنگام تست، این سرویس را به کنترلرهای خود تزریق کنید. اما قبل از DI باید برای این سرویس مشخص شود که برای مثال در هنگام مواجه شدن با یک درخواست از نوع Get و آدرس X چه خروجی برگشت داده شود. درست شبیه به رفتار mocking framework ها. فرض کنید شما کنترلری به شکل زیر دارید:

```
(function (module) {

    var myController = function ($scope, $http) {

        $http.get("/api/myData")
            .then(function (result) {
                $scope.data= result.data;
            });
    };

    module.controller("MyController",
        ["$scope", "$http", myController]);

}(angular.module("myApp")));
```

همان طور که می‌بینید در این کنترلر از $http استفاده شده است. حال برای تست آن می‌توان نوشت:

```
describe("myApp", function () {

    beforeEach(module('myApp'));

    describe("MyController", function () {

        var scope, httpBackend;
        beforeEach(inject(function ($rootScope, $controller, $httpBackend, $http) {
            scope = $rootScope.$new();
            httpBackend = $httpBackend;
            httpBackend.when("GET", "/api/myData").respond([{}, {}, {}]);
            $controller('MyController', {
                $scope: scope,
                $http: $http
            });
        }));

        it("should have 3 row", function () {
            httpBackend.flush();
            expect(scope.data.length).toBe(3);
        });
    });
});
```

httpBackend ساخته شده با استفاده از سرویس $controller به کنترلر مورد نظر تزریق می‌شود. حال اگر در یک کنترلر 5 بار از سرویس $http برای فراخوانی 5 resource متفاوت استفاده شده باشد باید برای هر حالت $httpBackend را طوری تنظیم کرد که بداند برای هر منبع چه خروجی در اختیار کنترلر قرار دهد.

عنوان: **معرفی چند پروژه‌ی مهم Typescript**
نویسنده: علی سبزواری
تاریخ: ۱۳۹۳/۱۲/۲۳ ۲:۵
آدرس: www.dotnettips.info
گروه‌ها: JavaScript, TypeScript

فلسفه‌ی بوجود آمدن زبان Typescript یکی از شنیدنی‌ترین‌ها در دنیای برنامه‌نویسی است. به یاد دارم روزهای اولی که با این زبان آشنا شدم (زمانی که حدوداً ورژن 0.6 منتشر شده بود)، افراد زیادی در مورد این زبان و اینکه آیا اصلاً به این زبان احتیاج داریم یا نه نظرات زیادی دادند. مثلاً Douglas Crockford در مورد این زبان بعد از تعریف و تمجیدهایی که از Anders Hejlsberg کرده گفته :

```
I think that JavaScript's loose typing is one of its best features and that type checking is way overrated. TypeScript adds sweetness, but at a price. It is not a price I am willing to pay.
```

اما به مرور زمان این زبان توفیق بیشتری پیدا کرد تا اینکه امروز پروژه‌های بسیار جالبی با این زبان در حال توسعه هستند.

چرا باید در مورد Typescript بدانیم؟

زبان Typescript نقاط قوت بسیاری دارد، از جمله‌ی آنها می‌توان به موارد زیر اشاره کرد:

این زبان یکی از مشکلات اصلی JavaScript را که نبودن Type Safety می‌باشد حل کرده‌است. اگر چه زبانی که type safe نباشد بسیاری اوقات مزیت است! زبان typescript در حقیقت یک زبان gradual typing است.
از آنجایی که typescript یک super set از زبان JavaScript است، برنامه‌نویس در لحظه از مزایای زبان JavaScript هم بهره‌مند است. مهم‌تر از آن این است که در زبان typescript به اقیانوس کتابخانه‌های JavaScript دسترسی دارید. این امکان در بسیاری زبان‌های دیگر جایگزین JavaScript وجود ندارد. حتی بهتر از آن، می‌تواند با این کتابخانه‌ها به‌صورت type safe برنامه بنویسید. تصور کنید که وقتی با $ در JQuery کار می‌کنید بتوانید از امکان intellisense استفاده کنید.
بازهم از آنجا که typescript یک super set از JavaScript است، typescript قرار نیست به اسمبلی کامپایل شود؛ بلکه به زبان شناخته شده‌ای به نام JavaScript تبدیل می‌شود. بنابراین حتی می‌توان از آن JavaScript نیز یاد گرفت.
کار با زبان typescript برای کسانی که با java یا سی شارپ آشنا هستند، راحت است. امکاناتی مانند genericها نیز در typescript وجود دارد.
نقشه‌ی راه typescript با EcmaScript هماهنگ است. بنابراین از یادگرفتن این زبان ضرر نمی‌کنید چون قابلیت‌های این زبان را به احتمال زیاد در نسخه‌ی بعدی EcmaScript خواهید دید.
این زبان توسط شرکت مایکروسافت پشتیبانی می‌شود، اوپن سورس است و تجربه‌ی Anders Hejlsberg در زمینه‌ی طراحی زبان‌های برنامه‌نویسی پشتیبان آن!
پروژه‌های جالبی که در ادامه به معرفی آنها می‌پردازیم، با این زبان در حال توسعه هستند.

در این مطلب تعدادی از این پروژه‌ها را که برای خودم جذاب هستند، به شما معرفی می‌کنم.

AngularJS 2

طبیعتاً مهم‌ترین اتفاقی که برای typescript در این روزهای اخیر افتاد این بود که تیم Angular اعلام کرد که نسخه‌ی ۲ این فریم‌ورک (که این روزها در حد JQuery در وب معروف شده و استفاده می‌شود) را با زبان Typescript توسعه می‌دهد و امکاناتی که قرار بود توسط زبان AtScript پیاده‌سازی شوند، به کمک Typescript توسعه پیدا می‌کنند. تیم Typescript هم بلافاصله اعلام کرد که در نسخه‌ی 1.5 که به‌زودی منتشر می‌شود بسیاری از امکانات AtScript قرار خواهد داشت. بنابراین می‌توانید منتظر قابلیتی شبیه به Attributeهای سی‌شارپ در typescript 1.5 باشید.
همانطور که می‌دانید AngularJS مهم‌ترین فریم‌ورک حال حاضر است که برای توسعه‌ی نرم‌افزارهای SPA وجود دارد. اعلام توسعه‌ی Angular 2 به‌وسیله‌ی Typescript مطمئناً خبر خوبی برای برنامه‌نویسان typescript خواهد بود، چون این اتفاق باعث بهبود سریع‌تر این زبان می‌شود.

Definitely Typed

اگرچه نمی‌توان این پروژه را در سطح دیگر پروژه‌هایی که در این مقاله معرفی می‌شود قرار داد، ولی اهمیت آن من را مجبور کرد که در این مقاله در موردش صحبت کنم. پروژه‌ی Definitely Typed در حقیقت استفاده از کتابخانه‌های دیگر JavaScript را در typescript ممکن می‌سازد. این پروژه برای پروژه‌های دیگری مانند Underscore، HighCharts، AngularJS، JQuery و هر چیزی که فکرش را بکنید Type Definition تعریف کرده. اگر هم کتابخانه‌ای که شما می‌خواستید در این پروژه نبود، دلیلش این است که اضافه کردن آن را به شما واگذار کرده‌اند! Type Definitionها در Typescript یکی از قابلیت‌های این زبان هستند برای اینکه بتوان با کتابخانه‌های JavaScript به‌صورت Type safe کار کرد.

shumway

حتماً از شنیدن اینکه این پروژه قرار است چه کاری انجام دهد شوکه خواهید شد! shumway که توسط موزیلا توسعه می‌یابد قرار است همان flash player باشد! البته این پروژه هنوز در مراحل اولیه‌ی توسعه است ولی اگر بخواهید می‌توانید دموی این پروژه را اینجا ببینید.

Fayde

پروژه‌ی Fayde هم Silverlight را هدف گرفته است. البته مانند shumway موسسه‌ی معروفی از آن حمایت نمی‌کند.

Doppio

پروژه‌ی Doppio در حقیقت یک Java Virtual Machine است که روی Browser هم می‌تواند اجرا شود. از جمله کارهای جالبی که با این پروژه می‌توان کرد، کامپایل کردن کد جاوا، Disassemble کردن یک فایل class، اجرای یک فایل JAR و حتی ارتباط با JavaScript هستند.

TypeFramework این پروژه برای افرادی خوب است که هم به NodeJS علاقمند هستند و هم به ASP.NET MVC. پروژه‌ی TypeFramework در حقیقت پیاده‌سازی مدل ASP.NET MVC در NodeJS است. Controllerها، Actionها، ActionResultها و حتی ActionFilterها با همان تعریف موجود در ASP.NET MVC در این فریم‌ورک وجود دارند.

MAYHEM این پروژه یک فریم‌ورک کاملی برای طراحی و توسعه‌ی نرم‌افزارهای Enterprise است. در شرح این پروژه آمده است که بر خلاف اینکه همه‌ی فریم‌ورک‌ها روی حجم فایل، سرعت و... تمرکز دارند این پروژه بر درستی معماری تأکید دارد. احتمالاً استفاده از این فریم‌ورک برای پروژه‌های طولانی مدت و بزرگ مناسب است. اگرچه از طرف دیگر احتمالاً یاد گرفتن این فریم‌ورک هم کار سختی خواهد بود.

حرف آخر

حرف آخر اینکه به نظر می‌رسد Typescript زبانی است که ارزش وقت گذاشتن دارد و اگر خواستید Typescript را یاد بگیرید نگاه کردن به کدهای این پروژه‌ها حتماً کلاس درس پرباری خواهد بود. چه کسی می‌داند، شاید شما بخواهید در توسعه‌ی یکی از این پروژه‌ها مشارکت کنید!

نکته‌ی بعد از آخر هم اینکه اگر خواستید به‌طور جدی با این زبان برنامه‌نویسی کنید نگاهی به tslint و typedoc هم بیاندازید.