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 $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.