

با توجه به [پست ها منتشر شده قبلی](#) درباره AngularJS به احتمال قوی شما نیز به این نتیجه رسیده اید که این فریم ورک برای انواع پروژه ها به ویژه پروژه هایی با مقیاس بزرگ بسیار مناسب است. منظور از ساختار پروژه Angular این است که به چه سبکی فایل های پروژه را سازمان دهی کنیم طوری که در هنگام توسعه و تغییرات با مشکل مواجه نشویم. عموماً کدهای مربوط به بخش frontend پروژه دارای ساختار قوی نمی باشند در نتیجه developerها بیشتر سلیقه ای کدهای مربوطه را می نویسند که با گذر زمان این مورد باعث بروز مشکل در امر توسعه نرم افزار می شود (نمونه بارز آن کدهای نوشته شده JQuery در صفحات است). AngularJS نیز همانند سایر کتابخانه ها و فریم ورک های جاوااسکریپتی دیگر از این امر مستثنی نیست و فایل های آن باید طبق روشی مناسب پیاده سازی و مدیریت شوند. انتخاب ساختار و روش سازمان دهی فایل ها وابستگی مستقیم به مقیاس پروژه دارد. ساختار پروژه های کوچک می تواند کاملاً متفاوت با ساختار پروژه های بزرگ باشد. در این پست به بررسی چند روش در این زمینه خواهیم پرداخت.

پروژه های کوچک عموماً دارای ساختاری مشابه تصویر ذیل می باشند:

- css/
- img/
- js/
 - app.js
 - controllers.js
 - directives.js
 - filters.js
 - services.js
- lib/
- partials/

این مورد، روش پیشنهادی در [Angular Seed](#) است و بدین صورت است که تعاریف ماژول ها در فایل app.js انجام می گیرد. تعاریف و پیاده سازی تمام کنترلر ها در فایل controller.js است. و همچنین دایرکتیوها و فیلترها و سرویس ها هر کدام در فایل ها جداگانه تعریف و پیاده سازی می شوند. این روش راه حلی سریع برای پروژه های کوچک با تعداد developer کم است. برای مثال زمانی که یک developer در حال ویرایش فایل controller.js است، از آن جا که فایل مورد نظر checkout خواهد شد در نتیجه سایر developerها امکان تغییر در فایل مورد نظر را نخواهند داشت. سورس فایل ها به مرور زیاد خواهد شد و در نتیجه debug آن سخت می شود.

روش دوم

در این حالت تعاریف کنترلر ها، مدل ها و سرویس ها هرکدام در یک دایرکتوری مجزا قرار خواهد گرفت. برای هر view یک کنترلر و بنا بر نیاز مدل تعریف می کنیم. ساختار آن به صورت زیر می شود:

- controllers/
 - LoginController.js
 - RegistrationController.js
 - ProductDetailController.js
 - SearchResultsController.js
- directives.js
- filters.js
- models/
 - CartModel.js
 - ProductModel.js
 - SearchResultsModel.js
 - UserModel.js
- services/
 - CartService.js
 - UserService.js
 - ProductService.js

دایرکتیوها و فیلترها عموماً در یک فایل قرار داده خواهند شد تا بنابر نیاز در جای مناسب رفرنس داده شوند. این روش ساختار مناسب تری نسبت به روش قبلی دارد اما دارای معایبی هم چون موارد زیر است:

«وابستگی بین فایل ها مشخص نیست در نتیجه بدون استفاده از کتابخانه هایی نظیر requireJs با مشکل مواجه خواهید شد.

«refactoring کدها تا حدودی سخت است.

روش سوم

این ساختار مناسب برای پیاده سازی پروژه ها به صورت ماژولار است و برای پروژه های بزرگ نیز بسیار مناسب است. در این حالت شما فایل های مربوط به هر ماژول را در دایرکتوری خاص آن قرار خواهید داد. به صورت زیر:

- cart/
 - CartModel.js
 - CartService.js
- common/
 - directives.js
 - filters.js
- product/
 - search/
 - SearchResultsController.js
 - SearchResultsModel.js
 - ProductDetailController.js
 - ProductModel.js
 - ProductService.js
- user/
 - LoginController.js
 - RegistrationController.js
 - UserModel.js
 - UserService.js

همان طور که ملاحظه می کنید سرویس ها، کنترلرها و حتی مدل های مربوط به هر بخش در یک مسیر جداگانه قرار می گیرند. علاوه بر آن فایل هایی که قابلیت اشتراکی دارند در مسیری به نام common وجود دارند تا بتوان در جای مناسب برای استفاده از آنها رفرنس داده شود. حتی اگر در پروژه خود فقط یک ماژول دارید باز سعی کنید از این روش برای مدیریت فایل های خود استفاده نمایید. اگر با [ngStart](#) آشنایی داشته باشید به احتمال زیاد با این روش بیگانه نیستید.

بررسی چند نکته درباره کدهای مشترک

در اکثر پروژه های بزرگ، فایل ها و کد هایی وجود خواهد داشت که حالت اشتراکی بین ماژول ها دارند. در این روش این فایل ها در مسیری به نام common یا shared ذخیره می شوند. علاوه بر آن در Angular تکنیک هایی برای به اشتراک گذاشتن این اطلاعات وجود دارد.

«اگر ماژول ها وابستگی شدیدی به فایل ها و سورس های مشترک دارند باید اطمینان حاصل نمایید که این ماژولها فقط به اطلاعات مورد نیاز دسترسی دارند. این اصل interface segregation principle اصول SOLID است.»

«توابعی که کاربرد زیادی دارند و اصطلاحا به عنوان Utility شناخته می شوند باید به `$rootScope` اضافه شوند تا `scope` های وابسته نیز به آنها دسترسی داشته باشند. این مورد به ویژه باعث کاهش تکرار وابستگی های مربوط به هر کنترلر می شود.»

«برای جداسازی وابستگی های بین دو component بهتر از eventها استفاده نمایید. AngularJS این امکان را با استفاده از سرویس های `$on` و `$emit` و `$broadcast` به راحتی میسر کرده است.»

نظرات خوانندگان

نویسنده: ایاک

تاریخ: ۱۳۹۲/۱۱/۲۷ ۹:۵۹

با سلام و تشکر از مقاله خوب شما. برای بارگذاری اسکریپت ها در روش سوم ، از آنجا که ممکن است تعداد دایرکتوری ها زیاد باشد ، شما چه روشی را پیشنهاد می کنید؟

نویسنده: مسعود پاکدل

تاریخ: ۱۳۹۲/۱۱/۲۷ ۱۰:۵۴

زمانی که تعداد فایل ها و دایرکتوری ها در پروژه زیاد می شود (البته این جزء جدانشدنی پروژه های مقیاس بزرگ است) برای جلوگیری از لود یک باره کنترلرها و دایرکتیوها، بهتر از lazy loading برای لود فایل های مورد نیاز استفاده شود. متأسفانه Angular به صورت رسمی از lazy loading پشتیبانی نمی کند اما با کمی تغییر در ساختار و استفاده از کتابخانه های جانبی مثل requireJs یا ScriptJS می توان به این مهم دست یافت. (با عنوان این مطلب که قصد داشتم این مورد را طی یک پست جداگانه بررسی کنم) برای مثال: ابتدا ماژول app خود را به این شکل تنظیم کنید:

```
(function()
{
    var app = angular.module('app', []);

    app.config(function($routeProvider, $controllerProvider, $compileProvider, $filterProvider,
    $provide)
    {
        app.controllerProvider = $controllerProvider;
        app.compileProvider     = $compileProvider;
        app.routeProvider       = $routeProvider;
        app.filterProvider       = $filterProvider;
        app.provider             = $provide;
    });
})();
```

با استفاده از سرویس \$controllerProvider می توان چرخه ساخت کنترلر را به دست گرفت. هم چنین سرویس \$compileProvider برای نمونه سازی دایرکتیوها و \$filterProvider برای فیلترها استفاده می شوند. ساخت کنترلرها و دایرکتیوها نیز به صورت زیر انجام خواهد شد:

```
angular.module('app').controllerProvider.resgister('SomeLazyController', function($scope)
{
    $scope.key = '...';
});
```

و هم چنین یک نمونه از ساخت directive

```
$compileProvider.directive('SomeLazyDirective', function()
{
    return {
        restrict: 'A',
        templateUrl: 'templates/some-lazy-directive.html'
    }
});
```

فقط کافیست در هنگام پیاده سازی routing (که در این [مقاله](#) شرح داده شده است) نوع بارگذاری کنترلرها و دایرکتیو و ... را به صورت lazy انجام دهید :

```
$routeProvider.when('/about', {templateUrl:'views/about.html', resolve:{deps:function($q, $rootScope)
{
    var deferred = $q.defer();
    var dependencies =
    [
        'controllers/AboutViewController.js',
        'directives/some-directive.js'
    ];

    /* نکته اول
    $script(dependencies, function()
    {
        // نکته دوم *
        $rootScope.$apply(function()
        {
            deferred.resolve();
        });
    });

    return deferred.promise;
}}})
```

*نکته اول: تمام وابستگی‌ها توسط scriptJs مدیریت می‌شوند.

*نکته دوم: تمام وابستگی‌ها مربوط به این scope بعد از فراخوانی تابع deferred.resolved بارگذاری خواهند شد.
نقطه شروع برنامه نیز به صورت زیر است:

```
$script(['appModule.js'], function()
{
    angular.bootstrap(document, ['app'])
});
```

[angular.bootstrap](#)

نویسنده: حمید صابری
تاریخ: ۹۵۳ ۱۳۹۲/۱۱/۲۸

ضمن تشکر فراوان از جناب آقای پاکدل عزیز، در این [مقاله](#) به خوبی درباره lazy loading در angularjs بحث شده. نکته مهم اینکه [حتما پروژه‌ای قابل اجرایی که در انتهای مقاله لینک شده](#) را ملاحظه کنید. نکاتی در این پروژه هست از جمله اینکه برای دسترسی به providerها برای lazy loading آنها به این ترتیب به app افزوده شده اند:

```
app.config([
    '$stateProvider',
    '$urlRouterProvider',
    '$locationProvider',
    '$controllerProvider',
    '$compileProvider',
    '$filterProvider',
    '$provide',

    function ($stateProvider, $urlRouterProvider, $locationProvider, $controllerProvider,
    $compileProvider, $filterProvider, $provide) {
        // برای رجیستر کردن غیر همروند اجزای انگیولاری در آینده
        app.lazy =
        {
            controller: $controllerProvider.register,
            directive: $compileProvider.directive,
            filter: $filterProvider.register,
            factory: $provide.factory,
            service: $provide.service
        };
    }
]);
```

(البته این کد از پروژه خودمان است و بعضی وابستگی‌های دیگر هم تزریق شده‌اند).

استفاده از app.lazy باعث سهولت بیشتر در استفاده و خواناتر شدن کد می‌شود. در ادامه به این ترتیب می‌توانید از app.lazy استفاده کنید:

```
angular.module('app').lazy.controller('myController',
    ['$scope', function($scope){
    }]);
```

به این ترتیب کد نوشته شده به دلیل نام گذاری ارجاع \$ controllerProvider با controller به حالت عادی شبیه است، و از طرفی lazy پیش از آن به فهم ماجرا کمک خواهد کرد.

این نقطه شروع یکی از پروژه‌های ماست که به عنوان نمونه بد نیست ملاحظه کنید:

```
<script type="text/javascript">
// --- Scriptjs ---
!function(a, b, c) { function t(a, c) { var e = b.createElement("script"), f = j; e.onload = e.onerror = e[o] = function () { e[m] && !/^c|load/.test(e[m]) || f || (e.onload = e[o] = null, f = 1, c()) }, e.async = 1, e.src = a, d.insertBefore(e, d.firstChild) } function q(a, b) { p(a, function (a) { return !b(a) }) } var d = b.getElementsByTagName("head")[0], e = {}, f = {}, g = {}, h = {}, i = "string", j = !1, k = "push", l = "DOMContentLoaded", m = "readyState", n = "addEventListener", o = "onreadystatechange", p = function (a, b) { for (var c = 0, d = a.length; c < d; ++c) if (!b(a[c])) return j; return 1 }; !b[m] && b[n] && (b[n](l, function r() { b.removeEventListener(l, r, j), b[m] = "complete" }, j), b[m] = "loading"); var s = function (a, b, d) { function o() { if (!--m) { e[l] = 1, j && j(); for (var a in g) p(a.split("|"), n) && !q(g[a], n) && (g[a] = []) } } function n(a) { return a.call ? a() : e[a] } a = a[k] ? a : [a]; var i = b && b.call, j = i ? b : d, l = i ? a.join("") : b, m = a.length; c(function () { q(a, function (a) { h[a] ? (l && (f[l] = 1), o()) : (h[a] = 1, l && (f[l] = 1), t(s.path ? s.path + a + ".js" : a, o)) } }, 0); return s }; s.get = t, s.ready = function (a, b, c) { a = a[k] ? a : [a]; var d = []; !q(a, function (a) { e[a] || d[k](a) }) && p(a, function (a) { return e[a] }) ? b() : !function (a) { g[a] = g[a] || [], g[a][k](b), c && c(d) }(a.join("|")); return s }; var u = a.$script; s.noConflict = function () { a.$script = u; return this }, typeof module != "undefined" && module.exports ? module.exports = s : a.$script = s }(this, document, setTimeout)

$script(['/Scripts/Lib/jquery/jquery-1.10.2.min.js'], function () {
    $script(['/Scripts/Lib/angular/angular.js'], function () {
        $script(['/Scripts/Lib/angular/angular-ui-router.min.js',
            '/Scripts/Lib/angular/angular-resource.min.js',
            '/Scripts/Lib/angular/angular-cache.min.js',
            '/Scripts/Lib/angular/angular-sanitize.min.js',
            '/Scripts/Lib/angular/angular-animate.min.js',
            '/Scripts/Lib/angular/angular-cookie.min.js',
            '/APP/Common/directives.js'
        ], function () {
            $script('/app/app.js', function () {
                angular.bootstrap(document, ['app']);
            });
        });
    });
});
</script>
```

این تگ script در صفحه شروع پروژه آمده است.

کد minify شده scriptjs در ابتدا قرار دارد، پس از آن فایل‌های js مورد نیاز با رعایت وابستگی‌های احتمالی به ترتیب بارگذاری شده‌اند.

این قسمت resolve یکی از بخش‌های مسیریابی است:

```
resolve: {
    fileDeps: ['$q', '$rootScope', function ($q, $rootScope) {
        var deferred = $q.defer();
        var deps = ['/app/HotStories/dataContextService.js',
            '/app/HotStories/hotStController.js'];
        $script(deps, function () {
            $rootScope.$apply(function () {
                deferred.resolve();
            });
        });
        return deferred.promise;
    }]
}
```

```
}
```

این نحوه تعریف سرویسی که فایل آن در وابستگی‌ها آمده و قرار است lazy load شود:

```
angular.module('app').lazy.service('dataContextService',
    ['$rootScope', '$resource', '$angularCacheFactory', '$q', function($rootScope, $resource,
    $cacheFactory, $q){
    ...
    }]);
```

و این هم نحوه تعریف کنترلری که فایل آن در وابستگی‌ها آمده و قرار است lazy load شود:

```
angular.module('app').lazy.controller('hotStController',
    ['$scope', 'ipCookie', 'dataContextService', function($scope, ipCookie, dataContextService){
    ...
    }]);
```

نویسنده:

علی فخرایی

تاریخ:

۱۳:۱۲ ۱۳۹۲/۱۲/۲۷

ممنون از مطلب مفیدتون. اگر ما یک area مثلا به نام administrator برای مدیریت داشته باشیم، آیا باید فایل‌ها را در مسیر ریشه مثلا در پوشه script قرار دهیم؟ یا باید در همان area؟ چون اگر در ریشه قرار دهیم جالب به نظر نمی‌رسد. ممکنه راهنمایی کنید؟

نویسنده:

مسعود پاکدل

تاریخ:

۱۷:۳۱ ۱۳۹۲/۱۲/۲۷

خیر. می‌توانید فایل‌های مورد نیاز هر ماژول و area را در مسیرهای جداگانه مربوط به area قرار دهید. پوشه Scripts صرفا برای قرار گیری فایل‌های مورد نیاز کتابخانه هاست (نظیر Jquery و angular و q و ...).

نویسنده:

علی فخرایی

تاریخ:

۲۰:۱ ۱۳۹۲/۱۲/۲۷

تشکر. شما در مورد مسیر یابی هم قطعه کدی قرار دادید که میشود وابستگی‌ها و ... را تزریق کرد. منتها اگر ما بیش از 100 مسیر داشته باشیم باید چه کنیم؟ یعنی به ازای هر مسیر باید این قطعه کد تکرار شود :

```
$routeProvider.when('/about', {templateUrl:'views/about.html', resolve:{deps:function($q, $rootScope)
{
    var deferred = $q.defer();
    var dependencies =
    [
        'controllers/AboutViewController.js',
        'directives/some-directive.js'
    ];
    /** نکته اول
    $script(dependencies, function()
    {
        // نکته دوم *
        $rootScope.$apply(function()
        {
            deferred.resolve();
        });
    });
    return deferred.promise;
}}})
```


راه حل پویایی وجود دارد؟

مثلا شما در ساختار سوم بیان کردید که فایل های مربوط به هر قسمت در کنار هم باشند. اعم از کنترلر و دایرکتیوها و فیلترها و ...

آیا میشود برای هر قسمت مثل cart, user, product, یک ماژول app جدا نوشت و در آن طبق مثال شما مسیریابی را تولید کرد؟ یعنی چندین ماژول انگولار app.js برای یک پروژه نوشت؟ استاندارد است؟ بدین صورت دیگر نگران تعداد مسیرهای زیاد نیستیم و مشخص میشود که مسیریابی هر قسمت در کنار آن وجود دارد. امکان پذیر است؟ اگر نیست شما چه راهی برای این کار دارید. ممنون

نویسنده: علی فخرایی

تاریخ: ۱۴:۲۳ ۱۳۹۳/۰۱/۰۲

میشه یک مثال ساده هم در مورد کامنت های دوم و سوم قرار بدید؟

من کلیه مراحل رو پیش رفتم و دو روز کامل درگیرش هستم، اما به نتیجه ای نمیرسم. خطاهای زیر رو در کنسول کروم دریافت میکنم.

```
Uncaught Error: [$injector:modulerr] Failed to instantiate module app due to:
Error: [$injector:nomod] Module 'app' is not available! You either misspelled the module name or forgot
to load it. If registering a module ensure that you specify the de...<omitted>...0) angular.js:78
Uncaught ReferenceError: app is not defined selectAllCheckbox.js:3
Uncaught Error: [$injector:modulerr] Failed to instantiate module app due to:
Error: [$injector:unpr] Unknown provider: $routeProvider
http://errors.angularjs.org/1.2.14/$injector/unpr?p0=%24routeProvider
at http://localhost:8417/Scripts/Angula...<omitted>...0)
```

نویسنده: حمید رضا منصوری

تاریخ: ۱۶:۲۷ ۱۳۹۳/۰۱/۰۳

با تشکر بابت راهنماییتون

لطفا میتونید یه sample ساده از این مطلبتون بزارید. البته اون مثال لینکی که گذاشته بودید رو دیدم ولی نتونستم اجراش کنم و نمونه داخل خودش هم کار نمی کرد.

نویسنده: مسعود پاکدل

تاریخ: ۱۰:۱۶ ۱۳۹۳/۰۱/۰۵

بخش اول سوال: بهتر است که کد مربوط به لود وابستگی ها در یک تابع مجزا نوشته شود و فقط در زمان نیاز این تابع را با پاس دادن وابستگی فراخوانی نمایید (با فرض اینکه نام این فایل dependencyResolver است):

```
(function()
{
  return function(dependencies)
  {
    var definition =
    {
      resolver: ['$q','$rootScope', function($q, $rootScope)
      {
        var deferred = $q.defer();
        $script(dependencies, function()
        {
          $rootScope.$apply(function()
          {
            deferred.resolve();
          });
        });
        return deferred.promise;
      }
    ]
  }
  return definition;
})
```

```
});
```

و برای لود وابستگی نیز تابع `dependencyResolver` را به این صورت فراخوانی نمایید:

```
angular.forEach(config.routes, function(route, path)
{
    $routeProvider.when(path, {templateUrl:route.templateUrl,
    resolve:dependencyResolver(route.dependencies)}});
});
```

در مورد سوال دوم نیز باید عنوان کنم که شما می‌توانید مسیریابی هر ماژول را به صورت جداگانه در تعاریف همان ماژول‌ها انجام دهید که البته روشی مرسوم و معمول است. فقط در هنگام عملیات bootstrapping ماژول اصلی برنامه، سایر ماژول‌ها به عنوان وابستگی آن تعیین می‌شوند. به صورت زیر(عنوان ماژول‌ها را یکتا انتخاب نمایید):

```
var app = angular.module('app', ['anotherModule1' , 'anotherModule2' , 'anotherModule3']);
```

نویسنده: حمید صابری
تاریخ: ۱۷:۰ ۱۳۹۳/۰۱/۰۸

دوست عزیز [اینجا](#) میتونید توضیحات بیشتر درباره lazy loading و [یک پیاده سازی ساده](#) از اونو مطالعه کنید.

نویسنده: حمید صابری
تاریخ: ۱۷:۱ ۱۳۹۳/۰۱/۰۸

دوست عزیز [اینجا](#) میتونید توضیحات بیشتر درباره lazy loading و [یک پیاده سازی ساده](#) از اونو مطالعه کنید.

نویسنده: ایاک
تاریخ: ۱۲:۴۰ ۱۳۹۳/۰۱/۲۷

با تشکر.

برای این قسمت در صورت امکان توضیح بیشتری میدید؟

```
angular.forEach(config.routes, function(route, path)
{
    $routeProvider.when(path, {templateUrl:route.templateUrl,
    resolve:dependencyResolver(route.dependencies)}});
});
```

این کد باید در کجا نوشته شود و مقدار `config.routes` از کجا دریافت می‌شود؟