# Angular代码规范

## Angular Team Endorsed

非常感谢领导Angular团队的Igor Minar对本指南做出的审查和贡献，并且委托我继续打理本指南。

## 目的

Angular规范[*@john\_papa*](https://twitter.com/john_papa)

如果你正在寻找一些关于语法、约定和结构化的Angular应用的一个有建设性的规范，那么你来对地方了。这里所包含的内容是基于我在团队中使用[Angular](https://angularjs.org/)的一些经验、一些演讲和[Pluralsight培训课程](http://pluralsight.com/training/Authors/Details/john-papa)。

这个规范的目的是为构建Angular应用提供指导，当然更加重要的是让大家知道我为什么要选择它们。

如果你喜欢这个规范，请在Pluralsight看看[Angular Patterns: Clean Code](http://jpapa.me/ngclean)。

[](http://jpapa.me/ngclean)

## Community Awesomeness and Credit

Angular社区是一个热衷于分享经验的令人难以置信的社区，尽管Todd Motto（他是我的一个朋友，也是Angular专家）和我合作了多种规范和惯例，但是我们也存在着一些分歧。我鼓励你去看看[Todd的指南](https://github.com/toddmotto/angularjs-styleguide)，在那里你能看到我们之间的区别。

我的许多规范都是从大量的程序会话[Ward Bell](http://twitter.com/wardbell)和我所拥有的而来的，我的好友Ward也影响了本规范的最终演变。

## 在示例App中了解这些规范

看示例代码有助于你更好地理解，你可以在modular文件夹下找到[命名为modular的示例应用程序](https://github.com/johnpapa/ng-demos)，随便克隆。

##翻译 [Angular规范翻译版本](https://github.com/johnpapa/angular-styleguide/tree/master/a1/i18n)。

##目录

1. [单一职责](https://github.com/johnpapa/angular-styleguide/blob/master/a1/i18n/zh-CN.md#%E5%8D%95%E4%B8%80%E8%81%8C%E8%B4%A3)
2. [IIFE](https://github.com/johnpapa/angular-styleguide/blob/master/a1/i18n/zh-CN.md#iife)
3. [Modules](https://github.com/johnpapa/angular-styleguide/blob/master/a1/i18n/zh-CN.md#modules)
4. [Controllers](https://github.com/johnpapa/angular-styleguide/blob/master/a1/i18n/zh-CN.md#controllers)
5. [Services](https://github.com/johnpapa/angular-styleguide/blob/master/a1/i18n/zh-CN.md#services)
6. [Factories](https://github.com/johnpapa/angular-styleguide/blob/master/a1/i18n/zh-CN.md#factories)
7. [Data Services](https://github.com/johnpapa/angular-styleguide/blob/master/a1/i18n/zh-CN.md#data-services)
8. [Directives](https://github.com/johnpapa/angular-styleguide/blob/master/a1/i18n/zh-CN.md#directives)
9. [解决Controller的Promises](https://github.com/johnpapa/angular-styleguide/blob/master/a1/i18n/zh-CN.md#%E8%A7%A3%E5%86%B3controller%E7%9A%84promises)
10. [手动依赖注入](https://github.com/johnpapa/angular-styleguide/blob/master/a1/i18n/zh-CN.md#%E6%89%8B%E5%8A%A8%E4%BE%9D%E8%B5%96%E6%B3%A8%E5%85%A5)
11. [压缩和注释](https://github.com/johnpapa/angular-styleguide/blob/master/a1/i18n/zh-CN.md#%E5%8E%8B%E7%BC%A9%E5%92%8C%E6%B3%A8%E9%87%8A)
12. [异常处理](https://github.com/johnpapa/angular-styleguide/blob/master/a1/i18n/zh-CN.md#%E5%BC%82%E5%B8%B8%E5%A4%84%E7%90%86)
13. [命名](https://github.com/johnpapa/angular-styleguide/blob/master/a1/i18n/zh-CN.md#%E5%91%BD%E5%90%8D)
14. [应用程序结构的LIFT准则](https://github.com/johnpapa/angular-styleguide/blob/master/a1/i18n/zh-CN.md#%E5%BA%94%E7%94%A8%E7%A8%8B%E5%BA%8F%E7%BB%93%E6%9E%84%E7%9A%84lift%E5%87%86%E5%88%99)
15. [应用程序结构](https://github.com/johnpapa/angular-styleguide/blob/master/a1/i18n/zh-CN.md#%E5%BA%94%E7%94%A8%E7%A8%8B%E5%BA%8F%E7%BB%93%E6%9E%84)
16. [模块化](https://github.com/johnpapa/angular-styleguide/blob/master/a1/i18n/zh-CN.md#%E6%A8%A1%E5%9D%97%E5%8C%96)
17. [启动逻辑](https://github.com/johnpapa/angular-styleguide/blob/master/a1/i18n/zh-CN.md#%E5%90%AF%E5%8A%A8%E9%80%BB%E8%BE%91)
18. [Angular $包装服务](https://github.com/johnpapa/angular-styleguide/blob/master/a1/i18n/zh-CN.md#angular-%E5%8C%85%E8%A3%85%E6%9C%8D%E5%8A%A1)
19. [测试](https://github.com/johnpapa/angular-styleguide/blob/master/a1/i18n/zh-CN.md#%E6%B5%8B%E8%AF%95)
20. [动画](https://github.com/johnpapa/angular-styleguide/blob/master/a1/i18n/zh-CN.md#%E5%8A%A8%E7%94%BB)
21. [注释](https://github.com/johnpapa/angular-styleguide/blob/master/a1/i18n/zh-CN.md#%E6%B3%A8%E9%87%8A)
22. [JSHint](https://github.com/johnpapa/angular-styleguide/blob/master/a1/i18n/zh-CN.md#js-hint)
23. [JSCS](https://github.com/johnpapa/angular-styleguide/blob/master/a1/i18n/zh-CN.md#jscs)
24. [常量](https://github.com/johnpapa/angular-styleguide/blob/master/a1/i18n/zh-CN.md#%E5%B8%B8%E9%87%8F)
25. [文件模板和片段](https://github.com/johnpapa/angular-styleguide/blob/master/a1/i18n/zh-CN.md#%E6%96%87%E4%BB%B6%E6%A8%A1%E6%9D%BF%E5%92%8C%E7%89%87%E6%AE%B5)
26. [Yeoman Generator](https://github.com/johnpapa/angular-styleguide/blob/master/a1/i18n/zh-CN.md#yeoman-generator)
27. [路由](https://github.com/johnpapa/angular-styleguide/blob/master/a1/i18n/zh-CN.md#%E8%B7%AF%E7%94%B1)
28. [任务自动化](https://github.com/johnpapa/angular-styleguide/blob/master/a1/i18n/zh-CN.md#%E4%BB%BB%E5%8A%A1%E8%87%AA%E5%8A%A8%E5%8C%96)
29. [Filters](https://github.com/johnpapa/angular-styleguide/blob/master/a1/i18n/zh-CN.md#filters)
30. [Angular文档](https://github.com/johnpapa/angular-styleguide/blob/master/a1/i18n/zh-CN.md#angularjs%E6%96%87%E6%A1%A3)
31. [贡献](https://github.com/johnpapa/angular-styleguide/blob/master/a1/i18n/zh-CN.md#%E8%B4%A1%E7%8C%AE)
32. [许可](https://github.com/johnpapa/angular-styleguide/blob/master/a1/i18n/zh-CN.md#%E8%AE%B8%E5%8F%AF)

## 单一职责

###规则一

###### [Style [Y001](https://github.com/johnpapa/angular-styleguide/blob/master/a1/i18n/zh-CN.md#style-y001)]

* 一个文件只定义一个组件。

下面的例子在同一个文件中定义了一个app的module和它的一些依赖、一个controller和一个factory。

/\* avoid \*/

angular

.module('app', ['ngRoute'])

.controller('SomeController', SomeController)

.factory('someFactory', someFactory);

function SomeController() { }

function someFactory() { }

推荐以下面的方式来做，把上面相同的组件分割成单独的文件。

/\* recommended \*/

// app.module.js

angular

.module('app', ['ngRoute']);

/\* recommended \*/

// someController.js

angular

.module('app')

.controller('SomeController', SomeController);

function SomeController() { }

/\* recommended \*/

// someFactory.js

angular

.module('app')

.factory('someFactory', someFactory);

function someFactory() { }

[**返回顶部**](https://github.com/johnpapa/angular-styleguide/blob/master/a1/i18n/zh-CN.md#%E7%9B%AE%E5%BD%95)

## IIFE

### JavaScript闭包

###### [Style [Y010](https://github.com/johnpapa/angular-styleguide/blob/master/a1/i18n/zh-CN.md#style-y010)]

* 把Angular组件包装到一个立即调用函数表达式中（IIFE）。

为什么？：把变量从全局作用域中删除了，这有助于防止变量和函数声明比预期在全局作用域中有更长的生命周期，也有助于避免变量冲突。

为什么？：当你的代码为了发布而压缩了并且被合并到同一个文件中时，可能会有很多变量发生冲突，使用了IIFE（给每个文件提供了一个独立的作用域），你就不用担心这个了。

/\* avoid \*/

// logger.js

angular

.module('app')

.factory('logger', logger);

// logger function会被当作一个全局变量

function logger() { }

// storage.js

angular

.module('app')

.factory('storage', storage);

// storage function会被当作一个全局变量

function storage() { }

/\*\*

\* recommended

\*

\* 再也不存在全局变量了

\*/

// logger.js

(function() {

'use strict';

angular

.module('app')

.factory('logger', logger);

function logger() { }

})();

// storage.js

(function() {

'use strict';

angular

.module('app')

.factory('storage', storage);

function storage() { }

})();

* 注：为了简洁起见，本规范余下的示例中将会省略IIFE语法。
* 注：IIFE阻止了测试代码访问私有成员（正则表达式、helper函数等），这对于自身测试是非常友好的。然而你可以把这些私有成员暴露到可访问成员中进行测试，例如把私有成员（正则表达式、helper函数等）放到factory或是constant中。

[**返回顶部**](https://github.com/johnpapa/angular-styleguide/blob/master/a1/i18n/zh-CN.md#%E7%9B%AE%E5%BD%95)

## Modules

###避免命名冲突

###### [Style [Y020](https://github.com/johnpapa/angular-styleguide/blob/master/a1/i18n/zh-CN.md#style-y020)]

* 每一个独立子模块使用唯一的命名约定。

为什么：避免冲突，每个模块也可以方便定义子模块。

###定义(aka Setters)

###### [Style [Y021](https://github.com/johnpapa/angular-styleguide/blob/master/a1/i18n/zh-CN.md#style-y021)]

* 不使用任何一个使用了setter语法的变量来定义modules。

为什么?：在一个文件只有一个组件的条件下，完全不需要为一个模块引入一个变量。

/\* avoid \*/

var app = angular.module('app', [

'ngAnimate',

'ngRoute',

'app.shared',

'app.dashboard'

]);

你只需要用简单的setter语法来代替。

/\* recommended \*/

angular

.module('app', [

'ngAnimate',

'ngRoute',

'app.shared',

'app.dashboard'

]);

###Getters

###### [Style [Y022](https://github.com/johnpapa/angular-styleguide/blob/master/a1/i18n/zh-CN.md#style-y022)]

* 使用module的时候，避免直接用一个变量，而是使用getter的链式语法。

为什么？：这将产生更加易读的代码，并且可以避免变量冲突和泄漏。

/\* avoid \*/

var app = angular.module('app');

app.controller('SomeController', SomeController);

function SomeController() { }

/\* recommended \*/

angular

.module('app')

.controller('SomeController', SomeController);

function SomeController() { }

###Setting vs Getting

###### [Style [Y023](https://github.com/johnpapa/angular-styleguide/blob/master/a1/i18n/zh-CN.md#style-y023)]

* 只能设置一次。

为什么？：一个module只能被创建一次，创建之后才能被检索到。

- 设置module，`angular.module('app', []);`。

- 获取module，`angular.module('app');`。

###命名函数 vs 匿名函数

###### [Style [Y024](https://github.com/johnpapa/angular-styleguide/blob/master/a1/i18n/zh-CN.md#style-y024)]

* 回调函数使用命名函数，不要用匿名函数。

为什么？：易读，方便调试，减少嵌套回调函数的数量。

/\* avoid \*/

angular

.module('app')

.controller('Dashboard', function() { })

.factory('logger', function() { });

/\* recommended \*/

// dashboard.js

angular

.module('app')

.controller('Dashboard', Dashboard);

function Dashboard () { }

// logger.js

angular

.module('app')

.factory('logger', logger);

function logger () { }

[**回到顶部**](https://github.com/johnpapa/angular-styleguide/blob/master/a1/i18n/zh-CN.md#%E7%9B%AE%E5%BD%95)

## Controllers

###controllerAs在View中的语法

###### [Style [Y030](https://github.com/johnpapa/angular-styleguide/blob/master/a1/i18n/zh-CN.md#style-y030)]

* 使用[controllerAs](http://www.johnpapa.net/do-you-like-your-angular-controllers-with-or-without-sugar/) 语法代替直接用经典的$scope定义的controller的方式。

为什么？：controller被构建的时候，就会有一个新的实例，controllerAs 的语法比经典的$scope语法更接近JavaScript构造函数。

为什么？：这促进在View中对绑定到“有修饰”的对象的使用（例如用customer.name 代替name），这将更有语境、更容易阅读，也避免了任何没有“修饰”而产生的引用问题。

为什么？：有助于避免在有嵌套的controllers的Views中调用 $parent。

<!-- avoid -->

<div ng-controller="Customer">

{{ name }}

</div>

<!-- recommended -->

<div ng-controller="Customer as customer">

{{ customer.name }}

</div>

###controllerAs在controller中的语法

###### [Style [Y031](https://github.com/johnpapa/angular-styleguide/blob/master/a1/i18n/zh-CN.md#style-y031)]

* 使用 controllerAs 语法代替 经典的$scope语法 语法。
* 使用controllerAs 时，controller中的$scope被绑定到了this上。

为什么？：controllerAs 是$scope的语法修饰，你仍然可以绑定到View上并且访问 $scope的方法。

为什么？：避免在controller中使用 $scope，最好不用它们或是把它们移到一个factory中。factory中可以考虑使用$scope，controller中只在需要时候才使用$scope，例如当使用[$emit](https://docs.angularjs.org/api/ng/type/$rootScope.Scope#$emit)， [$broadcast](https://docs.angularjs.org/api/ng/type/$rootScope.Scope#$broadcast)，或者 [$on](https://docs.angularjs.org/api/ng/type/$rootScope.Scope#$on)。

/\* avoid \*/

function Customer ($scope) {

$scope.name = {};

$scope.sendMessage = function() { };

}

/\* recommended - but see next section \*/

function Customer () {

this.name = {};

this.sendMessage = function() { };

}

###controllerAs with vm

###### [Style [Y032](https://github.com/johnpapa/angular-styleguide/blob/master/a1/i18n/zh-CN.md#style-y032)]

* 使用controllerAs语法时把this 赋值给一个可捕获的变量，选择一个有代表性的名称，例如vm代表ViewModel。

为什么？：this在不同的地方有不同的语义（就是作用域不同），在controller中的一个函数内部使用this时可能会改变它的上下文。用一个变量来捕获this的上下文从而可以避免遇到这样的坑。

/\* avoid \*/

function Customer() {

this.name = {};

this.sendMessage = function() { };

}

/\* recommended \*/

function Customer () {

var vm = this;

vm.name = {};

vm.sendMessage = function() { };

}

* 注：你可以参照下面的做法来避免 [jshint](http://www.jshint.com/)的警告。但是构造函数（函数名首字母大写）是不需要这个的.

/\* jshint validthis: true \*/

var vm = this;

* 注：在controller中用controller as创建了一个watch时，可以用下面的语法监测vm.\*的成员。（创建watch时要谨慎，因为它会增加更多的负载）

<input ng-model="vm.title"/>

function SomeController($scope, $log) {

var vm = this;

vm.title = 'Some Title';

$scope.$watch('vm.title', function(current, original) {

$log.info('vm.title was %s', original);

$log.info('vm.title is now %s', current);

});

}

###置顶绑定成员

###### [Style [Y033](https://github.com/johnpapa/angular-styleguide/blob/master/a1/i18n/zh-CN.md#style-y033)]

* 把可绑定的成员放到controller的顶部，按字母排序，并且不要通过controller的代码传播。

为什么？：虽然设置单行匿名函数很容易，但是当这些函数的代码超过一行时，这将极大降低代码的可读性。在可绑定成员下面定义函数（这些函数被提出来），把具体的实现细节放到下面，可绑定成员置顶，这会提高代码的可读性。

/\* avoid \*/

function Sessions() {

var vm = this;

vm.gotoSession = function() {

/\* ... \*/

};

vm.refresh = function() {

/\* ... \*/

};

vm.search = function() {

/\* ... \*/

};

vm.sessions = [];

vm.title = 'Sessions';

/\* recommended \*/

function Sessions() {

var vm = this;

vm.gotoSession = gotoSession;

vm.refresh = refresh;

vm.search = search;

vm.sessions = [];

vm.title = 'Sessions';

////////////

function gotoSession() {

/\* \*/

}

function refresh() {

/\* \*/

}

function search() {

/\* \*/

}

[](https://raw.githubusercontent.com/johnpapa/angular-styleguide/master/a1/assets/above-the-fold-1.png)

注：如果一个函数就是一行，那么只要不影响可读性就把它放到顶部。

/\* avoid \*/

function Sessions(data) {

var vm = this;

vm.gotoSession = gotoSession;

vm.refresh = function() {

/\*\*

\* lines

\* of

\* code

\* affects

\* readability

\*/

};

vm.search = search;

vm.sessions = [];

vm.title = 'Sessions';

/\* recommended \*/

function Sessions(dataservice) {

var vm = this;

vm.gotoSession = gotoSession;

vm.refresh = dataservice.refresh; // 1 liner is OK

vm.search = search;

vm.sessions = [];

vm.title = 'Sessions';

###函数声明隐藏实现细节

###### [Style [Y034](https://github.com/johnpapa/angular-styleguide/blob/master/a1/i18n/zh-CN.md#style-y034)]

* 使用函数声明来隐藏实现细节，置顶绑定成员，当你需要在controller中绑定一个函数时，把它指向一个在文件的后面会出现函数声明。更多详情请看[这里](http://www.johnpapa.net/angular-function-declarations-function-expressions-and-readable-code)。

为什么？：易读，易识别哪些成员可以在View中绑定和使用。

为什么？：把函数的实现细节放到后面，你可以更清楚地看到重要的东西。

为什么？：由于函数声明会被置顶，所以没有必要担心在声明它之前就使用函数的问题。

为什么？：你再也不用担心当 a依赖于 b时，把var a放到var b之前会中断你的代码的函数声明问题。

为什么？：函数表达式中顺序是至关重要的。

/\*\*

\* avoid

\* Using function expressions.

\*/

function Avengers(dataservice, logger) {

var vm = this;

vm.avengers = [];

vm.title = 'Avengers';

var activate = function() {

return getAvengers().then(function() {

logger.info('Activated Avengers View');

});

}

var getAvengers = function() {

return dataservice.getAvengers().then(function(data) {

vm.avengers = data;

return vm.avengers;

});

}

vm.getAvengers = getAvengers;

activate();

}

注意这里重要的代码分散在前面的例子中。 下面的示例中，可以看到重要的代码都放到了顶部。实现的详细细节都在下方，显然这样的代码更易读。

/\*

\* recommend

\* Using function declarations

\* and bindable members up top.

\*/

function Avengers(dataservice, logger) {

var vm = this;

vm.avengers = [];

vm.getAvengers = getAvengers;

vm.title = 'Avengers';

activate();

function activate() {

return getAvengers().then(function() {

logger.info('Activated Avengers View');

});

}

function getAvengers() {

return dataservice.getAvengers().then(function(data) {

vm.avengers = data;

return vm.avengers;

});

}

}

###把Controller中的逻辑延迟到Service中

###### [Style [Y035](https://github.com/johnpapa/angular-styleguide/blob/master/a1/i18n/zh-CN.md#style-y035)]

* 通过委派到service和factory中来延迟controller中的逻辑。

为什么？：把逻辑放到service中，并通过一个function暴露，就可以被多个controller重用。

为什么？：把逻辑放到service中将会使单元测试的时候更加容易地把它们分离，相反，如果在controller中调用逻辑就显得很二了。

为什么？：保持controller的简洁。

为什么？：从controller中删除依赖关系并且隐藏实现细节。

/\* avoid \*/

function Order($http, $q, config, userInfo) {

var vm = this;

vm.checkCredit = checkCredit;

vm.isCreditOk;

vm.total = 0;

function checkCredit () {

var settings = {};

// Get the credit service base URL from config

// Set credit service required headers

// Prepare URL query string or data object with request data

// Add user-identifying info so service gets the right credit limit for this user.

// Use JSONP for this browser if it doesn't support CORS

return $http.get(settings)

.then(function(data) {

// Unpack JSON data in the response object

// to find maxRemainingAmount

vm.isCreditOk = vm.total <= maxRemainingAmount

})

.catch(function(error) {

// Interpret error

// Cope w/ timeout? retry? try alternate service?

// Re-reject with appropriate error for a user to see

});

};

}

/\* recommended \*/

function Order (creditService) {

var vm = this;

vm.checkCredit = checkCredit;

vm.isCreditOk;

vm.total = 0;

function checkCredit () {

return creditService.isOrderTotalOk(vm.total)

.then(function(isOk) { vm.isCreditOk = isOk; })

.catch(showServiceError);

};

}

###保持Controller的专一性

###### [Style [Y037](https://github.com/johnpapa/angular-styleguide/blob/master/a1/i18n/zh-CN.md#style-y037)]

* 一个view定义一个controller，尽量不要在其它view中使用这个controller。把可重用的逻辑放到factory中，保证controller只服务于当前视图。

为什么？：不同的view用同一个controller是非常不科学的，良好的端对端测试覆盖率对于保证大型应用稳定性是必需的。

###分配Controller

###### [Style [Y038](https://github.com/johnpapa/angular-styleguide/blob/master/a1/i18n/zh-CN.md#style-y038)]

* 当一个controller必须匹配一个view时或者任何一个组件可能被其它controller或是view重用时，连同controller的route一起定义。

注：如果一个view是通过route外的其它形式加载的，那么就用ng-controller="Avengers as vm"语法。

为什么？：在route中匹配controller允许不同的路由调用不同的相匹配的controller和view，当在view中通过[ng-controller](https://docs.angularjs.org/api/ng/directive/ngController)分配controller时，这个view总是和相同的controller相关联。

/\* avoid - when using with a route and dynamic pairing is desired \*/

// route-config.js

angular

.module('app')

.config(config);

function config ($routeProvider) {

$routeProvider

.when('/avengers', {

templateUrl: 'avengers.html'

});

}

<!-- avengers.html -->

<div ng-controller="Avengers as vm">

</div>

/\* recommended \*/

// route-config.js

angular

.module('app')

.config(config);

function config ($routeProvider) {

$routeProvider

.when('/avengers', {

templateUrl: 'avengers.html',

controller: 'Avengers',

controllerAs: 'vm'

});

}

<!-- avengers.html -->

<div>

</div>

[**返回顶部**](https://github.com/johnpapa/angular-styleguide/blob/master/a1/i18n/zh-CN.md#%E7%9B%AE%E5%BD%95)

## Services

###单例

###### [Style [Y040](https://github.com/johnpapa/angular-styleguide/blob/master/a1/i18n/zh-CN.md#style-y040)]

* 用new实例化service，用this实例化公共方法和变量，由于这和factory是类似的，所以为了保持统一，推荐用facotry来代替。

注意：[所有的Angular services都是单例](https://docs.angularjs.org/guide/services)，这意味着每个injector都只有一个实例化的service。

// service

angular

.module('app')

.service('logger', logger);

function logger () {

this.logError = function(msg) {

/\* \*/

};

}

// factory

angular

.module('app')

.factory('logger', logger);

function logger () {

return {

logError: function(msg) {

/\* \*/

}

};

}

[**返回顶部**](https://github.com/johnpapa/angular-styleguide/blob/master/a1/i18n/zh-CN.md#%E7%9B%AE%E5%BD%95)

## Factories

###单一职责

###### [Style [Y051](https://github.com/johnpapa/angular-styleguide/blob/master/a1/i18n/zh-CN.md#style-y051)]

* factory应该是[单一职责](http://en.wikipedia.org/wiki/Single_responsibility_principle)，这是由其上下文进行封装的。一旦一个factory将要处理超过单一的目的时，就应该创建一个新的factory。

###单例

###### [Style [Y051](https://github.com/johnpapa/angular-styleguide/blob/master/a1/i18n/zh-CN.md#style-y051)]

* facotry是一个单例，它返回一个包含service成员的对象。

注：[所有的Angular services都是单例](https://docs.angularjs.org/guide/services)，这意味着每个injector都只有一个实例化的service。

###可访问的成员置顶###

###### [Style [Y052](https://github.com/johnpapa/angular-styleguide/blob/master/a1/i18n/zh-CN.md#style-y052)]

* 使用从[显露模块模式](http://addyosmani.com/resources/essentialjsdesignpatterns/book/#revealingmodulepatternjavascript)派生出来的技术把service（它的接口）中可调用的成员暴露到顶部，

为什么？：易读，并且让你可以立即识别service中的哪些成员可以被调用，哪些成员必须进行单元测试（或者被别人嘲笑）。

为什么？：当文件内容很长时，这可以避免需要滚动才能看到暴露了哪些东西。

为什么？：虽然你可以随意写一个函数，但当函数代码超过一行时就会降低可读性并造成滚动。通过把实现细节放下面、把可调用接口置顶的形式返回service的方式来定义可调用的接口，从而使代码更加易读。

/\* avoid \*/

function dataService () {

var someValue = '';

function save () {

/\* \*/

};

function validate () {

/\* \*/

};

return {

save: save,

someValue: someValue,

validate: validate

};

}

/\* recommended \*/

function dataService () {

var someValue = '';

var service = {

save: save,

someValue: someValue,

validate: validate

};

return service;

////////////

function save () {

/\* \*/

};

function validate () {

/\* \*/

};

}

这种绑定方式复制了宿主对象，原始值不会随着暴露模块模式的使用而更新。

[](https://raw.githubusercontent.com/johnpapa/angular-styleguide/master/a1/assets/above-the-fold-2.png)

###函数声明隐藏实现细节

###### [Style [Y053](https://github.com/johnpapa/angular-styleguide/blob/master/a1/i18n/zh-CN.md#style-y053)]

* 函数声明隐藏实现细节，置顶绑定成员，当你需要在controller中绑定一个函数时，把它指向一个函数声明，这个函数声明在文件的后面会出现。

为什么？：易读，易识别哪些成员可以在View中绑定和使用。

为什么？：把函数的实现细节放到后面，你可以更清楚地看到重要的东西。

为什么？：由于函数声明会被置顶，所以没有必要担心在声明它之前就使用函数的问题。

为什么？：你再也不用担心当 a依赖于 b时，把var a放到var b之前会中断你的代码的函数声明问题。

为什么？：函数表达式中顺序是至关重要的。

/\*\*

\* avoid

\* Using function expressions

\*/

function dataservice($http, $location, $q, exception, logger) {

var isPrimed = false;

var primePromise;

var getAvengers = function() {

// implementation details go here

};

var getAvengerCount = function() {

// implementation details go here

};

var getAvengersCast = function() {

// implementation details go here

};

var prime = function() {

// implementation details go here

};

var ready = function(nextPromises) {

// implementation details go here

};

var service = {

getAvengersCast: getAvengersCast,

getAvengerCount: getAvengerCount,

getAvengers: getAvengers,

ready: ready

};

return service;

}

/\*\*

\* recommended

\* Using function declarations

\* and accessible members up top.

\*/

function dataservice($http, $location, $q, exception, logger) {

var isPrimed = false;

var primePromise;

var service = {

getAvengersCast: getAvengersCast,

getAvengerCount: getAvengerCount,

getAvengers: getAvengers,

ready: ready

};

return service;

////////////

function getAvengers() {

// implementation details go here

}

function getAvengerCount() {

// implementation details go here

}

function getAvengersCast() {

// implementation details go here

}

function prime() {

// implementation details go here

}

function ready(nextPromises) {

// implementation details go here

}

}

[**返回顶部**](https://github.com/johnpapa/angular-styleguide/blob/master/a1/i18n/zh-CN.md#%E7%9B%AE%E5%BD%95)

## Data Services

###独立的数据调用

###### [Style [Y060](https://github.com/johnpapa/angular-styleguide/blob/master/a1/i18n/zh-CN.md#style-y060)]

* 把进行数据操作和数据交互的逻辑放到factory中，数据服务负责XHR请求、本地存储、内存存储和其它任何数据操作。

为什么？：controller的作用是查看视图和收集视图的信息，它不应该关心如何取得数据，只需要知道哪里需要用到数据。把取数据的逻辑放到数据服务中能够让controller更简单、更专注于对view的控制。

为什么？：方便测试。

为什么？：数据服务的实现可能有非常明确的代码来处理数据仓库，这可能包含headers、如何与数据交互或是其它service，例如$http。把逻辑封装到单独的数据服务中，这隐藏了外部调用者（例如controller）对数据的直接操作，这样更加容易执行变更。

/\* recommended \*/

// dataservice factory

angular

.module('app.core')

.factory('dataservice', dataservice);

dataservice.$inject = ['$http', 'logger'];

function dataservice($http, logger) {

return {

getAvengers: getAvengers

};

function getAvengers() {

return $http.get('/api/maa')

.then(getAvengersComplete)

.catch(getAvengersFailed);

function getAvengersComplete(response) {

return response.data.results;

}

function getAvengersFailed(error) {

logger.error('XHR Failed for getAvengers.' + error.data);

}

}

}

注意：数据服务被调用时（例如controller），隐藏调用的直接行为，如下所示。

/\* recommended \*/

// controller calling the dataservice factory

angular

.module('app.avengers')

.controller('Avengers', Avengers);

Avengers.$inject = ['dataservice', 'logger'];

function Avengers(dataservice, logger) {

var vm = this;

vm.avengers = [];

activate();

function activate() {

return getAvengers().then(function() {

logger.info('Activated Avengers View');

});

}

function getAvengers() {

return dataservice.getAvengers()

.then(function(data) {

vm.avengers = data;

return vm.avengers;

});

}

}

###数据调用返回一个Promise

###### [Style [Y061](https://github.com/johnpapa/angular-styleguide/blob/master/a1/i18n/zh-CN.md#style-y061)]

* 就像$http一样，调用数据时返回一个promise，在你的调用函数中也返回一个promise。

为什么？：你可以把promise链接到一起，在数据调用完成并且resolve或是reject这个promise后采取进一步的行为。

/\* recommended \*/

activate();

function activate() {

/\*\*

\* Step 1

\* Ask the getAvengers function for the

\* avenger data and wait for the promise

\*/

return getAvengers().then(function() {

/\*\*

\* Step 4

\* Perform an action on resolve of final promise

\*/

logger.info('Activated Avengers View');

});

}

function getAvengers() {

/\*\*

\* Step 2

\* Ask the data service for the data and wait

\* for the promise

\*/

return dataservice.getAvengers()

.then(function(data) {

/\*\*

\* Step 3

\* set the data and resolve the promise

\*/

vm.avengers = data;

return vm.avengers;

});

}

[**返回顶部**](https://github.com/johnpapa/angular-styleguide/blob/master/a1/i18n/zh-CN.md#%E7%9B%AE%E5%BD%95)

## Directives

###一个directive一个文件

###### [Style [Y070](https://github.com/johnpapa/angular-styleguide/blob/master/a1/i18n/zh-CN.md#style-y070)]

* 一个文件中只创建一个directive，并依照directive来命名文件。

为什么？：虽然把所有directive放到一个文件中很简单，但是当一些directive是跨应用的，一些是跨模块的，一些仅仅在一个模块中使用时，想把它们独立出来就非常困难了。

为什么？：一个文件一个directive也更加容易维护。

注： "**最佳实践**：Angular文档中有提过，directive应该自动回收，当directive被移除后，你可以使用element.on('$destroy', ...)或者scope.$on('$destroy', ...)来执行一个clean-up函数。"

/\* avoid \*/

/\* directives.js \*/

angular

.module('app.widgets')

/\* order directive仅仅会被order module用到 \*/

.directive('orderCalendarRange', orderCalendarRange)

/\* sales directive可以在sales app的任意地方使用 \*/

.directive('salesCustomerInfo', salesCustomerInfo)

/\* spinner directive可以在任意apps中使用 \*/

.directive('sharedSpinner', sharedSpinner);

function orderCalendarRange() {

/\* implementation details \*/

}

function salesCustomerInfo() {

/\* implementation details \*/

}

function sharedSpinner() {

/\* implementation details \*/

}

/\* recommended \*/

/\* calendarRange.directive.js \*/

/\*\*

\* @desc order directive that is specific to the order module at a company named Acme

\* @example <div acme-order-calendar-range></div>

\*/

angular

.module('sales.order')

.directive('acmeOrderCalendarRange', orderCalendarRange);

function orderCalendarRange() {

/\* implementation details \*/

}

/\* recommended \*/

/\* customerInfo.directive.js \*/

/\*\*

\* @desc sales directive that can be used anywhere across the sales app at a company named Acme

\* @example <div acme-sales-customer-info></div>

\*/

angular

.module('sales.widgets')

.directive('acmeSalesCustomerInfo', salesCustomerInfo);

function salesCustomerInfo() {

/\* implementation details \*/

}

/\* recommended \*/

/\* spinner.directive.js \*/

/\*\*

\* @desc spinner directive that can be used anywhere across apps at a company named Acme

\* @example <div acme-shared-spinner></div>

\*/

angular

.module('shared.widgets')

.directive('acmeSharedSpinner', sharedSpinner);

function sharedSpinner() {

/\* implementation details \*/

}

注：由于directive使用条件比较广，所以命名就存在很多的选项。选择一个让directive和它的文件名都清楚分明的名字。下面有一些例子，不过更多的建议去看[命名](#命名)章节。

###在directive中操作DOM

###### [Style [Y072](https://github.com/johnpapa/angular-styleguide/blob/master/a1/i18n/zh-CN.md#style-y072)]

* 当需要直接操作DOM的时候，使用directive。如果有替代方法可以使用，例如：使用CSS来设置样式、[animation services](https://docs.angularjs.org/api/ngAnimate)、Angular模板、[ngShow](https://docs.angularjs.org/api/ng/directive/ngShow)或者[ngHide](https://docs.angularjs.org/api/ng/directive/ngHide)，那么就直接用这些即可。例如，如果一个directive只是想控制显示和隐藏，用ngHide/ngShow即可。

为什么？：DOM操作的测试和调试是很困难的，通常会有更好的方法（CSS、animations、templates）。

###提供一个唯一的Directive前缀

###### [Style [Y073](https://github.com/johnpapa/angular-styleguide/blob/master/a1/i18n/zh-CN.md#style-y073)]

* 提供一个短小、唯一、具有描述性的directive前缀，例如acmeSalesCustomerInfo在HTML中声明为acme-sales-customer-info。

为什么？：方便快速识别directive的内容和起源，例如acme-可能预示着这个directive是服务于Acme company。

注：避免使用ng-为前缀，研究一下其它广泛使用的directive避免命名冲突，例如[Ionic Framework](http://ionicframework.com/)的ion-。

###限制元素和属性

###### [Style [Y074](https://github.com/johnpapa/angular-styleguide/blob/master/a1/i18n/zh-CN.md#style-y074)]

* 当创建一个directive需要作为一个独立元素时，restrict值设置为E（自定义元素），也可以设置可选值A（自定义属性）。一般来说，如果它就是为了独立存在，用E是合适的做法。一般原则是允许EA，但是当它是独立的时候这更倾向于作为一个元素来实施，当它是为了增强已存在的DOM元素时则更倾向于作为一个属性来实施。

为什么？：这很有意义！

为什么？：虽然我们允许directive被当作一个class来使用，但如果这个directive的行为确实像一个元素的话，那么把directive当作元素或者属性是更有意义的。

注：Angular 1.3 +默认使用EA。

<!-- avoid -->

<div class="my-calendar-range"></div>

/\* avoid \*/

angular

.module('app.widgets')

.directive('myCalendarRange', myCalendarRange);

function myCalendarRange () {

var directive = {

link: link,

templateUrl: '/template/is/located/here.html',

restrict: 'C'

};

return directive;

function link(scope, element, attrs) {

/\* \*/

}

}

<!-- recommended -->

<my-calendar-range></my-calendar-range>

<div my-calendar-range></div>

/\* recommended \*/

angular

.module('app.widgets')

.directive('myCalendarRange', myCalendarRange);

function myCalendarRange () {

var directive = {

link: link,

templateUrl: '/template/is/located/here.html',

restrict: 'EA'

};

return directive;

function link(scope, element, attrs) {

/\* \*/

}

}

###Directives和ControllerAs

###### [Style [Y075](https://github.com/johnpapa/angular-styleguide/blob/master/a1/i18n/zh-CN.md#style-y075)]

* directive使用controller as语法，和view使用controller as保持一致。

为什么？：因为不难且有必要这样做。

注意：下面的directive演示了一些你可以在link和directive控制器中使用scope的方法，用controllerAs。这里把template放在行内是为了在一个地方写出这些代码。

注意：关于依赖注入的内容，请看[手动依赖注入](https://github.com/johnpapa/angular-styleguide/blob/master/a1/i18n/zh-CN.md#%E6%89%8B%E5%8A%A8%E4%BE%9D%E8%B5%96%E6%B3%A8%E5%85%A5)。

注意：directive的控制器是在directive外部的，这种风格避免了由于注入造成的return之后的代码无法访问的情况。

<div my-example max="77"></div>

angular

.module('app')

.directive('myExample', myExample);

function myExample() {

var directive = {

restrict: 'EA',

templateUrl: 'app/feature/example.directive.html',

scope: {

max: '='

},

link: linkFunc,

controller : ExampleController,

controllerAs: 'vm',

bindToController: true // because the scope is isolated

};

return directive;

function linkFunc(scope, el, attr, ctrl) {

console.log('LINK: scope.min = %s \*\*\* should be undefined', scope.min);

console.log('LINK: scope.max = %s \*\*\* should be undefined', scope.max);

console.log('LINK: scope.vm.min = %s', scope.vm.min);

console.log('LINK: scope.vm.max = %s', scope.vm.max);

}

}

ExampleController.$inject = ['$scope'];

function ExampleController($scope) {

// Injecting $scope just for comparison

var vm = this;

vm.min = 3;

console.log('CTRL: $scope.vm.min = %s', $scope.vm.min);

console.log('CTRL: $scope.vm.max = %s', $scope.vm.max);

console.log('CTRL: vm.min = %s', vm.min);

console.log('CTRL: vm.max = %s', vm.max);

}

<!-- example.directive.html -->

<div>hello world</div>

<div>max={{vm.max}}<input ng-model="{{vm.max}}"/></div>

<div>min={{vm.min}}<input ng-model="{{vm.min}}"/></div>

注意：当你把controller注入到link的函数或可访问的directive的attributes时，你可以把它命名为控制器的属性。

// Alternative to above example

function linkFunc(scope, el, attr, vm) { // 和上面例子的区别在于把vm直接传递进来

console.log('LINK: scope.min = %s \*\*\* should be undefined', scope.min);

console.log('LINK: scope.max = %s \*\*\* should be undefined', scope.max);

console.log('LINK: vm.min = %s', vm.min);

console.log('LINK: vm.max = %s', vm.max);

}

###### [Style [Y076](https://github.com/johnpapa/angular-styleguide/blob/master/a1/i18n/zh-CN.md#style-y076)]

* 当directive中使用了controller as语法时，如果你想把父级作用域绑定到directive的controller作用域时，使用bindToController = true。

为什么？：这使得把外部作用域绑定到directive controller中变得更加简单。

注意：Angular 1.3.0才介绍了bindToController。

<div my-example max="77"></div>

angular

.module('app')

.directive('myExample', myExample);

function myExample() {

var directive = {

restrict: 'EA',

templateUrl: 'app/feature/example.directive.html',

scope: {

max: '='

},

controller: ExampleController,

controllerAs: 'vm',

bindToController: true

};

return directive;

}

function ExampleController() {

var vm = this;

vm.min = 3;

console.log('CTRL: vm.min = %s', vm.min);

console.log('CTRL: vm.max = %s', vm.max);

}

<!-- example.directive.html -->

<div>hello world</div>

<div>max={{vm.max}}<input ng-model="vm.max"/></div>

<div>min={{vm.min}}<input ng-model="vm.min"/></div>

[**返回顶部**](https://github.com/johnpapa/angular-styleguide/blob/master/a1/i18n/zh-CN.md#%E7%9B%AE%E5%BD%95)

## 解决Controller的Promises

###Controller Activation Promises

###### [Style [Y080](https://github.com/johnpapa/angular-styleguide/blob/master/a1/i18n/zh-CN.md#style-y080)]

* 在activate函数中解决controller的启动逻辑。

为什么？：把启动逻辑放在一个controller中固定的位置可以方便定位、有利于保持测试的一致性，并能够避免controller中到处都是激活逻辑。

为什么？：activate这个controller使得重用刷新视图的逻辑变得很方便，把所有的逻辑都放到了一起，可以让用户更快地看到视图，可以很轻松地对ng-view 或 ui-view使用动画，用户体验更好。

注意：如果你需要在开始使用controller之前有条件地取消路由，那么就用[route resolve](https://github.com/johnpapa/angular-styleguide/blob/master/a1/i18n/zh-CN.md#style-y081)来代替。

/\* avoid \*/

function Avengers(dataservice) {

var vm = this;

vm.avengers = [];

vm.title = 'Avengers';

dataservice.getAvengers().then(function(data) {

vm.avengers = data;

return vm.avengers;

});

}

/\* recommended \*/

function Avengers(dataservice) {

var vm = this;

vm.avengers = [];

vm.title = 'Avengers';

activate();

////////////

function activate() {

return dataservice.getAvengers().then(function(data) {

vm.avengers = data;

return vm.avengers;

});

}

}

###Route Resolve Promises

###### [Style [Y081](https://github.com/johnpapa/angular-styleguide/blob/master/a1/i18n/zh-CN.md#style-y081)]

* 当一个controller在激活之前，需要依赖一个promise的完成时，那么就在controller的逻辑执行之前在$routeProvider中解决这些依赖。如果你需要在controller被激活之前有条件地取消一个路由，那么就用route resolver。
* 当你决定在过渡到视图之前取消路由时，使用route resolve。

为什么？：controller在加载前可能需要一些数据，这些数据可能是从一个通过自定义factory或是[$http](https://docs.angularjs.org/api/ng/service/$http)的promise而来的。[route resolve](https://docs.angularjs.org/api/ngRoute/provider/$routeProvider)允许promise在controller的逻辑执行之前解决，因此它可能对从promise中来的数据做一些处理。

为什么？：这段代码将在路由后的controller的激活函数中执行，视图立即加载，promise resolve的时候将会开始进行数据绑定，可以（通过ng-view或ui-view）在视图的过渡之间加个loading状态的动画。

注意：这段代码将在路由之前通过一个promise来执行，拒绝了承诺就会取消路由，接受了就会等待路由跳转到新视图。如果你想更快地进入视图，并且无需验证是否可以进入视图，你可以考虑用[控制器 activate 技术](https://github.com/johnpapa/angular-styleguide/blob/master/a1/i18n/zh-CN.md#style-y080)。

/\* avoid \*/

angular

.module('app')

.controller('Avengers', Avengers);

function Avengers (movieService) {

var vm = this;

// unresolved

vm.movies;

// resolved asynchronously

movieService.getMovies().then(function(response) {

vm.movies = response.movies;

});

}

/\* better \*/

// route-config.js

angular

.module('app')

.config(config);

function config ($routeProvider) {

$routeProvider

.when('/avengers', {

templateUrl: 'avengers.html',

controller: 'Avengers',

controllerAs: 'vm',

resolve: {

moviesPrepService: function(movieService) {

return movieService.getMovies();

}

}

});

}

// avengers.js

angular

.module('app')

.controller('Avengers', Avengers);

Avengers.$inject = ['moviesPrepService'];

function Avengers (moviesPrepService) {

var vm = this;

vm.movies = moviesPrepService.movies;

}

注意：下面这个例子展示了命名函数的路由解决，这种方式对于调试和处理依赖注入更加方便。

/\* even better \*/

// route-config.js

angular

.module('app')

.config(config);

function config($routeProvider) {

$routeProvider

.when('/avengers', {

templateUrl: 'avengers.html',

controller: 'Avengers',

controllerAs: 'vm',

resolve: {

moviesPrepService: moviesPrepService

}

});

}

function moviesPrepService(movieService) {

return movieService.getMovies();

}

// avengers.js

angular

.module('app')

.controller('Avengers', Avengers);

Avengers.$inject = ['moviesPrepService'];

function Avengers(moviesPrepService) {

var vm = this;

vm.movies = moviesPrepService.movies;

}

注意：示例代码中的movieService不符合安全压缩的做法，可以到[手动依赖注入](https://github.com/johnpapa/angular-styleguide/blob/master/a1/i18n/zh-CN.md#%E6%89%8B%E5%8A%A8%E4%BE%9D%E8%B5%96%E6%B3%A8%E5%85%A5)和[压缩和注释](https://github.com/johnpapa/angular-styleguide/blob/master/a1/i18n/zh-CN.md#%E5%8E%8B%E7%BC%A9%E5%92%8C%E6%B3%A8%E9%87%8A)部分学习如何安全压缩。

[**返回顶部**](https://github.com/johnpapa/angular-styleguide/blob/master/a1/i18n/zh-CN.md#%E7%9B%AE%E5%BD%95)

## 手动依赖注入

### 压缩的不安全性

###### [Style [Y090](https://github.com/johnpapa/angular-styleguide/blob/master/a1/i18n/zh-CN.md#style-y090)]

* 声明依赖时避免使用缩写语法。

为什么？：组件的参数（例如controller、factory等等）将会被转换成各种乱七八糟错误的变量。例如，common和dataservice可能会变成a或者b，但是这些转换后的变量在Angular中是找不到的。

/\* avoid - not minification-safe\*/

angular

.module('app')

.controller('Dashboard', Dashboard);

function Dashboard(common, dataservice) {

}

这一段代码在压缩时会产生错误的变量，因此在运行时就会报错。

/\* avoid - not minification-safe\*/

angular.module('app').controller('Dashboard', d);function d(a, b) { }

###手动添加依赖

###### [Style [Y091](https://github.com/johnpapa/angular-styleguide/blob/master/a1/i18n/zh-CN.md#style-y091)]

* 用$inject手动添加Angular组件所需的依赖。

为什么？：这种技术反映了使用[ng-annotate](https://github.com/olov/ng-annotate)的技术，这就是我推荐的对依赖关系进行自动化创建安全压缩的方式，如果ng-annotate检测到已经有了注入，那么它就不会再次重复执行。

为什么？：可以避免依赖变成其它Angular找不到的变量，例如，common和dataservice可能会变成a或者b。

为什么？：避免创建内嵌的依赖，因为一个数组太长不利于阅读，此外，内嵌的方式也会让人感到困惑，比如数组是一系列的字符串，但是最后一个却是组件的function。

/\* avoid \*/

angular

.module('app')

.controller('Dashboard',

['$location', '$routeParams', 'common', 'dataservice',

function Dashboard($location, $routeParams, common, dataservice) {}

]);

/\* avoid \*/

angular

.module('app')

.controller('Dashboard',

['$location', '$routeParams', 'common', 'dataservice', Dashboard]);

function Dashboard($location, $routeParams, common, dataservice) {

}

/\* recommended \*/

angular

.module('app')

.controller('Dashboard', Dashboard);

Dashboard.$inject = ['$location', '$routeParams', 'common', 'dataservice'];

function Dashboard($location, $routeParams, common, dataservice) {

}

注意：当你的函数处于return语句后面，那么`$inject`是无法访问的（这会在directive中发生），你可以通过把Controller移到directive外面来解决这个问题。

/\* avoid \*/

// inside a directive definition

function outer() {

var ddo = {

controller: DashboardPanelController,

controllerAs: 'vm'

};

return ddo;

DashboardPanelController.$inject = ['logger']; // Unreachable

function DashboardPanelController(logger) {

}

}

/\* recommended \*/

// outside a directive definition

function outer() {

var ddo = {

controller: DashboardPanelController,

controllerAs: 'vm'

};

return ddo;

}

DashboardPanelController.$inject = ['logger'];

function DashboardPanelController(logger) {

}

###手动确定路由解析器依赖

###### [Style [Y092](https://github.com/johnpapa/angular-styleguide/blob/master/a1/i18n/zh-CN.md#style-y092)]

* 用$inject手动给Angular组件添加路由解析器依赖。

为什么？：这种技术打破了路由解析的匿名函数的形式，易读。

为什么？：$inject语句可以让任何依赖都可以安全压缩。

/\* recommended \*/

function config ($routeProvider) {

$routeProvider

.when('/avengers', {

templateUrl: 'avengers.html',

controller: 'AvengersController',

controllerAs: 'vm',

resolve: {

moviesPrepService: moviesPrepService

}

});

}

moviesPrepService.$inject = ['movieService'];

function moviesPrepService(movieService) {

return movieService.getMovies();

}

[**返回顶部**](https://github.com/johnpapa/angular-styleguide/blob/master/a1/i18n/zh-CN.md#%E7%9B%AE%E5%BD%95)

## 压缩和注释

###ng-annotate

###### [Style [Y100](https://github.com/johnpapa/angular-styleguide/blob/master/a1/i18n/zh-CN.md#style-y100)]

* 在[Gulp](http://gulpjs.com/)或[Grunt](http://gruntjs.com/)中使用[ng-annotate](https://github.com/olov/ng-annotate)，用/\*\* @ngInject \*/对需要自动依赖注入的function进行注释。

为什么？：可以避免代码中的依赖使用到任何不安全的写法。

为什么？：不推荐用[ng-min](https://github.com/btford/ngmin)。

我更喜欢Gulp，因为我觉得它是易写易读易调试的。

下面的代码没有注入依赖，显然压缩是不安全的。

angular

.module('app')

.controller('Avengers', Avengers);

/\* @ngInject \*/

function Avengers (storageService, avengerService) {

var vm = this;

vm.heroSearch = '';

vm.storeHero = storeHero;

function storeHero(){

var hero = avengerService.find(vm.heroSearch);

storageService.save(hero.name, hero);

}

}

当上面的代码通过ng-annotate运行时，就会产生如下的带有$inject注释的输出结果，这样的话压缩就会安全了。

angular

.module('app')

.controller('Avengers', Avengers);

/\* @ngInject \*/

function Avengers (storageService, avengerService) {

var vm = this;

vm.heroSearch = '';

vm.storeHero = storeHero;

function storeHero(){

var hero = avengerService.find(vm.heroSearch);

storageService.save(hero.name, hero);

}

}

Avengers.$inject = ['storageService', 'avengerService'];

注意：如果ng-annotate检测到已经有注入了（例如发现了@ngInject），就不会重复生成$inject代码了。

注意：路由的函数前面也可以用/\* @ngInject \*/

// Using @ngInject annotations

function config($routeProvider) {

$routeProvider

.when('/avengers', {

templateUrl: 'avengers.html',

controller: 'Avengers',

controllerAs: 'vm',

resolve: { /\* @ngInject \*/

moviesPrepService: function(movieService) {

return movieService.getMovies();

}

}

});

}

注意：从Angular 1.3开始，你就可以用[ngApp](https://docs.angularjs.org/api/ng/directive/ngApp)指令的 ngStrictDi参数来检测任何可能失去依赖的地方，当以“strict-di”模式创建injector时，会导致应用程序无法调用不使用显示函数注释的函数（这也许无法安全压缩）。记录在控制台的调试信息可以帮助追踪出问题的代码。我只在需要调试的时候才会用到ng-strict-di。 <body ng-app="APP" ng-strict-di>

###使用Gulp或Grunt结合ng-annotate

###### [Style [Y101](https://github.com/johnpapa/angular-styleguide/blob/master/a1/i18n/zh-CN.md#style-y101)]

* 在自动化任务中使用[gulp-ng-annotate](https://www.npmjs.org/package/gulp-ng-annotate)或[grunt-ng-annotate](https://www.npmjs.org/package/grunt-ng-annotate)，把/\* @ngInject \*/注入到任何有依赖关系函数的前面。

为什么？：ng-annotate会捕获大部分的依赖关系，但是有时候需要借助于/\* @ngInject \*/语法提示。

下面的代码是gulp任务使用ngAnnotate的例子。

gulp.task('js', ['jshint'], function() {

var source = pkg.paths.js;

return gulp.src(source)

.pipe(sourcemaps.init())

.pipe(concat('all.min.js', {newLine: ';'}))

// Annotate before uglify so the code get's min'd properly.

.pipe(ngAnnotate({

// true helps add where @ngInject is not used. It infers.

// Doesn't work with resolve, so we must be explicit there

add: true

}))

.pipe(bytediff.start())

.pipe(uglify({mangle: true}))

.pipe(bytediff.stop())

.pipe(sourcemaps.write('./'))

.pipe(gulp.dest(pkg.paths.dev));

});

\*\*[返回顶部](#目录)\*\*

## 异常处理

###修饰符

###### [Style [Y110](#style-y110)]

- 使用一个[decorator](https://docs.angularjs.org/api/auto/service/$provide#decorator)，在配置的时候用[`$provide`](https://docs.angularjs.org/api/auto/service/$provide)服务，当发生异常时，在[`$exceptionHandler`](https://docs.angularjs.org/api/ng/service/$exceptionHandler)服务中执行自定义的处理方法。

\*为什么？\*：在开发时和运行时提供了一种统一的方式来处理未被捕获的Angular异常。

注：另一个选项是用来覆盖service的，这个可以代替decorator，这是一个非常nice的选项，但是如果你想保持默认行为，那么推荐你扩展一个decorator。

```javascript

/\* recommended \*/

angular

.module('blocks.exception')

.config(exceptionConfig);

exceptionConfig.$inject = ['$provide'];

function exceptionConfig($provide) {

$provide.decorator('$exceptionHandler', extendExceptionHandler);

}

extendExceptionHandler.$inject = ['$delegate', 'toastr'];

function extendExceptionHandler($delegate, toastr) {

return function(exception, cause) {

$delegate(exception, cause);

var errorData = {

exception: exception,

cause: cause

};

/\*\*

\* Could add the error to a service's collection,

\* add errors to $rootScope, log errors to remote web server,

\* or log locally. Or throw hard. It is entirely up to you.

\* throw exception;

\*/

toastr.error(exception.msg, errorData);

};

}

```

###异常捕获器

###### [Style [Y111](#style-y111)]

- 创建一个暴露了一个接口的factory来捕获异常并以合适方式处理异常。

\*为什么？\*：提供了一个统一的方法来捕获代码中抛出的异常。

注：异常捕获器对特殊异常的捕获和反应是非常友好的，例如，使用XHR从远程服务获取数据时，你想要捕获所有异常并做出不同的反应。

```javascript

/\* recommended \*/

angular

.module('blocks.exception')

.factory('exception', exception);

exception.$inject = ['logger'];

function exception(logger) {

var service = {

catcher: catcher

};

return service;

function catcher(message) {

return function(reason) {

logger.error(message, reason);

};

}

}

```

###路由错误

###### [Style [Y112](#style-y112)]

- 用[`$routeChangeError`](https://docs.angularjs.org/api/ngRoute/service/$route#$routeChangeError)来处理并打印出所有的路由错误信息。

\*为什么？\*：提供一个统一的方式来处理所有的路由错误。

\*为什么？\*：当一个路由发生错误的时候，可以给展示一个提示信息，提高用户体验。

```javascript

/\* recommended \*/

var handlingRouteChangeError = false;

function handleRoutingErrors() {

/\*\*

\* Route cancellation:

\* On routing error, go to the dashboard.

\* Provide an exit clause if it tries to do it twice.

\*/

$rootScope.$on('$routeChangeError',

function(event, current, previous, rejection) {

if (handlingRouteChangeError) { return; }

handlingRouteChangeError = true;

var destination = (current && (current.title ||

current.name || current.loadedTemplateUrl)) ||

'unknown target';

var msg = 'Error routing to ' + destination + '. ' +

(rejection.msg || '');

/\*\*

\* Optionally log using a custom service or $log.

\* (Don't forget to inject custom service)

\*/

logger.warning(msg, [current]);

/\*\*

\* On routing error, go to another route/state.

\*/

$location.path('/');

}

);

}

```

\*\*[返回顶部](#目录)\*\*

## 命名

###命名原则

###### [Style [Y120](#style-y120)]

- 遵循以描述组件功能，然后是类型（可选）的方式来给所有的组件提供统一的命名，我推荐的做法是`feature.type.js`。大多数文件都有2个名字。

\* 文件名 (`avengers.controller.js`)

\* 带有Angular的注册组件名 (`AvengersController`)

\*为什么？\*：命名约定有助于为一目了然地找到内容提供一个统一的方式，在项目中和团队中保持统一性是非常重要的，保持统一性对于跨公司来说提供了巨大的效率。

\*为什么？\*：命名约定应该只为代码的检索和沟通提供方便。

###功能文件命名

###### [Style [Y121](#style-y121)]

- 遵循以“描述组件功能.类型（可选）”的方式来给所有的组件提供统一的命名，我推荐的做法是`feature.type.js`。

\*为什么？\*：为快速识别组件提供了统一的方式。

\*为什么？\*：为任何自动化的任务提供模式匹配。

```javascript

/\*\*

\* common options

\*/

// Controllers

avengers.js

avengers.controller.js

avengersController.js

// Services/Factories

logger.js

logger.service.js

loggerService.js

```

```javascript

/\*\*

\* recommended

\*/

// controllers

avengers.controller.js

avengers.controller.spec.js

// services/factories

logger.service.js

logger.service.spec.js

// constants

constants.js

// module definition

avengers.module.js

// routes

avengers.routes.js

avengers.routes.spec.js

// configuration

avengers.config.js

// directives

avenger-profile.directive.js

avenger-profile.directive.spec.js

```

注意：另外一种常见的约定就是不要用`controller`这个词来给controller文件命名，例如不要用`avengers.controller.js`，而是用`avengers.js`。所有其它的约定都坚持使用类型作为后缀，但是controller是组件中最为常用的类型，因此这种做法的好处貌似仅仅是节省了打字，但是仍然很容易识别。我建议你为你的团队选择一种约定，并且要保持统一性。我喜欢的命名方式是`avengers.controller.js`。

```javascript

/\*\*

\* recommended

\*/

// Controllers

avengers.js

avengers.spec.js

```

###测试文件命名

###### [Style [Y122](#style-y122)]

- 和组件命名差不多，带上一个`spec`后缀。

\*为什么？\*：为快速识别组件提供统一的方式。

\*为什么？\*：为[karma](http://karma-runner.github.io/)或是其它测试运行器提供模式匹配。

```javascript

/\*\*

\* recommended

\*/

avengers.controller.spec.js

logger.service.spec.js

avengers.routes.spec.js

avenger-profile.directive.spec.js

```

###Controller命名

###### [Style [Y123](#style-y123)]

- 为所有controller提供统一的名称，先特征后名字，鉴于controller是构造函数，所以要采用UpperCamelCase（每个单词首字母大写）的方式。

\*为什么？\*：为快速识别和引用controller提供统一的方式。

\*为什么？\*：UpperCamelCase是常规的识别一个可以用构造函数来实例化的对象的方式。

```javascript

/\*\*

\* recommended

\*/

// avengers.controller.js

angular

.module

.controller('HeroAvengersController', HeroAvengersController);

function HeroAvengers(){ }

```

###Controller命名后缀

###### [Style [Y124](#style-y124)]

- 使用`Controller`。

\*为什么？\*：`Controller`使用更广泛、更明确、更具有描述性。

```javascript

/\*\*

\* recommended

\*/

// avengers.controller.js

angular

.module

.controller('AvengersController', AvengersController);

function AvengersController(){ }

```

###Factory命名

###### [Style [Y125](#style-y125)]

- 一样要统一，对service和factory使用camel-casing（驼峰式，第一个单词首字母小写，后面单词首字母大写）方式。避免使用`$`前缀。

\*为什么？\*：可以快速识别和引用factory。

\*为什么？\*：避免与内部使用`$`前缀的服务发生冲突。

```javascript

/\*\*

\* recommended

\*/

// logger.service.js

angular

.module

.factory('logger', logger);

function logger(){ }

```

###Directive组件命名

###### [Style [Y126](#style-y126)]

- 使用camel-case方式，用一个短的前缀来描述directive在哪个区域使用（一些例子中是使用公司前缀或是项目前缀）。

\*为什么？\*：可以快速识别和引用controller。

```javascript

/\*\*

\* recommended

\*/

// avenger-profile.directive.js

angular

.module

.directive('xxAvengerProfile', xxAvengerProfile);

// usage is <xx-avenger-profile> </xx-avenger-profile>

function xxAvengerProfile(){ }

```

###模块

###### [Style [Y127](#style-y127)]

- 当有很多的模块时，主模块文件命名成`app.module.js`，其它依赖模块以它们代表的内容来命名。例如，一个管理员模块命名成`admin.module.js`，它们各自的注册模块名字就是`app`和`admin`。

\*为什么？\*：给多模块的应用提供统一的方式，这也是为了扩展大型应用。

\*为什么？\*：对使用任务来自动化加载所有模块的定义（先）和其它所有的angular文件（后）提供了一种简单的方式。

###配置

###### [Style [Y128](#style-y128)]

- 把一个模块的配置独立到它自己的文件中，以这个模块为基础命名。`app`模块的配置文件命名成`app.config.js`（或是`config.js`），`admin.module.js`的配置文件命名成`admin.config.js`。

\*为什么？\*：把配置从模块定义、组件和活跃代码中分离出来。

\*为什么？\*：为设置模块的配置提供了一个可识别的地方。

###路由

###### [Style [Y129](#style-y129)]

- 把路由的配置独立到单独的文件。主模块的路由可能是`app.route.js`，`admin`模块的路由可能是`admin.route.js`。即使是在很小的应用中，我也喜欢把路由的配置从其余的配置中分离出来。

\*\*[返回顶部](#目录)\*\*

## 应用程序结构的LIFT准则

###LIFT

###### [Style [Y140](#style-y140)]

- 构建一个可以快速定位（`L`ocate）代码、一目了然地识别（`I`dentify）代码、拥有一个平直（`F`lattest）的结构、尽量（`T`ry）坚持DRY（Don’t Repeat Yourself）的应用程序，其结构应该遵循这4项基本准则。

\*为什么是LIFT?\*: 提供一个有良好扩展的结构，并且是模块化的，更快的找到代码能够帮助开发者提高效率。另一种检查你的app结构的方法就是问你自己：你能多块地打开涉及到一个功能的所有相关文件并开始工作？

当我发现我的的代码结构很恶心的时候，我就重新看看LIFT准则。

1. 轻松定位代码（L）

2. 一眼识别代码（I）

3. 平直的代码结构（层级不要太多）（F）

4. 尽量保持不要写重复代码（T）

###Locate

###### [Style [Y141](#style-y141)]

- 更直观、更简单、更快捷地定位代码

\*为什么？\*：我发现这对于一个项目是非常重要的，如果一个团队不能快速找到他们需要工作的文件，这将不能使团队足够高效地工作，那么这个代码结构就得改变。你可能不知道文件名或是相关的文件放在了哪里，那么就把他们放在最直观的地方，放在一起会节省大量的时间。下面是一个参考目录结构。

/bower\_components /client /app /avengers /blocks /exception /logger /core /dashboard /data /layout /widgets /content index.html .bower.json

###Identify

###### [Style [Y142](#style-y142)]

- 当你看到一个文件时你应该能够立即知道它包含了什么、代表了什么。

\*为什么？\*：你花费更少的时间来了解代码代表了什么，并且变得更加高效。如果这意味着你需要更长的名字，那么就这么干吧。文件名一定要具有描述性，保持和文件内容互为一体。避免文件中有多个controller，多个service，甚至是混合的。

###Flat

###### [Style [Y143](#style-y143)]

- 尽可能长时间地保持一个平直的文件夹结构，如果你的文件夹层级超过7+，那么就开始考虑分离。

\*为什么？\*：没有谁想在一个7级文件夹中寻找一个文件，你可以考虑一下网页导航栏有那么多层。文件夹结构没有硬性规则，但是当一个文件夹下的文件有7-10个，那么就是时候创建子文件夹了，文件夹的层级一定要把握好。一直使用一个平直的结构，直到确实有必要（帮助其它的LIFT）创建一个新的文件夹。

###T-DRY（尽量坚持DRY）

###### [Style [Y144](#style-y144)]

- 坚持DRY，但是不要疯了一样的做却牺牲了可读性。

\*为什么？\*：保持DRY很重要，但是如果牺牲了其它LIFT，那么它就没那么重要了，这就是为什么说尽量坚持DRY。

\*\*[返回顶部](#目录)\*\*

## 应用程序结构

###总规范

###### [Style [Y150](#style-y150)]

- 有实施的短期看法和长远的目标，换句话说，从小处入手，但是要记住app的走向。app的所有代码都在一个叫做`app`的根目录下，所有的内容都遵循一个功能一个文件，每一个controller、service、module、view都是独立的文件。第三方脚本存放在另外的根文件夹中（`bower\_components`、`scripts`、`lib`）。

注：了解实例结构的具体信息看[Angular应用结构](http://www.johnpapa.net/angular-app-structuring-guidelines/)。

###Layout

###### [Style [Y151](#style-y151)]

- 把定义应用程序总体布局的组件放到`layout`文件夹中，如导航、内容区等等。

\*为什么？\*：复用。

###按功能划分文件夹结构

###### [Style [Y152](#style-y152)]

- 按照它们代表的功能来给创建的文件夹命名，当文件夹包含的文件超过7个（根据需要自行调整数量限制），就考虑新建文件夹。

\*为什么？\*：开发者可以快速定位代码、快速识别文件代表的意思，结构尽可能平直，没有重复，没有多余名字。

\*为什么？\*：LIFT规范都包括在内。

\*为什么？\*：通过组织内容和让它们保持和LIFT指导准则一致，帮助降低应用程序变得混乱的可能性。

\*为什么？\*：超过10个文件时，在一个一致性的文件夹中很容易定位，但是在一个平直的文件夹结构中确实很难定位。

```javascript

/\*\*

\* recommended

\*/

app/

app.module.js

app.config.js

directives/

calendar.directive.js

calendar.directive.html

user-profile.directive.js

user-profile.directive.html

services/

dataservice.js

localstorage.js

logger.js

spinner.js

layout/

shell.html

shell.controller.js

topnav.html

topnav.controller.js

people/

attendees.html

attendees.controller.js

people.routes.js

speakers.html

speakers.controller.js

speaker-detail.html

speaker-detail.controller.js

sessions/

sessions.html

sessions.controller.js

sessions.routes.js

session-detail.html

session-detail.controller.js

```

![实例App结构](https://raw.githubusercontent.com/johnpapa/angular-styleguide/master/a1/assets/modularity-2.png)

注意：不要使用按类型划分文件夹结构，因为如果这样的话，当做一个功能时，需要在多个文件夹中来回切换。当应用程序有5个、10个，甚至是25个以上的view、controller（或其他feature）时，这种方式将迅速变得不实用，这就使得它定位文件比按功能分文件夹的方式要困难的多。

```javascript

/\*

\* avoid

\* Alternative folders-by-type.

\* I recommend "folders-by-feature", instead.

\*/

app/

app.module.js

app.config.js

app.routes.js

directives.js

controllers/

attendees.js

session-detail.js

sessions.js

shell.js

speakers.js

speaker-detail.js

topnav.js

directives/

calendar.directive.js

calendar.directive.html

user-profile.directive.js

user-profile.directive.html

services/

dataservice.js

localstorage.js

logger.js

spinner.js

views/

attendees.html

session-detail.html

sessions.html

shell.html

speakers.html

speaker-detail.html

topnav.html

```

\*\*[返回顶部](#目录)\*\*

## 模块化

###许多小的、独立的模块

###### [Style [Y160](#style-y160)]

- 创建只封装一个职责的小模块。

\*为什么？\*：模块化的应用程序很容易添加新的功能。

###创建一个App Module

###### [Style [Y161](#style-y161)]

- 创建一个应用程序的根模块，它的职责是把应用程序中所有的模块和功能都放到一起。

\*为什么？\*：Angular鼓励模块化和分离模式。创建根模块的作用是把其它模块都绑定到一起，这为增加或是删除一个模块提供了非常简单的方法。

\*为什么？\*：应用程序模块变成了一个描述哪些模块有助于定义应用程序的清单。

###保持App Module的精简

###### [Style [Y162](#style-y162)]

- app module中只放聚合其它模块的逻辑，具体功能在它们自己的模块中实现。

\*为什么？\*：添加额外的代码（获取数据、展现视图、其它和聚合模块无关的代码）到app module中使app module变得很糟糕，也使得模块难以重用和关闭。

###功能区域就是模块

###### [Style [Y163](#style-y163)]

- 创建代表功能区的模块，例如布局、可重用、共享服务、仪表盘和app的特殊功能（例如客户、管理、销售）。

\*为什么？\*：自包含的模块可以无缝地被添加到应用程序中。

\*为什么？\*：项目进行功能迭代时，可以专注于功能，在开发完成启用它们即可。

\*为什么？\*：把功能拆分成不同模块方便测试。

###可重用的块就是模块

###### [Style [Y164](#style-y164)]

- 为通用service创建代表可重用的应用程序块的模块，例如异常处理、日志记录、诊断、安全性和本地数据储藏等模块。

\*为什么？\*：这些类型的功能在很多应用程序中都需要用到，所以把它们分离到自己的模块中，它们可以变成通用的应用程序，也能被跨应用地进行重用。

###模块依赖

###### [Style [Y165](#style-y165)]

- 应用程序根模块依赖于应用程序特定的功能模块、共享的和可复用的模块。

![模块化和依赖](https://raw.githubusercontent.com/johnpapa/angular-styleguide/master/a1/assets/modularity-1.png)

\*为什么？\*：主程序模块包含一个能快速识别应用程序功能的清单。

\*为什么？\*：每个功能区都包含一个它依赖了哪些模块的列表，因此其它应用可以把它当作一个依赖引入进来。

\*为什么？\*：程序内部的功能，如共享数据的服务变得容易定位，并且从`app.core`中共享。

注意：这是保持一致性的一种策略，这里有很多不错的选择，选择一种统一的，遵循Angular依赖规则，这将易于维护和扩展。

> 我的不同项目间的结构略有不同，但是它们都遵循了这些结构和模块化的准则,具体的实施方案会根据功能和团队发生变化。也就是说，不要在一棵树上吊死，但是心中一定要记得保持一致性、可维护性和效率。

> 小项目中，你可以直接把所有依赖都放到app module中，这对于小项目来说比较容易维护，但是想在此项目外重用模块就比较难了。

\*\*[返回顶部](#目录)\*\*

## 启动逻辑

### 配置

###### [Style [Y170](#style-y170)]

- 必须在angular应用启动前进行配置才能把代码注入到[模块配置](https://docs.angularjs.org/guide/module#module-loading-dependencies)，理想的一些case应该包括providers和constants。

\*为什么？\*：这使得在更少的地方进行配置变得容易。

```javascript

angular

.module('app')

.config(configure);

configure.$inject =

['routerHelperProvider', 'exceptionHandlerProvider', 'toastr'];

function configure (routerHelperProvider, exceptionHandlerProvider, toastr) {

exceptionHandlerProvider.configure(config.appErrorPrefix);

configureStateHelper();

toastr.options.timeOut = 4000;

toastr.options.positionClass = 'toast-bottom-right';

////////////////

function configureStateHelper() {

routerHelperProvider.configure({

docTitle: 'NG-Modular: '

});

}

}

```

### 运行代码块

###### [Style [Y171](#style-y171)]

- 任何在应用程序启动时需要运行的代码都应该在factory中声明，通过一个function暴露出来，然后注入到[运行代码块](https://docs.angularjs.org/guide/module#module-loading-dependencies)中。

\*为什么？\*：直接在运行代码块处写代码将会使得测试变得很困难，相反，如果放到facotry则会使的抽象和模拟变得很简单。

```javascript

angular

.module('app')

.run(runBlock);

runBlock.$inject = ['authenticator', 'translator'];

function runBlock(authenticator, translator) {

authenticator.initialize();

translator.initialize();

}

[**返回顶部**](https://github.com/johnpapa/angular-styleguide/blob/master/a1/i18n/zh-CN.md#%E7%9B%AE%E5%BD%95)

##Angular $包装服务

###$document和$window

###### [Style [Y180](https://github.com/johnpapa/angular-styleguide/blob/master/a1/i18n/zh-CN.md#style-y180)]

* 用[$document](https://docs.angularjs.org/api/ng/service/$document)和[$window](https://docs.angularjs.org/api/ng/service/$window)代替document和window。

为什么？：使用内部包装服务将更容易测试，也避免了你自己去模拟document和window。

###$timeout和$interval

###### [Style [Y181](https://github.com/johnpapa/angular-styleguide/blob/master/a1/i18n/zh-CN.md#style-y181)]

* 用[$timeout](https://docs.angularjs.org/api/ng/service/$timeout)和[$interval](https://docs.angularjs.org/api/ng/service/$interval)代替setTimeout和setInterval 。

为什么？：易于测试，处理Angular消化周期从而保证数据的同步绑定。

[**返回顶部**](https://github.com/johnpapa/angular-styleguide/blob/master/a1/i18n/zh-CN.md#%E7%9B%AE%E5%BD%95)

## 测试

单元测试有助于保持代码的清晰，因此我加入一些关于单元测试的基础和获取更多信息的链接。

###用故事来编写测试

###### [Style [Y190](https://github.com/johnpapa/angular-styleguide/blob/master/a1/i18n/zh-CN.md#style-y190)]

* 给每一个故事都写一组测试，先创建一个空的测试，然后用你给这个故事写的代码来填充它。

为什么？：编写测试有助于明确规定你的故事要做什么、不做什么以及你如何判断是否成功。

it('should have Avengers controller', function() {

//TODO

});

it('should find 1 Avenger when filtered by name', function() {

//TODO

});

it('should have 10 Avengers', function() {

//TODO (mock data?)

});

it('should return Avengers via XHR', function() {

//TODO ($httpBackend?)

});

// and so on

###测试库

###### [Style [Y191](https://github.com/johnpapa/angular-styleguide/blob/master/a1/i18n/zh-CN.md#style-y191)]

* 用[Jasmine](http://jasmine.github.io/)或者[Mocha](http://mochajs.org/)进行单元测试。

为什么？：Angular社区中Jasmine和Mocha都用的很广，两者都很稳定，可维护性好，提供强大的测试功能。

注意：使用Mocha时你可以考虑选择一个类似[Chai](http://chaijs.com/)的提示库。

###测试运行器

###### [Style [Y192](https://github.com/johnpapa/angular-styleguide/blob/master/a1/i18n/zh-CN.md#style-y192)]

* [Karma](http://karma-runner.github.io/)。

为什么？：Karma容易配置，代码发生修改时自动运行。

为什么？：可以通过自身或是Grunt、Gulp方便地钩入持续集成的进程。

为什么？：一些IDE已经开始集成Karma了，如[WebStorm](http://www.jetbrains.com/webstorm/)和[Visual Studio](http://visualstudiogallery.msdn.microsoft.com/02f47876-0e7a-4f6c-93f8-1af5d5189225)。

为什么？：Karma可以很好的和自动化任务工具如[Grunt](http://www.gruntjs.com/)（带有[grunt-karma](https://github.com/karma-runner/grunt-karma)）和[Gulp](http://www.gulpjs.com/)（带有[gulp-karma](https://github.com/lazd/gulp-karma)）合作。

###Stubbing和Spying

###### [Style [Y193](https://github.com/johnpapa/angular-styleguide/blob/master/a1/i18n/zh-CN.md#style-y193)]

* 用[Sinon](http://sinonjs.org/)。

为什么？：Sinon可以和Jasmine和Mocha合作良好，并且可以扩展它们提供的stubbing和spying。

为什么？：如果你想试试Jasmine和Mocha，用Sinon在它们中间来回切换是很方便的。我更喜欢Mocha。

为什么？：测试失败Sinon有一个具有描述性的信息。

###Headless Browser

###### [Style [Y194](https://github.com/johnpapa/angular-styleguide/blob/master/a1/i18n/zh-CN.md#style-y194)]

* 在服务器上使用[PhantomJS](http://phantomjs.org/)来运行你的测试。

为什么？：PhantomJS是一个headless browser，无需一个“可视”的浏览器来帮助你运行测试。因此你的服务器上不需要安装Chrome、Safari、IE或是其它浏览器。

注意：你仍然需要在你的环境下测试所有浏览器，来满足用户的需求。

###代码分析

###### [Style [Y195](https://github.com/johnpapa/angular-styleguide/blob/master/a1/i18n/zh-CN.md#style-y195)]

-在你的测试上运行JSHint。

\*为什么？\*：测试也是代码，JSHint能够帮你识别代码中可能导致测试无法正常工作的的质量问题。

###对测试降低全局JSHint规则

###### [Style [Y196](https://github.com/johnpapa/angular-styleguide/blob/master/a1/i18n/zh-CN.md#style-y196)

* 对你的测试代码放宽规则，这样可以允许使用describe和expect等类似通用的全局方法。对表达式放宽规则，就行Mocha一样。

为什么？：测试也是代码，因此要和对待其它生产代码一样重视测试代码的质量。然而，测试框架中允许使用全局变量，例如，在你的测试单例中允许使用this。

/\* jshint -W117, -W030 \*/

或者你也可以把下面的这几行加入到你的JSHint Options文件中。

"jasmine": true,

"mocha": true,

[](https://raw.githubusercontent.com/johnpapa/angular-styleguide/master/a1/assets/testing-tools.png)

### 组织测试

###### [Style [Y197](https://github.com/johnpapa/angular-styleguide/blob/master/a1/i18n/zh-CN.md#style-y197)]

* 将单元测试文件(specs)同被测试客户端代码并列放在同一个文件夹下，将多个组件共用的测试文件以及服务器集成测试的文件放到一个单独的tests文件夹下。

为什么？：单元测试和源代码中的每一个组件和文件都有直接的相关性。

为什么？：这样它就会一直在你的视野中，很容易让它们保持在最新状态。编码的时候无论你做TDD还是在开发过程中测试，或者开发完成后测试，这些单测都不会脱离你的视线和脑海，这样就更容易维护，也有助于保持代码的覆盖率。

为什么？：更新源代码的时候可以更简单地在同一时间更新测试代码。

为什么？：方便源码阅读者了解组件如何使用，也便于发现其中的局限性。

为什么？：方便找。

为什么？：方便使用grunt或者gulp。

/src/client/app/customers/customer-detail.controller.js

/customer-detail.controller.spec.js

/customers.controller.js

/customers.controller.spec.js

/customers.module.js

/customers.route.js

/customers.route.spec.js

[**返回顶部**](https://github.com/johnpapa/angular-styleguide/blob/master/a1/i18n/zh-CN.md#%E7%9B%AE%E5%BD%95)

## 动画

###用法

###### [Style [Y210](https://github.com/johnpapa/angular-styleguide/blob/master/a1/i18n/zh-CN.md#style-y210)]

* 在页面过渡时使用[Angular动画](https://docs.angularjs.org/guide/animations)，包括[ngAnimate模块](https://docs.angularjs.org/api/ngAnimate)。三个关键点是细微、平滑、无缝。

为什么？：使用得当的话能够提高用户体验。

为什么？：当视图过渡时，微小的动画可以提高感知性。

###Sub Second

###### [Style [Y211](https://github.com/johnpapa/angular-styleguide/blob/master/a1/i18n/zh-CN.md#style-y211)]

* 使用短持续性的动画，我一般使用300ms，然后调整到合适的时间。

为什么？：长时间的动画容易造成用户认为程序性能太差的影响。

###animate.css

###### [Style [Y212](https://github.com/johnpapa/angular-styleguide/blob/master/a1/i18n/zh-CN.md#style-y212)]

* 传统动画使用[animate.css](http://daneden.github.io/animate.css/)。

为什么？：css提供的动画是快速的、流畅的、易于添加到应用程序中的。

为什么？：为动画提供一致性。

为什么？：animate.css被广泛使用和测试。

注意：参阅[Matias Niemelä的关于Angular动画的文章](http://www.yearofmoo.com/2013/08/remastered-animation-in-angularjs-1-2.html)

[**返回顶部**](https://github.com/johnpapa/angular-styleguide/blob/master/a1/i18n/zh-CN.md#%E7%9B%AE%E5%BD%95)

## 注释

### jsDoc

###### [Style [Y220](https://github.com/johnpapa/angular-styleguide/blob/master/a1/i18n/zh-CN.md#style-y220)]

* 如果你准备做一个文档，那么就使用[jsDoc](http://usejsdoc.org/)的语法来记录函数名、描述、参数和返回值。使用@namespace和@memberOf来匹配应用程序结构。

为什么？：你可以从代码中生成（重新生成）文档，而不必从头开始编写文档。

为什么？：使用业内通用工具保持了统一性。

/\*\*

\* Logger Factory

\* @namespace Factories

\*/

(function() {

angular

.module('app')

.factory('logger', logger);

/\*\*

\* @namespace Logger

\* @desc Application wide logger

\* @memberOf Factories

\*/

function logger ($log) {

var service = {

logError: logError

};

return service;

////////////

/\*\*

\* @name logError

\* @desc Logs errors

\* @param {String} msg Message to log

\* @returns {String}

\* @memberOf Factories.Logger

\*/

function logError(msg) {

var loggedMsg = 'Error: ' + msg;

$log.error(loggedMsg);

return loggedMsg;

};

}

})();

[**返回顶部**](https://github.com/johnpapa/angular-styleguide/blob/master/a1/i18n/zh-CN.md#%E7%9B%AE%E5%BD%95)

## JS Hint

###使用一个Options文件

###### [Style [Y230](https://github.com/johnpapa/angular-styleguide/blob/master/a1/i18n/zh-CN.md#style-y230)]

* 用JS Hint来分析你的JavaScript代码，确保你自定义了JS Hint选项文件并且包含在源控制里。详细信息：[JS Hint文档](http://www.jshint.com/docs/)。

为什么？：提交代码到原版本之前先发出警告。

为什么？：统一性。

{

"bitwise": true,

"camelcase": true,

"curly": true,

"eqeqeq": true,

"es3": false,

"forin": true,

"freeze": true,

"immed": true,

"indent": 4,

"latedef": "nofunc",

"newcap": true,

"noarg": true,

"noempty": true,

"nonbsp": true,

"nonew": true,

"plusplus": false,

"quotmark": "single",

"undef": true,

"unused": false,

"strict": false,

"maxparams": 10,

"maxdepth": 5,

"maxstatements": 40,

"maxcomplexity": 8,

"maxlen": 120,

"asi": false,

"boss": false,

"debug": false,

"eqnull": true,

"esnext": false,

"evil": false,

"expr": false,

"funcscope": false,

"globalstrict": false,

"iterator": false,

"lastsemic": false,

"laxbreak": false,

"laxcomma": false,

"loopfunc": true,

"maxerr": false,

"moz": false,

"multistr": false,

"notypeof": false,

"proto": false,

"scripturl": false,

"shadow": false,

"sub": true,

"supernew": false,

"validthis": false,

"noyield": false,

"browser": true,

"node": true,

"globals": {

"angular": false,

"$": false

}

}

[**返回顶部**](https://github.com/johnpapa/angular-styleguide/blob/master/a1/i18n/zh-CN.md#%E7%9B%AE%E5%BD%95)

## JSCS

### 用一个Options文件

###### [Style [Y235](https://github.com/johnpapa/angular-styleguide/blob/master/a1/i18n/zh-CN.md#style-y235)]

* 使用JSCS检查代码规范，确保你的代码控制中有定制的JSCS options文件，在这里[JSCS docs](http://www.jscs.info/)查看更多信息。

为什么？：提交代码前第一时间提供一个预警。

为什么？：保持团队的一致性。

{

"excludeFiles": ["node\_modules/\*\*", "bower\_components/\*\*"],

"requireCurlyBraces": [

"if",

"else",

"for",

"while",

"do",

"try",

"catch"

],

"requireOperatorBeforeLineBreak": true,

"requireCamelCaseOrUpperCaseIdentifiers": true,

"maximumLineLength": {

"value": 100,

"allowComments": true,

"allowRegex": true

},

"validateIndentation": 4,

"validateQuoteMarks": "'",

"disallowMultipleLineStrings": true,

"disallowMixedSpacesAndTabs": true,

"disallowTrailingWhitespace": true,

"disallowSpaceAfterPrefixUnaryOperators": true,

"disallowMultipleVarDecl": null,

"requireSpaceAfterKeywords": [

"if",

"else",

"for",

"while",

"do",

"switch",

"return",

"try",

"catch"

],

"requireSpaceBeforeBinaryOperators": [

"=", "+=", "-=", "\*=", "/=", "%=", "<<=", ">>=", ">>>=",

"&=", "|=", "^=", "+=",

"+", "-", "\*", "/", "%", "<<", ">>", ">>>", "&",

"|", "^", "&&", "||", "===", "==", ">=",

"<=", "<", ">", "!=", "!=="

],

"requireSpaceAfterBinaryOperators": true,

"requireSpacesInConditionalExpression": true,

"requireSpaceBeforeBlockStatements": true,

"requireLineFeedAtFileEnd": true,

"disallowSpacesInsideObjectBrackets": "all",

"disallowSpacesInsideArrayBrackets": "all",

"disallowSpacesInsideParentheses": true,

"validateJSDoc": {

"checkParamNames": true,

"requireParamTypes": true

},

"disallowMultipleLineBreaks": true,

"disallowCommaBeforeLineBreak": null,

"disallowDanglingUnderscores": null,

"disallowEmptyBlocks": null,

"disallowMultipleLineStrings": null,

"disallowTrailingComma": null,

"requireCommaBeforeLineBreak": null,

"requireDotNotation": null,

"requireMultipleVarDecl": null,

"requireParenthesesAroundIIFE": true

}

[**返回顶部**](https://github.com/johnpapa/angular-styleguide/blob/master/a1/i18n/zh-CN.md#%E7%9B%AE%E5%BD%95)

## 常量

###供应全局变量

###### [Style [Y240](https://github.com/johnpapa/angular-styleguide/blob/master/a1/i18n/zh-CN.md#style-y240)]

* 为供应库中的全局变量创建一个Angular常量。

为什么？：提供一种注入到供应库的方法，否则就是全局变量。通过让你更容易地了解你的组件之间的依赖关系来提高代码的可测试性。这还允许你模拟这些依赖关系，这是很有意义的。

// constants.js

/\* global toastr:false, moment:false \*/

(function() {

'use strict';

angular

.module('app.core')

.constant('toastr', toastr)

.constant('moment', moment);

})();

###### [Style [Y241](https://github.com/johnpapa/angular-styleguide/blob/master/a1/i18n/zh-CN.md#style-y241)]

* 对于一些不需要变动，也不需要从其它service中获取的值，使用常量定义，当一些常量只是在一个模块中使用但是有可能会在其它应用中使用的话，把它们写到一个以当前的模块命名的文件中。把常量集合到一起是非常有必要的，你可以把它们写到constants.js的文件中。

为什么？：一个可能变化的值，即使变动的很少，也会从service中重新被检索，因此你不需要修改源代码。例如，一个数据服务的url可以被放到一个常量中，但是更好的的做法是把它放到一个web service中。

为什么？：常量可以被注入到任何angular组件中，包括providers。

为什么？：当一个应用程序被分割成很多可以在其它应用程序中复用的小模块时，每个独立的模块都应该可以操作它自己包含的相关常量。

// Constants used by the entire app

angular

.module('app.core')

.constant('moment', moment);

// Constants used only by the sales module

angular

.module('app.sales')

.constant('events', {

ORDER\_CREATED: 'event\_order\_created',

INVENTORY\_DEPLETED: 'event\_inventory\_depleted'

});

[**返回顶部**](https://github.com/johnpapa/angular-styleguide/blob/master/a1/i18n/zh-CN.md#%E7%9B%AE%E5%BD%95)

## 文件模板和片段

为了遵循一致的规范和模式，使用文件模板和片段，这里有针对一些web开发的编辑器和IDE的模板和（或）片段。

###Sublime Text

###### [Style [Y250](https://github.com/johnpapa/angular-styleguide/blob/master/a1/i18n/zh-CN.md#style-y250)]

* Angular片段遵循这些规范。
  + 下载[Sublime Angular snippets](https://github.com/johnpapa/angular-styleguide/blob/master/a1/i18n/assets/sublime-angular-snippets?raw=true)
  + 把它放到Packages文件夹中
  + 重启Sublime
  + 在JavaScript文件中输入下面的命令然后按下TAB键即可：
* ngcontroller // creates an Angular controller
* ngdirective // creates an Angular directive
* ngfactory // creates an Angular factory
* ngmodule // creates an Angular module
* ngservice // creates an Angular service

ngfilter // creates an Angular filter

###Visual Studio

###### [Style [Y251](https://github.com/johnpapa/angular-styleguide/blob/master/a1/i18n/zh-CN.md#style-y251)]

* Angular文件遵循[SideWaffle](http://www.sidewaffle.com/)所介绍的规范。
  + 下载Visual Studio扩展文件[SideWaffle](http://www.sidewaffle.com/)
  + 运行下载的vsix文件
  + 重启Visual Studio

###WebStorm

###### [Style [Y252](https://github.com/johnpapa/angular-styleguide/blob/master/a1/i18n/zh-CN.md#style-y252)]

* 你可以把它们导入到WebStorm设置中:
  + 下载[WebStorm Angular file templates and snippets](https://github.com/johnpapa/angular-styleguide/blob/master/assets/webstorm-angular-file-template.settings.jar?raw=true)
  + 打开WebStorm点击File菜单
  + 选择Import Settings菜单选项
  + 选择文件点击OK
  + 在JavaScript文件中输入下面的命令然后按下TAB键即可：
* ng-c // creates an Angular controller
* ng-f // creates an Angular factory

ng-m // creates an Angular module

### Atom

###### [Style [Y253](https://github.com/johnpapa/angular-styleguide/blob/master/a1/i18n/zh-CN.md#style-y253)]

* Angular片段遵循以下规范。
* apm install angularjs-styleguide-snippets

或

* + 打开Atom，打开包管理器(Packages -> Settings View -> Install Packages/Themes)
  + 搜索'angularjs-styleguide-snippets'
  + 点击'Install' 进行安装
* JavaScript文件中输入以下命令后以TAB结束
* ngcontroller // creates an Angular controller
* ngdirective // creates an Angular directive
* ngfactory // creates an Angular factory
* ngmodule // creates an Angular module
* ngservice // creates an Angular service

ngfilter // creates an Angular filter

### Brackets

###### [Style [Y254](https://github.com/johnpapa/angular-styleguide/blob/master/a1/i18n/zh-CN.md#style-y254)]

* Angular代码片段遵循以下规范。
  + 下载[Brackets Angular snippets](https://github.com/johnpapa/angular-styleguide/blob/master/a1/i18n/assets/brackets-angular-snippets.yaml?raw=true)
  + 拓展管理器( File > Extension manager )
  + 安装['Brackets Snippets (by edc)'](https://github.com/chuyik/brackets-snippets)
  + Click the light bulb in brackets' right gutter
  + Click Settings and then Import
  + Choose the file and select to skip or override
  + Click Start Import
* JavaScript文件中输入以下命令后以TAB结束
* // These are full file snippets containing an IIFE
* ngcontroller // creates an Angular controller
* ngdirective // creates an Angular directive
* ngfactory // creates an Angular factory
* ngapp // creates an Angular module setter
* ngservice // creates an Angular service
* ngfilter // creates an Angular filter
* // These are partial snippets intended to chained
* ngmodule // creates an Angular module getter
* ngstate // creates an Angular UI Router state defintion
* ngconfig // defines a configuration phase function
* ngrun // defines a run phase function

ngroute // creates an Angular routeProvider

### vim

###### [Style [Y255](https://github.com/johnpapa/angular-styleguide/blob/master/a1/i18n/zh-CN.md#style-y255)]

* vim代码片段遵循以下规范。
  + 下载[vim Angular代码段](https://github.com/johnpapa/angular-styleguide/blob/master/a1/i18n/assets/vim-angular-snippets?raw=true)
  + 设置[neosnippet.vim](https://github.com/Shougo/neosnippet.vim)
  + 粘贴到snippet路径下
* ngcontroller // creates an Angular controller
* ngdirective // creates an Angular directive
* ngfactory // creates an Angular factory
* ngmodule // creates an Angular module
* ngservice // creates an Angular service

ngfilter // creates an Angular filter

[**返回顶部**](https://github.com/johnpapa/angular-styleguide/blob/master/a1/i18n/zh-CN.md#%E7%9B%AE%E5%BD%95)

## Yeoman Generator

###### [Style [Y260](https://github.com/johnpapa/angular-styleguide/blob/master/a1/i18n/zh-CN.md#style-y260)]

你可以使用[HotTowel yeoman generator](http://jpapa.me/yohottowel)来创建一个遵循本规范的Angular入门应用。

1. 安装generator-hottowel

npm install -g generator-hottowel

1. 创建一个新的文件夹并定位到它

mkdir myapp

cd myapp

1. 运行生成器

yo hottowel helloWorld

[**返回顶部**](https://github.com/johnpapa/angular-styleguide/blob/master/a1/i18n/zh-CN.md#%E7%9B%AE%E5%BD%95)

## 路由

客户端路由对于在视图和很多小模板和指令组成的构成视图中创建导航是非常重要的。

###### [Style [Y270](https://github.com/johnpapa/angular-styleguide/blob/master/a1/i18n/zh-CN.md#style-y270)]

* 用[AngularUI Router](http://angular-ui.github.io/ui-router/)来做路由控制。

为什么？：它包含了Angular路由的所有特性，并且增加了一些额外的特性，如嵌套路由和状态。

为什么？：语法和Angular路由很像，很容易迁移到UI Router。

* 注意：你可以在运行期间使用routerHelperProvider配置跨文件状态
* // customers.routes.js
* angular
* .module('app.customers')
* .run(appRun);
* /\* @ngInject \*/
* function appRun(routerHelper) {
* routerHelper.configureStates(getStates());
* }
* function getStates() {
* return [
* {
* state: 'customer',
* config: {
* abstract: true,
* template: '<ui-view class="shuffle-animation"/>',
* url: '/customer'
* }
* }
* ];

}

// routerHelperProvider.js

angular

.module('blocks.router')

.provider('routerHelper', routerHelperProvider);

routerHelperProvider.$inject = ['$locationProvider', '$stateProvider', '$urlRouterProvider'];

/\* @ngInject \*/

function routerHelperProvider($locationProvider, $stateProvider, $urlRouterProvider) {

/\* jshint validthis:true \*/

this.$get = RouterHelper;

$locationProvider.html5Mode(true);

RouterHelper.$inject = ['$state'];

/\* @ngInject \*/

function RouterHelper($state) {

var hasOtherwise = false;

var service = {

configureStates: configureStates,

getStates: getStates

};

return service;

///////////////

function configureStates(states, otherwisePath) {

states.forEach(function(state) {

$stateProvider.state(state.state, state.config);

});

if (otherwisePath && !hasOtherwise) {

hasOtherwise = true;

$urlRouterProvider.otherwise(otherwisePath);

}

}

function getStates() { return $state.get(); }

}

}

###### [Style [Y271](https://github.com/johnpapa/angular-styleguide/blob/master/a1/i18n/zh-CN.md#style-y271)]

* Define routes for views in the module where they exist，Each module should contain the routes for the views in the module.

为什么？：每个模块应该是独立的。

为什么？：当删除或增加一个模块时，应用程序只包含指向现存视图的路由。（也就是说删除模块和增加模块都需更新路由）

为什么？：这使得可以在不关心孤立的路由时很方便地启用或禁用应用程序的某些部分。

[**返回顶部**](https://github.com/johnpapa/angular-styleguide/blob/master/a1/i18n/zh-CN.md#%E7%9B%AE%E5%BD%95)

## 任务自动化

用[Gulp](http://gulpjs.com/)或者[Grunt](http://gruntjs.com/)来创建自动化任务。Gulp偏向于代码优先原则(code over configuration)而Grunt更倾向于配置优先原则(configuration over code)。我更倾向于使用gulp，因为gulp写起来比较简单。

可以在我的[Gulp Pluralsight course](http://jpapa.me/gulpps)了解更多gulp和自动化任务的信息

###### [Style [Y400](https://github.com/johnpapa/angular-styleguide/blob/master/a1/i18n/zh-CN.md#style-y400)]

* 用任务自动化在其它JavaScript文件之前列出所有模块的定义文件\*.module.js。

为什么？：Angular中，模块使用之前必须先注册。

为什么？：带有特殊规则的模块命名，例如\*.module.js，会让你很轻松地识别它们。

var clientApp = './src/client/app/';

// Always grab module files first

var files = [

clientApp + '\*\*/\*.module.js',

clientApp + '\*\*/\*.js'

];

[**返回顶部**](https://github.com/johnpapa/angular-styleguide/blob/master/a1/i18n/zh-CN.md#%E7%9B%AE%E5%BD%95)

## Filters

###### [Style [Y420](https://github.com/johnpapa/angular-styleguide/blob/master/a1/i18n/zh-CN.md#style-y420)]

* 避免使用filters扫描一个复杂对象的所有属性，应该用filters来筛选选择的属性。

为什么？：不恰当的使用会造成滥用并且会带来糟糕的性能问题，例如对一个复杂的对象使用过滤器。

[**返回顶部**](https://github.com/johnpapa/angular-styleguide/blob/master/a1/i18n/zh-CN.md#%E7%9B%AE%E5%BD%95)

## Angular文档

[Angular文档](https://docs.angularjs.org/api)。

## 贡献

先打开一个问题讨论潜在的变化和增加。如果你对这篇规范有任何疑惑，随时在仓库中提出问题。如果你发现了一个错字，创建一个pull request。这样做是为了保持内容的更新，使用github的原生功能通过问题和PR来帮助讲述这个故事，具体做法可以google一下。为什么？因为如果你有问题，其他人可能有同样的问题，你在这里可以学到如何贡献。

贡献代码到这个仓库就意味着你同意了本仓库的许可证内容

###过程 1. 在Github Issue中讨论这个问题。 2. 拉取一个pull request，引用这个问题，解释你做的修改和为什么要这样做。 3. pull request将会被进行评估，结果就是合并或是拒绝。

## 许可

* **tldr;** 如果可以的话，使用本规范的时候还是指明归属吧。

### Copyright

Copyright (c) 2014-2015 [John Papa](http://johnpapa.net/)

### (The MIT License)

Permission is hereby granted, free of charge, to any person obtaining a copy of this software and associated documentation files (the 'Software'), to deal in the Software without restriction, including without limitation the rights to use, copy, modify, merge, publish, distribute, sublicense, and/or sell copies of the Software, and to permit persons to whom the Software is furnished to do so, subject to the following conditions:

The above copyright notice and this permission notice shall be included in all copies or substantial portions of the Software.

THE SOFTWARE IS PROVIDED 'AS IS', WITHOUT WARRANTY OF ANY KIND, EXPRESS OR IMPLIED, INCLUDING BUT NOT LIMITED TO THE WARRANTIES OF MERCHANTABILITY, FITNESS FOR A PARTICULAR PURPOSE AND NONINFRINGEMENT. IN NO EVENT SHALL THE AUTHORS OR COPYRIGHT HOLDERS BE LIABLE FOR ANY CLAIM, DAMAGES OR OTHER LIABILITY, WHETHER IN AN ACTION OF CONTRACT, TORT OR OTHERWISE, ARISING FROM, OUT OF OR IN CONNECTION WITH THE SOFTWARE OR THE USE OR OTHER DEALINGS IN THE SOFTWARE.

[**返回顶部**](https://github.com/johnpapa/angular-styleguide/blob/master/a1/i18n/zh-CN.md#%E7%9B%AE%E5%BD%95)