عنوان: **چطور باید یک پروژه سورس باز را خوب مدیریت کرد؟**
نویسنده: وحید نصیری
تاریخ: ۱۳۹۲/۰۶/۲۸ ۱۲:۰
آدرس: www.dotnettips.info
برچسب‌ها: Tips, Open Source, Project Management

اگر مایل هستید که پروژه خود را به صورت سورس باز ارائه دهید، نیاز است یک سری شرایط را رعایت کنید تا کاربران این پروژه بتوانند به سادگی از آن استفاده نمایند.

- فایل ReadMe را فراموش نکنید

حتی اگر پروژه شما از یک سایت اختصاصی استفاده می‌کند، اولین محلی که عموم کاربران برای دریافت اطلاعات کار با پروژه، به آن مراجعه می‌کنند، فایل ReadMe برنامه است. این فایل می‌تواند حاوی مشخصات ذیل باشد:

الف) وابستگی‌های پروژه را مشخص کنید

واقعیت این است که برخلاف شمای برنامه نویس، عموم استفاده کنندگان، آشنایی چندانی با جزئیات محیط و شرایط تهیه برنامه شما ندارند. به این ترتیب بسیاری از مسایلی که برای شما بدیهی هستند، برای عموم اینگونه نخواهند بود. بنابراین مساله‌ای که به سرعت می‌تواند سبب خشم کاربران و صرفنظر از کار شما گردد، مشخص نبودن نحوه نصب و وابستگی‌های لازم برای اجرای برنامه است.

ب) وضعیت بلوغ پروژه خود را مشخص کنید

آیا از این برنامه، مدتی است که در محیط کاری استفاده می‌کنید؟ آیا به نظر شما هنوز ناتمام است؟ آیا API کتابخانه شما در نگارش بعدی کاملا دگرگون خواهد شد؟ تمام این مسایل و سؤالات را به نحو واضحی توضیح دهید و مشخص کنید. همین توضیحات کوتاه می‌توانند ساعت‌های بسیاری از زندگی دیگران را صرفه جویی کند.

ج) اگر پروژه شما یک کتابخانه است، نوع زبان و Runtimeهای پشتیبانی شده را مشخص کنید

برای مثال اگر یک کتابخانه دات نتی را ارائه می‌دهید، مشخص کنید که از کدام نگارش دات نت به بعد را پشتیبانی می‌کنید.

د) مجوز استفاده از پروژه را مشخص کنید

مطلب مقایسه مجوزهای سورس باز را یکبار مطالعه نمائید و سپس مجوز صحیحی را برای کار خود انتخاب کنید. همچنین آن‌را به نحو واضحی در مستندات پروژه خود قید نمائید.
به علاوه به‌خاطر داشته باشید که امکان ارائه مجوزهای دوگانه مانند AGPL نیز وجود دارند. در این حالت کاربر یا باید سورس محصول خودش را ارائه دهد، یا مجوز کتابخانه شما را خریداری کند. مانند RavenDB که از این نوع مجوز استفاده می‌کند.

- یک پروژه نیاز به مستندات دارد

مستند سازی کار، سخت و زمانبر است؛ اما بهترین لطفی است که می‌توانید به کاربران خود نمائید. مستندات نه تنها زمان جستجوی بسیاری را صرفه جویی خواهند کرد، همچنین حس اطمینان خاطر را به کاربر القاء می‌کنند. از این جهت که احساس می‌کنند شما برای کارتان ارزش قائل بوده‌اید و احتمال اینکه این برنامه در آینده نزدیک به یک abandonware تبدیل شود، کم است (منظور یک برنامه فراموش شده و خاتمه یافته).

- به روز رسانی را ساده کنید

بالاخره زمانی نیاز خواهد بود تا نگارش جدیدی از کار خود را ارائه دهید. در این حالت نیاز است یک سری از شرایط را مدنظر داشته باشید:

الف) سازگاری قبلی را مدنظر داشته باشید

یکی از بدترین حالات به روز رسانی یک کتابخانه زمانی است که کاربر آن با ده‌ها خطای کامپایل حاصل از به روز رسانی مواجه شود. اگر نیاز است قسمتی از کد خود را حذف کنید یا تغییر دهید، استفاده از ویژگی Obsolete را فراموش نکنید و اینکار باید مرحله به مرحله انجام شود. در یک نگارش، ویژگی Obsolete را معرفی کنید. در دو نگارش بعد، API را تغییر دهید.

ب) حتما یک Change log را تکمیل کنید

پس از ارائه یک نگارش جدید، حداقل در چند سطر مشخص کنید که چه مواردی تغییر کرده‌اند، چه مواردی اضافه شده‌اند و چه مواردی را حذف کرده‌اید.
همچنین اگر مواردی تغییر کرده‌اند، نحوه ارتقاء کدهای قدیمی را به نگارش جدید، شرح دهید. اگر مورد جدیدی اضافه شده‌است، لینکی را به مثالی درباره‌ی آن ارائه دهید.

- نگارش‌های جدید را اعلام کنید
برای مثال در طی ارائه یک مطلب جدید در وبلاگ خود، ارائه نگارش جدیدی از کتابخانه یا برنامه خود را به عموم اعلام کنید. در این حالت، حتما لینکی را به change log، ارائه داده و مشخص کنید که وضعیت سازگاری آن با قبل چگونه است.

- محلی را برای دریافت بازخوردهای پروژه خود مشخص کنید
نیاز است بتوانید پروژه خود را پشتیبانی کنید یا به سؤالات مربوطه پاسخ دهید. اگر سورس کنترل یا برنامه مدیریت پروژه شما، امکان پرسش و پاسخ را دارد، که بسیار خوب. اگر خیر، می‌توانید مثلا یک گروه گوگل جدید و امثال آن‌را برای دریافت بازخوردهای پروژه ایجاد کنید.
همچنین نیاز است لینک به این محل را در فایل ReadME پروژه به صراحت مشخص کنید.

- گذر از پروژه
بالاخره روزی فراخواهد رسید که دیگر علاقه‌ای به نگهداری پروژه نداشته باشید. این مساله را در مکان جمع آوری بازخوردهای خود اعلام کنید یا شخص دیگری را به نگهداری پروژه دعوت نمائید. اگر این کار را انجام ندهید، سبب خواهید شد forkهای متعددی از این پروژه بی‌جهت ایجاد شده و در نهایت مشخص نباشد که کدامیک بهتر است و کدامیک مشکلات کمتری دارند.

عنوان: **ساختار پروژه های Angular**
نویسنده: مسعود پاکدل
تاریخ: ۱۳۹۲/۱۱/۲۷ ۹:۵
آدرس: www.dotnettips.info
گروه‌ها: AngularJS, Project Management

با توجه به پست‌ها منتشر شده قبلی درباره AngularJs به احتمال قوی شما نیز به این نتیجه رسیده اید که این فریم ورک برای انواع پروژه‌ها به ویژه پروژه هایی با مقیاس بزرگ بسیار مناسب است. منظور از ساختار پروژه Angular این است که به چه سبکی فایل‌های پروژه را سازمان دهی کنیم طوری که در هنگام توسعه و تغییرات با مشکل مواجه نشویم. عموما کدهای مربوط به بخش frontend پروژه دارای ساختار قوی نمی‌باشند در نتیجه developerها بیشتر سلیقه ای کدهای مربوطه را می‌نویسند که با گذر زمان این مورد باعث بروز مشکل در امر توسعه نرم افزار می‌شود (نمونه بارز آن کدهای نوشته شده Jquery در صفحات است). AngularJs نیز همانند سایر کتابخانه‌ها و فریم ورک‌های جاوااسکریپتی دیگر از این امر مستثنی نیست و فایل‌های آن باید طبق روشی مناسب پیاده سازی و مدیریت شوند. انتخاب ساختار و روش سازمان دهی فایل‌ها وابستگی مستقیم به مقیاس پروژه دارد. ساختار پروژه‌های کوچک می‌تواند کاملا متفاوت با ساختار پروژه‌های بزرگ باشد. در این پست به بررسی چند روش در این زمینه خواهم پرداخت.
پروژه‌های کوچک عموما دارای ساختاری مشابه تصویر ذیل می‌باشند:

- 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 ماژول‌ها در فایل app.js انجام می‌گیرد. تعاریف و پیاده سازی تمام کنترلرها در فایل controller.js است. و همچنین دایرکتیوها و فیلترها و سرویس‌ها هر کدام در فایل‌ها جداگانه تعریف و پیاده سازی می‌شوند. این روش راه حلی سریع برای پروژه‌های کوچک با تعداد developerهای 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 فایل‌ها به مرور زیاد خواهد شد و در نتیجه debug آن سخت می‌شود.

روش دوم

در این حالت تعاریف کنترلر ها، مدل‌ها و سرویس‌ها هرکدام در یک دایرکتوری مجزا قرار خواهد گرفت. برای هر view یک کنترلر و بنا بر نیاز مدل تعریف می‌کنیم. ساختار آن به صورت زیر می‌شود:

دایرکتیوها و فیلترها عموما در یک فایل قرار داده خواهند شد تا بنابر نیاز در جای مناسب رفرنس داده شوند. این روش ساختار مناسب‌تری نسبه به روش قبلی دارد اما دارای معایبی هم چون موارد زیر است:
»وابستگی بین فایل‌ها 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 پروژه‌ها به صورت ماژولار است و برای پروژه‌های بزرگ نیز بسیار مناسب است. در این حالت شما فایل‌های 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 می‌کنید سرویس ها، کنترلرها و حتی مدل‌های مربوط به هر بخش در یک مسیر جداگانه قرار می‌گیرند. 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 فایل‌های 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 پروژه‌های بزرگ، فایل‌ها و کد هایی وجود خواهد داشت که حالت اشتراکی بین ماژول‌ها دارند. در این روش این فایل‌ها در مسیری به نام common یا shared ذخیره می‌شوند. علاوه بر آن در Angular تکنیک هایی برای به اشتراک گذاشتن این اطلاعات وجود دارد.

»اگر ماژول‌ها وابستگی شدیدی به فایل‌ها و سورس‌های 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 می‌شوند باید به $rootScope اضافه شوند تا scopeهای وابسته نیز به آن‌ها دسترسی داشته باشند. این مورد به ویژه باعث کاهش تکرار وابستگی‌های مربوط به هر کنترلر می‌شود.

»برای جداسازی وابستگی‌های بین دو component بهتر از eventها استفاده نمایید. AngularJs این امکان را با استفاده از سرویس‌های $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 اسکریپت‌ها در روش سوم ، از آنجا که ممکن است تعداد دایرکتوری‌ها زیاد باشد ، شما چه روشی را پیشنهاد می‌کنید؟

نویسنده: مسعود پاکدل
تاریخ: ۱۳۹۲/۱۱/۲۷ ۱۰:۵۴

زمانی که تعداد فایل‌ها و دایرکتوری‌ها در پروژه زیاد می‌شود(البته این جزء جدانشدنی پروژه‌های مقیاس بزرگ است) برای جلوگیری از لود یک باره کنترلرها و دایرکتیوها، بهتر از lazy loading برای لود فایل‌های مورد نیاز استفاده شود. متاسفانه Angular به صورت رسمی از lazy loading پشتیبانی نمی‌کند اما با کمی تغییر در ساختار و استفاده از کتابخانه‌های جانبی مثل requireJs یا ScriptJs می‌توان 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.provide            = $provide;

    });
})();
```

با استفاده از سرویس $controllerProvider می‌توان چرخه ساخت کنترلر را به دست گرفت. هم چنین سرویس $compileProvider برای نمونه سازی دایرکتیوها و $filterProvider برای فیلترها استفاده می‌شوند.
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 مروبط به این scope بعد از فراخوانی تابع deffered.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: ۱۳۹۲/۱۱/۲۸ ۹:۵۳

ضمن تشکر فراوان از جناب آقای پاکدل عزیز، در این مقاله به خوبی درباره lazy loading در angularjs بحث شده. نکته مهم اینکه حتما پروژه‌ی 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|loade/.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: ۱۳۹۲/۱۲/۲۷ ۱۳:۱۲

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: ۱۳۹۲/۱۲/۲۷ ۱۷:۳۱

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 Jqeury و angular و q و ...).

نویسنده: علی فخرایی
تاریخ: ۱۳۹۲/۱۲/۲۷ ۲۰:۱

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 فایل‌های 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: ۱۳۹۳/۰۱/۰۲ ۱۴:۲۳

میشه یک مثال ساده هم در مورد کامنت‌های 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: ۱۳۹۳/۰۱/۰۳ ۱۶:۲۷

با تشکر بابت راهنمایتون
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 نمی‌کرد.

نویسنده: مسعود پاکدل
تاریخ: ۱۳۹۳/۰۱/۰۵ ۱۰:۱۶

بخش اول سوال: بهتر است که کد مربوط به لود وابستگی‌ها 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: ۱۳۹۳/۰۱/۰۸ ۱۷:۰

دوست عزیز اینجا میتونید توضیحات بیشتر درباره lazy loading و یک پیاده سازی ساده از اونو مطالعه کنید.

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

دوست عزیز اینجا میتونید توضیحات بیشتر درباره lazy loading و یک پیاده سازی ساده از اونو مطالعه کنید.

نویسنده: ایاک
تاریخ: ۱۳۹۳/۰۱/۲۷ ۱۲:۴۰

با تشکر.
برای این قسمت در صورت امکان توضیح بیشتری میدهید؟

```
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؟

عنوان: **به‌روز رسانی اسمبلی‌های خارجی**
نویسنده: میثم نوایی
تاریخ: ۱۳۹۳/۰۷/۲۷ ۱۶:۲۵
آدرس: www.dotnettips.info
گروه‌ها: Assembly, Project Management, batch file

در اکثر شرکت‌های بزرگ و متوسط نرم افزاری، بخش مشترکی از پروژه‌ها تحت عنوان فریم ورک و یا پروژه‌های مشترک (Common) از پروژه‌های جاری فاکتور گرفته میشود و ارتباط با آنها با ارجاعی (Reference) به اسمبلی آنها انجام مشود.
اما مشکل همیشگی این است که برای حفظ استقلال، مستقیما از پروژه‌های جاری به اسمبلی‌های پایه ارجاع داده نمی‌شود؛ چون ممکن است بنا بر پایسته بودن نسخه پروژه جاری، قصد نداشته باشیم همیشه آخرین ورژن اسمبلی‌های خارجی را دریافت کنیم، بلکه ارجاعی به اسمبلی‌ها در یک پوشه در خود پروژه انجام میشود و شما بعد از هر بار تغییر در فریم ورک، باید اسمبلی‌های جدید را به داخل پوشه، در تک تک پروژه‌های جاری تان کپی کنید.
برای خلاصی از این کار مدام و تکراری می‌توانید از یک Batch فایل شبیه کد زیر استفاده کنید:

```
xcopy /s /y D:\Project\Framework\Framework.Web\bin\Debug\Framework.dll D:\Project\Current\DependentDLL
xcopy /s /y D:\Project\Framework\Framework.Web\bin\Debug\Framework.Common.dll
D:\Project\Current\DependentDLL
xcopy /s /y D:\Project\Framework\Framework.Web\bin\Debug\Framework.Business.dll
D:\Project\Current\DependentDLL
xcopy /s /y D:\Project\Framework\Framework.Web\bin\Debug\Framework.Web.dll
D:\Project\Current\DependentDLL
```

کافی است این دستورات را در Note Pad کپی کنید و سپس با پسوند bat و مثلا با نام Update ذخیره کنید. این فایل را در پوشه اسمبلی‌های وابسته در پروژه‌های جاری تان کپی کنید و از این به بعد هر وقت خواستید آخرین ورژن اسمبلی‌های خارجی را دریافت کنید دوبار روی این فایل کلیک کنید. برای شخصی سازی بیشتر دستورات انتقال فایل در Batch فایل‌ها اینجا و اینجا را بخوانید.

نظرات خوانندگان

نویسنده: محسن موسوی
تاریخ: ۱۳۹۳/۰۷/۲۸ ۱۰:۲۱

البته استفاده از Nuget به صورت محلی گزینه‌ی بهتری هست. نوگت راه حل‌های مختلفی برای این کار ارئه میده. بصورت یک پوشه اشتراکی و یا ایجاد یک سرویس نوگت.

نویسنده: میثم نوایی
تاریخ: ۱۳۹۳/۰۷/۲۸ ۱۴:۱۷

بله استفاده از روش مطروحه شما استاندارد و اصولی‌تر میباشد.
راهکار من سر راست‌تر و سریع‌تر میباشد و برای افراد مبتدی کاربرد بیشتری دارد به اضافه اینکه خیلی از اوقات اسمبلی‌های خارجی در پروژه مدام دستخوش تغییر مسیر،تغییر نام و یا حتی ممکن است بکلی از پروژه کنار گذاشته شوند.

عنوان: **برنامه ریزی به روش چابک**
نویسنده: پگاه
تاریخ: ۱۳۹۳/۱۱/۰۹ ۱۱:۳۰
آدرس: www.dotnettips.info
گروه‌ها: Agile, Schedule, Project Management, time management, time schedule, Agile planning

شاید شما هم مثل من فکر می‌کنید، به اندازه‌ای که درحرفه یا زندگی شخصی خود زحمت می‌کشید، نتیجه نمی‌گیرید! چندی پیش کتابی خواندم از آقای J.D Meier (مهندس نرم افزار و مدیر پروژه در شرکت مایکروسافت) که نحوه‌ی برنامه ریزی و زمان بندی من در کار و زندگی ام را به طور شگفت آوری متحول کرد. به همین دلیل به این فکر افتادم که در قالب چند مقاله به معرفی روش ایشان که یک روش بسیار آسان برای گرفتن نتایج بهتر و سریع‌تر در کار و زندگی است، بپردازم. امیدوارم همانطور که من این روش را مفید و کاربردی یافتم، برای خوانندگان این مقاله هم تأثیرگزار باشد.

این مدل که روش چابک (Agile way) نام دارد برپایه چهار اصل اساسی به نام‌های **قانون سه تایی، دیدگاه شنبه، خروجی روزانه و بازخورد پنجشنبه** شکل گرفته است که درادامه به توضیح هریک از این اصول پرداخته شده است.

1) قانون سه تایی:
این قانون یک قانون ساده است و به شما کمک میکند تا در زمانی که خیلی پرمشغله هستید، با یک روش ساده بتوانید به کارهای خود سروسامان دهید و در زمان کوتاه به بهترین نتایج دلخواه‌تان دست یابید.

قانون سه تایی یعنی به صورت سه تایی فکر کردن. به این معنی که شما سه نتیجه‌ای را که مایل هستید در انتهای یک روز، یک هفته یا یک ماه بدست آورید را مشخص می‌کنید. با این سیستم سه در سه (3 x 3) شما نقشه آنچه را که می‌خواهید به دست آورید، به صورت خیلی ساده و سریع در مجموعه‌های سه تایی طراحی می‌کنید که به راحتی قابل بخاطر سپردن و ویرایش سریع است.

داشتن یک برنامه‌ی هفتگی، قلب مدل "روش چابک" است. الگوی برنامه‌ی هفتگی در این روش براساس دیدگاه شنبه، خروجی روزانه، بازخورد پنجشنبه و با کمک قانون سه تایی بنا شده است. جدول زیر نحوه طراحی و برنامه ریزی در این روش را نشان می‌دهد.

بازخورد پنجشنبه		خروجی روزانه						دیدگاه شنبه	نقاط جوش
اصلاح شود	با موفقیت انجام شده	٥شنبه	٤شنبه	٣شنبه	٢شنبه	١شنبه	شنبه	سه دست آورد برای هفته	زندگی
١.	١.	١.	١.	١.	١.	١.	١.	١.	کار
٢.	٢.	٢.	٢.	٢.	٢.	٢.	٢.	٢.	
٣.	٣.	٣.	٣.	٣.	٣.	٣.	٣.	٣.	مسایل شخصی
مرور کردن برنامه		انجام دادن برنامه						برنامه ریزی	

این مدل به شما این اجازه را می‌دهد که در هر روز و هرهفته یک شروع جدید داشته باشید و بدین صورت از یک طرف یک برنامه ریزی دقیق، قابل اعتماد و از طرفی دیگر قابل انعطاف برای کار و زندگی روزمره خود داشته باشید.

2) دیدگاه شنبه
در این مدل در روز شنبه (اولین روز هفته) سه نتیجه‌ای را که مایل هستید در آخر هفته به دست بیاورید را مشخص می‌کنید. به طور مثال شما می‌خواهید طراحی اسلایدهای مربوط به جلسه‌ی هفته آینده را تا آخر هفته جاری به اتمام برسانید و یا مایل هستید که پس از یک هفته تمرین، به راحتی بتوانید دو مایل را در روز بدوید.

3) خروجی روزانه

در این مرحله، به صورت روزانه سه دستاوردی را که مایل هستید در انتهای یک روز داشته باشید را مشخص می‌کنید. به طورمثال در ارتباط با طراحی اسلایدهای جلسه آینده، شما می‌خواهید در پایان روز اسلایدهای مربوط به فاز اول پروژه را تکمیل کرده باشید و یا به طور مثال در ارتباط با رکورد دویدن، شما هدف هفت‌صد متر را برای خود در روز مشخص می‌کنید. به طور طبیعی خواهید دید که سه خروجی که برای هر روز خود در نظر می‌گیرید، باید در راستای سه دستاوردی باشند که برای هفته‌ی خود مشخص کرده‌اید.

4) بازتاب پنجشنبه

در روز پنجشنبه شما این امکان را دارید که یک قدم به عقب برگردید و نگاهی به برنامه ریزی هفته و دستاوردهای حاصل از آن بیاندازید و ازخودتان بپرسید «کدام سه فعالیت هستند که باید بیشتر بر روی آن کار کنم؟» و «کدام سه فعالیت هستند که به خوبی پیش رفته اند؟». با این روش شما یاد می‌گیرید ظرفیتهای خود را شناسایی کرده و به صورتی شفاف مشخص کنید کدام فعالیتها نیاز به تمرکز، وقت و انرژی بیشتری دارند.

نکته کلیدی: در روش چابک، نکته‌ی کلیدی این است که موفقیت‌های خود را حتی اگر بسیار کوچک هم باشند، جشن بگیرید. به این دلیل که شما کار یا فعالیتی مشخص شده را خواه اینکه کوچک یا بزرگ باشد، به طور کامل انجام داده‌اید. توجه کنید که این به نوع نگاه شما بستگی دارد که بتوانید سه نتیجه‌ی مورد نظر خود در هفته و روز را بر اساس اولویت‌های خود و به صورت درست انتخاب کنید. بازتاب پنجشنبه به شما این امکان را می‌دهد که اولویت‌های خود را بازبینی و اصلاح کنید و از این بازخورد برای طراحی و برنامه ریزی هفته آینده و روزهای آتی استفاده کنید.

نقاط جوش

در جدول برنامه ریزی مدل چابک ستونی به نام نقاط جوش (hot spots) گنجانده شده است. اگر مسایل مهم زندگی خود را در نظر بگیرید، احتمالاً آنها جزء یکی از مسایل مربوط به تفکرات و یا جسم شما، مسایل احساسی ویا مالی، معاشرت با دیگران و تفریحات خواهند بود. نقاط جوش زندگی در حقیقت بخش هایی از زندگی شما هستند که شما بر روی آنها تمرکز بیشتری دارید و به طور ساده آنها می‌توانند بیانگر درد و رنج فراوان، یا موفقیت‌ها و موقعیت‌های فراوان برای شما باشند. نکته‌ی مهم این است که این نقاط جوش هرچه باشند، بخش‌های مهمی در زندگی شما، از نظر شما هستند.

سعی کنید یک لیست از نقاط جوش زندگی خود تهیه کنید. مثلا لذت بردن از زندگی، داشتن اندامی متناسب، به پایان رساندن یک مقطع تحصیلی، ارتقا یافتن در محل کار و یا داشتن یک رابطه خوب، می‌تواند مثالهایی از نقاط جوش زندگی باشند. مشخص کردن این نقاط جوش به شما کمک می‌کنند تا بتوانید بین کار، زندگی شخصی، مسایل مالی و سرگرمی، یک تعادل ایجاد کنید. به این معنی که به صورت متعادل و منصفانه، انرژی، زمان و تمرکز خود را صرف آنها کنید. به این نکته توجه کنید که وقتی به صورت متعادل بر روی هریک از نقاط جوش زندگی خود سرمایه گزاری می‌کنید، به طور ضمنی توانایی خود را برای رویارویی با اتفاقات جدید در تمام زمینه‌های یاد شده در بالا، افزایش می‌دهید. علاوه براین، اگر در یکی از زمینه‌ها مثلا زندگی شخصی خود به اندازه‌ی کافی سرمایه گزاری نکنید، تاثیرات منفی آن را در محیط کار و حرفه‌ی خود مشاهده خواهید کرد.

خلاصه‌ی مطلب

1) قانون سه تایی به عنوان یک روش ساده برای جلوگیری از بی نظمی و هرج ومرج و به سرانجام رساندن کارهای موردنظر با کیفیت بالا در مدت زمان مناسب به کار می‌رود.

2) ابتدا سه نقطه‌ی جوش را در زندگی خود انتخاب کنید. سه نتیجه‌ی کلی و نهایی را برای هریک، براساس اولویت‌ها و اهمیت آنها مشخص کنید. به طور مثال این سوال را از خود بپرسید که دوست دارید در سال آینده در هر یک از این بخشهای زندگی خود، چه دستاوردی را داشته باشید.

3) انتخاب سه دستاورد دلخواه را به صورت ماهیانه، هفتگی و روزانه تکرار کنید. توجه کنید که خروجی هر روز باید در راستای خروجی هفته و خروجی هر هفته در راستای خروجی هر ماه، باشد.

4) در پایان هر هفته کارهای انجام شده را با خروجی مورد انتظار مقایسه کنید. مشخص کنید که چه کارهایی خوب پیش رفته اند و

چه کارهایی نیاز به وقت، تمرکز و یا زمان بیشتری نیاز دارند و از اطلاعات و بازخورد به دست آمده برای برنامه ریزی هفته‌ی آینده استفاده کنید.

در مقالات آینده درباره اینکه چطور به صورت بهینه و کارآمد دیدگاه‌های هفتگی و خروجی‌های روزانه را مشخص و برنامه ریزی کنیم صحبت خواهیم کرد.

نظرات خوانندگان

نویسنده: امیر بختیاری
تاریخ: ۱۳۹۳/۱۱/۱۱ ۱۰:۳۴

با تشکر از مطلب مفیدتون
می‌خواستم اسم کتاب به زبان اصلی و اگر آدرس لینک کتاب رو دارید قرار بدید

نویسنده: پگاه
تاریخ: ۱۳۹۳/۱۱/۱۱ ۱۵:۵۶

خواهش می‌کنم. اسم کتاب " Getting Results the Agile Way: A Personal Results System for Work and Life " است که می‌شود از آمازون خریداری کرد. اما می‌توانید یک خلاصه بسیار مفید از این کتاب را که توسط خود نویسنده کتاب تهیه شده در این آدرس مطالعه کنید. موفق باشید.

عنوان: **اولویت بندی: رمز موفقیت در برنامه ریزی به روش چابک**
نویسنده: پگاه
تاریخ: ۱۳۹۳/۱۱/۱۹ ۲:۱۵
آدرس: www.dotnettips.info
گروه‌ها: Agile, Project Management, Agile Developement, time management, time schedule, Agile planning

در مقاله « برنامه ریزی به روش چابک » به قانون سه تایی اشاره کردیم. در این قانون سه خروجی یا دستاوردی را که مایل هستیم در ماه، هفته و یا یک روز داشته باشیم، به عنوان دیدگاه‌های هفته و خروجی‌های روزانه‌ی خود مشخص می‌کنیم. اما اگر تعداد کارها و دستاوردهای مورد نظرمان از سه مورد برای یک ماه، یک هفته و یا یک روز بیشتر باشد چه باید کرد؟ این مقاله درباره اینکه چطور به صورت بهینه و کارآمد دیدگاه‌های هفتگی و خروجی‌های روزانه را مشخص و برنامه ریزی کنیم می‌پردازد.

رمز موفقیت در روش برنامه ریزی چابک، اولویت بندی مؤثر کارها و خواسته‌هاست. زمانیکه شما احساس کنید دارید بر روی کار و هدفی درست، در زمانی مناسب کار می‌کنید تمرکز بیشتری برروی کارتان خواهید داشت و بنابراین نتایج بهتری از انجام آن کار خواهید گرفت.

همه‌ی ما با اولویت بندی آشنا هستیم و معمولا با اختصاص دادن شماره به هر مورد، آن مورد را اولویت بندی می‌کنیم. به طور مثال، «نوشتن مقاله برنامه ریزی به روش چابک» اولویت 2، «شبیه سازی الگوریتم همزمان سازی» اولویت 1 و «دویدن به مدت 30 دقیقه» اولویت 3، مثال‌هایی از اولویت بندی به روش سنتی است. اما آقای Meier .J.D در کتاب خودش روش مؤثرتری را که بر اساس **بایدها** و **نبایدها** بنا شده است، پیشنهاد می‌کند که در ادامه به آن اشاره می‌کنیم.

اولویت‌ها در مدل چابک

در این روش سه درجه از اولویت وجود دارند. درجه‌ی اول، کارهایی هستند که حتما **باید** انجام بگیرند. درجه دوم، کارهایی هستند که بهتراست (بایستی) انجام شوند و درجه سوم، کارهایی هستند که می‌شود انجام شوند و یا نشوند. آقای Meier این سه درجه را به ترتیب با سه واژه " Must "، " Should " و " Could " مشخص کرده است.

روش اولویت بندی در مدل چابک

برای تهیه سه دیدگاه هفته و خروجی روزانه، ابتدا لیستی از اهداف و کارهای مورد نظر خود را تهیه کنید. سپس از خود بپرسید: (1) کدامیک از اقلام این لیست را باید انجام دهید؟ (2) کدامیک را بهتر است که انجام دهید؟ (3) کدامیک را می‌توانید انجام دهید؟ پس از مشخص کردن اولویت‌ها، سه خروجی روزانه و یا سه دیدگاه هفتگی خود را از میان اقلامی که در دسته اول، یعنی **بایدها** قرار می‌گیرند، انتخاب کنید.

مزایای اولویت بندی

-1 نتیجه‌ای که از اولویت بندی کارها نصیبتان می‌شود ارزش این را دارد که روی فرآیند اولویت بندی، مدت زمانی را صرف کنید. بدون اولویت بندی شما نگران یک لیست طولانی از کارهای خود هستید؛ در حالیکه فکر می‌کنید مشغول انجام یک کار مهم هستید. در آخر روز متوجه می‌شوید که کاری که باید انجام می‌گرفته است، انجام نشده است.

-2 با تعیین اولویت‌ها برای هفته و هر روز خود، شما حداقل کارهایی را که باید برای هفته یا هر روز خود انجام دهید، مشخص کرده‌اید. زمانیکه این **بایدها** را انجام دهید، بقیه هفته و یا روز برای شما خواهد بود و می‌توانید از آن لذت ببرید!

-3 اولویت بندی به شما قابلیت انعطاف می‌دهد. اگر یک کار یا فعالیت جدید پیش بیاید مثلا اگر رییس شما کار جدیدی را به شما محول کند، شما می‌توانید با تعیین درجه اولویت آن کار و مقایسه آن با بایدهای درحال انجام (سه خروجی روزانه یا دیدگاه هفتگی) تصمیم بگیرید که فعالیت محول شده جدید را انجام دهید یا انجام آن را به زمان دیگری موکول کنید.

بدون شک اولویت بندی یک لیست طولانی از کارها و فعالیت‌ها، کار مشکل و زمان بری است. در مقاله‌ی آینده درباره‌ی اینکه چگونه لیست کارها و فعالیت‌های خود را با مهارت، انتخاب و سازماندهی کنیم، خواهیم پرداخت.

عنوان: **دسته بندی چابکانه**
نویسنده: پگاه
تاریخ: ۱۳۹۳/۱۲/۰۳ ۲۳:۲۵
آدرس: www.dotnettips.info
گروه‌ها: Agile, Schedule, Project Management, Agile Developement, time management, time schedule, Agile planning, Project Organizer

تا کنون با روش برنامه ریزی چابک ، اهمیت و نحوه اولویت بندی فعالیت‌ها در این مدل آشنا شدیم. اما بدون شک اولویت بندی یک لیست طولانی از کارها و فعالیت‌ها کار مشکل و زمان بری خواهد بود، به‌خصوص اگر فردی پر مشغله با مسئولیت‌های فراوان باشید. در مدل برنامه ریزی به روش چابک، پیشنهاد شده‌است که لیست فعالیت‌های کاری خود را دسته بندی کنید. در این روش، دسته بندی با الهام از روش کار بخش اورژانس بیمارستان‌ها، براساس درجه ضرورت کار و اضطراری بودن آن فعالیت انجام می‌شود.

در این روش چهار دسته وجود دارد که هر فعالیت می‌تواند در آن دسته قرار بگیرد: (1) **انجام** شود، (2) **موکول** شود، (3) **زمانبندی** شود، (4) **واگذار** شود.

1) دسته اول کارهایی هستند که **هم اکنون** زمان انجام آنها فرا رسیده است. به این معنی که یا اکنون بهترین زمان انجام آنهاست، یا اگر آنها را به آینده موکول کنید مجبور خواهید بود انرژی و وقت بیشتری را صرف انجام آنها کنید.

2) دسته دوم کارهایی هستند که **می‌بایستی** انجام شوند اما اکنون زمان مناسبی برای انجام آنها نیست. بنابراین شما می‌توانید آنها را در **صف انتظار** قرار دهید و در آینده تصمیم بگیرید که با آنها چه کنید.

3) دسته سوم کارهایی هستند که برای شما **اهمیت بالایی** دارند، اما اکنون زمان مناسبی برای انجام آنها نیست. آنها را **زمانبندی** کنید. به این معنی که در تقویم خود برای انجام آنها یک زمان مشخص درنظر بگیرید تا در آن زمان انجام شوند.

با تخصیص دادن زمان به انجام یک کار در آینده فکر خود را از مشغولیت درباره آن آزاد می‌کنید. به طور مثال، شما می‌دانید که باید گزارش ماهیانه خود را در پایان ماه برای رییس خود آماده و ارایه کنید. بنابراین برای نوشتن گزارش، زمانی را مانند آخرین هفته‌ی ماه، درنظر می‌گیرید. به این ترتیب هرگاه مشغول انجام کار دیگری هستید و ناگهان فکر نوشتن گزارش به شما هجوم می‌آورد، می‌دانید که برای انجام این کار زمان خاصی تعیین شده‌است و آن را در زمان مناسب انجام خواهید داد. بدین ترتیب، آرامش فکری خود را باز می‌گردانید.

4) دسته چهارم کارهایی هستند که در حقیقت می‌توانند توسط افراد دیگری انجام شوند. بنابراین آنها را واگذار کنید. توجه کنید که واگذار کردن کارها باید با مهارت انجام شود. بدین معنی که فرد موردنظر باید تخصص و انگیزه انجام آن کار را داشته باشد. به طورمثال، اگر شما سرپرست یک تیم در محیط کار خود هستید، کارها را با مهارت به اعضای گروه اختصاص دهید؛ به جای اینکه سعی کنید خودتان همه کارها را انجام دهید.

سؤالات زیر را در نظر بگیرید:

-1 چه دستاوردی را می‌خواهید داشته باشید؟

-2 آیا واقعا مهم است؟

-3 چقدر مهم است؟

-4 به انجام رساندن این کار چه تاثیر یا نتایجی خواهد داشت؟

-5 بهترین کاری که می‌توانید الان انجام دهید چیست؟

-6 آیا حتما باید توسط من انجام گیرد؟

-7 و...

این‌ها نمونه سؤالاتی هستند که می‌توانید در زمان دسته بندی کردن کارها از خود بپرسید. سؤالاتی از این قبیل شما را راهنمایی می‌کنند تا دسته بندی کارها را با مهارت بیشتر و بهتری انجام دهید.

بعد از دسته بندی کارها، حالا لیستی از کارهایی را دارید که **هم اکنون** زمان انجام آنها فرا رسیده است. آنها را توسط سیستم اولویت بندی چابک، اولویت بندی نمایید و 3 تا از بایدهای لیست را به عنوان دیدگاه هفتگی و یا خروجی روزانه انتخاب کنید.

نظرات خوانندگان

نویسنده: فرید بکران
تاریخ: ۱۳۹۳/۱۲/۱۶ ۱۷:۶

سلام.
فرق موکول و زمان بندی دقیقا چیست؟
هر دو در زمان آینده انجام خواهند شد؛ ولی برای موکول‌ها زمان بندی در نظر گرفته نمی‌شود؟ در واقع اولویت آن‌ها کمتر از زمان بندی‌ها است. درسته؟

نویسنده: پگاه
تاریخ: ۱۳۹۳/۱۲/۱۶ ۲۰:۲۰

سلام. درست است. در حقیقت کاری را زمانبندی می‌کنیم که در آینده نزدیک **باید** انجام شود. به طور مثال بخشی از یک پروژه که باید تا تاریخ خاصی به اتمام برسد را در نظر بگیرید. اگر اکنون زمان مناسبی برای انجام این پروژه نیست بهتر است برای انجام آن زمان مشخصی را در تقویم خود ثبت کنید. حال فرض کنید شما می‌خواهید راجع به ایده جدیدی با همکار خود بحث و گفتگو کنید اگر اکنون زمان مناسبی برای انجام آن نباشد بهتر است آن را به آینده موکول کنید. ناگفته پیداست این تصمیمات کاملا شخصی است و وابسته به شرایط هر فرد است.