

# AngularJS-Chinese-API

@西瓜橘子葡萄冰

Published  
with GitBook



## 目錄

介紹	0
官方模块 - Modules	1
auto	1.1
ng	1.2
ngAnimate	1.3
ngAria	1.4
ngCookies	1.5
ngMessageFormat	1.6
ngMessages	1.7
ngMock	1.8
ngMockE2E	1.9
ngResource	1.10
ngRoute	1.11
ngSanitize	1.12
ngTouch	1.13
公共方法 - Functions	2
ng.angular.module	2.1
ng.angular.bootstrap	2.2
ng.angular.element	2.3
ng.angular.copy	2.4
ng.angular.extend	2.5
ng.angular.merge	2.6
ng.angular.forEach	2.7
ng.angular.toJson	2.8
ng.angular.fromJson	2.9
ng.angular.injector	2.10
ng.angular.equals	2.11

ng.angular.isArray	2.12
ng.angular.isDate	2.13
ng.angular.isDefined	2.14
ng.angular.isElement	2.15
ng.angular.isFunction	2.16
ng.angular.isNumber	2.17
ng.angular.isObject	2.18
ng.angular.isString	2.19
ng.angular.isUndefined	2.20
ng.angular.reloadWithDebugInfo	2.21
ng.angular.lowercase	2.22
ng.angular.uppercase	2.23
ng.angular.bind	2.24
ng.angular.identity	2.25
ng.angular.noop	2.26
ng.angular.version	2.27
ngMock.angular.mock.dump	2.28
ngMock.angular.mock.inject	2.29
ngMock.angular.mock	2.30
ngMock.angular.mock.module	2.31
<b>服务 - Services</b>	<b>3</b>
ng.\$anchorScroll	3.1
ng.\$animate	3.2
ng.\$cacheFactory	3.3
ng.\$controller	3.4
ng.\$http	3.5
ng.\$locale	3.6
ng.\$location	3.7
ng.\$log	3.8
ng.\$parse	3.9

ng.\$q	3.10
ng.\$rootElement	3.11
ng.\$rootScope	3.12
ng.\$sce	3.13
ng.\$sceDelegate	3.14
ng.\$templateCache	3.15
ng.\$templateRequest	3.16
ng.\$timeout	3.17
ng.\$interval	3.18
ng.\$window	3.19
ng.\$document	3.20
指令 - Directives	4
ng.a	4.1
ng.ng-app	4.2
ng.ng-bind-html	4.3
ng.ng-bind-template	4.4
ng.ng-bind	4.5
ng.ng-non-bindable	4.6
ng.input	4.7
ng.form	4.8
ng.ng-form	4.9
ng.textarea	4.10
ng.ng-focus	4.11
ng.ng-blur	4.12
ng.ng-change	4.13
ng.ng-checked	4.14
ng.ng-disabled	4.15
ng.ng-readonly	4.16
ng.ng-selected	4.17
ng.ng-open	4.18

<a href="#">ng.ng-submit</a>	4.19
<a href="#">ng.ng-class</a>	4.20
<a href="#">ng.ng-class-even</a>	4.21
<a href="#">ng.ng-class-odd</a>	4.22
<a href="#">ng.ng-click</a>	4.23
<a href="#">ng.ng-dblclick</a>	4.24
<a href="#">ng.ng-cloak</a>	4.25
<a href="#">ng.ng-controller</a>	4.26
<a href="#">ng.ng-copy</a>	4.27
<a href="#">ng.ng-csp</a>	4.28
<a href="#">ng.ng-cut</a>	4.29
<a href="#">ng.ng-paste</a>	4.30
<a href="#">ng.ng-hide</a>	4.31
<a href="#">ng.ng-show</a>	4.32
<a href="#">ng.ng-href</a>	4.33
<a href="#">ng.ng-src</a>	4.34
<a href="#">ng.ng-srcset</a>	4.35
<a href="#">ng.ng-if</a>	4.36
<a href="#">ng.ng-switch</a>	4.37
<a href="#">ng.ng-include</a>	4.38
<a href="#">ng.ng-init</a>	4.39
<a href="#">ng.ng-jq</a>	4.40
<a href="#">ng.ng-keydown</a>	4.41
<a href="#">ng.ng-keypress</a>	4.42
<a href="#">ng.ng-keyup</a>	4.43
<a href="#">ng.ng-list</a>	4.44
<a href="#">ng.ng-model-options</a>	4.45
<a href="#">ng.ng-model</a>	4.46
<a href="#">ng.ng-mousedown</a>	4.47
<a href="#">ng.ng-mouseenter</a>	4.48

ng.ng-mouseleave	4.49
ng.ng-mousemove	4.50
ng.ng-mouseover	4.51
ng.ng-mouseup	4.52
ng.select	4.53
ng.ng-options	4.54
ng.ng-value	4.55
ng.ng-pluralize	4.56
ng.ng-repeat	4.57
ng.ng-style	4.58
ng.ng-transclude	4.59
ng.script	4.60
ngMessage.ng-message-exp	4.61
ngMessage.ng-message	4.62
ngMessage.ng-messages-include	4.63
ngMessage.ng-messages	4.64
ngRoute.ng-view	4.65
ngTouch.ng-click	4.66
ngTouch.ng-swipe-left	4.67
ngTouch.ng-swipe-right	4.68
过滤器 - Filters	5
ng.currency	5.1
ng.date	5.2
ng.filter	5.3
ng.json	5.4
ng.lowercase	5.5
ng.uppercase	5.6
ng.number	5.7
ng.limitTo	5.8
ng.orderBy	5.9

<a href="#">ngSanitize.linky</a>	5.10
<a href="#">心得</a>	6
<a href="#">关于模块</a>	6.1
<a href="#">关于服务</a>	6.2
<a href="#">关于指令</a>	6.3
<a href="#">关于过滤器</a>	6.4

[blog](#)

[gitbook 地址](#) 可以下载完里先看哦。

为了学习英语和 angular 诶嘿嘿。混入了不同版本的 API, 有时间我会抖到更新到最新版 */(\w+)/*

weibo: [@西瓜橘子葡萄冰](#)

twitter: [@fsm114000](#)

mail: fsm114000@163.com

注意:

- 出现在标签或者函数参数中的 `[ ]` 代表可选, 并不代表真正书写时需要书写它

进度 ★☆

```
| - 指令/
| - ★ ng.a@1.4.7
| - ★ ng.form@1.4.7
| - ★ ng.ng-form@1.4.7
| - ☆ ng.input@1.4.7
| - ★ ng.ng-app@1.4.7
| - ★ ng.ng-bind@1.4.7
| - ★ ng.ng-bind-html@1.4.7
| - ★ ng.ng-bind-template@1.4.7
| - ★ ng.ng-blur@1.4.7
| - ★ ng.ng-change@1.4.7
| - ★ ng.ng-checked@1.4.7
| - ★ ng.ng-class@1.4.7
| - ★ ng.ng-class-even@1.4.7
| - ★ ng.ng-class-odd@1.4.7
| - ★ ng.ng-click@1.4.7
| - ★ ng.ng-cloak@1.4.7
| - ★ ng.ng-controller@1.4.7
| - ★ ng.ng-copy@1.4.7
| - ★ ng.ng-csp@1.4.7
| - ★ ng.ng-cut@1.4.7
| - ★ ng.ng-dblclick@1.4.7
| - ★ ng.ng-disabled@1.4.7
| - ★ ng.ng-focus@1.4.7
```



- | - ★ ng.ng-form@1.4.7
- | - ★ ng.ng-hide@1.4.7
- | - ★ ng.ng-href@1.4.7
- | - ★ ng.ng-if@1.4.7
- | - ★ ng.ng-include@1.4.7
- | - ★ ng.ng-init@1.4.7
- | - ★ ng.ng-jq@1.4.7
- | - ★ ng.ng-keydown@1.4.7
- | - ★ ng.ng-keypress@1.4.7
- | - ★ ng.ng-keyup@1.4.7
- | - ★ ng.ng-list@1.4.7
- | - ☆ ng.ng-model-options@1.4.7
- | - ★ ng.ng-model@1.4.7
- | - ★ ng.ng-mousedown@1.4.7
- | - ★ ng.ng-mouseenter@1.4.7
- | - ★ ng.ng-mouseleave@1.4.7
- | - ★ ng.ng-mousemove@1.4.7
- | - ★ ng.ng-mouseover@1.4.7
- | - ★ ng.ng-mouseup@1.4.7
- | - ★ ng.ng-non-bindable@1.4.7
- | - ★ ng.ng-open@1.4.7
- | - ☆ ng.ng-options@1.4.7
- | - ★ ng.ng-paste@1.4.7
- | - ☆ ng.ng-pluralize@1.4.7
- | - ★ ng.ng-readonly@1.4.7
- | - ★ ng.ng-repeat@1.4.7
- | - ★ ng.ng-selected@1.4.7
- | - ★ ng.ng-show@1.4.7
- | - ★ ng.ng-src@1.4.7
- | - ★ ng.ng-srcset@1.4.7
- | - ★ ng.ng-style@1.4.7
- | - ★ ng.ng-submit@1.4.7
- | - ☆ ng.ng-switch@1.4.7
- | - ★ ng.ng-transclude@1.4.7
- | - ★ ng.ng-value@1.4.7
- | - ★ ng.script@1.4.7
- | - ☆ ng.select@1.4.7
- | - ☆ ng.textarea@1.4.7
- | - ☆ ngMessage.ng-message-exp@1.4.7
- | - ☆ ngMessage.ng-message@1.4.7

```
| - ☆ ngMessage.ng-messages@1.4.7
| - ☆ ngMessage.ng-messages-include@1.4.7
| - ☆ ngRoute.ng-view@1.4.7
| - ☆ ngTouch.ng-click@1.4.7
| - ☆ ngTouch.ng-swipe-left@1.4.7
| - ☆ ngTouch.ng-swipe-right@1.4.7
...
|- 过滤器/ 撒花! ☆, °*: . ☆ \ ( ▽ ) / $: * . ° ☆* 。
| - ☆ ng.currency@1.4.7
| - ☆ ng.date@1.4.7
| - ☆ ng.filter@1.4.7
| - ☆ ng.json@1.4.7
| - ☆ ng.limitTo@1.4.7
| - ☆ ng.lowercase@1.4.7
| - ☆ ng.number@1.4.7
| - ☆ ng.orderBy@1.4.7
| - ☆ ng.uppercase@1.4.7
| - ☆ ngSanitize.linky@1.4.7
...
|- 公共函数/ 撒花! ☆, °*: . ☆ \ ( ▽ ) / $: * . ° ☆* 。
| - ☆ ng.angular.bind@1.4.7
| - ☆ ng.angular.bootstrap@1.4.7
| - ☆ ng.angular.copy@1.4.7
| - ☆ ng.angular.element@1.4.7
| - ☆ ng.angular.equals@1.4.7
| - ☆ ng.angular.extend@1.4.7
| - ☆ ng.angular.forEach@1.4.7
| - ☆ ng.angular.fromJson@1.4.7
| - ☆ ng.angular.identity@1.4.7
| - ☆ ng.angular.injector@1.4.7
| - ☆ ng.angular.isArray@1.4.7
| - ☆ ng.angular.isDate@1.4.7
| - ☆ ng.angular.isDefined@1.4.7
| - ☆ ng.angular.isElement@1.4.7
| - ☆ ng.angular.isFunction@1.4.7
| - ☆ ng.angular.isNumber@1.4.7
| - ☆ ng.angular.isObject@1.4.7
| - ☆ ng.angular.isString@1.4.7
| - ☆ ng.angular.isUndefined@1.4.7
| - ☆ ng.angular.lowercase@1.4.7
```

```
| - ★ ng.angular.merge@1.4.7
| - ★ ng.angular.module@1.4.7
| - ★ ng.angular.noop@1.4.7
| - ★ ng.angular.reloadWithDebugInfo@1.4.7
| - ★ ng.angular.toJson@1.4.7
| - ★ ng.angular.uppercase@1.4.7
| - ★ ng.angular.version@1.4.7
| - ★ ngMock.angular.mock.dump@1.4.7
| - ★ ngMock.angular.mock.inject@1.4.7
| - ★ ngMock.angular.mock@1.4.7
| - ★ ngMock.angular.mock.module@1.4.7
...
| - 模块/
|   | - ☆ auto@1.4.7
|   | - ☆ main@1.4.7
|   | - ☆ ng@1.4.7
|   | - ☆ ngAnimate@1.4.7
|   | - ☆ ngAria@1.4.7
|   | - ☆ ngCookies@1.4.7
|   | - ☆ ngMessageFormat@1.4.7
|   | - ☆ ngMessages@1.4.7
|   | - ☆ ngMock@1.4.7
|   | - ☆ ngMockE2E@1.4.7
|   | - ☆ ngResource@1.4.7
|   | - ☆ ngRoute@1.4.7
|   | - ☆ ngSanitize@1.4.7
|   | - ☆ ngTouch@1.4.7
|   ...
| - 服务/
|   | - ★ ng.$anchorScroll@1.4.3
|   | - ★ ng.$animate@1.4.3
|   | - ★ ng.$cacheFactory@1.4.3
|   | - ★ ng.$controller@1.4.3
|   | - ★ ng.$document@1.4.3
|   | - ★ ng.$filter@1.4.3
|   | - ☆ ng.$http@1.4.3
|   | - ★ ng.$interval@1.4.3
|   | - ★ ng.$locale@1.4.3
|   | - ★ ng.$location@1.4.3
|   | - ★ ng.$log@1.4.3
```

```
| - ★ ng.$parse@1.4.3
| - ★ ng.$q@1.4.3
| - ★ ng.$rootElement@1.4.3
| - ★ ng.$rootScope@1.4.3
| - ★ ng.$sce@1.4.3
| - ★ ng.$sceDelegate@1.4.3
| - ★ ng.$templateCache@1.4.3
| - ★ ng.$templateRequest@1.4.3
| - ★ ng.$timeout@1.4.3
| - ★ ng.$window@1.4.3
...
| - 类型/
...
```

# Angular\_1.4.3 API 文档

欢迎来到 AngularJS API 文档网页。这些网页包含了 AngularJS 1.4.3 版本的一些参考资料。

在一个 AngularJS 的应用中，包含了各种组件的文件被组织到不同的模块中。这些组件是 directives, services, filters, providers, 模板, 全局APIs和测试模块。

Angular 前缀 \$ 和 \$\$:

为了防止与你的代码发生命名冲突，Angular 公共对象的前缀名以 `\$` 开头，而私有对象的前缀名以 `\$\$` 开头。请尽量避免在你的代码中使用 `\$` 或 `\$\$` 作为

## Angular 模块

### ng (核心模块)

这是默认提供的模块，并包含了 AngularJS 的核心组件。

Directives	这是核心的指令(directive)集, 你可以将其应用到你的模板代码中来建立一个 AngularJS 的应用。 其中包含了: <code>ngClick</code> , <code>ngInclude</code> , <code>ngRepeat</code> 等等
Services / Factories	这是可以供你的应用的 DI使用的核心的服务(service)集 其中包含了: <code>\$compile</code> , <code>\$http</code> , <code>\$location</code> , 等等
Filters	在模板数据被指令和表达式渲染之前, ng模块中的核心过滤器 (filter) 会对其做出一些转换或改变。 其中包含了: <code>filter</code> , <code>date</code> , <code>currency</code> , <code>lowercase</code> , <code>uppercase</code> , 等等
全局 APIs	核心的全局API函数被绑定到 <code>angular</code> 对象上。这些核心的函数使有助于你在你的应用中使用底层的 Javascript 操作。 其中包含了: <code>angular.copy()</code> , <code>angular.equals()</code> , <code>angular.element()</code> , 等等...

## ngRoute

ngRoute 可以使你可以在你的应用实现 URL 路由 ngRoute 模块可以通过 hashbang 和 HTML5 pushState 两种方式支持 URL 管理。

引入 angular-route.js 文件，并在你的应用是设置一个 ngRoute 依赖

Services / Factories	下面的服务是用来管理路由的： <ul style="list-style-type: none"> <li>- \$routeParams 用来访问 URL 当前的查询字符串值</li> <li>- \$route 用来获取当前被访问路由的详细信息。</li> <li>- \$routeProvider 用来为应用注册路由</li> </ul>
Directives	ngView 指令将会在页面中展示当前路由的模板。

## ngAnimate

使用 ngAnimate 就可以在你的应用中实现动画效果。当ngAnimate 被引入时，许多核心的 ng 指令都会 在你的应用中提供动画钩子函数。动画可以使用 CSS transition / animations 或者 Javascript回调函数来定义

引入 angular-animate.js 文件，并在你的应用是设置一个 ngAnimate 依赖

Services / Factories	在你的指令代码中用 \$animate 来触发动画操作
以 CSS 为基础的动画	在 Angular 中，使用 ngAnimate 的 CSS 命名规则来引用 CSS 的 transition 或关键帧动画。一次定义，动画就总是会被 HTML模板代码中引用的 CSS 类触发。
以 JS 为基础的动画	调用 module.animation() 方法来注册一个 Javascript 动画，一次注册，动画就总是会被 HTML模板代码中引用的 CSS 类触发。

## ngAria 可能不是这样的凹

使用 ngAria 注入公共的可访问属性到指令中并提升残疾人用户的体验

引入 angular-aria.js 文件，并在你的应用是设置一个 ngAria 依赖

Services	\$aria 服务包含了辅助方法来申请 ARIA 属性值为 HTML.\$ariaProvider 用于配置ARIA的属性。
----------	--

## ngResource

当你需要使用一个 REST API 来获取和发送数据时，你可以使用

`ngResource`

引入 `angular-resource.js` 文件，并在你的应用是设置一个 `ngResource` 依赖

Services / Factories	<code>\$resource</code> 服务用来定义 使用 REST API 通信的 RESTful 对象
----------------------	---

## ngCookies

在你的应用中你可以使用 `ngCookies` 模块处理 cookie 的管理。

引入 `angular-cookies.js` 文件，并在你的应用是设置一个 `ngCookies` 依赖

Services / Factories	下面的服务可用于cookie 的管理: <ul style="list-style-type: none"><li>- <code>\$cookie</code> 服务是一个浏览器 cookie 的便捷的封装，可以用来存储一些简单的数据</li><li>- <code>\$cookieStore</code> 用于序列化存储更为复杂的数据。</li></ul>
----------------------	---

## ngTouch

当你为手机浏览器或设备开发时可以使用 `ngTouch` 模块。

引入 `angular-touch.js` 文件，并在你的应用是设置一个 `ngTouch` 依赖

Services / Factories	<code>\$swipe</code> 服务用来注册和处理移动端的 DOM 事件
Directives	<code>ngTouch</code> 中有很多指令可以模拟移动端的 DOM 事件。

## ngSanitize

在你的应用中，`ngSanitize` 可以帮你安全的解析和操纵 HTML 数据

引入 `angular-sanitize.js` 文件，并在你的应用是设置一个 `ngSanitize` 依赖

Services / Factories	<code>\$sanitize</code> 服务是一个用来清理危险的HTML代码的快速且便利的途径。
Filters	<code>linky</code> 过滤器用于，在提供的字符串内将 URL 转化成 HTML 链接

## ngMock

在你的单元测试中使用 `ngMock` 注入并测试 `module` , `factory` , `service` , `provider`

引入 `angular-mocks.js` 文件，来测试运行

Services / Factories	<code>ngMock</code> 会以同步的方式拓展许多核心服务的行为来使它们拥有良好的测试性并易于管理 举一些例子: <code>\$timeout</code> , <code>\$interval</code> , <code>\$log</code> , <code>\$httpBackend</code> , 等等
Global APIs	更有多种帮助性的函数来注入并测试模块在单元测试代码中 例如 <code>inject()</code> , <code>module()</code> , <code>dump()</code> , 等等































## angular.module

- Angular@1.4.7
- `ng` 模块中的函数

`angular.module` 是一个用于创建，注册和检索 Angular 模块的全局方法。一个应用中所需要使用的所有模块（angular 核心或第三方）都需要使用这种机制注册。

传入一个参数是检索一个存在的 Angular 模块（`angular.Module`），传入多个参数则是创建一个新的 Angular 模块。

## 模块

模块是服务（services），指令（directives），控制器（controllers），过滤器（filters）和配置信息的集合。`angular.module` 被用于配置 `$injector`

```
// 创建一个新的模块
var myModule = angular.module('myModule', []);

// 注册一个新服务
// 译：创建服务有很多种方式 value service factory provider 等
myModule.value('appName', 'MyCoolApp');

// 将现有的服务配置到初始化部分中。
myModule.config(['$locationProvider', function($locationProvider) {
    // 配置现有的 providers
    $locationProvider.hashPrefix('!');
}]);
```

你可以像下面这样创建注射器，并加载模块：

```
var injector = angular.injector(['ng', 'myModule'])
```

然而更多可能是你仅仅会使用 `ngApp` 或者 `angular.bootstrap` 来简化这个过程。

## 用法

```
angular.module(name, [requires], [configFn])
```

### 参数

参数	形式	详细
name	<code>string</code>	创建或检索的模块的名字。
requires (可选)	<code>!Array.&lt;string&gt;=</code>	如果给定则新建一个模块，没有传入则检索模块做进一步配置。
configFn (可选)	<code>Function=</code>	模块的可选配置函数。和 <code>Module#config()</code> 相同。

### 返回

`module` 带有 `angular.module` api 的新模块

## angular.bootstrap

- Angular@1.4.7
- `ng` 模块中的函数

使用这个函数来手动启动 angular 应用。

请看开发向导中的 Bootstrap

注意：基于 Protractor 的 E2E 测试不能使用这个函数来手动启动应用。必须使用 `ngApp` 指令。Angular 会检测重复加载，并只允许首次加载的脚本启动，而为之加载的脚本在控制台提出警告。这会防止在应用中有多个 Angular 实例一同试图工作在 DOM 上而发生奇怪现象。

```
<!doctype html>
<html>
<body>
<div ng-controller="WelcomeController">
  {{greeting}}
</div>

<script src="angular.js"></script>
<script>
  var app = angular.module('demo', [])
  .controller('WelcomeController', function($scope) {
    $scope.greeting = 'Welcome!';
  });
  angular.bootstrap(document, ['demo']);
</script>
</body>
</html>
```

## 用法

```
angular.bootstrap(element, [modules], [config])
```

## 参数



参数	形式	详细
element	<code>DOMElement</code>	代表 angular 应用的根的 DOM 元素
modules (可选)	<code>Array&lt;String/Function/Array&gt;=</code>	一个想要在空中引入的模块的数组。数组中的每一项应该是一个预设模块的名字，或者是一个 (DI 注解，译：依赖注入) 函数，它将被注射器当作一个配置块调用。请看 <code>angular.module</code>
config (可选)	<code>Object</code>	定义应用配置项的对象。支持一下项目： - <code>strictDi</code> - 是应用的自动函数注解失效，有助于在压缩的代码中找到错误。默认为 <code>false</code>

## 返回

`auto.$injector` 返回为这个应用新创建的注射器。

## angular.element

- Angular@1.4.7
- `ng` 模块中的函数

将一个原生的 DOM 元素或者 HTML 字符串包装成一个 jQuery 元素。

如果引入了 jQuery。 `angular.element` 则将是 jQuery 函数的别名。如果没有引入 jQuery，则 `angular.element` 则表示 Angular 内建的 jQuery 的子级，叫做“jQuery 精简版”或者“jqLite”。

jqLite 是 jQuery 的一个精简的，并且API兼容的自己，它允许 Angular 使用跨浏览器。jqLite 的目标是尽可能小影响代码，所以仅实现最常用功能。

如果使用 jQuery，需要确认它在 `angular.js` 文件之前载入。

注意：在 Angular 中，所有的元素总是被 `jQuery` 或 `jqLite` 封装过的，它们不会是原生的 DOM。

## Angular's jqLite

jqLite 仅提供以下的 jQuery 方法：

- `addClass()`
- `after()`
- `append()`
- `attr()` - 不支持传入参数
- `bind()` - 不支持命名空间，选择器和事件数据
- `children()` - 不支持选择器
- `clone()`
- `contents()`
- `css()` - 仅仅取回行内样式，不会调用 `getComputedStyle()` 方法。当作设置元素样式的时候，不会将数值转化为字符串和添加 px。
- `data()`
- `detach()`
- `empty()`

- `eq()`
- `find()` - 限制到只可以用标签名查找
- `hasClass()`
- `html()`
- `next()` - 不支持选择器
- `on()` - 不支持命名空间, 选择器和事件数据
- `off()` - 不支持命名空间, 选择器和事件对象
- `one()` - 不支持命名空间, 选择器
- `parent()` - 不支持选择器
- `prepend()`
- `prop()`
- `ready()`
- `remove()`
- `removeAttr()`
- `removeClass()`
- `removeData()`
- `replaceWith()`
- `text()`
- `toggleClass()`
- `triggerHandler()` - 传入一个虚拟事件对象来处理。
- `unbind()` - 不支持命名空间, 选择器和事件对象
- `val()`
- `wrap()`

## jQuery/jqLite 附加功能

Angular 还为 jQuery 和 jqLite 提供了一下的附加方法。

### 事件

- `$destroy` - AngularJS 拦截了所有的 qLite/jQuery 的摧毁 DOM 的 API 并且在所有 DOM 节点被删除时触发这个事件。这个事件可以用来在 DOM 元素被移除之前清除绑定在它上的任何第三方绑定。

### 方法

- `controller(name)` - 检索当前元素或其父级的控制器，默认检索与 `ngController` 指令关联的控制器。如果 `name` 为驼峰形式的指令名，则检索这个指令的控制器（例如：`ngModel`）。
- `injector()` - 检索当前元素或其父级的注射器。
- `scope()` - 检索当前元素或其父级的作用域，请求调试数据（Requires Debug Data）会被启用。
- `isolateScope()` - 如果当前元素有一个独立的作用域时，可以使用这个方法检索。这个获取方法只能用在包含有独立作用域指令的元素上。在此元素调用 `scope()` 只返回原有的非独立作用域。请求调试数据会被启用。
- `inheritedData()` - 与 `data()` 用法相同，但是除非找到一个值，或者到达顶层父元素，否则不会中断。

## 用法

```
angular.element(element)
```

## 参数

参数	形式	详细
<code>element</code>	<code>string</code> <code>DOMElement</code>	需要被封装进 jQuery 的 HTML 字符串或者 DOM 元素。

## 返回

`Object` jQuery 对象.

## angular.copy

- Angular@1.4.7
- `ng` 模块中的函数

深度复制一个源对象或者数组。

- 如果没给定目标（`destination`），则会创建这个对象或者数组的拷贝。
- 如果提供了目标，它自身的元素（如果是数组）或者属性（如果是对象）会被删除，之后会从复制源处复制的所有元素（或属性）
- 如果复制源不是一个对象或者数组（`null` 和 `undefined`），则返回复制源。
- 如果源和目标是相同的，则会抛出一个例外。

## 用法

```
angular.copy(source, [destination])
```

## 参数

参数	形式	详细
source	*	用来拷贝的源，可以使任何形式。包括原函数， <code>null</code> ， <code>undefined</code> 。
destination（可选）	Object Array	源将被复制到的目标中，如果提供了这个参数，请确保它与源的类型保持相同。

## 返回

\* 复制的源或者是更新传入了的目标。

## angular.extend

- Angular@1.4.7
- `ng` 模块中的函数

使用复制源对象的枚举属性到 `dst` 对象上的方式来扩展目标对象（`dst`），你可以指定多个源对象，如果你想保留原来的对象，你可以传入一个空对象作为目标对象：  
`var object = angular.extend({}, object1, object2)`

注意：`angular.extend` 不支持递归合并（深拷贝）。可以使用 `angular.merge` 实现。

## 用法

```
angular.extend(dst, src)
```

## 参数

参数	形式	详细
<code>dst</code>	<code>object</code>	目标对象。
<code>src</code>	<code>object</code>	源对象。

## 返回

`object` `dst`.

## angular.merge

- Angular@1.4.7
- `ng` 模块中的函数

通过在源对象上枚举属性并复制到自己身上来达成深度扩展目标对象 `dst`。你可以指定多个源对象，如果你想保留原始对象，你可以传入一个空对象作为目标点

```
var object = angular.merge({}, object1, object2)
```

不像 `extend()` 那样，`merge()` 会递归地深入到源对象的对象属性中，进行一个深度的复制。

## 用法

```
angular.merge(dst, src)
```

## 参数

参数	形式	详细
<code>dst</code>	<code>object</code>	目标对象。
<code>src</code>	<code>object</code>	源对象。

## 返回

`object dst.`

## angular.forEach

- Angular@1.4.7
- `ng` 模块中的函数

为对象或数组中的每一个项调用一次迭代函数。迭代函数被 `iterator(value, key, obj)` 调用，其中 `value` 为对象的一个属性的值，或者数组中的一个元素，`key` 是这个对象属性的键名，或者为数组元素的索引，`obj` 则为对象或数组本身，可以为这个函数指定一个执行上下文环境。

值得注意的是 `.forEach` 不会遍历继承的属性，因为它会使用 `hasOwnProperty` 方法进行过滤。

不像 ES262 标准的 `Array.prototype.forEach`，会在 `obj` 中的值有 `undefined` 或是 `null` 抛出类型错误，我们只会直接返回这些值。

```
var values = {name: 'misko', gender: 'male'};
var log = [];
angular.forEach(values, function(value, key) {
    this.push(key + ': ' + value);
}, log);
expect(log).toEqual(['name: misko', 'gender: male']);
```

## 用法

```
angular.forEach(obj, iterator, [context])
```

## 参数

参数	形式	详细
<code>obj</code>	<code>Object</code> <code>Array</code>	迭代的对象。
<code>iterator</code>	<code>function</code>	迭代函数
<code>context</code> (可选)	<code>Object</code>	这个对象会成为迭代器函数的上下文（即 <code>this</code> 的指向）

## 返回



object / array obj

## angular.fromJson

- Angular@1.4.7
- `ng` 模块中的函数

将输入序列化成 JSON 形式的字符串。以 `$$` 开头的属性将被略过，因为这代表它是 angular 内部使用的标记。

## 用法

```
angular.toJson(obj, pretty)
```

## 参数

参数	形式	详细
obj	string array Date object number	需要序列化成 JSON 的输入
pretty (可选)	boolean number	如果设置为 <code>true</code> ，输出的 JSON 会包含换行符和空白。 如果设置为整数，输出的 JSON 将会在每个缩进处使用给定个数的空白。（默认为2）

## 返回

`string` `undefined` 表示 `obj` 的JSON形式的字符串。

## angular.fromJson

- Angular@1.4.7
- `ng` 模块中的函数

反序列化 JSON 字符串

### 用法

```
angular.fromJson(json)
```

### 参数

参数	形式	详细
json	<code>string</code>	需要反序列化的 JSON 字符串

### 返回

`object` `array` `string` `number` 反序列化的结果

## angular.injector

- Angular@1.4.7
- `ng` 模块中的函数

创建一个可用于查找服务和依赖注入的注射器对象。（请看开发向导中的依赖注入）

## 用法

```
angular.injector(modules, [strictDi])
```

## 参数

参数	形式	详细
modules	<code>Array. &lt;string/Function&gt;</code>	模块函数或其代称的列表，请看 <code>angular.module</code> 。 <code>ng</code> 模块必须明确地添加进来。
strictDi（可选）	<code>boolean</code>	注射器是否使用严格注入模式，这种模式不允许使用参数名注解自身推断的方式（默认: <code>false</code> ）

## 返回

`injector` 注射器对象，详细请看 `$injector`

## 例子

典型用法：

```
// 创建一个注射器
var $injector = angular.injector(['ng']);

// 使用注射器启动你的应用
// 使用类型推断方式自动注入参数，或者使用静默注入
$injector.invoke(function($rootScope, $compile, $document) {
    $compile($document)($rootScope);
    $rootScope.$digest();
});
```

有时候你希望从外部的 Angular 访问当前运行的 Angular 应用的注射器。或者，你希望在应用启动后注入或者编译一些标记装饰器。你可以使用在 JQuery/jqLite 元素上拓展的 `injector()` 方法来实现。请看 `angular.element`。

在下面的例子中，使用了 JQuery 将一个包含了 `ng-controller` 指令的新的 HTML 块插入了 `body` 的末尾。然后我们编译并将其连接到当前的 Angular 作用域。

```
var $div = $('<div ng-controller="MyCtrl">{{content.label}}</div>');
$(document.body).append($div);

angular.element(document).injector().invoke(function($compile) {
    var scope = angular.element($div).scope();
    $compile($div)(scope);
});
```

## angular.equals

- Angular@1.4.7
- `ng` 模块中的函数

确定两个对象或者两个值是否相等。支持的值类型，正则表达式，数组，对象。

如果两个对象或是两个数组会被认定为相等，那它们至少要满足下面的某一个条件的值为 `true`：

- 两个对象或数组通过 `===` 判定。
- 两个对象或数组是同一类型并且他们的属性通过 `angular.equal` 方法验证也相等。
- 两个值为 NaN（在 JavaScript 中，`NaN == NaN => false`，但是我们认定两个 NaN 相等）
- 两个值代表着同样的表达式（在 JavaScript 中，`/abc/ == /abc/ => false`，但是在他们的文字表述一致的情况下我们认定两个正则表达式相等）

在对一个属性进行校验的时候，函数形式的属性和以 `$` 开头的属性名的属性将忽略判断。

作用域和 `DOMWindow` 对象只能使用判定（`===`）来校验相等

## 用法

```
angular.equals(o1, o2)
```

## 参数

参数	形式	详细
<code>o1</code>	<code>*</code>	需要判断相等的对象或值。
<code>o2</code>	<code>*</code>	需要判断相等的对象或值。

## 返回

`boolean` 如果相等返回 `true`

## angular.isArray

- Angular@1.4.7
- `ng` 模块中的函数

判断是否为数组。

## 用法

```
angular.isArray(value)
```

## 参数

参数	形式	详细
value	*	检测值

## 返回

`boolean` 如果 `value` 为数组则返回 `true`

## angular.isDate

- Angular@1.4.7
- `ng` 模块中的函数

判断是否为日期对象（Date）。

## 用法

```
angular.isDate(value)
```

## 参数

参数	形式	详细
value	<code>*</code>	检测值

## 返回

`boolean` 如果 `value` 为日期对象则返回 `true`



## angular.isDefined

- Angular@1.4.7
- `ng` 模块中的函数

判断值是否定义了。

## 用法

```
angular.isDefined(value)
```

## 参数

参数	形式	详细
value	*	检测值

## 返回

boolean 如果 value 定义了返回 true

## angular.isElement

- Angular@1.4.7
- `ng` 模块中的函数

判断值是否是一个 DOM 元素（或者被封装的 JQuery 元素）。

## 用法

```
angular.isElement(value)
```

## 参数

参数	形式	详细
value	*	检测值

## 返回

`boolean` 如果 `value` 是一个 DOM 元素（或者被封装的 JQuery 元素），则返回 `true`

## angular.isFunction

- Angular@1.4.7
- `ng` 模块中的函数

判断值是否是一个函数。

## 用法

```
angular.isFunction(value)
```

## 参数

参数	形式	详细
value	<code>*</code>	检测值

## 返回

`boolean` 如果 `value` 是一个函数，则返回 `true`

## angular.isNumber

- Angular@1.4.7
- `ng` 模块中的函数

判断值是否是一个数值。

包括‘特殊’的数值 `NaN` , `+Infinity` , `-Infinity`

如果你希望排除这些, 可以使用原生的 `isFinite` 方法。

## 用法

```
angular.isNumber(value)
```

## 参数

参数	形式	详细
value	<code>*</code>	检测值

## 返回

`boolean` 如果 `value` 是一个数值, 则返回 `true`

## angular.isFunction

- Angular@1.4.7
- `ng` 模块中的函数

判断值是否是一个对象。并不像 JavaScript 中 `typeof` 显示的那样。 `null` 并不会被当做对象。 并且要注意 JavaScript 数组也是对象。

## 用法

```
angular.isObject(value)
```

## 参数

参数	形式	详细
value	*	检测值

## 返回

`boolean` 如果 `value` 是一个对象，并且不是 `null`，则返回 `true`

## angular.isString

- Angular@1.4.7
- `ng` 模块中的函数

判断值是否是一个字符串。

### 用法

```
angular.isString(value)
```

### 参数

参数	形式	详细
value	<code>*</code>	检测值

### 返回

`boolean` 如果 `value` 是一个字符串，则返回 `true`

## angular.isUndefined

- Angular@1.4.7
- `ng` 模块中的函数

判断值是否没有被定义了。

## 用法

```
angular.isUndefined(value)
```

## 参数

参数	形式	详细
value	<code>*</code>	检测值

## 返回

`boolean` 如果 `value` 未定义返回 `true`

## angular.reloadWithDebugInfo

- Angular@1.4.7
- `ng` 模块中的函数

使用这个函数来重新加载当前的应用，并打开调试信息。它的优先级高于

`$compileProvider.debugInfoEnabled(false)`

[查看](#) `$compileProvider` [服务](#) [查看更多](#)



## angular.lowercase

- Angular@1.4.7
- `ng` 模块中的函数

将指定的字符串转化为小写。

### 用法

```
angular.lowercase(string)
```

### 参数

参数	形式	详细
string	*	需要转化为小写形式的字符串

### 返回

`boolean` 小写形式的字符串。

## angular.uppercase

- Angular@1.4.7
- `ng` 模块中的函数

将指定的字符串转化为大写。

### 用法

```
angular.uppercase(string)
```

### 参数

参数	形式	详细
string	*	需要转化为大写形式的字符串

### 返回

`boolean` 大写形式的字符串。

## angular.bind

- Angular@1.4.7
- `ng` 模块中的函数

返回一个绑定在 `self` 上的 `fn` 函数（`self` 会成为 `fn` 中的 `this`）。你可以提供一些可选的参数（`args`）预先绑定到函数上。这个功能也被称为部分应用程序，或柯里话。

## 用法

```
angular.bind(self, fn, args)
```

## 参数

参数	形式	详细
<code>self</code>	<code>object</code>	<code>fn</code> 对应的上下文环境。
<code>fn</code>	<code>function()</code>	被绑定的函数
<code>args</code>	<code>*</code>	预先绑定到 <code>fn</code> 函数上的参数

## 返回

`function()` 使用指定绑定了的 `fn` 的封装

## angular.identity

- Angular@1.4.7
- `ng` 模块中的函数

一个返回它第一个参数的函数，这个函数用于函数式编程方式。

```
function transformer(transformationFn, value) {  
  return (transformationFn || angular.identity)(value);  
};
```

## 用法

```
angular.identity(value)
```

## 参数

参数	形式	详细
value	*	被返回的值

## 返回

\* 传入的值

## angular.noop

- Angular@1.4.7
- `ng` 模块中的函数

不执行任何操作的函数，这个函数用于函数式编程方式。

```
function foo(callback) {  
  var result = calculateResult();  
  (callback || angular.noop)(result);  
}
```

## 用法

```
angular.noop()
```

## angular.version

- Angular@1.4.7
- `ng` 模块中的对象

一个包含当前 AngularJS 版本信息的对象。

这个对象有下列属性。

- `full` – `{string}` – 完整版本字符串, 如 "0.9.18".
- `major` – `{number}` – 主版本号, 如 "0".
- `minor` – `{number}` – 次版本号, 如 "9".
- `dot` – `{number}` – 末版本号, 如 "18".
- `codeName` – `{string}` – 发布代号, 如 "jiggling-armfat".

## angular.mock.dump

- Angular@1.4.7
- `ngMock` 模块中的函数

NOTE: 这不是一个可注入的实例。仅仅是一个全局函数。

将 `angular` 对象（`scope`, `elements`,等）序列化成字符串的方法。用于调试。

这个方法也可以从 `window` 上得到来在控制台中展示对象。

## 用法

```
angular.mock.dump(object)
```

## 参数

参数	形式	详细
<code>object</code>	<code>*</code>	任何需要转化成字符串的对象

## 返回

`string` 参数序列化成的字符串。

## angular.mock.inject

- Angular@1.4.7
- ngMock 模块中的函数

注意: 为了方便访问, 这个方法也被暴露在了 `window` 上。注意: 这个方法只会在使用 `jasmine` 或者 `mocha` 运行测试时被声明。

这个注射函数将一个函数包裹进了一个可注入的函数。 `inject()` 在每个测试中都创建了一个 `$injector` 实例, 之后会用于解析引用。

### 解析引用 (下划线包装)

通常, 我们只会注入一个引用一次到 `beforeEach()` 块中, 并在多个 `it()` 中使用。为了达到这个目的我们必须将引用赋值给一个在 `describe()` 块中声明了的变量。我们想要拥有一个命名的引用, 这就会导致一个问题传入 `inject()` 的函数中的参数会将外部的变量覆盖掉。

为了解决这个问题, 被注入的参数可以使用下划线来封闭装饰, 当引用名被解析的时候下划线会被注射器忽略。

举个例子: 参数 `_myService_` 将会被当做 `myService` 引用来解析, 由于它可以在函数体内当做 `myService` 被使用, 所以之后我们可以将其赋值给外部作用域定义的变量。

```
// 在外部定义的引用变量
var myService;

// 将参数用下划线包装
beforeEach( inject( function(_myService_){
    myService = _myService_;
})));

// 在这个系列的测试中使用 myService
it('makes use of myService', function() {
    myService.doStuff();
});
```



同样可以查看 `angular.mock.module`

## 例子

一个典型的 jasmine 测试 Example of what a typical jasmine tests looks like with the inject method.

```
angular.module('myApplicationModule', [])
  .value('mode', 'app')
  .value('version', 'v1.0.1');

describe('MyApp', function() {

  // 你需要去载入你想要测试的模块
  // 默认只会加载 "ng" 模块
  beforeEach(module('myApplicationModule'));

  // inject() 用来注入所有给出函数的参数
  it('需要提供一个版本', inject(function(mode, version) {
    expect(version).toEqual('v1.0.1');
    expect(mode).toEqual('app');
  }));

  // inject 和 module 方法同样可以在 it 或 beforeEach 的内部使用
  it('需要覆盖版本并且测试新的版本被注入', function() {
    // module() 可以传入函数或字符串(模块的别名)
    module(function($provide) {
      $provide.value('version', 'overridden'); // 此处覆盖了版本
    });

    inject(function(version) {
      expect(version).toEqual('overridden');
    });
  });
});
```

## 用法

```
angular.mock.inject(fns)
```

## 参数

参数	形式	详细
fns	...Function	任意数量的需要使用注射器注入的函数。

## angular.mock

- Angular@1.4.7
- ngMock 模块中的对象

'angular-mocks.js' 的命名空间，包含了测试相关的代码。

## angular.mock.module

- Angular@1.4.7
- ngMock 模块中的函数

注意: 为了方便访问, 这个方法也被暴露在了 `window` 上。注意: 这个方法只会在使用 `jasmine` 或者 `mocha` 运行测试时被声明。

这个函数注册了一个模块配置代码。当 `inject` 创建注射器时, 这个函数会用来收集配置信息。

用法示例可查看 `inject` (译: `angular.mock.inject`)

## 用法

```
angular.mock.module(fns)
```

## 参数

参数	形式	详细
fns	string function() object	任何数量的, 表示为字符串的别名或匿名模块的初始化函数。这些模块用于配置注射器。 <code>ng</code> 和 <code>ngMock</code>

模块会自动加载。如果传入一个对象的字面量, 则在这个模块中它会被注册成 `value`。键名成为模块名, 键值成为返回值。|

## Angular\_1.4.3 API 服务篇 \$anchorScroll

- `$anchorScrollProvider`
- `ng` 模块中的服务

当被调用的时候，页面会滚动到与元素相关联的指定的 `hash` 处，或者滚动到当前 `$location.hash()` 处，是依照HTML5 spec 的规则制定的。

它当然也会监听 `$location.hash()` 并且无论锚点值何时变化，都会自动地滚动到相应的位置。当不需要它时，可以调

用 `$anchorScrollProvider.disableAutoScrolling()` 让它失效。

另外我们可以使用它的 `yOffset` 属性来指定一个垂直滚动偏移量（既可以是定值也可以是动态值）。

### 依赖

`$window` `$location` `$rootScope`

### 用法

```
$anchorScroll([hash]) ;
```

#### 参数

参数	形式	具体
<code>hash</code> (可选)	<code>string</code>	<code>hash</code> 将会指定元素滚动到的位置，如果省略参数，则将使用 <code>\$location.hash()</code> 作为默认值。

### 属性

`yOffset`

---	---
<div>number</div> <div>function()</div> <div>jqLite</div>	<p>如果设置了这个值，将会指定一个垂直的滚动的偏移量。这种场景经常用于在页面顶部有固定定位的元素, 如导航条，头部等（让出头部空间）。</p> <p><code>yoffset</code> 可以用多种途径指定:</p> <ul style="list-style-type: none"><li>- <b>number</b> : 一个固定的像素值可以使用（无单位）。</li><li>- <b>function</b> : 每次 <code>\$anchorScroll()</code> 执行时这个函数都会被调用，它必须返回一个代表位移的数字（无单位像素值）。</li><li><b>jqLite</b> : 一个jqLite/jQuery元素可以被指定为位移值。这个位移值会取页面的顶部到该元素底部的距离。</li></ul> <p>注意: 只有有元素的定位方式是固定定位时才会应该被纳入考虑之中。这个设置 在响应式的导航条/头部需要调整他们的高度亦或 根据视图来定位时很有用处。</p>

为了使 `yoffset` 正确地工作，滚动必须是在文档的根节点，而不是子节点。

## 例子

html

```
<div id="scrollArea" ng-controller="ScrollController">

  <a ng-click="gotoBottom()">Go to bottom</a>
  <a id="bottom"></a> You're at the bottom!

</div>
```

javascript

```
angular.module('anchorScrollExample', [])

.controller('ScrollController', ['$scope', '$location', '$anchorScroll',
function ($scope, $location, $anchorScroll) {

    $scope.gotoBottom = function() {

        // 将location.hash的值设置为
        // 你想要滚动到的元素的id
        $location.hash('bottom');

        // 调用 $anchorScroll()
        $anchorScroll();

    };
}]);
```

#### CSS

```
#scrollArea {

    height: 280px;
    overflow: auto;

}

#bottom {

    display: block;
    margin-top: 2000px;

}
```

下面的例子将说明如何使用一个垂直滚动偏移（指定了一个固定值）关于  
`$anchorScroll.yOffset` 的详情请看上方介绍

#### html

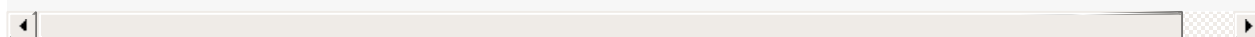
```
<div class="fixed-header" ng-controller="headerCtrl">

  <a href="" ng-click="gotoAnchor(x)" ng-repeat="x in [1,2,3,4,5]">
    Go to anchor {{x}}
  </a>

</div>
<div id="anchor{{x}}" class="anchor" ng-repeat="x in [1,2,3,4,5]">

  Anchor {{x}} of 5

</div>
```



javascript



```
angular.module('anchorScrollOffsetExample', [])
.run(['$anchorScroll', function($anchorScroll) {

    $anchorScroll.yOffset = 50;    // 总是滚动额外的50像素

}])
.controller('headerCtrl', ['$anchorScroll', '$location', '$scope',
    function ($anchorScroll, $location, $scope) {

        $scope.gotoAnchor = function(x) {

            var newHash = 'anchor' + x;
            if ($location.hash() !== newHash) {

                // 将$location.hash设置为`newHash` and
                // $anchorScroll也将自动滚到该处

                $location.hash('anchor' + x);

            } else {

                // 显式地调用 $anchorScroll()方法 ,
                // 因为 $location.hash 并没有改变
                $anchorScroll();

            }
        };
    }
]);
```

CSS

```
body {  
  
    padding-top: 50px;  
  
}  
  
.anchor {  
  
    border: 2px dashed DarkOrchid;  
    padding: 10px 10px 200px 10px;  
  
}  
  
.fixed-header {  
  
    background-color: rgba(0, 0, 0, 0.2);  
    height: 50px;  
    position: fixed;  
    top: 0; left: 0; right: 0;  
  
}  
  
.fixed-header > a {  
  
    display: inline-block;  
    margin: 5px 15px;  
  
}
```

## Angular\_1.4.3 API 服务篇 \$animate

- \$animateProvider
- ng 模块中的服务

`$animate` 服务暴露了一系列支持动画钩子的 DOM 实用方法。默认行为是DOM操作的应用程序，然而，一个动画被检测到（并且动画是可用的），`$animate` 会做出大量的操作来确保那些触发DOM操作的动画。

默认情况下，`$animate` 不会触发动画，这是因为 `ngAnimate` 模块没有被包含进来，而只有当 `ngAnimate` 被依赖之后，动画钩子 `$animate` 触发器才能使用。依赖之后所有以 `ng-` 开头的指令 都将会根据他们相关的 DOM 操作来执行动画。其他的指令如：`ngClass`，`ngShow`，`ngHide`，`ngMessage` 也都提供了对动画的支持。

我们推荐你，总是在指令中使用 `$animate` 服务来执行DOM相关的程序。

了解更多的动画支持的使用，点解这访问 [ngAnimate 模块页面](#)

## 方法

### 1. on(event, container, callback)

无论何时动画事件（enter, leave, move等）在被给定的元素（或其子元素）上执行都会触发你设置的事件监听器。一旦监听器被触发，提供的回调函数就会接收到以下参数并触发。

```
$animate.on('enter', container,
  function callback(element, phase) {
    // 酷，我们检测到容器内的进入动画
  }
);
```

### 参数

参数	形式	详细
event	string	将被捕获的动画事件 (例如 : enter, leave, move, addClass, removeClass, 等等)
container	DOMElement	容器元素将会捕获每一次动画事件在它自身以及子元素身上执行。
callback	fn	回调函数在监听器被触发后执行。 回调函数接收的参数如下： element - 被捕获的需要执行动画的 DOM 元素。 phase - 动画的阶段，动画开始和结束时都有两个可能出现的状态。

## 2. off(event, [container], [callback])

注销基于已与所提供的元素相关联的事件的事件监听器。这个方法可以依据参数有三种不同的使用方法：

```
// 删除所有监听 `enter` 事件的动画监听器。
$animate.off('enter');

// 删除所有绑定在给定元素（及其子元素）上的监听 `enter` 事件的动画事件监听器
$animate.off('enter', container);

// 删除由 `listenerFn` 提供用来监听
// `container` 及其子元素身上的 `enter` 事件监听器函数
$animate.off('enter', container, callback);
```

### 参数

参数	形式	详细
event	string	动画事件 (如 enter, leave, move, addClass, removeClass等)
container (可选)	DOMElement	事件监听器被放置的容器元素
callback (可选)	Function=	被注册为监听器的回调函数

### 3. `pin(element, parentElement)`

将提供的元素和一个父元素联系在一起，允许即使它存在于 Angular 应用的DOM结构的外部时也可以出发动画。这样一来，无论这个元素是在应用域外还是在另一个应用中 `$animate` 触发的动画都可以在该元素上生效。例如说，假如应用在一个 `<body>` 内部的某一个元素上已经启动了。但是我们仍需要使一个元素成为 `document.body` 的直接子元素。这时我们可以通过使用 `$animate.pin(element)` 来实现这个连接这个元素。别忘了，调用 `$animate.pin(element, parentElement)` 并不会实际地将其插入到 DOM 中。它仅仅是被连接起来的。

注意：这个特点仅仅在 `ngAnimate` 模块被引入的情况下可以使用。

#### 参数

参数	形式	详细
<code>element</code>	<code>DOMElement</code>	需要被连接的外部元素
<code>parentElement</code>	<code>DOMElement</code>	将会被外部元素连接的主父元素

### 4. `enabled([element], [enabled])`

用于获取和设置动画是否启用与否对整个应用程序或元素及其子上。这个函数可以使用以下四种方式调用：

```
// 返回 true 或者 false
$animate.enabled();

// 改变所有动画的启用状态
$animate.enabled(false);
$animate.enabled(true);

// 返回 true 或者 false 判断一个元素的动画是否启用

// 改变一个元素及其子元素动画的启用状态
$animate.enabled(element, true);
$animate.enabled(element, false);
```

#### 参数

参数	形式	详细
element (可选)	DOMElement	将被考虑检测/设置启用状态的元素
enabled (可选)	boolean	对于 element 动画是否启用

返回

boolean - 动画是否启用

## 5. cancel(animationPromise)

取消提供的动画

参数

参数	形式	详细
animationPromise	Promise	一个动画开始时返回的动画 promise

## 6. enter(element, parent, [after], [options])

将元素插入DOM或 after 元素(如果提供了)之后, 或者作为父元素的第一个元素, 之后会触发动画。当动画执行完毕之后的 digest 将会返回一个被解析的 promise。

参数

参数	形式	详细
element	DOMElement	需要插入 DOM 的元素
parent	DOMElement	作为插入元素父级的元素(只要该元素之后不存在)
after (可选)	DOMElement	作为插入元素的兄弟元素
options (可选)	object	一个可选的集合, 其中的项目和样式都将应用到元素上

返回

Promise - 动画回调的 promise

## 7. move(element, parent, [after], [options])

插入或移动元素到一个 DOM 中的新的位置或 `after` 元素（如果提供的话）之后或作为母体元件内的第一个子元素，然后触发一个动画。当动画执行完毕之后的 `digest` 将会返回一个被解析的 `promise`。

#### 参数

参数	形式	详细
<code>element</code>	<code>DOMElement</code>	将要移动到新位置的元素
<code>parent</code>	<code>DOMElement</code>	作为插入元素父级的元素(只要该元素之后不存在)
<code>after</code> （可选）	<code>DOMElement</code>	作为插入元素的兄弟元素
<code>options</code> （可选）	<code>object</code>	一个可选的集合，其中的项目和样式都将应用到元素上

#### 返回

`Promise` - 动画的回调 `promise`

## 8. `leave(element, [options])`

触发一个动画并在之后重DOM中删除这个元素。当动画执行完毕之后的 `digest` 将会返回一个被解析的 `promise`。

#### 参数

参数	形式	详细
<code>element</code>	<code>DOMElement</code>	将从 DOM 中删除的元素
<code>options</code> （可选）	<code>object</code>	一个可选的集合，其中的项目和样式都将应用到元素上

#### 返回

`Promise` - 动画的回调 `promise`

## 9. `addClass(element, className, [options])`

当添加了一个提供的 CSS 类名时会触发一个 `addClass` 动画。关于执行方面，`addClass` 操作仅会在下一次 `digest` 后被执行。但如果元素已经包含了这个 CSS 类名亦或这个类名被稍后的动作删除了，那动画也不会被触发。需要注

意的是，基于类名的动画要区别于结构动画（像 `enter`，`move`，`leave`），因为如果 CSS 或 Javascript 动画被使用，CSS 类名可能会在不同的地方被依赖使用（添加或删除）

#### 参数

参数	形式	详细
<code>element</code>	<code>DOMElement</code>	将要应用 CSS 类名的元素
<code>className</code>	<code>string</code>	将要被添加的 CSS 类名（多类名由空格分开）
<code>options</code> （可选）	<code>object</code>	一个可选的集合，其中的项目和样式都将应用到元素上

#### 返回

`Promise` - 动画的回调 `promise`

## 10. `removeClass(element, className, [options])`

当删除一个提供 CSS 类名时会触发一个 `removeClass` 动画。。关于执行方面，`removeClass` 操作仅会在下一次 `digest` 后被执行。如果元素没有包含 CSS 类名，或在稍后会添加这个类名则动画就不会被触发。需要注意的是，基于类名的动画要区别于结构动画（像 `enter`，`move`，`leave`），因为如果 CSS 或 Javascript 动画被使用，CSS 类名可能会在不同的地方被依赖使用（添加或删除）

#### 参数

参数	形式	详细
<code>element</code>	<code>DOMElement</code>	CSS 类名要应用的元素
<code>className</code>	<code>string</code>	将被移除的 CSS 类名（多类名由空格分开）
<code>options</code> （可选）	<code>object</code>	一个可选的集合，其中的项目和样式都将应用到元素上

#### 返回

`Promise` - 动画的回调 `promise`

## 11. `setClass(element, add, remove, [options])`



同时执行一个元素上 CSS 类名的添加和删除（在这个过程中）并依据添加或删除触发动画。与 `$animate.addClass` 和 `$animate.removeClass` 类似，`setClass` 仅会在这次 `digest` 完成后对类名进行一次修改。需要注意的是，基于类名的动画要区别于结构动画（像 `enter`，`move`，`leave`），因为如果 CSS 或 Javascript 动画被使用，CSS 类名可能会在不同的地方被依赖使用（添加或删除）

### 参数

参数	形式	详细
<code>element</code>	<code>DOMElement</code>	CSS 类名要应用的元素
<code>add</code>	<code>string</code>	将被添加的 CSS 类名（多类名由空格分开）
<code>remove</code>	<code>string</code>	将被删除的 CSS 类名（多类名由空格分开）
<code>options</code> （可选）	<code>object</code>	一个可选的集合，其中的项目和样式都将应用到元素上

### 返回

`Promise` - 动画的回调 `promise`

## 12. `animate(element, from, to, [className], [options])`

在元素上执行一个指定了起始和结束的样式的内联动画。如果任何检测到的 CSS 过渡，关键帧或 JavaScript 匹配了提供的 `className` 的值，则该动画将使用他们所提供的样式。例如，如果一个过渡动画设置为了给定的 `className`，那么所提供的 `from` 和 `to` 的样式将沿着给定的过渡施加。如果检测到了 JavaScript 动画就将设定的样式最为函数的参数传递到动画的方法中（或者作为选项参数的一部分）

### 参数

参数	形式	详细
element	DOMElement	CSS 样式应用的元素
from	object	会被应用到元素和整个动画过程的初始的 CSS 样式
to	object	会被应用到元素和整个动画过程的结束的 CSS 样式
className (可选)	string	一个将要在动画的执行过程中应用到元素上的可设置的 CSS 类。 如果这个这保留为空，那么一个 ng-inline-animate CSS 类将被应用到元素中。 (注意，如果没有检测到动画，那么，这个这应不会应用到元素身上)
options (可选)	object	一个将被应用到元素的可控的设置/样式的集合

返回

Promise - 动画的回调 promise

## Angular\_1.4.3 API 服务篇 \$cacheFactory

- `ng` 模块中的服务

以工厂模式构造`cache`对象，并且使它们可以被访问。

javascript

```
var cache = $cacheFactory('cacheId');

expect($cacheFactory.get('cacheId')).toBe(cache);
expect($cacheFactory.get('noSuchCacheId')).not.toBeDefined();

cache.put("key", "value");
cache.put("another key", "another value");

// 创建时我们没有指定配置项
expect(cache.info()).toEqual({id: 'cacheId', size: 2});
```

### 用法

```
$cacheFactory(cacheId, [配置项]);
```

参数	形式	具体
cacheId	string	新缓存的名称或ID。
配置项 (选填)	object	配置对象会指定缓存的行为。 性能: {number=} 容量 — 将缓存转化成LRU缓存。

返回

---	---
object	<p>新创建的缓存对象有以下的配置方法:</p> <ul style="list-style-type: none"> <li>- <code>{object} info()</code> — 返回 id, 大小, 和缓存的配置。</li> <li>- <code>*, put({string} key, {*} value)</code> — 向缓存中插入以个新的键值对并将它返回。</li> <li>- <code>*, get({string} key)</code> — 返回与 <code>key</code> 对应的 <code>value</code> 值, 如果未命中则返回 <code>undefined</code> 。</li> <li>- <code>{void} remove({string} key)</code> — 从缓存中删除一个键值对</li> <li>- <code>{void} removeAll()</code> — 删除所有缓存中的数据</li> <li>- <code>{void} destroy()</code> — 删除从 <code>\$cacheFactory</code> 引用的这个缓存.</li> </ul>

## 方法

### 1. `info()`; 获取所有被创建的缓存的信息

返回

`Object` 返回一个关于 `cacheId` 的键值

### 2. `get(cacheId)`; 如果与`cacheId`相对应的缓存对象被创建, 则获取它

参数

参数	形式	具体
<code>cacheId</code>	<code>string</code>	一个可以通过的缓存名字或ID

返回

Returns `Object` - 通过 `cacheId` 确认的缓存对象, 或是确认失败的 `undefined`

## 例子

html

```

<div ng-controller="CacheController">

  <input ng-model="newCacheKey" placeholder="Key">
  <input ng-model="newCacheValue" placeholder="Value">

  <button ng-click="put(newCacheKey,newCacheValue)">Cache</button>

  <p ng-if="keys.length">Cached Values</p>
  <div ng-repeat="key in keys">
    <span ng-bind="key"></span>
    <span>: </span>
    <b ng-bind="cache.get(key)"></b>
  </div>

  <p>Cache Info</p>
  <div ng-repeat="(key, value) in cache.info()">
    <span ng-bind="key"></span>
    <span>: </span>
    <b ng-bind="value"></b>
  </div>
</div>

```

javascript

```

angular.module('cacheExampleApp', []).
  controller('CacheController', ['$scope', '$cacheFactory', function($scope, $cacheFactory) {
    $scope.keys = [];
    $scope.cache = $cacheFactory('cacheId');
    $scope.put = function(key, value) {
      if ($scope.cache.get(key) === undefined) {
        $scope.keys.push(key);
      }
      $scope.cache.put(key, value === undefined ? null : value);
    };
  }]);

```

## css

```
p {  
  margin: 10px 0 3px;  
}
```

## Angular\_1.4.3 API 服务篇 \$controller

- \$controllerProvider
- ng 模块中的服务

`$controller` 服务负责实例化控制器。

它仅仅是对 `$injector` 的简单调用，之所以提取成服务，是为了方便BC版本可以覆盖这个服务。

### 依赖

`$injector`

### 用法

`$controller(constructor, locals)`

#### 参数

参数	形式	详细
constructor	<code>function()</code> <code>string</code>	如果传入一个函数，那么它被认为是控制器的构造函数。否则它会被认为是用于检索所述控制器构造函数的一个字符串。 用法如以下步骤： <ul style="list-style-type: none"><li>- 检查给定的名字是否已经通过 <code>\$controllerProvider</code> 注册成为控制器</li><li>- 检查字符串是否会在当前作用域返回一个构造函数，如果为 <code>\$controllerProvider#allowGlobals</code>，则在全局的 <code>window</code> 对象上验证 <code>window[constructor]</code> (不建议这么做)</li><li>- 当控制器实例在某个作用域内被发布为指定的属性时，该字符串可以使用控制器的属性语法；该作用域必须被注入到局部的参数中以确保能正常工作。</li></ul>
locals	Object	控制器要注入的域

#### 返回

Object - 给定控制器的实例

# \$http

- \$httpProvider
- ng 模块中的服务

`$http` 服务是一个Angular内的有助于通过浏览器的 `XMLHttpRequest` 对象或 `JSONP` 的方式连接远程 HTTP 服务器核心服务。

对使用 `$http` 服务的单元测试的应用，请参见 `$httpBackend` 测试

对于更高级别的抽象化，请查看 `$resource` 服务

`$http` 的 API 是基于 `$q` 服务导出的 `deferred/promise` API.

虽然简单的用法格式与此没有多大关系，但是对于更为高级的用法它使非常重要的，它所提供的这些 API 和保障会使你更为熟悉你自己写的代码。

## 一般用法

`$http` 服务是一个函数，它需要单一的一个参数 - 一个配置对象 - 它被用来生成一个 HTTP 请求并返回一个 拥有两个具体方法的 `promise` : `success` 和 `error`

```
// 简单的 GET 请求示例：
$http.get('/someUrl').
  success(function(data, status, headers, config) {
    // 当响应成功了
    // 这个回调函数会被异步调用
  }).
  error(function(data, status, headers, config) {
    // 如果发生错误或则相应返回了错误的状态吗
    // 这个回调函数会被异步调用
  });
```



```
// 简单的 POST 请求示例 (passing data) :
$http.post('/someUrl', {msg: 'hello word!'}).
  success(function(data, status, headers, config) {
    // 当响应成功了
    // 这个回调函数会被异步调用
  }).
  error(function(data, status, headers, config) {
    // 如果发生错误或则相应返回了错误的状态吗
    // 这个回调函数会被异步调用
  });
```

由于调用 `$http` 函数的返回值是一个 `promise`，因此你可以使用 `then` 方法注册回调函数，并且这些回调函数 将会接收唯一的参数 - 请求返回的响应对象。请查看下方关于 API 和格式信息的详细信息。

响应状态码在 200 - 299 之间会被认定为成功的状态，并在成功的回调函数被调用时将返回的结果传递给它。注意，如果响应是一个重定向，`XMLHttpRequest` 依旧会执行，意思是 `error` 回调不会被响应调用。

## 使用 `$http` 编写单元测试

```
$httpBackend.expectGET(...);
$http.get(...);
$httpBackend.flush();
```

## 简写方法

简写方法也可以使用，所有的简写方法都需要传入 URL，并且需要为 `POST/PUT` 请求传入请求数据。

```
$http.get('/someUrl').success(successCallback);
$http.post('/someUrl', data).success(successCallback);
```

简写方法的完整列表:

- `$http.get`
- `$http.head`
- `$http.post`
- `$http.put`
- `$http.delete`
- `$http.jsonp`
- `$http.patch`

## 设置 HTTP 头部信息

`$http` 服务会自动为所有的请求添加确定的 HTTP 头。这些默认的设置可以通过 `$httpProvider.defaults.headers` 组态对象来完全重置。包含着一下的组态：

- `$httpProvider.defaults.headers.common` (所有请求公用的头部):
  - `Accept: application/json, text/plain, * / *`
- `$httpProvider.defaults.headers.post` : (POST 请求的默认头部)
  - `Content-Type: application/json`
- `$httpProvider.defaults.headers.put` (PUT 请求的头部)
  - `Content-Type: application/json`

添加或者覆盖这些默认配置，只需从组态对象中添加或者删除一个属性即可。为除了 POST 或 PUT 的 HTTP 方法添加一个头部，则只需添加一个以这个方法名字的小写形式作为 key 的新的对象即可。例如

```
$httpProvider.defaults.headers.get = { 'My-Header' : 'value' } .
```

这些默认的设置运行时仍然可以通过 `$http.defaults` 对象实现改变，例如：

```
module.run(function($http) {  
  $http.defaults.headers.common.Authorization = 'Basic YmVlcDpib29v  
})
```

此外，你可以在 `$http(config)` 调用的时候在传入 `config` 对象的时候提供一个 `header` 属性，以此来覆盖默认配置而并不用全局地改变他们。

为了明确的删除在每一个请求的通过 `$httpProvider.default.headers` 自动添加的头部，使用 `headers` 的属性，设置需要的属性为 `undefined`。例如：

```
var req = {
  method: 'POST',
  url: 'http://example.com',
  headers: {
    'Content-Type': undefined
  },
  data: { test: 'test' }
}

$http(req).success(function(){...}).error(function(){...});
```

## 改造请求和响应

请求和响应均可以被改造函数改造：`transformRequest` 和 `transformResponse`。这两个属性可以是返回改造值一个单独的函数（`function(data, headersGetter, status)`），或者是一个含有改造函数的一个数组，它可以允许你向其中 `push` 或 `unshift` 一个新改造函数到你的改造链中。

## 默认改造

`$httpProvider` 提供器和 `$http` 服务导出了 `defaults.transformRequest` 和 `defaults.transformResponse` 属性。如果以一个请求没有一共它自己的改造那么以上的实行将被应用。

你可以通过向数组添加或替换改造函数来修改这些属性以扩大或者取代默认的改造。

Angular 提供了以下的默认改造。

请求改造 ( `$httpProvider.defaults.transformRequest` 和 `$http.defaults.transformRequest` ):

- 如果 `request` 的组态对象的 `data` 属性中包含对象，那么这个对象将被 JSON 格式化。

响应改造 ( `$httpProvider.defaults.transformResponse` and `$http.defaults.transformResponse` ):

- 如果检测到 `XSRF` 前缀, 则过滤掉它(请看下面的 Security Considerations 部分)。
- 如果检测到 `JSON response`, 则使用 `JSON parser` 将其反序列化。

## 覆盖每一个请求的默认改造

如果你仅仅想覆盖某一个单独的 `request/response` 的改造, 那么将带有 `transformRequest` `transformResponse` 的对象传入 `$http` 中即可。

Note that if you provide these properties on the config object the default transformations will be overwritten. If you wish to augment the default transformations then you must include them in your local transformation array.

The following code demonstrates adding a new response transformation to be run after the default response transformations have been run.

```
function appendTransform(defaults, transform) {  
  
    // We can't guarantee that the default transformation is an array  
    defaults = angular.isArray(defaults) ? defaults : [defaults];  
  
    // Append the new transformation to the defaults  
    return defaults.concat(transform);  
}  
  
$http({  
    url: '...',  
    method: 'GET',  
    transformResponse: appendTransform($http.defaults.transformRespor  
        return doTransform(value);  
    })  
});
```

## Caching

To enable caching, set the request configuration `cache` property to `true` (to use default cache) or to a custom cache object (built with `$cacheFactory` ). When the cache is enabled, `$http` stores the response from the server in the specified cache. The next time the same request is made, the response is served from the cache without sending a request to the server.

Note that even if the response is served from cache, delivery of the data is asynchronous in the same way that real requests are.

If there are multiple GET requests for the same URL that should be cached using the same cache, but the cache is not populated yet, only one request to the server will be made and the remaining requests will be fulfilled using the response from the first request.

You can change the default cache to a new object (built with `$cacheFactory` ) by updating the `$http.defaults.cache` property. All requests who set their cache property to true will now use this cache object.

If you set the default cache to false then only requests that specify their own custom cache object will be cached.

## Interceptors

Before you start creating interceptors, be sure to understand the `$q` and `deferred/promise` APIs.

For purposes of global error handling, authentication, or any kind of synchronous or asynchronous pre-processing of request or postprocessing of responses, it is desirable to be able to intercept requests before they are handed to the server and responses before they are handed over to the application code that initiated these requests. The interceptors leverage the promise APIs to fulfill this need for both synchronous and asynchronous pre-processing.

The interceptors are service factories that are registered with the `$httpProvider` by adding them to the `$httpProvider.interceptors` array. The factory is called and injected with dependencies (if specified) and returns the interceptor.

There are two kinds of interceptors (and two kinds of rejection interceptors):

- `request` : interceptors get called with a `http config` object. The function is free to modify the `config` object or create a new one. The function needs to return the `config` object directly, or a promise containing the `config` or a new `config` object.
- `requestError` : interceptor gets called when a previous interceptor threw an error or resolved with a rejection.
- `response` : interceptors get called with `http response` object. The function is free to modify the `response` object or create a new one. The function needs to return the `response` object directly, or as a promise containing the `response` or a new `response` object.
- `responseError` : interceptor gets called when a previous interceptor threw an error or resolved with a rejection.

```
// register the interceptor as a service
$provide.factory('myHttpInterceptor', function($q, dependency1, dependency2) {
  return {
    // optional method
    'request': function(config) {
      // do something on success
      return config;
    },

    // optional method
    'requestError': function(rejection) {
      // do something on error
      if (canRecover(rejection)) {
        return responseOrNewPromise
      }
      return $q.reject(rejection);
    },

    // optional method
    'response': function(response) {
      // do something on success
      return response;
    },
  };
});
```

```
// optional method
'responseError': function(rejection) {
  // do something on error
  if (canRecover(rejection)) {
    return responseOrNewPromise
  }
  return $q.reject(rejection);
};
});

$httpProvider.interceptors.push('myHttpInterceptor');

// alternatively, register the interceptor via an anonymous factory
$httpProvider.interceptors.push(function($q, dependency1, dependency2) {
  return {
    'request': function(config) {
      // same as above
    },

    'response': function(response) {
      // same as above
    }
  };
});
```

## Security Considerations

When designing web applications, consider security threats from:

- JSON vulnerability
- XSRF Both server and the client must cooperate in order to eliminate these threats. Angular comes pre-configured with strategies that address these issues, but for this to work backend server cooperation is required.

## JSON Vulnerability Protection

A JSON vulnerability allows third party website to turn your JSON resource URL into JSONP request under some conditions. To counter this your server can prefix all JSON requests with following string `" )]]}',\n"` . Angular will automatically strip the prefix before processing it as JSON.

For example if your server needs to return:

```
[ 'one', 'two' ]
```

which is vulnerable to attack, your server can return:

```
'.....)]]',  
[ 'one', 'two' ]
```

Angular will strip the prefix, before processing the JSON.

## Cross Site Request Forgery (XSRF) Protection

XSRF is a technique by which an unauthorized site can gain your user's private data. Angular provides a mechanism to counter XSRF. When performing XHR requests, the `$http` service reads a token from a cookie (by default, XSRF-TOKEN) and sets it as an HTTP header (X-XSRF-TOKEN). Since only JavaScript that runs on your domain could read the cookie, your server can be assured that the XHR came from JavaScript running on your domain. The header will not be set for cross-domain requests.

To take advantage of this, your server needs to set a token in a JavaScript readable session cookie called XSRF-TOKEN on the first HTTP GET request. On subsequent XHR requests the server can verify that the cookie matches X-XSRF-TOKEN HTTP header, and therefore be sure that only JavaScript running on your domain could have sent the request. The token must be unique for each user and must be verifiable by the server (to prevent the JavaScript from making up its own tokens). We recommend that the token is a digest of your site's authentication cookie with a salt for added security.



The name of the headers can be specified using the `xsrfHeaderName` and `xsrfCookieName` properties of either `$httpProvider.defaults` at config-time, `$http.defaults` at run-time, or the per-request config object.

In order to prevent collisions in environments where multiple Angular apps share the same domain or subdomain, we recommend that each application uses unique cookie name.

## 依赖

`$httpBackend` `$cacheFactory` `$rootScope` `$q` `$injector`

## 用法

`$http(config)`

### 参数

参数	形式	详细
config	object	

Object describing the request to be made and how it should be processed. 该对象有以下属性:

- **method** – {string} – HTTP 方法 (如. 'GET', 'POST', 等)
- **url** – {string} – 需要请求的资源的绝对或相对 URL
- **params** – {Object.<string|Object>} – 它会被 `paramSerializer` 序列化字符串或对象的 `map` 形式, 并添加到 GET 参数中。
- **data** – {string|Object} – 将被当做请求的数据被发送。
- **headers** – {Object} – 字符串或函数的 `map` 形式 代表 HTTP 头

Map of strings or functions which return strings representing HTTP headers to send to the server. If the return value of a function is null, the header will not be sent. Functions accept a config object as an argument.

- **xsrfHeaderName** – {string} – HTTP 头部名 Name of HTTP header to populate with the XSRF token.
- **xsrfCookieName** – {string} – Name of cookie containing the XSRF

token.

- **transformRequest** – `{function(data, headersGetter)|Array.<function(data, headersGetter)>}` – transform function or an array of such functions. The transform function takes the http request body and headers and returns its transformed (typically serialized) version. See [Overriding the Default Transformations](#)
- **transformResponse** – `{function(data, headersGetter, status)|Array.<function(data, headersGetter, status)>}` – transform function or an array of such functions. The transform function takes the http response body, headers and status and returns its transformed (typically deserialized) version. See [Overriding the Default Transformation](#)
- **paramSerializer** - `{string|function(Object<string, string>):string}` - A function used to prepare the string representation of request parameters (specified as an object). If specified as string, it is interpreted as function registered with the \$injector, which means you can create your own serializer by registering it as a service. The default serializer is the \$httpParamSerializer; alternatively, you can use the \$httpParamSerializerJQLike
- **cache** – `{boolean|Cache}` – If true, a default \$http cache will be used to cache the GET request, otherwise if a cache instance built with \$cacheFactory, this cache will be used for caching.
- **timeout** – `{number|Promise}` – 设置毫秒延时, timeout in milliseconds, or promise that should abort the request when resolved.
- **withCredentials** - `{boolean}` - 是否在 XHR 对象上设置 withCredentials 。 请看[使用证书的请求](#) 以获取更多信息。
- **responseType** - `{string}` - 请看 [XMLHttpRequest.responseType](#). |

返回

`HttpPromise` - Returns a promise object with the standard then method and two http specific methods: success and error. The then method takes two arguments a success and an error callback which will be called with a response object. The success and error methods take a single argument - a function that will be called when the request succeeds or fails respectively. The arguments passed into these functions are destructured representation of the response object passed into the then method. The response object has these properties:

- **data** – {string|Object} – The response body transformed with the transform functions.
- **status** – {number} – HTTP status code of the response.
- **headers** – {function([headerName])} – Header getter function.
- **config** – {Object} – The configuration object that was used to generate the request.
- **statusText** – {string} – HTTP status text of the response.

## 方法

### 1. `get(url, [config])`

执行 `GET` 请求的快捷方法。

参数

参数	形式	详细
url	string	指定请求目的地的相对或绝对路径。
config (可选)	Object	可选的配置对象

返回

HttpPromise - promise

### 2. `delete(url, [config])`

执行 `DELETE` 请求的快捷方法。

参数

参数	形式	详细
url	string	指定请求目的地的相对或绝对路径。
config (可选)	Object	可选的配置对象

返回

HttpPromise - promise

### 3. head(url, [config])

执行 HEAD 请求的快捷方法。

参数

参数	形式	详细
url	string	指定请求目的地的相对或绝对路径。
config (可选)	Object	可选的配置对象

返回

HttpPromise - promise

### 4. jsonp(url, [config])

执行 JSONP 请求的快捷方法。

参数

参数	形式	详细
url	string	指定请求目的地的相对或绝对路径。回调函数的名字必须是字符串 JSON_CALLBACK
config (可选)	Object	可选的配置对象

返回

HttpPromise - promise

### 5. post(url, data, [config])

执行 POST 请求的快捷方法。

参数

参数	形式	详细
url	string	指定请求目的地的相对或绝对路径。
data	*	请求内容
config (可选)	Object	可选的配置对象

返回

HttpPromise - promise

## 6. put(url, data, [config])

执行 PUT 请求的快捷方法。

参数

参数	形式	详细
url	string	指定请求目的地的相对或绝对路径。
data	*	请求内容
config (可选)	Object	可选的配置对象

返回

HttpPromise - promise

## 7. patch(url, data, [config])

执行 PATCH 请求的快捷方法。

参数

参数	形式	详细
url	string	指定请求目的地的相对或绝对路径。
data	*	请求内容
config (可选)	Object	可选的配置对象

返回

HttpPromise - promise

## 属性

### 1. pendingRequests

`Array.<Object>` - 配置对象的当前待处理请求的数组，它的主要作用是为了调试。

## 2. defaults

`$httpProvider.defaults` 属性的运行时当量。允许默认头配置, `withCredentials`, 以及请求和响应的变化。

详细请看上面的 "设置 HTTP Headers", "Transforming Requests and Responses"。

## 例子

index.html

```
<div ng-controller="FetchController">
  <select ng-model="method" aria-label="Request method">
    <option>GET</option>
    <option>JSONP</option>
  </select>
  <input type="text" ng-model="url" size="80" aria-label="URL" />
  <button id="fetchbtn" ng-click="fetch()">fetch</button><br>
  <button id="samplegetbtn" ng-click="updateModel('GET', 'http-hel...
  <button id="samplejsonpbtn"
    ng-click="updateModel('JSONP',
                        'https://angularjs.org/greet.php?callback=JSON_C/
    Sample JSONP
  </button>
  <button id="invalidjsonpbtn"
    ng-click="updateModel('JSONP', 'https://angularjs.org/doesntext...
    Invalid JSONP
  </button>
  <pre>http status code: {{status}}</pre>
  <pre>http response data: {{data}}</pre>
</div>
```

script.js

```
angular.module('httpExample', [])
.controller('FetchController', ['$scope', '$http', '$templateCache']
function($scope, $http, $templateCache) {
    $scope.method = 'GET';
    $scope.url = 'http-hello.html';

    $scope.fetch = function() {
        $scope.code = null;
        $scope.response = null;

        $http({method: $scope.method, url: $scope.url, cache: $templateCache}).
            success(function(data, status) {
                $scope.status = status;
                $scope.data = data;
            }).
            error(function(data, status) {
                $scope.data = data || "Request failed";
                $scope.status = status;
            });
    };

    $scope.updateModel = function(method, url) {
        $scope.method = method;
        $scope.url = url;
    };
}]);
```

http-hello.html

Hello, \$http!

protactor.js

```
var status = element(by.binding('status'));
var data = element(by.binding('data'));
var fetchBtn = element(by.id('fetchbtn'));
var sampleGetBtn = element(by.id('samplegetbtn'));
var sampleJsonpBtn = element(by.id('samplejsonpbtn'));
var invalidJsonpBtn = element(by.id('invalidjsonpbtn'));

it('should make an xhr GET request', function() {
  sampleGetBtn.click();
  fetchBtn.click();
  expect(status.getText()).toMatch('200');
  expect(data.getText()).toMatch(/Hello, \${http!}/);
});

// Commented out due to flakes. See https://github.com/angular/angular.js/issues/10987
// it('should make a JSONP request to angularjs.org', function() {
//   sampleJsonpBtn.click();
//   fetchBtn.click();
//   expect(status.getText()).toMatch('200');
//   expect(data.getText()).toMatch(/Super Hero!/);
// });

it('should make JSONP request to invalid URL and invoke the error handler', function() {
  invalidJsonpBtn.click();
  fetchBtn.click();
  expect(status.getText()).toMatch('0');
  expect(data.getText()).toMatch('Request failed');
});
```



## Angular\_1.4.3 API 服务篇 \$locale

- `ng` 模块的服务

`$locale` 服务为许多 Angular 组件提供了本地化规则，截止目前只提供一个公共的 API:

`id` – { `string` } – 语言环境 id 格式化成 `语言Id-国家Id` 的形式(如 `en-us`)

## Angular\_1.4.3 API 服务篇 \$location

- \$locationProvider
- ng 模块中的服务

`$location` 服务格式化了浏览器地址栏中的 URL (基于 `window.location` 对象), 并让它在你的应用中可以使用。改变地址栏中的 URL 将反馈到 `$location` 服务中而且改变 `$location` 也同样会反馈到浏览器地址栏中。

`$location` 服务：

- 暴露了浏览器地址栏中的当前 URL, 所以你可以：
  - 观察监听 URL.
  - 修改 URL.
- 与浏览器的 URL 保持同步, 当用户：
  - 修改地址栏
  - 点击了前进或者后退按钮(或点击了一个历史链接).
  - 页面中点击了一个链接
- 用一组方法表示了 URL 对象 ( `protocol` , `host` , `port` , `path` , `search` , `hash` ).

更多信息请参见开发者文档：使用 `$location`

### 依赖

`$rootElement`

### 方法

#### 1. `absUrl()`

这个方法仅用于获取

```
// 地址栏中的 URL http://example.com/#/some/path?foo=bar&baz=xoxo

var absUrl = $location.absUrl();

// => sting : "http://example.com/#/some/path?foo=bar&baz=xoxo"
```

返回

string - 返回带有所有根据 RFC 3986 编码的部分的完整 URL

## 2. url([url])

这个方法既可以获取也可以设置

没传入参数时返回 url (如 : /path?a=b#hash)

当传入参数时则会修改 path , search 和 ·hash , 并返回 \$location`。

```
// 地址栏中的 URL http://example.com/#/some/path?foo=bar&baz=xoxo

var url = $location.url()

// => sting : "/some/path?foo=bar&baz=xoxo"
```

参数

参数	形式	详细
url (可选)	string	新的 url, 不需指定基础前缀 (如 : '/path?a=b#hash' )

返回

string - url

## 3. protocol()

这个方法仅用于获取

返回当前 url 的协议

```
// 地址栏中的 URL http://example.com/#/some/path?foo=bar&baz=xoxo

var protocol = $location.protocol()

// => sting : "http"
```

返回

string - 返回当前 url 的协议

## 4. host()

这个方法仅用于获取

返回当前 URL 的域名、主机名

Note: 相比于不使用 Angular 版本的 `location.host` 返回的 域名:端口 , `$location.host()` 仅返回域名。

```
// 地址栏中的 URL http://example.com/#/some/path?foo=bar&baz=xoxo

var host = $location.host();

// => "example.com"

// 地址栏中的 URL http://user:password@example.com:8080/#/some/path?

host = $location.host();

// => "example.com"

host = location.host;

// => "example.com:8080"
```

返回

`string` - 当前 URL 的域名、主机名

## 5. `port()`

这个方法仅用于获取

返回当前 URL 的端口号

```
// 地址栏中的 URL http://example.com/#/some/path?foo=bar&baz=xoxo

var port = $location.port()

// => number : 80
```

返回

`number` - 端口

## 6. `path([path])`

这个方法既可以获取也可以设置

不传入参数则返回当前 URL 的路径部分

传入参数则修改路径部分并返回 `$location`

注意：路径必须以斜杠开头（ / ），如果没有斜杠这个方法将会将其补充上

```
// 地址栏中的 URL http://example.com/#/some/path?foo=bar&baz=xoxo

var path = $location.path()

// => string : "/some/path"
```

参数

参数	形式	详细
path （可选）	<code>string</code> <code>number</code>	新的路径部分

返回

string - 路径部分

## 7. search(search, [paramValue])

这个方法既可以获取也可以设置

当没传入参数时， 返回当前 URL 的搜索部分 (以对象的形式) 即 ? 后的部分

传入参数时， 将会改变搜索部分并返回 \$location

```
// 地址栏中的 URL  http://example.com/#/some/path?foo=bar&baz=xoxo

var searchObject = $location.search()

// => object : {foo: 'bar', baz: 'xoxo'}

// set foo to 'yipee'
$location.search('foo', 'yipee')

// => $location
// $location.search() => object {foo: 'yipee', baz: 'xoxo'}
```

参数

参数	形式	详细
search	string Object. <string> Object. <Array. <string>>	新的搜索部分的参数 - 字符串或哈希对象 当只传入一个参数时，这个方法是一个设置动作，将 \$location 的搜索部分组件设置为一个指定的值。 如果传入的参数是一个包含一组值的哈希对象，那么这些值将会被作为URL 中复制的搜索部分参数被编码
paramValue (可选)	string Number Array. <string> boolean	如果搜索部分是一个数字或者字符串，paramValue 将只覆盖一个单独的搜索部分的属性。 如果 paramValue 是一个数组，它会覆盖 \$location 的通过第一个参数指定的搜索组件的属性 如果 paramValue 是 null ，通过第一个参数指定的实行将会被删除 如果 paramValue 是 true ，通过第一个参数指定的实行将会被添加进去，但是不会赋值也没有尾随等号

返回

`Object` - 如果没传递参数则返回格式化后的搜索部分的对象，如果传递了多于一个的参数则返回 `$location` 对象

## 8. `hash([hash])`

这个方法既可以获取也可以设置

没传入参数时返回哈希码

传入参数则修改哈希码并返回 `$location`

```
// 地址栏中的 URL http://example.com/#/some/path?foo=bar&baz=xoxo#ha  
  
var hash = $location.hash();  
  
// => string : "hashValue"
```

参数

参数	形式	详细
hash (可选)	string number	新的哈希码

返回

`string` - 哈希码

## 9. `replace()`

如果调用，所有当前 `$digest` 下的对 `$location` 的修改都将会取代历史记录。而不是添加一个新的。

## 10. `state([state])`

这个方法既可以获取也可以设置

没传入参数时返回历史状态对象，当传入参数时修改历史状态对象并返回

`$location` . 这个状态对象稍后会传递给 `pushState` 或 `replaceState`

注意：这个方法仅在HTML5模式下被支持，而且只有在浏览器支持 HTML5 历史 API 的情况下可以使用。（即，`pushState` 和 `replaceState` ），若你想兼容更旧的浏览器（想 IE9 或 安卓4.0以下的设备），就不要使用这个方法。

## 参数

参数	形式	详细
state （可选）	object	可传递给 <code>pushState</code> 或 <code>replaceState</code> 的对象

## 返回

object - 状态

# 事件

## 1. `$locationChangeStart`

在 URL 修改前向下传播

针对 URL 的修改可以通过调用事件对象中的 `preventDefault` 方法来阻止。详情请参阅 `$rootScope.Scope` 来对事件对象有更详细的了解。一旦 URL 成功修改 `$locationChangeSuccess` 将会被调用。

当浏览器支持 HTML5 History API 并在 HTML5 模式下时，`newState` 和 `oldState` 参数才会被定义

方式: `broadcast` 向下传播

起始: 根作用域

## 参数



参数	形式	详细
angularEvent	object	合成的事件对象
newUrl	string	新的 URL
oldUrl (可选)	string	被修改之前的 URL
newState (可选)	string	新的历史状态对象
oldState (可选)	string	被修改之前的 URL

## 2. locationChangeSuccess

URL 修改后向下传播

当浏览器支持 HTML5 History API 并在 HTML5 模式下时，newState 和 oldState 参数才会被定义

方式: broadcast 向下传播

起始: 根作用域

参数

参数	形式	详细
angularEvent	object	合成的事件对象
newUrl	string	新的 URL
oldUrl (可选)	string	被修改之前的 URL
newState (可选)	string	新的历史状态对象
oldState (可选)	string	被修改之前的 URL

## Angular\_1.4.3 API 服务篇 \$log

- `$logProvider`
- `ng` 模块中的服务

简单的打印日志的服务。默认实现安全的写入信息到浏览器的控制台（如果存在的话）。

这个服务最主要的目的是简化调试和排除故障。

默认的设置是打印调试信息。你可以通过 `ng.$logProvider#debugEnabled` 来设置。

### 依赖

`$window`

### 方法

`log();` - 打印日志消息

`info();` - 打印信息消息

`warn();` - 打印警告消息

`error();` - 打印错误消息

`debug();` - 打印调试消息

### 例子

html

```
<div ng-controller="LogController">

  <p>
    Reload this page with open console,
    enter text and hit the log button...
  </p>
  <label>Message:
  <input type="text" ng-model="message" /></label>

  <button ng-click="$log.log(message)">log</button>
  <button ng-click="$log.warn(message)">warn</button>
  <button ng-click="$log.info(message)">info</button>
  <button ng-click="$log.error(message)">error</button>
  <button ng-click="$log.debug(message)">debug</button>

</div>
```

## javascript

```
angular.module('logExample', [])
.controller('LogController', ['$scope', '$log', function($scope, $log) {

  $scope.$log = $log;
  $scope.message = 'Hello World!';

}]);
```

## Angular\_1.4.3 API 服务篇 \$parse

- \$parseProvider
- ng 模块中的服务

将 Angular 表达式转化为函数。

```
var getter = $parse('user.name');
var setter = getter.assign;
var context = {user:{name:'angular'}};
var locals = {user:{name:'local'}};

expect(getter(context)).toEqual('angular');
setter(context, 'newValue');
expect(context.user.name).toEqual('newValue');
expect(getter(context, locals)).toEqual('local');
```

### 用法

```
$parse(expression);
```

#### 参数

参数	形式	详细
expression	string	编译字符串表达式

#### 返回

function(context, locals) - 一个代表编译表达式的函数:

- context - {object} - 这个对象中的值将在表达式转换时被忽略
- locals - {object=} - 局部变量的上下文函数, 用于在此上下文中覆盖 value 。 这个函数返回值会有跟随属性:
  - literal - {boolean} - 表达式的顶级节点是否为 Javascript 字面量。
  - constant - {boolean} - 表达式是否完全由 Javascript 常量组成。
  - assign - {?function(context, value)} - 如果表达式是可分配

的，那么它可以调用这个函数，并由传入的 `context` 来改变 `value` 。

## Angular\_1.4.3 API 服务篇 \$q

- ng模块下的服务 `$q` [官方文档](#)

这是一个能帮助你异步地运行函数，并且在他们仍在进程中的时候使用他们的返回值（或者捕捉异常）的服务

`$q` 的灵感源自[Kris Kowal的 Q](#)，它实现是一个 `promises / deferred` 对象。

`$q` 可以以下面两种形式来使用：

- 类似于Kris Kowal的 `Q` 或者jQuery实现的 `Deferred`
- 某种程度上类似于ES6规范中的 `promises`

### `$q` 的构造函数

`$q` 可以作为一个构造函数，并可以像流式ES6风格的 `promise` 一样使用。它需要一个解析器函数作为第一个参数。这与源自ES6 Harmony 的原生Promise的实现方式是十分相似的。详情见 [MDN](#). 同样也支持构造风格的编程方式，但是目前还没有完全支持ES6 Harmony实现的Promise内的所有方法。

可以被这样使用：

```
// 此例中，我们假设变量`$q`和`okToGreet`已经存在于当前作用域
// 他们可以通过依赖注入和传递参数来获取

function asyncGreet(name) {
  // 执行一些异步操作
  // 适当的时候解析(resolve)或者阻止(reject) Promise.

  return $q(function(resolve, reject) {

    setTimeout(function() {

      if (okToGreet(name)) {
        resolve('Hello, ' + name + '!');
      } else {
        reject('Greeting ' + name + ' is not allowed.');
```

贴士：进度/通知的回调目前还未在ES6风格里得到支持。

不过，更多的传统的CommonJS的风格依然可用，并且记录如下。 [The CommonJS Promise 指南](#) 描述 promise 是一个为某个代表着异步操作结果的对象提供互动的接口，而且它的执行也并不依赖于某个特定的时间节点。

从用错误处理解决问题的角度来看 `deferred` 和 `promise` 的一些API是异步编程而 `try`、`catch` 和 `throw` 关键字是同步编程.



```
// 此例中, 我们假设变量`$ q`和`okToGreet`已经存在于当前作用域
// 他们可以通过依赖注入和传递参数来获取

function asyncGreet(name) {

    var deferred = $q.defer();

    setTimeout(function() {

        deferred.notify('About to greet ' + name + '.');

        if (okToGreet(name)) {
            deferred.resolve('Hello, ' + name + '!');
        } else {
            deferred.reject('Greeting ' + name + ' is not allowed.');
```

开始的时候这样做的好处不是十分明显的，我们为什么要增加额外的复杂度？看起来十分麻烦。这样做的好处是源自 `promise` 和 `deferred` API文档确立的，详述请看[链接](#)。此外，`promise` API 允许构造是因为使用传统的回调形式十分困难。更多相关请参考[Q 文档](#) 特别是关于 `promise` 的串行和并行连接的部分。

## Deferred API

一个新的 `deferred` 实例可以通过调用 `$q.defer()` 来创建。

这样暴露出了相关 `promise` 实例，而且API可以用来广播成功或失败的完成，以及任务的状态。

### 方法

- `resolve(value)` – 用 `value` 来解析返回的 `promise`。如果 `value` 是通过 `$q.reject` 返回的拒绝构造，那么这个 `promise` 也将被拒绝。
- `reject(reason)` – 以 `reason` 拒绝返回的 `promise`。这等同于由 `$q.reject` 拒绝构造来解析它。
- `notify(value)` - 在 `promise` 执行的状态下提供更新。因此在 `promise` 尚未被解析或是拒绝之前会调用很多次。

### 属性

- `promise` - `{Promise}` – 与 `deferred` 相关联的 `promise` 对象。

## Promise API

当一个新的 `deferred` 实例被创建出来，并且可以通过使用 `deferred.promise` 检索出来，那么实际上一个新的 `promise` 实例也被创建出来了。`promise` 对象的作用是，允许相关部分获得当 `deferred` 任务完成时的结果。

### 方法

1. `then(successCallback, errorCallback, notifyCallback)` – 不管 `promise` 已经被解析（拒绝）还是将要发生这些。一旦结果为可知的，`then` 都会调用一个成功或者失败的异步回调函数。这些回调函数被调用是只会被传递一个单独的参数：成功结果或是失败被拒绝的原因。此外

Additionally, the 在 `promise` 被解析（拒绝）之前，为了提供了一个进度指示 通知回调（`notify`）可以被调用0到若干次。这个方法返回了一个根据成功（或失败）回调函数返回结果来解析的新的 `promise`（除非返回的结果是一个 `promise`，而这个 `promise` 在这种情况下已经由存在于 `promise` 链的 `promise` 的 `value` 解析）。依然会通过 `notifyCallback` 方法的返回值发布通知。`promise` 不能由这个 `notifyCallback` 方法解析或者拒绝。

2. `catch(errorCallback)` - `promise.then(null, errorCallback)` 的简写
3. `finally(callback, notifyCallback)` - 虽然允许你观测一个 `promise` 的实现(拒绝)，但并不会修改最终值。这对于发布资源或者为那些需要做处理的被解析(拒绝)的 `promise` 做一些清理工作十分有用。

## 链接 Promise

由于调用 `promise` 下的 `then` 方法会返回一个源 `promise`，这使得创建一条 `promise` 链变得很容易：

```
promiseB = promiseA.then(function(result) {  
    return result + 1;  
});  
  
// 当promiseA被解析之后promiseB会紧接着被解析  
// 其值将是promiseA的加1的结果
```

这有可能会创建一个任意长度的链,因为一个 `promise` 可以被另一个 `promise` 解析(而这个 `promise` 将会被进一步的推迟解析)，也有可能在这条链的任意节点暂停/推迟解析。这将实现像 `$http` 的相应拦截器那样强有力的API成为可能。

## Kris Kowal's Q 与 \$q 之间的区别

以下是他们之间最主要的区别：

- `$q` 是通过 `$rootScope.Scope` 集成的，在Angular中作用域模型观测机制会更快的在你的 `model` 之间传播解析或拒绝，这避免不必要的浏览器重绘导致的UI闪烁。
- 尽管 `Q` 比 `$q` 有更多的特性，但是这也会带来增加更多字节的问题。虽然 `$q` 是微型的，但是它包含了大多数异步任务需要的所有重要功能。

## 测试部分

```
it('should simulate promise', inject(function($q, $rootScope) {

    var deferred = $q.defer();
    var promise = deferred.promise;
    var resolvedValue;

    promise.then(function(value) {

        resolvedValue = value;
    });

    expect(resolvedValue).toBeUndefined();

    // 模拟解析 promise

    deferred.resolve(123);

    // 注意！ then函数没有被同步调用。
    // 这是因为无论 promise 是被同步或是异步调用的，
    // 我们都希望它的 API 总是异步的
    expect(resolvedValue).toBeUndefined();

    // 使用$apply向 then方法传递 promise 解析。
    $rootScope.$apply();
    expect(resolvedValue).toEqual(123);

}));
```

## 依赖

```
$rootScope
```

## 用法

```
$q(resolver);
```

### 参数

参数	形式	具体
resolver	<code>function(function, function)</code>	函数是为了响应新创建 <code>promise</code> , 函数的第一个参数是用来解析 <code>promise</code> 的函数, 函数的第二个参数是用来拒绝 <code>promise</code> 的函数.

### 返回

`Promise` - 是新创建出的 `promise` .

## 方法

### 1. `defer()`;

创建一个新的 `deferred` 对象来表示一个将要在未来完成的任务.

### 返回

`Deferred` - 是一个 `deferred` 的新实例.

### 2. `reject(reason)`;

创建一个由被指定 `reason` 的拒绝而解析的新的 `promise` . 它的API将被放到 `promise` 链的前面, 所以如果你正在处理这个 `promise` 链的最后一个 `promise` , 你并不需要为此感到担心.

当将 `deferreds/promises` 和更为熟悉的 `try/catch/throw` 行为相比较时, 把 `reject` 当做Javascript里的 `throw` 关键字. 同样地如果你“捕获 `catch` ” `promise` 失败回调返回的 `error` 并且你想将此 `error` 掷到由当前 `promise` 派生出的 `promise` 中, 你将不得通过借由 `reject` 返回的一个拒绝构造来“重抛”这个 `error` .

```
promiseB = promiseA.then(function(result) {
  // 成功： 执行函数体内部的操作
  //          并用既有或新的结果解析 promiseB
  return result;

}, function(reason) {
  // 错误： 尽可能处理错误，
  //          并用 newPromiseOrValue 解析 promiseB,
  //          否则向promiseB转发拒绝

  if (canHandle(reason)) {
    // 处理错误并恢复
    return newPromiseOrValue;

  }
  return $q.reject(reason);
});
```

参数

参数	形式	具体
reason	*	常量, 信息, 例外或一个代表拒绝原因的对象.

返回

Promise 返回一个由 reason 拒绝的 promise .

4. when(value);

把一个 value 或者一个(第三方)可以使用 then 的 promise 包装到一个 \$q promise .这样做的好处是当你再处理一个对象时，你不用再考虑它是不是一个 promise ,也不必考虑某个外源的 promise 是否是值得信任的.

参数

参数	形式	具体
value	*	value 或一个 promise

返回

`Promise` - 返回由一个由验证过的 `value` 或一个 `promise` 包裹成的新的 `promise` .

## 5. `resolve(value);`

`when`的别名，与ES6保持一致.

### 参数

参数	形式	具体
<code>value</code>	*	<code>value</code> 或一个 <code>promise</code>

### 返回

`Promise` 返回由一个由验证过的 `value` 或一个 `promise` 包裹成的新的 `promise` .

## 6. `all(promises);`

将多个 `promise` 合并成一个 `promise` ,它的解析将会在所有输入的 `promise` 解析之后执行.

### 参数

参数	形式	具体
<code>promises</code>	<code>Array.&lt;Promise&gt;</code> <code>Object.&lt;Promise&gt;</code>	一个数组或者 <code>promises</code> 的哈希值.

### 返回

`Promise` 返回一个单独的 `promise` , 它将被一组数组/哈希 `value` 解析。在这个单独的 `promise` 中每一个 `value` 都会对应到 `promise` 数组/哈希的索引中. 如果参数中的任何一个 `promise` 被拒绝, 这将使由 `all` 方法返回的 `promise` 被同样的理由 (上面的拒绝) 拒绝掉.

## Angular\_1.4.3 API 服务篇 \$rootElement

- `ng` 模块中的服务

`$rootElement` 是Angular应用中的根元素。它或是 `ngApp` 声明处的元素，或是传入 `Angular.bootstrap` 方法中的元素。它代表应用的根元素。它也是应用的 `$injector` 服务被暴露的位置，并能用 `$rootElement.injector()` 方法检索到。



## Angular\_1.4.3 API 服务篇 \$rootScope

- `$rootScopeProvider`
- `ng` 模块中的服务

每个应用都有一个单独的根作用域。所有在这个应用的其他的的作用域都是它的子代作用域。作用域通过一个监听模型变化的机制来为在视图和模型之间提供分离功能。他们还可以提供事件的冒泡/捕获和和订阅功能。详情参见 `scope` 的开发文档。

## Angular\_1.4.3 API 服务篇 \$sce

`$sce` 是一个为 AngularJS 提供严格上下文模式 (Strict Contextual Escaping) 的服务

### 严格上下文模式

在Angular中，严格上下文模式 (SCE)用来将绑定的内容输出为该内容的安全值的形式。一个用法就是：我们将 `ng-bind-html` 控制的一个随意的 html 绑定内容转化为特定形式或者SCE形式的内容。

Angular 1.2.x以后，将 `$sce` 放进了启动文件中

注意：启动时，IE11以下浏览器的怪异模式是不支持Angular `SCE` 的。它允许使用 `()`表达式 执行任意语法。如果你想了解更多相关。你可以将你处于标准模式的文档中的头部的 `<!doctype html>` 移除。

SCE 会给编写代码带来两点好处: a.使用默认方式确保安全, b.审查诸如XSS，点击劫持等安全漏洞做也会变得容易

以下是一个特定形式的绑定用例:

html

```
<input ng-model="userHtml" aria-label="User input">
<div ng-bind-html="userHtml"></div>
```

请注意，`ng-bind-html` 是通过 `user` 绑定到 `userHtml` 控制上的。由于 SCE 失效，这个应用将会允许用户将任意的 HTML 渲染到 DIV 上。一个更为实际的情景是，需要渲染用户评论，博客文章的时候。（这里HTML是一个上下文环境，它渲染的由用户控制的输入会造成安全漏洞）

针对这个HTML案例，你可能会在客户端或者服务器端使用一个库，在HTML绑定到值上并渲染到文档之前矫正 它的不安全问题。

然而你如何确保你使用的库会在你使用双向绑定时生效（或者你的服务器返回的貌似安全的渲染）。你又将如何保证你不会不小心删除了矫正值的界线，或者重命名了一些属性/域，但是却忘记更新了矫正值绑定。

在默认情况下保证安全，你想要确保任何绑定都是不允许的，除非你能确定一些明确的表述它是对于在对应的上下文中使用一个值去绑定是安全的。然后你可以审核你的代码（一个简单的 `grep` 命令就可以）仅仅是为了保证你可以更简单的确定那些值是安全的-因为它们将从服务器端发送并被接收，通过你的库来校正等等。你可以组织你的代码库来帮助你完成这些工作-也许仅允许具体路径下的文件来完成这个工作。确保由代码暴露的内部API不会将任意的值认定为安全的，这就成了一个更易于管理任务。

在 Angular SCE 服务的案例中，使用 `$sce.trustAs`（或者像 `$sce.trustAsHtml` 等的简洁方法）来获得符合 SCE 或者指定的上下文环境的值。

## 它使如何工作的？

在特殊设置的上下文环境中。指令和代码相比于直接的值更倾向于绑定到 `$sce.getTrusted(context, value)` 的结果上。指令相比 `$parse` 监视属性的绑定更倾向使用 `$sce.parseAs`，因为它将执行 `$sce.getTrusted` 在非常量的字符场景之后。

举个例子，`ngBindHtml` 用了 `$sce.parseAsHtml(binding expression)` 方法，以下是代码 (稍微简化了一些)

javascript

```
var ngBindHtmlDirective = ['$sce', function($sce) {
  return function(scope, element, attr) {
    scope.$watch($sce.parseAsHtml(attr.ngBindHtml), function(value) {
      element.html(value || '');
    });
  };
}];
```

## 对加载文档的影响

该服务应用于 `ng-include` 指令的效果与 指令内部 `templateUrl` 指定的模板效果一样好。

默认情况下，Angular 仅仅通过应用内部一致的域或协议来加载模板。可以通过在模板的URL上调用 `$sce.getTrustedResourceUrl` 方法来实现这个需求。或者加载来自其他域或协议的模板，要么将他们加入白名单，要么将它们包装成一个可以信任的值。

请注意：浏览器的同源策略(Same Origin Policy)和跨域资源共享（Cross-Origin Resource Sharing - CORS）策略的应用会进一步限制模板的加载的成功。这意味着，如果没有正确的CORS策略，从一个不同的域下加载模板，是不会兼容所有浏览器的。同样地，从 `file://` 路径下加载的文档也会有一些浏览器不支持。

## 这让人感觉得不偿失

需要谨记 SCE 仅可应用于插值表达式中。

如果你的表达式是恒定不变的字符。它将自动地被信任并且不需要在它们身上调用 `$sce.trustAs` 就可以工作了（记得引用 `ngSanitize` 模块）。

example

```
<div ng-bind-html="'<b>implicitly trusted</b>'"></div>
```

此外，`a[href]` 和 `img[src]` 自动校正它们的路径，并不会通过 `$sce.getTrusted` 传递它们。SCE 此处不会发挥效用。

引入 `$sceDelegate` 合理的设置默认值来允许你从你应用的域里加载模板到 `ng-include` 中，而不需要知道SCE。它会阻止从其他域加载模板或者从 https 服务的文档向 http 加载文档。你可以通过设置你自己习惯的白名单和黑名单去匹配一些URL来改变这些。

这显著地降低了开销。相比于在应用启动一段时间之后指定安全策略它真是容易的多，花费很少的时间就拥有了一个安全的可审计验证的更为方便的应用。

## 支持那些收信人的上下文形式？

上下文	注意事项
<code>\$sce.HTML</code>	对于应用安全的 HTML 源。 <code>ngBindHtml</code> 指令使用他来绑定。如果遇到一个不安全的值并且 <code>\$sanitize</code> 模块当前可使用，那么它将矫正这个值而不是抛出错误。
<code>\$sce.CSS</code>	对于应用安全的 CSS 源。当前未使用。可以在你的指令了随意使用它。
<code>\$sce.URL</code>	对于作为链接安全的URL。当前未使用 ( <code>&lt;a href=</code> 和 <code>&lt;img src=</code> 校正它们的路径并且不会构成一个SCE上下文环境。)
<code>\$sce.RESOURCE_URL</code>	不仅仅是对于作为链接安全的URL。并且链接引入到你应用中的内容也是安全的，例如引入 <code>ng-include</code> 或是 <code>src / ngSrc</code> 绑定的 IMG 外的其他标签 (e.g. IFRAME, OBJECT, etc.) 需要注意的是 <code>\$sce.RESOURCE_URL</code> 相比 <code>\$sce.URL</code> 做了更为强大的声明。因此 <code>\$sce.RESOURCE_URL</code> 信任的上下文要求值可以用于任何地方，而 <code>\$sce.URL</code> 的则不行。
<code>\$sce.JS</code>	可以在你的应用上下文中安全执行的 Javascript。当前未使用。可以在你的指令了随意使用它。

## 格式化在资源路径白、黑名单中的项目

这些数组中的任意一个元素必须符合下面描述的其中一项：

- `self`
  - 特殊的字符串， `self` ，可用于匹配校验使用了相同协议的应用文档的相同域下的所有URL。
- 字符串 (除了特殊值 `self` )
  - 字符串符合完全标准化/资源被测试的绝对路径。(substring 匹配不够好.)  
有两个通配符 - `*` 和 `**` . 所有其他字符都可以和它们匹配。 `*` : 匹配没有，或者多次出现的任意字符但不包括以下六个字符中的任意一个：  
`:` , `/` , `.` , `?` , `&` 和 `;` . 在白名单的设置中它使十分有用的。  
`**` : 匹配没有，或者多次出现的任意字符。像这样，他不适合匹配格式和域名。因为它将匹配太多(e.g. `http://**.example.com/` 将会匹配 `http://evil.com/?ignore=.example.com/` 这并不是我们所期望的)  
它一般用于一个路径的结尾。(e.g. `http://foo.example.com/templates/**` ).

正则表达式 (看下面的警告)

警告: 虽然正则表达式威力十足并且提供了极大的灵活性, 他们的句法 (和所有不可避免的逃逸)使它们更难于维护。 它十分容易当某个人更新了一个复杂的表达式时十分偶然地引入一个错误 (恕我直言, 所有的正则表达式都应该有良好的测试覆盖率)。例如 `.` 在正则中的用途仅在很少的情况下会被正确使用。

A `.` 字符在正则中会在匹配格式或者子域为 `a :` 或者 字面量 `.` 的时候使用。很可能不如我们所愿。 我们强烈推荐使用字符串模式, 只有在没有其他办法的时候使用正则表达式。

正则表达式必须是一个 `RegExp` 的实例。(即, 不是一个字符串。)它匹配了整个标准化/资源被测试的绝对路径 (甚至当 `RegExp` 没有 `^` 和 `$` 的时候也是这样)。另外, 出现在正则表达式中的任何标志 (诸如多行`m`、全局`g`、忽略大小写`i`) 都会被忽略。

如果你是从其他模板引擎生成你的JavaScript (不建议如此, 例如 #4006 issue), 记得避免你的正则表达式 (要意识到你可能需要根据你的模板引擎和你插值的方式逃避的多个级别。) 请使用您的平台的逃逸机制, 因为它可能是编码自己以前不够好。例如 Ruby 有 `Regexp.escape(str)`, Python 有 `re.escape`。 Javascript 内部则缺少一个类似的机制用于逃逸。看一看Google Closure library's

`goog.string.regExpEscape(s)`。

参阅 `$sceDelegateProvider` 的例子。

## 为你展示一个使用**SCE**的例子

`index.html`

```

<div ng-controller="AppController as myCtrl">
  <i ng-bind-html="myCtrl.explicitlyTrustedHtml" id="explicitlyTrusted">
    <b>用户评论</b><br>
    默认情况下，如果在 $sanitize 可用，不是明确可信的HTML（例如 Alice的评论）
    如果$sanitize不可用，这将导致一个错误，而不是开发。

    <div class="well">
      <div ng-repeat="userComment in myCtrl.userComments">
        <b>{{userComment.name}}</b>:
        <span ng-bind-html="userComment.htmlComment" class="htmlComment">
          <br>
        </div>
      </div>
    </div>
  </div>

```

script.js

```

angular.module('mySceApp', ['ngSanitize'])
.controller('AppController', ['$http', '$templateCache', '$sce',

function($http, $templateCache, $sce) {

  var self = this;

  $http.get("test_data.json", {cache: $templateCache}).success(function(data) {
    self.userComments = data;
  });

  self.explicitlyTrustedHtml = $sce.trustAsHtml(
    '<span onmouseover="this.textContent=&quot;Explicitly trusted HTML without sanitization.&quot;">Hover over this text.</span>');
}]);

```

test\_data.json

```
[
  { "name": "Alice",
    "htmlComment":
      "<span onmouseover='this.textContent=\\\"PWN3D!\\\"'>Is <i>anyo
  },
  { "name": "Bob",
    "htmlComment": "<i>Yes!</i>  Am I the only other one?"
  }
]
```

protractor.js

```
describe('SCE doc demo', function() {
  it('should sanitize untrusted values', function() {
    expect(element.all(by.css('.htmlComment')).first().getInnerHtml()
      .toBe('<span>Is <i>anyone</i> reading this?</span>');
  });

  it('should NOT sanitize explicitly trusted values', function() {
    expect(element(by.id('explicitlyTrustedHtml')).getInnerHtml())
      .toBe('<span onmouseover="this.textContent=&quot;Explicitly trust
      'sanitization.&quot;">Hover over this text.</span>');
  });
});
```

## 我也以在应用开始生效时禁用 SCE 吗？

当然可以。然而，我们强烈建议你不要这么做。SCE 将给予你非常多的安全益处和很少的代码开销。掌控一个 SCE失效的应用更为困难。不论是自己去验证安全，或者在之后的阶段启动SCE。某些情况下禁用SCE是合理的，有很多在引入SCE前就编写了的代码并且你在迁移他们到一个模块中。

所以，以下是如何在应用开始生效时禁用 SCE:

javascript



```
angular.module('myAppWithSceDisabledmyApp', []).config(function($sceProvider)
    // 应用开始生效时禁用 SCE。 仅供演示！
    // 不再新项目中使用。
    $sceProvider.enabled(false);
});
```

## 用法

`$sce`

## 方法

### 1. `isEnabled()`

返回一个布尔值来说明 SCE 是否启用了

返回

`Boolean` - 如果启用则返回 `true`，其他情况则返回 `false`，如果你想设置这个值，你可以在模块初始化配置时使用 `$sceProvider` 来完成。

### 2. `parseAs(type, expression)`

将Angular 表达式转化成函数，这像是当表达式是常量时的 `$parse`。除此之外，它通过调用 `$sce.getTrusted(type, result)` 来包装这个表达式。

参数

参数	形式	详细
<code>type</code>	<code>string</code>	输出的结果使用哪一种 SCE 上下文种类
<code>expression</code>	<code>string</code>	用于编译的字符串表达式

返回

`function(context, locals)` 一个代表了编译表达式的函数:

`context` – `{object}` – 一个针对字符串形式的表达式的评估结果的对象(通常为一个 `scope` 对象). `locals` – `{object=}` – 局部变量上下文对象, 可用于覆盖上下文范围内的值。

### 3. `trustAs(type, value)`

代理给 `$sceDelegate.trustAs` 。像这样, 返回一个受Angular信任的在规定的严格的上下文环境中逸出使用 (如 `ng-bind-html` , `ng-include` , 任何的 `src` 属性的插值, 任何 `dom` 事件绑定的属性插值如 `onclick` 等) 使用所提供的值。了解 \*\$sce 关于启用严格的语境转义。

#### 参数

参数	形式	详细
<code>type</code>	<code>string</code>	指定使用的上下文类型。如 <code>url</code> , <code>resourceUrl</code> , <code>html</code> , <code>js</code> 或 <code>css</code> .
<code>value</code>	<code>*</code>	被认为是可信/安全的值。

#### 返回

\* 在那些Angular期望的得到的 `$sce.trustAs()` 返回值的提供值

### 4. `trustAsHtml(value)`

简洁方法。 `$sce.trustAsHtml(value)` → `$sceDelegate.trustAs($sce.HTML, value)`

#### 参数

参数	形式	详细
<code>value</code>	<code>*</code>	<code>trustAs</code> 的值.

#### 返回

\* 一个可以传给 `$sce.getTrustedHtml(value)` 后获得原值的对象 (自定义的指令仅能接收 常量表达式或者通过 `$sce.trustAs` 返回的值)

### 5. `trustAsUrl(value)`

简洁方法。 `$sce.trustAsUrl(value)` → `$sceDelegate.trustAs($sce.URL, value)`

#### 参数

参数	形式	详细
value	*	trustAs 的值.

#### 返回

\* 一个可以传给 `$sce.getTrustedUrl(value)` 后获得原值的对象（自定义的指令仅能接收 常量表达式或者通过 `$sce.trustAs` 返回的值）

## 6. `trustAsResourceUrl(value);`

简洁方法。 `$sce.trustAsResourceUrl(value)` → `$sceDelegate.trustAs($sce.RESOURCE_URL, value)`

#### 参数

参数	形式	详细
value	*	trustAs 的值.

#### 返回

\* 一个可以传给 `$sce.getTrustedResourceUrl(value)` 后获得原值的对象（自定义的指令仅能接收 常量表达式或者通过 `$sce.trustAs` 返回的值）

## 7. `tgetTrusted(type, maybeTrusted);`

代理给 `$sceDelegate.getTrusted` 像这样, 拿到一个 `$sce.trustAs()` 调用的结果, 如果查询的上相问类型是创建类型的超类型则返回最初传入的参数。如果不成立, 抛出一个例外。

#### 参数

参数	形式	详细
type	string	指定上下文的类型.
maybeTrusted	*	先被调用的 <code>\$sce,trustAs</code> 的结果。

## 返回

\* 如果在上下文环境中是合法的返回的则是开始提供给 `$sce.trustAs` 的参数。否则抛出一个例外。

## 8. `tgetTrustedHtml(value);`

简洁方法。 `$sce.getTrustedHtml(value)` → `$sceDelegate.getTrusted($sce.HTML, value)`

## 参数

参数	形式	详细
value	*	传给 <code>\$sce.getTrusted</code> 的值.

## 返回

\* `$sce.getTrusted($sce.HTML, value)`的返回值

## 9. `getTrustedCss(value);`

简洁方法。 `$sce.getTrustedCss(value)` → `$sceDelegate.getTrusted($sce.CSS, value)`

## 参数

参数	形式	详细
value	*	传给 <code>\$sce.getTrusted</code> 的值.

## 返回

\* `$sce.getTrusted($sce.CSS, value)`的返回值

## 10. `getTrustedUrl(value);`

简洁方法。 `$sce.getTrustedUrl(value)` →  
`$sceDelegate.getTrusted($sce.URL, value)`

#### 参数

参数	形式	详细
value	*	传给 <code>\$sce.getTrusted</code> 的值.

#### 返回

\* `$sce.getTrusted($sce.URL, value)`的返回值

## 10. `getTrustedResourceUrl(value);`

简洁方法。 `$sce.getTrustedResourceUrl(value)` →  
`$sceDelegate.getTrusted($sce.RESOURCE_URL, value)`

#### 参数

参数	形式	详细
value	*	传给 <code>\$sce.getTrusted</code> 的值.

#### 返回

\* `$sce.getTrusted($sce.RESOURCE_URL, value)`的返回值

## 11. `getTrustedJs(value);`

简洁方法。 `$sce.getTrustedJs(value)` →  
`$sceDelegate.getTrusted($sce.JS, value)`

#### 参数

参数	形式	详细
value	*	传给 <code>\$sce.getTrusted</code> 的值.

#### 返回

\* `$sce.getTrusted($sce.JS, value)`的返回值

## 12. `parseAsHtml(expression);`

简洁方法。 `$sce.parseAsHtml(expression string) → $sce.parseAs($sce.HTML, value)`

### 参数

参数	形式	详细
type	string	输出的结果使用哪一种 SCE 上下文种类
expression	string	用于编译的字符串表达式

### 返回

`function(context, locals)` 一个代表了编译表达式的函数:

`context` - `{object}` - 一个针对字符串形式的表达式的评估结果的对象(通常为一个 `scope` 对象). `locals` - `{object=}` - 局部变量上下文对象, 可用于覆盖上下文范围内的值。

## 13. `parseAsCss(expression);`

简洁方法。 `$sce.parseAsCss(value) → $sce.parseAs($sce.CSS, value)`

### 参数

参数	形式	详细
type	string	输出的结果使用哪一种 SCE 上下文种类
expression	string	用于编译的字符串表达式

### 返回

`function(context, locals)` 一个代表了编译表达式的函数:

`context` - `{object}` - 一个针对字符串形式的表达式的评估结果的对象(通常为一个 `scope` 对象). `locals` - `{object=}` - 局部变量上下文对象, 可用于覆盖上下文范围内的值。

## 14. `parseAsUrl(expression);`

简洁方法。 `$sce.parseAsUrl(value)` → `$sce.parseAs($sce.URL, value)`

### 参数

参数	形式	详细
type	string	输出的结果使用哪一种 SCE 上下文种类
expression	string	用于编译的字符串表达式

### 返回

`function(context, locals)` 一个代表了编译表达式的函数:

`context` – {object} – 一个针对字符串形式的表达式的评估结果的对象(通常为一个 `scope` 对象). `locals` – {object=} – 局部变量上下文对象, 可用于覆盖上下文范围内的值。

## 15. `parseAsResourceUrl(expression);`

简洁方法。 `$sce.parseAsResourceUrl(value)` → `$sce.parseAs($sce.RESOURCE_URL, value)`

### 参数

参数	形式	详细
type	string	输出的结果使用哪一种 SCE 上下文种类
expression	string	用于编译的字符串表达式

### 返回

`function(context, locals)` 一个代表了编译表达式的函数:

`context` – {object} – 一个针对字符串形式的表达式的评估结果的对象(通常为一个 `scope` 对象). `locals` – {object=} – 局部变量上下文对象, 可用于覆盖上下文范围内的值。

## 16. `parseAsJs(expression);`

简洁方法。 `$sce.parseAsJs(value)` → `$sce.parseAs($sce.JS, value)`

## 参数

参数	形式	详细
type	string	输出的结果使用哪一种 SCE 上下文种类
expression	string	用于编译的字符串表达式

## 返回

`function(context, locals)` 一个代表了编译表达式的函数:

`context` - `{object}` - 一个针对字符串形式的表达式的评估结果的对象(通常为一个 `scope` 对象). `locals` - `{object=}` - 局部变量上下文对象, 可用于覆盖上下文范围内的值。



## Angular\_1.4.3 API 服务篇 \$sceDelegate

- \$sceDelegateProvider
- ng模块中的服务

`$sceDelegate` 是一个基于 `$sce` 为 AngularJS 提供SCE 的服务。

通常情况下，你可以自定义 SCE 来初始化或者覆盖 `$sceDelegate` 来代替 `$sce` 服务在 AngularJS 中工作。这是因为虽然 `$sce` 提供了众多的简洁写法的方法，但你真的只需要覆盖3个核心的函数（`trustAs`，`getTrusted` 和 `valueOf`）就可以改变他们的工作方式，原因在于 `$sce` 将这些操作委托给 `$sceDelegate` 去做了。

调用 `$sceDelegateProvider` 来初始化这个服务。

默认的 `$sceDelegate` 实例将会制定出有一点蛋疼的规则，虽然你可以在应用启动后覆盖它来改变 `$sce` 的行为，但是常见的解决办法是调用

`$sceDelegateProvider`，而不是通过设置你自己的白名单和黑名单来使用信任的 URL 加载模板这类 AngularJS 资源。详请参见

`$sceDelegateProvider.resourceUrlWhitelist` 和  
`$sceDelegateProvider.resourceUrlBlacklist`

## 用法

`$sceDelegate()`

## 方法

### 1. `trustAs(type, value)`

返回一个受Angular信任的在规定的严格的上下文环境中逸出使用（如 `ng-bind-html`，`ng-include`，任何的 `src` 属性的插值，任何 `dom` 事件绑定的属性插值如 `onclick` 等）使用所提供的值。了解 \*\$sce 关于启用严格的语境转义。

参数

参数	形式	具体
type	string	指定使用的上下文类型。如 <code>url</code> , <code>resourceUrl</code> , <code>html</code> , <code>js</code> 或 <code>css</code> .
value	*	被认为是可信/安全的值。

返回

\* 在那些Angular期望的得到的 `$sce.trustAs()` 返回值的\*\*地方\*\*提供值

## 2. `valueOf(value)`

如果传入的参数已经被之前调用过的 `$sceDelegate.trustAs` 那么则返回已经被传入过 `$sceDelegate.trustAs` 的值。如果参数不是一个被 `$sceDelegate.trustAs` 返回的值，则返回它自己。

参数

参数	形式	具体
value	*	先前调用 <code>\$sceDelegate.trustAs</code> 的结果，或者任何其他值。

返回

\* 在那些Angular期望的得到的 `$sce.trustAs()` 返回值的\*\*地方\*\*提供值。如果值是这样一个调用的结果，那么它将是最初提供给 `$sceDelegate.trustAs` 的值。否则返回的值不变。

## 3. `getTrusted(type, maybeTrusted)`

获取一个 `$sceDelegate.trustAs` 调用的结果，并在查询的上下文类型是创建类型的超集的情况下返回最初提供的值。如果不满足这个条件，则抛出一个异常。

参数

参数	形式	具体
type	string	这个值被用于何种类型的上下文
maybeTrusted	*	先前调用 <code>\$sceDelegate.trustAs</code> 的结果。

返回 \* - 在上下文合法的情况下，返回的是最先提供给 `$sceDelegate.trustAs` 的值。否则，将抛出一个异常

## Angular\_1.4.3 API 服务篇 \$templateCache

- `ng` 模块中的服务

在模板第一次被引用时,它会被载入到模板缓存中以便快速的检索。你可以直接使用一个 `script` 标签来添加一个 模板, 或者直接通过 `$templateCache` 服务来达到相同的效果。

通过 `script` 标签添加:

html

```
<script type="text/ng-template" id="templateId.html">
  <p>This is the content of the template</p>
</script>
```

注意: `script` 标签包含的模板并不需要 `document` 的头部, 但它必须是 `$rootElement` 的后代(IE, 带有 `ng-app` 属性的元素), 否则这个模板将被忽略掉。

通过 `$templateCache` 服务添加:

javascript

```
var myApp = angular.module('myApp', []);
myApp.run(function($templateCache) {
  $templateCache.put('templateId.html', 'This is the content of the');
});
```

在检索模板之后, 你就可以在你的 HTML 里轻松的使用了。

html

```
<div ng-include=" 'templateId.html' "></div>
```

或者通过 Javascript 得到它:

javascript

```
$templateCache.get('templateId.html')
```

请看 [\\$cacheFactory](#) .

# Angular\_1.4.3 API 服务篇

## \$templateRequest

- `ng` 模块中的服务

`$templateRequest` 服务

`$templateRequest` 服务使用 `$http` 服务下载模板，并在之后进行安全检测。如果成功，则内容会被存储到 `$templateCache` 中。如果HTTP请求失败或者该请求的响应数据为空，会抛出一个编译（`$compile`）错误（这个行为可以通过将函数的第二个参数设置为 `true` 来阻止）。

需要注意的是，`$templateCache` 的内容是可以信任的，因此当 `tpl` 是以字符串形式出现的时候，或者通过 `$templateCache` 作为入口时，是可以不必去调用 `$sce.getTrustedUrl(tpl)` 来验证的。

## 用法

```
$templateRequest(tpl, [ignoreRequestError]);
```

### 参数

参数	形式	详细
<code>tpl</code>	<code>string</code> <code>TrustedResourceUrl</code>	HTTP 请求的模板路径(URL)
<code>ignoreRequestError</code> (可选)	<code>boolean</code>	是否忽略请求失败或者模板为空的情况

### 返回

`promise` - 一个表示通过给定的路径请求得到的 HTTP 相应数据的 `promise`

## 属性

`totalPendingRequests` - `number` - 所有被需求下载模板的总数

## Angular\_1.4.3 API 服务篇 \$timeout

- ng 模块中的服务

`window.setTimeout` 的Angular封装，这个 `fn` 函数被封装成了一个 `try / catch` 块并且授 `$ExceptionHandler` 服务以任何例外。

调用 `$timeout` 的返回值是一个 `promise`，这个 `promise` 将会在延时已经结束时被解析，超时函数（如果有的话）被执行了。

退出一个超时请求，调用 `$timeout.cancel(promise)`。如果在测试中你可以使用 `$timeout.flush()` 来同步刷新 `deferred` 函数的队列。而如果你仅仅想要得到一个在一个指定延时时间之后会被解析的 `promise`，你可以仅仅调用 `$timeout` 而不传入 `fn` 函数。

### 用法

```
$timeout([fn], [delay], [invokeApply], [Pass]);
```

#### 参数

参数	形式	详细
fn(可选)	<code>function()</code> =	已经将在延时之后被执行的函数。
delay(可选)	<code>number</code>	以毫秒计的延时时间。(默认值: 0)
invokeApply(可选)	<code>boolean</code>	如果设置为 <code>false</code> 则跳过模型的脏值检测，否则将在 <code>fn</code> 内调用 <code>\$apply</code> 块。(默认值: <code>true</code> )
Pass(可选)	*	函数执行的附加参数

#### 返回

`Promise` - `promise` 将会在超时达成后被解析，它的值会被 `fn` 的返回值解析。

### 方法

`cancel([promise]);` - 取消一个与 `promise` 相关联的任务。这个结果会导致，`promise` 会被拒绝解析。

参数

参数	形式	详细
<code>promise</code> (可选)	<code>promise</code>	<code>\$timeout</code> 函数返回的 <code>promise</code>

返回

`boolean` - 如果任务没有被执行就被成功取消了，则会返回 `true` 。

## Angular\_1.4.3 API 服务篇 \$interval

- ng 模块中的服务

Angular对 `window.setInterval` 的封装。 `fn` 函数将在每次延时的时候执行。一个注册的间隔函数的返回值是一个 `promise`。这个 `promise` 将会在interval每次调用的时候得到广播，并且在计数迭代之后被解析，或者在未指定次数的情况下既执行无数次时解析。这个通知的值将是已运行迭代的次数。如果想终端一个interval，则调用 `$interval.cancel(promise)`。

测试中你可以使用 `$interval.flush(millis)` 设置其中的 `millis` 为毫秒时间来到达指定的时间点，并且会触发在此过程中任何的函数。

注意: 用此服务创建的interval必须要在完成之后被明文销毁。特别地，在控制器的作用域和指令的元素被销毁时， `interval` 也不会被自动销毁。你需要将此纳入考虑之中，确保在适合的时间退出 `interval`。详述之后的例子将会 介绍何时与怎样去做这些处理。

### 用法

```
$interval(fn, delay, [count], [invokeApply], [Pass]);
```

#### 参数

参数	形式	详细
fn	<code>function()</code>	一个将被反复调用的函数
delay	<code>number</code>	毫秒记的间隔时间
count (可选)	<code>number</code>	循环的次数.如果不传入或者为0，那么\$interval将会无限循环. (默认值: 0)
invokeApply (可选)	<code>boolean</code>	如果设置为 <code>true</code> 则跳过脏值检测，否则将在 <code>fn</code> 中执行 <code>\$apply</code> (默认值: <code>true</code> )
Pass (可选)	<code>*</code>	函数执行的附加参数

#### 返回

`promise` - 一个每次迭代都会被通知的 `promise`。



## 方法

`cancel([promise]);` - 取消一个与 `promise` 相关的任务。

### 参数

参数	形式	详细
promise (可选)	<code>promise</code>	通过 <code>\$interval</code> 函数返回。

### 返回

`boolean` - 如果任务被成功取消则返回 `true` 。

## 例子

html

```
<div>
  <div ng-controller="ExampleController">

    <label>Date format: <input ng-model="format"></label> <hr/>
    Current time is: <span my-current-time="format"></span>
    <hr/>

    Blood 1 : <font color='red'>{{blood_1}}</font>
    Blood 2 : <font color='red'>{{blood_2}}</font>

    <button type="button" data-ng-click="fight()">Fight</button>
    <button type="button" data-ng-click="stopFight()">StopFight</button>
    <button type="button" data-ng-click="resetFight()">resetFight</button>
  </div>
</div>
```

javascript

```
angular.module('intervalExample', [])
.controller('ExampleController', ['$scope', '$interval',
```

```
function($scope, $interval) {

    $scope.format = 'M/d/yy h:mm:ss a';
    $scope.blood_1 = 100;
    $scope.blood_2 = 120;

    var stop;
    $scope.fight = function() {
        // 如果$scope.fight进行中, 则不会开启新的 $scope.fight
        if ( angular.isDefined(stop) ) return;

        stop = $interval(function() {
            if ($scope.blood_1 > 0 && $scope.blood_2 > 0) {
                $scope.blood_1 = $scope.blood_1 - 3;
                $scope.blood_2 = $scope.blood_2 - 4;
            } else {
                $scope.stopFight();
            }
        }, 100);
    };

    $scope.stopFight = function() {
        if (angular.isDefined(stop)) {
            $interval.cancel(stop);
            stop = undefined;
        }
    };

    $scope.resetFight = function() {
        $scope.blood_1 = 100;
        $scope.blood_2 = 120;
    };

    $scope.$on('$destroy', function() {
        // 保证interval已经被销毁
        $scope.stopFight();
    });
}

// 用工厂方法注册一个 'myCurrentTime' 指令。
// 我们注入 $interval 服务和源自 DI 的 dateFilter 服务。
.directive('myCurrentTime', ['$interval', 'dateFilter',
```

```
function($interval, dateFilter) {

    // 返回指令链接的函数。（不需要编译函数）
    return function(scope, element, attrs) {
        var format, // 数据格式
            stopTime; // 以便我们可以取消时间更新

        // 用于更新 UI
        function updateTime() {
            element.text(dateFilter(new Date(), format));
        }

        // 监视表达式，并在改变时更新UI。
        scope.$watch(attrs.myCurrentTime, function(value) {
            format = value;
            updateTime();
        });

        stopTime = $interval(updateTime, 1000);

        // 监听 DOM 销毁(去除)事件，并取消下一次的UI更新
        // 是为了在 DOM 被删除之后防止再次更新次数。
        element.on('$destroy', function() {
            $interval.cancel(stopTime);
        });
    }
}]]);
```

## Angular\_1.4.3 API 服务篇 \$window

- ng 模块中的服务

这是一个浏览器端 `window` 对象的引用. 由于Javascript 中的 `window` 是一个全局变量, 它会给可测试性带来问题. 在Angular中我们总会通过 `$window` 服务来引用它, 因此在测试总它可以被改写、删除或模拟。

在下面的例子中, 表达式可以像定义一个 `ngClick` 指令一样, 在当前作用域被严格的评估. 因此, 在这种依赖于一个全局变量表达式中, 不会因为不经意的编码而带来风险。

javascript and html

```
angular.module('windowExample', [])

.controller('ExampleController', ['$scope', '$window', function($scope) {
    $scope.greeting = 'Hello, World!';
    $scope.doGreeting = function(greeting) {
        $window.alert(greeting);
    };
}]);
```

```
<div ng-controller="ExampleController">
  <input type="text" ng-model="greeting" aria-label="greeting" />
  <button ng-click="doGreeting(greeting)">ALERT</button>
</div>
```

protractor

```
it('should display the greeting in the input box', function() {  
  
    element(by.model('greeting')).sendKeys('Hello, E2E Tests');  
    // If we click the button it will block the test runner  
    // element(':button').click();  
});
```

## Angular\_1.4.3 API 服务篇 \$document

- `ng` 模块中的服务

一个对于浏览器中的 `window.document` 对象的[jQuery](#) 或 [jqLite](#) 封装。

### 依赖

`$window`

### 例子

html

```
<div ng-controller="ExampleController">

  <p>$document title: <b ng-bind="title"></b></p>
  <p>window.document title: <b ng-bind="windowTitle"></b></p>

</div>
```

javascript

```
angular.module('documentExample', [])

.controller('ExampleController', ['$scope', '$document',
  function($scope, $document) {

    $scope.title = $document[0].title;
    $scope.windowTitle = angular.element(window.document)[0].title;

  }]);
```

## <a>

- Angular@1.4.7
- `ng` 模块中的指令

修改了 `html` 中 `A` 标签的默认行为，以阻止当 `href` 属性为空时触发的默认行为。

这个修改会让使用 `ngClick` 指令创建的简写链接不会不会改变 `location` 或者引起页面的重载。如 `<a href="" ng-click="list.addItem()">Add Item</a>`

## 指令信息

这个指令的执行优先级为0级

## 用法

像个元素（标签）一样使用

`html`

```
<a>  
...  
</a>
```

## ng-app

- Angular@1.4.7
- `ng` 模块中的指令

使用这个指令来自动启动（auto-bootstrap）一个 AngularJS 应用。`ngApp` 指令指定了一个应用的根元素并且通常被置于距离页面根元素比较近的地方。例如，放在 `<body>` 或者 `<html>` 标签上。

每一个 HTML 文档只能自动启动一个 AngularJS 应用。页面中第一个被找到的存放 `ngApp` 的标签会被定义为这个应用的根元素来自动启动这个应用。要是想在一个 HTML 文档中启动多个应用，则你必须使用 `angular.bootstrap` 来手动启动他们。而且 AngularJS 之间不能互相嵌套。

你可以指定一个 AngularJS 模块用作一个应用的根模块。当应用启动时，这个模块将会被载入到 `$injector`（注射器）。它应该包含了应用所需的代码，或者它依赖的其他模块包含了这些代码。可以查看 `angular.module` 来获取更多信息。

下面的例子里，如果 `ngApp` 指令没有被设定到 `html` 元素上，则文档就不会执行编译。`AppController` 就不会被实例化出来，因此 `{{ a + b }}` 也不会被解析成 `3`。

`ngApp` 是最简单、最常用的方式来启动一个应用。

index.html

```
<div ng-controller="ngAppDemoController">
  I can add: {{a}} + {{b}} = {{ a+b }}
</div>
```

script.js

```
angular.module('ngAppDemo', []).controller('ngAppDemoController', function($scope) {
  $scope.a = 1;
  $scope.b = 2;
});
```



如果使用 `ngStrictDi`（严格的依赖注入），会像下面这样：

index.html

```
<div ng-app="ngAppStrictDemo" ng-strict-di>
  <div ng-controller="GoodController1">
    I can add: {{a}} + {{b}} =  {{ a+b }}

    <p>这里通过显式的标注形式渲染（render）出了结果，说明了控制器实例化成功
  </div>

  <div ng-controller="GoodController2">
    Name: <input ng-model="name"><br />
    Hello, {{name}}!

    <p>这里通过显式的标注形式渲染（render）出了结果，说明了控制器实例化成功
  </div>

  <div ng-controller="BadController">
    I can add: {{a}} + {{b}} =  {{ a+b }}

    <p>这个控制器不能被实例化是由于使用了自动地函数标注（即严格模式下会失败
    。因此这一部分的内容没有被插值渲染出来，而且还会在你浏览器的控制台报
    </p>
  </div>
</div>
```

script.js

```
angular.module('ngAppStrictDemo', [])
// BadController 实例化失败,
// 是由于使用了自动地函数标注儿没有使用明确的标注
// 译: 如果不使用 `ngStrictDi` 这种注入方式也是可以的, 只不过代码压缩的时候
.controller('BadController', function($scope) {
  $scope.a = 1;
  $scope.b = 2;
})
// 不像 BadController, GoodController1 and GoodController2 并不会实例化
// 因为分别使用了明确数组形式和 注入 (`$inject`) 属性的方式
using the array style and $inject property, respectively.
.controller('GoodController1', ['$scope', function($scope) {
  $scope.a = 1;
  $scope.b = 2;
}])
.controller('GoodController2', GoodController2);
function GoodController2($scope) {
  $scope.name = "World";
}
GoodController2.$inject = ['$scope'];
```

#### style.css

```
div[ng-controller] {
  margin-bottom: 1em; padding: .5em; border: 1px solid; border-radius: 4px;
}
div[ng-controller^=Good] {
  border-color: #d6e9c6; background-color: #dff0d8; color: #3c763c;
}
div[ng-controller^=Bad] {
  border-color: #ebccd1; background-color: #f2dede; color: #a94442;
  margin-bottom: 0;
}
```

## 指令信息

这个指令的执行优先级为0级

## 用法

以元素属性的形式使用:

```
<ANY
  ng-app="angular.Module"
  [ng-strict-di="boolean"]>
  ...
</ANY>
```

## 参数

参数	形式	详细
ng-app	angular.Module	一个可以加载的应用模块名
ngStrictDi (可选)	boolean	如果这个属性被设置在了应用的元素上，则注入模式将被设置为严格依赖注入的模式。

这意味着没用使用显示函数标注的应用将不能成功的调用函数（而且这种形式也不利于压缩）， 严格的方式也可以作为依赖注入（Dependency Injection）的描述指引和有助于追查错误的根源的调试信息。

## ng-bind-html

- Angular@1.4.7
- `ng` 模块中的指令

解析表达式并用一种安全的方式将得到的 HTML 插入到元素中。默认情况下，得到的 HTML 内容会使用 `$sanitize` 服务进行校验。为了可以使用 `ng-bind-html`，需要确保 `$sanitize` 是可用的，举个例子，在你的模块依赖中引入 `ngSanitize`（它并不在 Angular 的核心代码中）。为了在你的模块中依赖 `ngSanitize`，你需要再你的应用中引入 "angular-sanitize.js" 这个文件。

当你知道某些值是安全的，你也可以跳过这个校验。想要做到这一点需要通过 `$sce.trustAsHtml` 绑定信任的值。可以在 Strict Contextual Escaping (SCE) 下查看例子（译：在服务里的 `ng.$sce` 中）

注意：如果 `$sanitize` 服务不能使用并且绑定的值也没有被明确地信任，你会获得以个意外（而不是一个漏洞）

## 指令信息

这个指令的执行优先级为0级

## 用法

用作属性：

```
<ANY ng-bind-html="expression">
...
</ANY>
```

## 参数

参数	形式	详细
<code>ngBindHtml</code>	<code>expression</code>	需要解析的表达式

## 例子

index.html

```
<div ng-controller="ExampleController">
  <p ng-bind-html="myHTML"></p>
</div>
```

script.js

```
angular.module('bindHtmlExample', ['ngSanitize'])
.controller('ExampleController', ['$scope', function($scope) {
  $scope.myHTML =
    'I am an <code>HTML</code>string with ' +
    '<a href="#">links!</a> and other <em>stuff</em>';
}]);
```

protractor.js

```
it('should check ng-bind-html', function() {
  expect(element(by.binding('myHTML')).getText()).toBe(
    'I am an HTMLstring with links! and other stuff');
});
```

## ng-bind-template

- Angular@1.4.7
- `ng` 模块中的指令

`ngBindTemplate` 指令会将元素的文本内容赋值为它所代表的模板的插值。并不像 `ngBind` 那样, `ngBindTemplate` 可以包含 `{{ }}` 表达式。这个指令的存在是由于一些 HTML 标签元素 (如 `<title>` 和

## 指令信息

这个指令的执行优先级为0级

## 用法

用作属性：

```
<ANY ng-bind-template="string">
...
</ANY>
```

## 参数

参数	形式	详细
<code>ngBindTemplate</code>	<code>string</code>	为了转化的 的模板组

## 例子

试一试: 在输入框输入内容观察变化。

index.html

```
<script>
angular.module('bindExample', [])
.controller('ExampleController', ['$scope', function($scope) {
  $scope.salutation = 'Hello';
  $scope.name = 'World';
}]);
</script>
<div ng-controller="ExampleController">
  <label>Salutation: <input type="text" ng-model="salutation"></label>
  <label>Name: <input type="text" ng-model="name"></label><br>
  <pre ng-bind-template="{{salutation}} {{name}}!"></pre>
</div>
```

protractor.js

```
it('should check ng-bind', function() {
  var salutationElem = element(by.binding('salutation'));
  var salutationInput = element(by.model('salutation'));
  var nameInput = element(by.model('name'));

  expect(salutationElem.getText()).toBe('Hello World!');

  salutationInput.clear();
  salutationInput.sendKeys('Greetings');
  nameInput.clear();
  nameInput.sendKeys('user');

  expect(salutationElem.getText()).toBe('Greetings user!');
});
```

## ng-bind

- Angular@1.4.7
- `ng` 模块中的指令

`ngBind` 指令告知 Angular 使用表达式（expression）给定的 HTML 元素来替代文本内容，并且当表达式的值改变时去更新文本内容。

通常情况下，你不会直接的使用 `ngBind` 指令，而是使用双花括号来代替，就像这样，这两种方法是类似的，但是后者更为简洁。

如果一个模板会在 Angular 编译完成之前，浏览器就会将它的原始状态瞬间展示出来的话，那么使用 `ngBind` 来代替 `{{ 表达式 }}` 会更好一点。因为 `ngBind` 是一个元素的属性，它会在页面加载的过程中无形地绑定到用户上。

另一种替代的解决方案是使用 `ngCloak` 指令。

## 指令信息

这个指令的执行优先级为0级

## 用法

用作属性：

```
<ANY ng-bind="expression">
...
</ANY>
```

用作 CSS 类

```
<ANY class="ng-bind: expression;"> ... </ANY>
```

## 参数



参数	形式	详细
ngBind	expression	需要解析的表达式

## 例子

在实时预览的文本盒子内输入一个名字，下面的内容会瞬间改变。

index.html

```
<script>
angular.module('bindExample', [])
.controller('ExampleController', ['$scope', function($scope) {
    $scope.name = 'Whirled';
}]);
</script>
<div ng-controller="ExampleController">
    <label>Enter name: <input type="text" ng-model="name"></label><br>
    Hello <span ng-bind="name"></span>!
</div>
```

protractor.js

```
it('should check ng-bind', function() {
    var nameInput = element(by.model('name'));

    expect(element(by.binding('name')).getText()).toBe('Whirled');
    nameInput.clear();
    nameInput.sendKeys('world');
    expect(element(by.binding('name')).getText()).toBe('world');
});
```

## ng-non-bindable

- Angular@1.4.7
- `ng` 模块中的指令

`ngNonBindable` 指令告知 Angular 不要在当前 DOM 元素上编译或绑定内容。这个指令用于，如果元素包含了类似于 Angular 的指令或者绑定的内容时，让 Angular 忽略它。可能是这种情况，你想在你的网站中展示代码片段。

## 指令信息

这个指令的执行优先级为1000级

## 用法

用作属性：

```
<ANY>
...
</ANY>
```

用作 CSS 类：

```
<ANY class=""> ... </ANY>
```

## 例子


在这个例子中有两个部分使用了简单的插值绑定（`{{表达式}}`），但是其中一个被 `ngNonBindable` 而没有生效。

index.html

```
<div>Normal: {{1 + 2}}</div>
<div ng-non-bindable>Ignored: {{1 + 2}}</div>
```

protractor.js

```
it('should check ng-non-bindable', function() {  
    expect(element(by.binding('1 + 2')).getText()).toContain('3');  
    expect(element.all(by.css('div')).last().getText()).toMatch(/1 \+ 2 = 3/);  
});
```



## <input>

- Angular@1.4.7
- `ng` 模块中的指令

HTML input element control. When used together with `ngModel`, it provides data-binding, input state control, and validation. Input control follows HTML5 input types and polyfills the HTML5 validation behavior for older browsers.

Note: Not every feature offered is available for all input types. Specifically, data binding and event handling via `ng-model` is unsupported for `input[file]`.

## 指令信息

这个指令的执行优先级为0级

## 用法

像个元素（标签）一样使用

```
<input
  ng-model="string"
  [name="string"]
  [required="string"]
  [ng-required="boolean"]
  [ng-minlength="number"]
  [ng-maxlength="number"]
  [ng-pattern="string"]
  [ng-change="string"]
  [ng-trim="boolean"]>
...
</input>
```

## 参数

参数	形式	详细
ngModel	string	Assignable angular expression to data-bind to.
name (可选)	string	Property name of the form under which the control is published.
required (可选)	string	Sets required validation error key if the value is not entered.
ngRequired (可选)	boolean	Sets required attribute if set to true
ngMinlength (可选)	number	Sets minlength validation error key if the value is shorter than minlength.
ngMaxlength (可选)	number	Sets maxlength validation error key if the value is longer than maxlength. Setting the attribute to a negative or non-numeric value, allows view values of any length.
ngPattern (可选)	string	Sets pattern validation error key if the ngModel value does not match a RegExp found by evaluating the Angular expression given in the attribute value. If the expression evaluates to a RegExp object, then this is used directly. If the expression evaluates to a string, then it will be converted to a RegExp after wrapping it in ^ and \$ characters. For instance, "abc" will be converted to new RegExp('^abc\$').

Note: Avoid using the g flag on the RegExp, as it will cause each successive search to start at the index of the last search's match, thus not taking the whole input value into account.

| ngChange (可选) | string |Angular expression to be executed when input changes due to user interaction with the input element.

| ngTrim (可选) | boolean |If set to false Angular will not automatically trim the input. This parameter is ignored for input[type=password] controls, which will never trim the input.(默认值: true)

## 例子

index.html

```
<script>
angular.module('inputExample', [])
.controller('ExampleController', ['$scope', function($scope) {
    $scope.user = {name: 'guest', last: 'visitor'};
}]);
```

```

</script>
<div ng-controller="ExampleController">
  <form name="myForm">
    <label>
      User name:
      <input type="text" name="userName" ng-model="user.name" required="" />
    </label>
    <div role="alert">
      <span class="error" ng-show="myForm.userName.$error.required">
        Required!</span>
    </div>
    <label>
      Last name:
      <input type="text" name="lastName" ng-model="user.last"
        ng-minlength="3" ng-maxlength="10" />
    </label>
    <div role="alert">
      <span class="error" ng-show="myForm.lastName.$error.minlength">
        Too short!</span>
      <span class="error" ng-show="myForm.lastName.$error.maxlength">
        Too long!</span>
    </div>
  </form>
  <hr>
  <tt>user = {{user}}</tt><br/>
  <tt>myForm.userName.$valid = {{myForm.userName.$valid}}</tt><br/>
  <tt>myForm.userName.$error = {{myForm.userName.$error}}</tt><br/>
  <tt>myForm.lastName.$valid = {{myForm.lastName.$valid}}</tt><br/>
  <tt>myForm.lastName.$error = {{myForm.lastName.$error}}</tt><br/>
  <tt>myForm.$valid = {{myForm.$valid}}</tt><br/>
  <tt>myForm.$error.required = {{!!myForm.$error.required}}</tt><br/>
  <tt>myForm.$error.minlength = {{!!myForm.$error.minlength}}</tt><br/>
  <tt>myForm.$error.maxlength = {{!!myForm.$error.maxlength}}</tt><br/>
</div>

```

protractor.js

```

var user = element(by.exactBinding('user'));
var userNameValid = element(by.binding('myForm.userName.$valid'));

```

```
var lastNameValid = element(by.binding('myForm.lastName.$valid'));
var lastNameError = element(by.binding('myForm.lastName.$error'));
var formValid = element(by.binding('myForm.$valid'));
var userNameInput = element(by.model('user.name'));
var userLastInput = element(by.model('user.last'));

it('should initialize to model', function() {
  expect(user.getText()).toContain('{"name":"guest","last":"visitor"}');
  expect(userNameValid.getText()).toContain('true');
  expect(formValid.getText()).toContain('true');
});

it('should be invalid if empty when required', function() {
  userNameInput.clear();
  userNameInput.sendKeys('');

  expect(user.getText()).toContain('{"last":"visitor"}');
  expect(userNameValid.getText()).toContain('false');
  expect(formValid.getText()).toContain('false');
});

it('should be valid if empty when min length is set', function() {
  userLastInput.clear();
  userLastInput.sendKeys('');

  expect(user.getText()).toContain('{"name":"guest","last":""}');
  expect(lastNameValid.getText()).toContain('true');
  expect(formValid.getText()).toContain('true');
});

it('should be invalid if less than required min length', function() {
  userLastInput.clear();
  userLastInput.sendKeys('xx');

  expect(user.getText()).toContain('{"name":"guest"}');
  expect(lastNameValid.getText()).toContain('false');
  expect(lastNameError.getText()).toContain('minlength');
  expect(formValid.getText()).toContain('false');
});
```

```
it('should be invalid if longer than max length', function() {  
  userLastInput.clear();  
  userLastInput.sendKeys('some ridiculously long name');  
  
  expect(user.getText()).toContain('{"name":"guest"}');  
  expect(lastNameValid.getText()).toContain('false');  
  expect(lastNameError.getText()).toContain('maxlength');  
  expect(formValid.getText()).toContain('false');  
});
```



## <form>

- Angular@1.4.7
- `ng` 模块中的指令

实例化 `FormController` (表单控制器在类型Type目录中)的指令

如果表单的 `name` 属性被指定, 那么表单的 `controller` (控制器) 会以 `name` 的值命名, 并被发布到当前的作用域中。

## 别名: `ngForm`

在 Angular 中, 表 ( `form` ) 是可以嵌套的。当所有的字表都有效时 ( `valid` ), 外部包含着他们的表才有效。但是, 浏览器实际上是不支持 `<form>` 元素嵌套的。因此 Angular 提供了 `ngForm` 指令来支持嵌套, 且他们的行为与 `<form>` 相同。这样你就可以嵌套着使用表了, 当在表中使用了 Angular 验证指令时, 这种方式会体现出很大的好处。它可以被 `ngRepeat` 指令动态地生成出来。由于你不能使用插值来动态生成表单 ( `<input>` ) 元素的 `name` 属性。所以你不能不使用一个 `ngForm` 指令来包裹每组被循环 ( `repeat` ) 出来的表单, 然后再使用一个表元素来包裹这些。

## CSS 类

- `.ng-valid` 如果表是有效的, 这个CSS类名会被添加。
- `.ng-invalid` 如果表是无效的, 这个CSS类名会被添加。
- `.ng-pending` 如果表是待定的, 这个CSS类名会被添加。
- `.ng-pristine` 如果表没有修改, 这个CSS类名会被添加。
- `.ng-dirty` 如果表修改了, 这个CSS类名会被添加。
- `.ng-submitted` 如果表提交过了, 这个CSS类名会被添加。

`ngAnimate` 模块会检测到这些类中每一个的添加与移除

## 提交表数据并防止默认的动作 (`[action]`)。

由于在 Angular 设备端应用中表的任务 (role) 不用于传统的往返型 (roundtrip) 应用, 理想的情况是, 不需要浏览器重载整个页面来使向服务器端提交表单, 而是通过应用特定的方式触发相应的 javascript 的逻辑来处理表单的提交。

因此, Angular 阻止了默认动作(action, 表提交到服务端的动作), 除非你为 `<form>` 元素指定了一个 `action` 属性。

你可以使用以下两种方法中的任意一种来调用 javascript 方法来完成表单的提交 (译: 这个方法自己写)

- 在表元素上添加 `ngSubmit` 指令
- 在第一个按钮或者 `type` 属性为 `submit` 的表单元素的上面添加 `ngClick` 指令

为了避免重复的执行处理, 请确保仅仅使用 `ngSubmit` 和 `ngClick` 的其中一个。这样做的原因是下面给出的在 HTML 规范中给出的表提交的规则:

- 如果一个表中仅仅只有一个输入框, 在这个输入框上的的输入行为会触发表单的提交 ( `ngSubmit` )
- 如果一个表中有两个输入框并且没有按钮和提交按钮 ( `input[type=submit]` ), 则输入行为不会触发表单的提交。
- 如果一个表中有一个以上的输入框, 并且有一个以上的按钮或提交按钮, 则在任意一个输入框上的输入行为都将会触发第一个按钮或者提交按钮上的单击处理程序和封闭表单上的提交处理程序。

当一个闭合的表被提交了, 任何待定的 `ngModelOptions` 的改变都会立即生效。需要注意 `ngClick` 事件将会在模型 (model) 更新之前就会触发。所以使用 `ngSubmit` 来允许更新模型。

## 动画钩子

`ngForm` 中的动画, 每当任何一个相关联的 CSS 类被添加或移除时都会触发。这些相关的类包括: `.ng-pristine`, `.ng-dirty`, `.ng-invalid` 和 `.ng-valid`, 还有其他一些在表中使用的验证 (译: 更详细的在类型-**FormController** 中)。 `ngForm` 中的动画同 `ngClass` 的工作原理一致, 动画行为会被钩取使用 CSS 的 `transition` 动画、关键帧 (`@keyframe`) 动画同样也可以使用 JS 的动画。

下面的例子展示了一个利用 CSS `transition` 动画将表中的一个被验证过为无效的元素特定样式渲染出来的栗子。 [N/w/Y](#)

```
/* 确保引入了 ngAnimate 模块可以挂接到下面的动画上 */
/* 译：模块的引入, 请看_公共方法_中的_ng.angular.module_ */
.my-form {
  transition:0.5s linear all;
  background: white;
}
.my-form.ng-invalid {
  background: red;
  color:white;
}
```

## 指令信息

这个指令的执行优先级为0级

## 用法

像个元素（标签）一样使用

html

```
<form [name="string"]>
...
</form>
```

## 参数

参数	形式	详细
name (可选)	string	表的名字，如果被指定了，这个表的控制器就会被发布到关联的作用域（scope）中。

## 例子

index.html

```
<script>
angular.module('formExample', [])
.controller('FormController', ['$scope', function($scope) {
    $scope.userType = 'guest';
}]);
</script>
<style>
.my-form {
    transition:all linear 0.5s;
    background: transparent;
}
.my-form.ng-invalid {
    background: red;
}
</style>
<form name="myForm" ng-controller="FormController" class="my-form">
    userType: <input name="input" ng-model="userType" required>
    <span class="error" ng-show="myForm.input.$error.required">Required
    <code>userType = {{userType}}</code><br>
    <code>myForm.input.$valid = {{myForm.input.$valid}}</code><br>
    <code>myForm.input.$error = {{myForm.input.$error}}</code><br>
    <code>myForm.$valid = {{myForm.$valid}}</code><br>
    <code>myForm.$error.required = {{!!myForm.$error.required}}</code>
</form>
```

protractor.js

```
it('模型应该被初始化: should initialize to model', function() {
  var userType = element(by.binding('userType'));
  var valid = element(by.binding('myForm.input.$valid'));

  expect(userType.getText()).toContain('guest');
  expect(valid.getText()).toContain('true');
});

it('如果为空, 则无效: should be invalid if empty', function() {
  var userType = element(by.binding('userType'));
  var valid = element(by.binding('myForm.input.$valid'));
  var userInput = element(by.model('userType'));

  userInput.clear();
  userInput.sendKeys('');

  expect(userType.getText()).toEqual('userType =');
  expect(valid.getText()).toContain('false');
});
```

## ng-form

- Angular@1.4.7
- `ng` 模块中的指令

可嵌套表单指令的别名。HTML 规定不允许表（`form`）元素的嵌套。但是它可以用于嵌套表单。例如，一个子组的有效性（`valid`）需要确定。

注意：`ngForm` 的目的是 组的控制，但是不能取代 `<form>` 标签的所有功能。例如；发送数据（`post`）到服务器。

## 指令信息

这个指令的执行优先级为0级

## 用法

以元素的形式 (可以作为普通的元素使用，但是要注意 IE 的限制).

```
<ng-form [name="string"]>
...
</ng-form>
```

以元素属性的形式使用:

```
<ANY [ng-form="string"]>
...
</ANY>
```

以 CSS 类使用:

```
<ANY class="[ng-form: string;]">
...
</ANY>
```

## 参数

参数	形式	详细
ngForm 或 name (可选)	string	表的名字，如果被指定了，这个表的控制器就会被发布到关联的作用域（ scope ）中。.

**<textarea>**



## ng-focus

- Angular@1.4.7
- `ng` 模块中的指令

为聚焦事件（`focus`）设置自定义行为。

注意：由于当 `input.focus()` 调用时聚焦事件是被同步执行的。所以，如果事件被触发，AngularJS 会使用 `scope.$evalAsync` 来解析执行表达式并使用 `$apply` 以确保拥有一致的状态。

## 指令信息

这个指令的执行优先级为0级

## 用法

用作属性：

```
<window, input, select, textarea, a
  ng-focus="expression">
...
</window, input, select, textarea, a>
```

## 参数

参数	形式	详细
<code>ngFocus</code>	<code>expression</code>	在聚焦时需要解析的表达式 (事件对象可以通过 <code>\$event</code> 获得)

## 例子

请看 `ngClick`

## ng-blur

- Angular@1.4.7
- `ng` 模块中的指令

为失焦事件（`blur`）设置自定义行为。

注意：由于在 DOM 操作中失焦事件的执行方式是同步执行（如：移除聚焦的表单元素），所以，如果事件被触发，AngularJS 会使用 `scope.$evalAsync` 来解析执行表达式并使用 `$apply` 以确保拥有一致的状态。

## 指令信息

这个指令的执行优先级为0级

## 用法

用作属性：

```
<window, input, select, textarea, a
  ng-blur="表达式">
<!-- 这些都是有失焦事件的元素 -->
...
</window, input, select, textarea, a>
```

## 参数

参数	形式	详细
<code>ngBlur</code>	<code>expression</code>	在失去焦点时需要解析的表达式 (事件对象可以通过 <code>\$event</code> 获得)

## 例子

请看 `ngClick`

## ng-change

- Angular@1.4.7
- `ng` 模块中的指令

当用户改变（`onChange`）表单时，解析给定的表达式。表达式会被立即解析，并不像 JavaScript 中的 `onChange` 事件那样仅仅会在改变结束之后触发一次。（就是通常当用户离开表中的元素或者按下回车键的时候）

`ngChange` 表达式仅会在表单产生了一个的新值，并且这个新的值被提交到模型（`model`）的时候被解析执行。

以下情况不会解析执行：

- 从 `$parsers` 改造管道返回的值没有变化。
- 由于模型将会置空（`null`）表单为无效。
- 模型由程序改变而不是由表单元素的值来改变时。

注意：这个指令需要 `ngModel` 作为前提。

## 指令信息

这个指令的执行优先级为0级

## 用法

用作属性：

```
<input ng-change="expression">
...
</input>
```

## 参数

参数	形式	详细
<code>ngChange</code>	<code>expression</code>	在表单内的值改变时，需要解析的表达式

## 例子

index.html

```
<script>
angular.module('changeExample', [])
.controller('ExampleController', ['$scope', function($scope) {
  $scope.counter = 0;
  $scope.change = function() {
    $scope.counter++;
  };
}]);
</script>
<div ng-controller="ExampleController">
  <input type="checkbox" ng-model="confirmed" ng-change="change()" />
  <input type="checkbox" ng-model="confirmed" id="ng-change-example2" />
  <label for="ng-change-example2">Confirmed</label><br />
  <tt>debug = {{confirmed}}</tt><br/>
  <tt>counter = {{counter}}</tt><br/>
</div>
```

protractor.js

```
var counter = element(by.binding('counter'));
var debug = element(by.binding('confirmed'));

it('如果从视图 `view` 改变的, 则解析表达式', function() {
    expect(counter.getText()).toContain('0');

    element(by.id('ng-change-example1')).click();

    expect(counter.getText()).toContain('1');
    expect(debug.getText()).toContain('true');
});

it('如果是从模型 `model` 直接改变的, 则不解析表达式', function() {
    element(by.id('ng-change-example2')).click();

    expect(counter.getText()).toContain('0');
    expect(debug.getText()).toContain('true');
});
```

## ng-checked

- Angular@1.4.7
- `ng` 模块中的指令

如果 `ngChecked` 的值为真，则在元素上设置选中（`checked`）属性。

注意：这个指令需要和 `ngModel` 一同使用，否则可能会产生意外行为。

## 为什么我们需要 `ngCheck` ？

HTML 规范并没有要求浏览器保存布尔值类型属性的值，如：`checked`。（他们存在则代表 `true`，如果没有这个属性则代表 `false`）如果我们将一个 Angular 的插值表达式放到一个这样的属性中，当浏览器山吃这个属性的时候我们就会失去绑定在这个属性上的信息。`ngCheck` 指令为选中属性解决了这个问题。这个补充指令并不会被浏览器删掉，为绑定信息提供了一个永久可靠的存储位置。

## 指令信息

这个指令的执行优先级为100级（译：终于有不是零的了 `o(*≥▽≤)ツ`）

## 用法

用作属性：

```
<INPUT ng-checked="expression">
...
</INPUT>
```

## 参数

参数	形式	详细
<code>ngChecked</code>	<code>expression</code>	如果表达式的值为真，则将会在元素上添加 <code>checked</code> 属性。

## 例子

index.html

```
<label>
  Check me to check both:
  <input type="checkbox" ng-model="master">
</label>
<br/>
<input id="checkSlave" type="checkbox" ng-checked="master" aria-label="checkSlave">
```

protractor.js

```
it('should check both checkBoxes', function() {
  expect(element(by.id('checkSlave')).getAttribute('checked')).toBe(false);
  element(by.model('master')).click();
  expect(element(by.id('checkSlave')).getAttribute('checked')).toBe(true);
});
```

## ng-disabled

- Angular@1.4.7
- `ng` 模块中的指令

如果 `ngDisabled` 内部的表达式解析的结果为真值，则这个指令会在元素上设置失效（`disabled`）的属性。

由于不能在 `disabled` 属性上使用插值，所以我们需要这样一个特殊的指令。下面的例子会使 chrome/Firefox 中的按钮失效，但是在老版本的 IE 中不会生效。

```
<!-- See below for an example of ng-disabled being used correctly  
<div ng-init="isDisabled = false">  
  <button disabled="{{isDisabled}}">Disabled</button>  
</div>
```

这是因为 HTML 规范不需要浏览器去保存像 `disabled` 这样的属性的布尔值（这些属性存在则意味着生效（`true`），这些属性不存在则意味着无效（`false`））。如果我们将 Angular 的插值表达式放入到这样一个属性中，那么当浏览器移除这个属性的时候我们就是遗失绑定在上面的信息。

## 指令信息

这个指令的执行优先级为100级（译：终于有不是零的了  $o(*\geq \nabla \leq)$  ツ）

## 用法

用作属性：

```
<INPUT ng-disabled="expression">  
...  
</INPUT>
```

## 参数



参数	形式	详细
ngDisabled	expression	如果表达式的值为真，则将会在元素上添加 disabled 属性。

## 例子

index.html

```
<label>点击我切换: <input type="checkbox" ng-model="checked"></label>  
<button ng-model="button" ng-disabled="checked">Button</button>
```

protractor.js

```
it('需要切换按钮状态', function() {  
  expect(element(by.css('button')).getAttribute('disabled')).toBeFalse;  
  element(by.model('checked')).click();  
  expect(element(by.css('button')).getAttribute('disabled')).toBeTrue;  
});
```

## ng-readonly

- Angular@1.4.7
- `ng` 模块中的指令

HTML 规范中没有要求浏览器保存属性的布尔值，如 `readonly`（这些属性存在则代表 `true`，不存在则代表 `false`）如果我们将一个 Angular 的插值表达式放入一个这样的属性中，那么当浏览器删除这个属性时，我们将丢失绑定的信息。`ngReadonly` 指令解决了 `readonly` 属性的这个问题。这个补充的属性不会被浏览器移除 并且提供了一个永久可靠的地方来存储绑定信息。

## 指令信息

这个指令的执行优先级为100级

## 用法

用作属性：

```
<INPUT
  ng-readonly="expression">
...
</INPUT>
```

## 参数

参数	形式	详细
<code>ngReadonly</code>	<code>expression</code>	如果表达式为真，则特殊的属性 <code>readonly</code> 将会被设置到元素上

## 例子

index.html

```
<label>Check me to make text readonly: <input type="checkbox" ng-model="checked">  
<input type="text" ng-readonly="checked" value="I'm Angular" aria-label="Text input" />
```

protractor.js

```
it('should toggle readonly attr', function() {  
  expect(element(by.css('[type="text"]')).getAttribute('readonly')).toBeFalsy();  
  element(by.model('checked')).click();  
  expect(element(by.css('[type="text"]')).getAttribute('readonly')).toBeTruthy();  
});
```

## ng-selected

- Angular@1.4.7
- `ng` 模块中的指令

HTML 规范中没有要求浏览器保存属性的布尔值，如 `selected`（这些属性存在则代表 `true`，不存在则代表 `false`）如果我们将一个 Angular 的插值表达式放入一个这样的属性中，那么当浏览器删除这个属性时，我们将丢失绑定的信息。`ngSelected` 指令解决了 `selected` 属性的这个问题。这个补充的属性不会被浏览器移除 并且提供了一个永久可靠的地方来存储绑定信息。

## 指令信息

这个指令的执行优先级为100级

## 用法

用作属性：

```
<OPTION
  ng-selected="expression">
...
</OPTION>
```

## 参数

参数	形式	详细
<code>ngSelected</code>	<code>expression</code>	如果表达式为真，则特殊的属性 <code>selected</code> 将会被设置到元素上

## 例子

`index.html`

```
<label>Check me to select: <input type="checkbox" ng-model="selected">  
<select aria-label="ngSelected demo">  
  <option>Hello!</option>  
  <option id="greet" ng-selected="selected">Greetings!</option>  
</select>
```

protractor.js

```
it('should select Greetings!', function() {  
  expect(element(by.id('greet')).getAttribute('selected')).toBeFalse;  
  element(by.model('selected')).click();  
  expect(element(by.id('greet')).getAttribute('selected')).toBeTrue;  
});
```

## ng-open

- Angular@1.4.7
- `ng` 模块中的指令

HTML 规范中没有要求浏览器保存属性的布尔值，如 `open`（这些属性存在则代表 `true`，不存在则代表 `false`）如果我们将一个 Angular 的插值表达式放入一个这样的属性中，那么当浏览器删除这个属性时，我们将丢失绑定的信息。`ngOpen` 指令解决了 `open` 属性的这个问题。这个补充的属性不会被浏览器移除 并且提供了一个永久可靠的地方来存储绑定信息。

## 指令信息

这个指令的执行优先级为100级

## 用法

用作属性：

```
<DETAILS
  ng-open="expression">
  ...
</DETAILS>
```

## 参数

参数	形式	详细
<code>ngOpen</code>	<code>expression</code>	如果表达式为真，则特殊的属性 <code>open</code> 将会被设置到元素上

## 例子

index.html

```
<label>Check me check multiple: <input type="checkbox" ng-model="open">  
<details id="details" ng-open="open">  
  <summary>Show/Hide me</summary>  
</details>
```

protractor.js

```
it('should toggle open', function() {  
  expect(element(by.id('details')).getAttribute('open')).toBeFalsy();  
  element(by.model('open')).click();  
  expect(element(by.id('details')).getAttribute('open')).toBeTruthy();  
});
```

## ng-submit

- Angular@1.4.7
- `ng` 模块中的指令

可以将 `angular` 表达式绑定到 `onsubmit` 事件上。

此外,如果表没有包含 `action` `data-action` 或者 `x-action` 属性时, 这个指令会阻止默认行为 (表

会向服务器发送请求并重载当前页面)

注意：小心同时使用 ``ngClick`` 和 ``ngSubmit``，这可能引起“重复提交”。  
查看 ``form`` 指令文档关于 ``ngSubmit`` 何时会被触发部分的解释。

## 指令信息

这个指令的执行优先级为0级

## 用法

用作属性：

```
<form
  ng-submit="expression">
  ...
</form>
```

## 参数

参数	形式	详细
<code>ngSubmit</code>	<code>expression</code>	需要解析的表达式 (事件对象可以通过 <code>\$event</code> 获得)

## 例子



## index.html

```

<script>
  angular.module('submitExample', [])
    .controller('ExampleController', ['$scope', function($scope) {
      $scope.list = [];
      $scope.text = 'hello';
      $scope.submit = function() {
        if ($scope.text) {
          $scope.list.push(this.text);
          $scope.text = '';
        }
      };
    }]);
</script>
<form ng-submit="submit()" ng-controller="ExampleController">
  Enter text and hit enter:
  <input type="text" ng-model="text" name="text" />
  <input type="submit" id="submit" value="Submit" />
  <pre>list={{list}}</pre>
</form>

```

## protractor.js

```

it('should check ng-submit', function() {
  expect(element(by.binding('list')).getText()).toBe('list=[]');
  element(by.css('#submit')).click();
  expect(element(by.binding('list')).getText()).toContain('hello');
  expect(element(by.model('text')).getAttribute('value')).toBe('');
});
it('should ignore empty strings', function() {
  expect(element(by.binding('list')).getText()).toBe('list=[]');
  element(by.css('#submit')).click();
  element(by.css('#submit')).click();
  expect(element(by.binding('list')).getText()).toContain('hello');
});

```

## ng-class

- Angular@1.4.7
- `ng` 模块中的指令

`ngClass` 指令允许你在 HTML 元素上通过数据绑定一个表达式动态的方式设置 CSS 类，这个表达式代表了所有要被添加的 CSS 类名。

操作指令有三种不同的方式，取决于解析表达式的三种形式：

- 如果表达式解析为一个字符串，这个字符串应该使用空格来分割 CSS 类名。
- 如果表达式解析为一个对象，那么每个键值对（key-value）中，如果值（value）为真值（true）那么与它对应的键名（key）将被当做 CSS 类名添加进去。（译：{red: true, blue: 0} `red` 会被添加，`blue` 则不会）
- 如果表达式解析为一个数组，那么数组中的每个元素的形式都应该为类型1中所述的字符串形式，或者类型2中所述的对象形式。也就是说你可以在一个数组中混合使用这两种表达式来更好的控制出现的 CSS 类名。下面的例子会展现出来。

`ngClass` 不会重复添加一个已经被设置了的CSS类名

如果表达式改变了，先删除之前添加的类，然后才加入新的类。

## 指令信息

这个指令的执行优先级为0级

## 用法

用作属性：

```
<ANY ng-class="expression">
...
</ANY>
```

## 用作 CSS 类

```
<ANY class="ng-class: expression;">
...
</ANY>
```

## 动画

**add** - 会在 CSS 类被应用到元素之前触发。

**remove** - 会在 CSS 类从元素上移除之前触发。

点击[这里](#)了解更多关于参与动画的步骤。（译：链接就是到 `$animate` 服务中）

## 参数

参数	形式	详细
ngClass	expression	解析的结果可以是一个使用空格分隔的 CSS 类名的字符串，一个数组，或者一个 CSS 类名到布尔值的映射（map）。

如果是映射（译：也就是之前说的 `object` ），属性值为真值的属性名会被作为 CSS 类名添加到元素上。|

## 例子

这个例子听过使用 `ngClass` 指令来说明基础的绑定。

index.html

```

<p ng-class="{strike: deleted, bold: important, 'has-error': error}">
<label>
  <input type="checkbox" ng-model="deleted">
    deleted (apply "strike" class)
</label><br>
<label>
  <input type="checkbox" ng-model="important">
    important (apply "bold" class)
</label><br>
<label>
  <input type="checkbox" ng-model="error">
    error (apply "has-error" class)
</label>
<hr>
<p ng-class="style">Using String Syntax</p>
<input type="text" ng-model="style"
  placeholder="Type: bold strike red" aria-label="Type: bold s
<hr>
<p ng-class="[style1, style2, style3]">Using Array Syntax</p>
<input ng-model="style1"
  placeholder="Type: bold, strike or red" aria-label="Type: bo
<input ng-model="style2"
  placeholder="Type: bold, strike or red" aria-label="Type: bo
<input ng-model="style3"
  placeholder="Type: bold, strike or red" aria-label="Type: bo
<hr>
<p ng-class="[style4, {orange: warning}]">Using Array and Map Synta
<input ng-model="style4" placeholder="Type: bold, strike" aria-labe
<label><input type="checkbox" ng-model="warning"> warning (apply "c

```

style.css

```
.strike {  
    text-decoration: line-through;  
}  
.bold {  
    font-weight: bold;  
}  
.red {  
    color: red;  
}  
.has-error {  
    color: red;  
    background-color: yellow;  
}  
.orange {  
    color: orange;  
}
```

protractor.js

```
var ps = element.all(by.css('p'));

it('should let you toggle the class', function() {

  expect(ps.first().getAttribute('class')).not.toMatch(/bold/);
  expect(ps.first().getAttribute('class')).not.toMatch(/has-error/);

  element(by.model('important')).click();
  expect(ps.first().getAttribute('class')).toMatch(/bold/);

  element(by.model('error')).click();
  expect(ps.first().getAttribute('class')).toMatch(/has-error/);
});

it('should let you toggle string example', function() {
  expect(ps.get(1).getAttribute('class')).toBe('');
  element(by.model('style')).clear();
  element(by.model('style')).sendKeys('red');
  expect(ps.get(1).getAttribute('class')).toBe('red');
});

it('array example should have 3 classes', function() {
  expect(ps.get(2).getAttribute('class')).toBe('');
  element(by.model('style1')).sendKeys('bold');
  element(by.model('style2')).sendKeys('strike');
  element(by.model('style3')).sendKeys('red');
  expect(ps.get(2).getAttribute('class')).toBe('bold strike red');
});

it('array with map example should have 2 classes', function() {
  expect(ps.last().getAttribute('class')).toBe('');
  element(by.model('style4')).sendKeys('bold');
  element(by.model('warning')).click();
  expect(ps.last().getAttribute('class')).toBe('bold orange');
});
```

## 动画例子

index.html

```
<input id="setbtn" type="button" value="set" ng-click="myVar='my-cl
<input id="clearbtn" type="button" value="clear" ng-click="myVar='
<br>
<span class="base-class" ng-class="myVar">Sample Text</span>
```

style.css

```
.base-class {
  transition:all cubic-bezier(0.250, 0.460, 0.450, 0.940) 0.5s;
}

.base-class.my-class {
  color: red;
  font-size:3em;
}
```

protractor.js

```
it('should check ng-class', function() {
  expect(element(by.css('.base-class')).getAttribute('class')).not
    toMatch(/my-class/);

  element(by.id('setbtn')).click();

  expect(element(by.css('.base-class')).getAttribute('class')).
    toMatch(/my-class/);

  element(by.id('clearbtn')).click();

  expect(element(by.css('.base-class')).getAttribute('class')).not
    toMatch(/my-class/);
});
```

## ngClass and pre-existing CSS3 Transitions/Animations

`ngClass` 指令也支持 CSS3 的过渡动画和关键帧动画，甚至不需要不遵从 `ngAnimate` 的 CSS 命名规则。使用动画时，`ngAnimate` 会应用补充的 CSS 类名来规定一个动画的起止，但是并不会阻止已经存在于这个元素上的之前存在的任何一个 CSS 过渡动画。想要了解基于 CSS 类的动画更详细的内容，仔细阅读以下 `$animate.addClass` 和 `$animate.removeClass`。（这两个方法都在 `$animate` 服务中。o(∩\_∩)d）



## ng-class-even

- Angular@1.4.7
- `ng` 模块中的指令

`ngClassOdd` 与 `ngClassEven` 指令工作方式其实与 `ngClass` 是一致的。只不过它们是配合 `ngRepeat` 使用的，并且只在奇数行（或偶数行）生效。

这个指令仅限于在一个 `ngRepeat` 指令的作用域内使用。

## 指令信息

这个指令的执行优先级为0级

## 用法

用作属性：

```
<ANY ng-class-even="expression">
...
</ANY>
```

用作 CSS 类

```
<ANY class="ng-class-even: expression;"> ... </ANY>
```

## 参数

参数	形式	详细
<code>ngClassEven</code>	<code>expression</code>	需要解析转化的表达式。解析的结果会是一个以空格分割的代表多个 CSS 类名或者一个数组的字符串

## 例子

index.html

```
<ol ng-init="names=['John', 'Mary', 'Cate', 'Suz']">
  <li ng-repeat="name in names">
    <span ng-class-odd="'odd'" ng-class-even="'even'">
      {{name}} &nbsp; &nbsp; &nbsp; &nbsp;
    </span>
  </li>
</ol>
```

## style.css

```
.odd {
  color: red;
}
.even {
  color: blue;
}
```

## protractor.js

```
it('should check ng-class-odd and ng-class-even', function() {
  expect(element(by.repeater('name in names').row(0).column('name'))
    toMatch(/odd/);
  expect(element(by.repeater('name in names').row(1).column('name'))
    toMatch(/even/);
});
```

## ng-class-odd

- Angular@1.4.7
- `ng` 模块中的指令

`ngClassOdd` 与 `ngClassEven` 指令工作方式其实与 `ngClass` 是一致的。只不过它们是配合 `ngRepeat` 使用的，并且只在奇数行（或偶数行）生效。

这个指令仅限于在一个 `ngRepeat` 指令的作用域内使用。

## 指令信息

这个指令的执行优先级为0级

## 用法

用作属性：

```
<ANY ng-class-odd="expression">
...
</ANY>
```

用作 CSS 类

```
<ANY class="ng-class-odd: expression;"> ... </ANY>
```

## 参数

参数	形式	详细
<code>ngClassOdd</code>	<code>expression</code>	需要解析转化的表达式。解析的结果会是一个以空格分割的代表多个 CSS 类名或者一个数组的字符串

## 例子

同 `ng-class-even`



## ng-click

- Angular@1.4.7
- `ng` 模块中的指令

`ngClick` 指令允许你在一个元素被点击后，为其指定自定义的行为。

## 指令信息

这个指令的执行优先级为0级

## 用法

用作属性：

```
<ANY ng-click="expression">
...
</ANY>
```

## 参数

参数	形式	详细
<code>ngClick</code>	<code>expression</code>	通过点击操作触发需要解析的表达式（事件对象可以通过 <code>\$event</code> 获得）

## 例子

index.html

```
<button ng-click="count = count + 1" ng-init="count=0">
  Increment
</button>
<span>
  count: {{count}}
</span>
```

protractor.js

```
it('should check ng-click', function() {
  expect(element(by.binding('count')).getText()).toMatch('0');
  element(by.css('button')).click();
  expect(element(by.binding('count')).getText()).toMatch('1');
});
```

## ng-dblclick

- Angular@1.4.7
- `ng` 模块中的指令

`ngClick` 指令允许为双击事件指定自定义的行为。

## 指令信息

这个指令的执行优先级为0级

## 用法

用作属性：

```
<ANY ng-dblclick="expression">
...
</ANY>
```

## 参数

参数	形式	详细
<code>ngDbclick</code>	<code>expression</code>	通过双击操作触发需要解析的表达式（事件对象可以通过 <code>\$event</code> 获得）

## 例子

index.html

```
<button ng-dblclick="count = count + 1" ng-init="count=0">
  Increment (on double click)
</button>
count: {{count}}
```

## ng-cloak

- Angular@1.4.7
- `ng` 模块中的指令

`ngCloak` 指令用来在你的应用加载时防止浏览器短暂地显示出 Angular html 模板的原始（未编译）的状态。使用这个指令避免由 html 模板显示而引起的不良的闪烁效果。

这个指令虽然可以应用到 `<body>` 元素上，但是为了使浏览器视图能够渐进地渲染，首选还是在页面上的多个小部分来使用它。

`ngCloak` 会同下面的 css 规则被一同嵌入到 `angular.js` 和 `angular.min.js` 中。使用 CSP 模式时请在你的 html 文件中添加 `angular-csp.css`（请看 `ngCsp`）。

```
[ng\:cloak], [ng-cloak], [data-ng-cloak], [x-ng-cloak], .ng-cloak,  
  display: none !important;  
}
```

在这条 css 被浏览器载入后，所有带有 `ngCloak` 的 html 元素（包括他们的子类）会被标记为隐藏。当 Angular 在模板编译时遇见这个指令时，Angular 会删除元素上的 `ngCloak` 属性，使编译后的元素可见。

为了达到最好的效果，`angular.js` 脚本必须在 html 文档的头部（head）被加载进来，或者作为替代的方法，上面的 css 规则必须被应用外部的样式表包含了。

## 指令信息

这个指令的执行优先级为0级

## 用法

用作属性：



```
<ANY>  
</ANY>
```

用作 CSS 类：

```
<ANY class=""> ... </ANY>
```

## 例子

index.html

```
<div id="template1" ng-cloak>{{ 'hello' }}</div>  
<div id="template2" class="ng-cloak">{{ 'world' }}</div>
```

protractor.js

```
it('should remove the template directive and css class', function()  
    expect($('#template1').getAttribute('ng-cloak')).  
        toBeNull();  
    expect($('#template2').getAttribute('ng-cloak')).  
        toBeNull();  
});
```

## ng-controller

- Angular@1.4.7
- `ng` 模块中的指令

`ngController` 会在视图上添加一个控制器类。这是 angular 如何支持 MVC（Model-View-Controller）设计模式原则的关键因素。

angular 中的 MVC 组件：

- **Model** — 模型是作用域（scope）的属性；作用域依附在 DOM 元素上，作用域的属性则可以通过绑定被访问到。
- **View** — 被渲染到视图（View）中的模板（拥有数据绑定的 HTML）
- **Controller** — `ngController` 指令指定了一个控制器（Controller）类；这个类包含了应用（app）使用函数和值来修饰作用域背后的业务逻辑

注意：通过使用 `$route` 服务，你可以将控制器声明到路由定义中从而使该控制器关联到 DOM 上。

一个常见的错误：使用 `ng-controller` 再一次把控制器声明在模板上，这会导致控制器被设置和执行两次。

## 指令信息

这个指令创建了新的作用域 这个指令的执行优先级为500级

## 用法

用作属性：

```
<ANY ng-controller="expression">
...
</ANY>
```

## 参数

参数	形式	详细
ngController	expression	使用当前 <code>\$controllerProvider</code> 注册的构造函数的名字，或者是当前作用域上的一个表达式解析出的构造函数的名字。 通过指定 <code>ng-controller="as propertyName"</code> 将控制器实例发布到一个作用域（scope）属性中。 如果当前的 <code>\$controllerProvider</code> 配置为使用全局（通过使用 <code>\$controllerProvider.allowGlobals()</code> ），这也会成为一个全局性的可访问的构造函数的名字（不推荐这么做）。

## 例子

这是一个简单的编辑用户联系信息的表。添加，删除，清空和展示方法被生命在控制器上（见源标签）。 这些方法可以轻易地从 `angular` 标记中调用。任何对数据的操作都会自动映射到视图中而不需要手动去更新。

下面介绍了两种不同风格的声明方式：

一种是将方法和属性直接绑定到控制器上，像这样：`ng-controller="SettingsController1 as settings"`

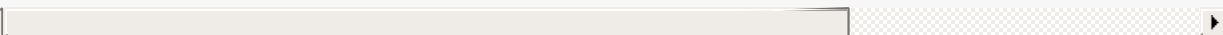
一种是将 `$scope` 注入到控制器中：`ng-controller="SettingsController2"`

第二种方式在 `angular` 社区中更为流行，而且在文档中也更经常使用这种方式作为样板。但是，避免使用作用域而直接将属性绑定到控制器上还是有很多优势的。

- 当多个控制器作用在一个元素上时，使用 `controller as` 让被访问模板上的控制器容易识别出来。
  - 如果你将你的控制器写成类的方式，则你会更方便的访问那些作用域中和控制器代码中的属性和方法
  - 由于绑定中总是有一个 `.`，所以你没有必要担心原型继承会覆盖原始代码。
- 这个例子展示了 `controller as` 的句法。

index.html

```
<div id="ctrl-as-exmpl" ng-controller="SettingsController1 as sett:
  <label>Name: <input type="text" ng-model="settings.name"/></label>
  <button ng-click="settings.greet()">greet</button><br/>
  Contact:
  <ul>
    <li ng-repeat="contact in settings.contacts">
      <select ng-model="contact.type" aria-label="Contact method" :
        <option>phone</option>
        <option>email</option>
      </select>
      <input type="text" ng-model="contact.value" aria-labelledby='
      <button ng-click="settings.clearContact(contact)">clear</butt
      <button ng-click="settings.removeContact(contact)" aria-label
    </li>
    <li><button ng-click="settings.addContact()">add</button></li>
  </ul>
</div>
```



app.js

```
angular.module('controllerAsExample', [])
  .controller('SettingsController1', SettingsController1);

function SettingsController1() {
  this.name = "John Smith";
  this.contacts = [
    {type: 'phone', value: '408 555 1212'},
    {type: 'email', value: 'john.smith@example.org'} ];
}

SettingsController1.prototype.greet = function() {
  alert(this.name);
};

SettingsController1.prototype.addContact = function() {
  this.contacts.push({type: 'email', value: 'yourname@example.org'}];
};

SettingsController1.prototype.removeContact = function(contactToRemove) {
  var index = this.contacts.indexOf(contactToRemove);
  this.contacts.splice(index, 1);
};

SettingsController1.prototype.clearContact = function(contact) {
  contact.type = 'phone';
  contact.value = '';
};
```

protractor.js

```
it('should check controller as', function() {
  var container = element(by.id('ctrl-as-exmpl'));
  expect(container.element(by.model('settings.name'))
    .getAttribute('value')).toBe('John Smith');

  var firstRepeat =
    container.element(by.repeater('contact in settings.contacts'));
  var secondRepeat =
    container.element(by.repeater('contact in settings.contacts'));

  expect(firstRepeat.element(by.model('contact.value')).getAttribute(
    'value')).toBe('408 555 1212');

  expect(secondRepeat.element(by.model('contact.value')).getAttribute(
    'value')).toBe('john.smith@example.org');

  firstRepeat.element(by.buttonText('clear')).click();

  expect(firstRepeat.element(by.model('contact.value')).getAttribute(
    'value')).toBe('');

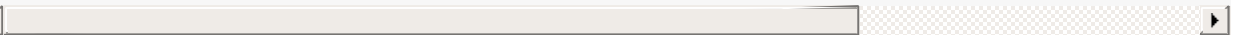
  container.element(by.buttonText('add')).click();

  expect(container.element(by.repeater('contact in settings.contacts')
    .element(by.model('contact.value'))
    .getAttribute('value'))
    .toBe('yourname@example.org'));
});
```

这个例子展示的是控制器的“附属到 `$scope` 上”形式。

index.html

```
<div id="ctrl-exmpl" ng-controller="SettingsController2">
  <label>Name: <input type="text" ng-model="name"/></label>
  <button ng-click="greet()">greet</button><br/>
  Contact:
  <ul>
    <li ng-repeat="contact in contacts">
      <select ng-model="contact.type" id="select_{{$index}}">
        <option>phone</option>
        <option>email</option>
      </select>
      <input type="text" ng-model="contact.value" aria-labelledby='
      <button ng-click="clearContact(contact)">clear</button>
      <button ng-click="removeContact(contact)">X</button>
    </li>
    <li>[ <button ng-click="addContact()">add</button> ]</li>
  </ul>
</div>
```



app.js

```
angular.module('controllerExample', [])
  .controller('SettingsController2', ['$scope', SettingsController2]);

function SettingsController2($scope) {
  $scope.name = "John Smith";
  $scope.contacts = [
    {type:'phone', value:'408 555 1212'},
    {type:'email', value:'john.smith@example.org'} ];

  $scope.greet = function() {
    alert($scope.name);
  };

  $scope.addContact = function() {
    $scope.contacts.push({type:'email', value:'yourname@example.org'});
  };

  $scope.removeContact = function(contactToRemove) {
    var index = $scope.contacts.indexOf(contactToRemove);
    $scope.contacts.splice(index, 1);
  };

  $scope.clearContact = function(contact) {
    contact.type = 'phone';
    contact.value = '';
  };
}
```

protractor.js



```
it('should check controller', function() {
  var container = element(by.id('ctrl-exmpl'));

  expect(container.element(by.model('name'))
    .getAttribute('value')).toBe('John Smith');

  var firstRepeat =
    container.element(by.repeater('contact in contacts').row(0));
  var secondRepeat =
    container.element(by.repeater('contact in contacts').row(1));

  expect(firstRepeat.element(by.model('contact.value')).getAttribute(
    'value')).toBe('408 555 1212');
  expect(secondRepeat.element(by.model('contact.value')).getAttribute(
    'value')).toBe('john.smith@example.org');

  firstRepeat.element(by.buttonText('clear')).click();

  expect(firstRepeat.element(by.model('contact.value')).getAttribute(
    'value')).toBe('');

  container.element(by.buttonText('add')).click();

  expect(container.element(by.repeater('contact in contacts').row(2))
    .element(by.model('contact.value'))
    .getAttribute('value'))
    .toBe('yourname@example.org');
});
```

## ng-copy

- Angular@1.4.7
- `ng` 模块中的指令

为复制（copy）事件指定自定义的行为。

## 指令信息

这个指令的执行优先级为0级

## 用法

用作属性：

```
<window, input, select, textarea, a
  ng-copy="expression">
...
</window, input, select, textarea, a>
```

## 参数

参数	形式	详细
ngCopy	<code>expression</code>	通过复制操作触发需要解析的表达式（事件对象可以通过 <code>\$event</code> 获得）

## 例子

index.html

```
<input ng-copy="copied=true" ng-init="copied=false; value='copy me'
copied: {{copied}}>
```

## ng-csp

- Angular@1.4.7
- `ng` 模块中的指令

Angular 的一些特性会破坏某些 CSP（Content Security Policy，内容安全策略）规则。


如果你想使用这些规则就需要告知 Angular 不要使用破坏这些规则的特性。

当开发想 Google Chrome 扩展或者通用的 Windows 应用时，这么做还是很有必要的。

下面的规则会影响 Angular：

- `unsafe-eval`：这个规则禁止应用使用 `eval` 或者 `Function(string)` 生成函数（以及其中携带的其他东西）。Angular 在 `$parse` 服务中使用了这种方式来为解析 Angular 表达式来增加 30% 的速度。
- `unsafe-inline`：这个规则禁止应用将自定义的样式注入到文档中。Angular 使用了这种方式添加了一些 CSS 规则（如：`ngCloak` 和 `ngHide`）。为了使这些只能能在 CSP 规则下正常工作，你必须手动引入 `angular-csp.css`

如果你不提供 `ngCsp` Angular 会尝试自动检测，如果 CSP 阻止了不安全的 `eval` (`unsafe-eval`) Angular 就会自动关闭 `$parse` 服务中的这个特性。虽然有这个自动检测，但是会触发一个 CSP 的错误。这个错误会被打印到控制台中：



```
拒绝将一个字符串解析为 JavaScript，因为在 CSP 指令中 'unsafe-eval': "default" 是一个不被认可的脚本源。请注意 'script-src' 没有被明确的设置。所以 'default'
```

这个错误虽然无害但是很招人烦。为了防止报错，你可以在引入 `angular.js` 文件的那个 `<script>` 之前的一个元素上添加 `ngCsp` 指令。

注意：这个指令仅会在 `ng-csp` 和 `data-ng-csp` 这两种属性形式下生效。

你可以为 `ng-csp` 属性赋值来控制 Angular 应该关闭那些与 CSP 相关的特性。设置参见下面：

- `no-inline-style`：禁止 Angular 向 DOM 中注入 CSS 样式。

- `no-unsafe-eval` : 禁止 Angular 使用以不安全的 `eval` 字符串的方式优化了的 `$parse`

你可以想下面的方式那样组合着使用这些值：

不声明意味着 Angular 会假设你允许使用了行内样式，但是它在执行时会为不安全的 `eval` 做检查。例如 `<body>`。这与 Angular 之前的版本向后兼容。

一个简单的 `ng-csp`（或者 `data-ng-csp`）属性就可以告知 Angular 将行内样式和不安全的 `eval` 特性全部禁止。例如：`<body ng-csp>`。这与 Angular 之前的版本向后兼容。

仅仅赋值 `no-unsafe-eval` 则告诉 Angular 不能使用 `eval`。但是我们仍然可以使用行内样式。例如：`<body ng-csp="no-unsafe-eval">`

仅仅赋值 `no-inline-style` 则告诉 Angular 不能注入样式。但是我们仍然可以使用 `eval` - 不会发生自动检查不安全的 `eval` 的情况。例如：`<body ng-csp="no-inline-style">`

将 `no-inline-style` 和 `no-unsafe-eval` 都赋值给 `ngCsp` 则告诉 Angular 我们不能使用 `eval` 和注入样式。这种情况与置空效果是一样的：`ng-csp`。例如 `<body ng-csp="no-inline-style;no-unsafe-eval">`

## 指令信息

这个指令的执行优先级为0级

## 用法

用作属性：

```
<html>
...
</html>
```

## 例子

这个指令展示了如何在 `html` 标签上使用 `ngCsp` 指令。

```
<!doctype html>
<html ng-app ng-csp>
...
...
</html>
```

// 注意: the suffix .csp in the example name triggers // csp mode in our http server!

index.html

```
<div ng-controller="MainController as ctrl">
  <div>
    <button ng-click="ctrl.inc()" id="inc">Increment</button>
    <span id="counter">
      {{ctrl.counter}}
    </span>
  </div>

  <div>
    <button ng-click="ctrl.evil()" id="evil">Evil</button>
    <span id="evilError">
      {{ctrl.evilError}}
    </span>
  </div>
</div>
```

script.js

```
angular.module('cspExample', [])
.controller('MainController', function() {
  this.counter = 0;
  this.inc = function() {
    this.counter++;
  };
  this.evil = function() {
    // jshint evil:true
    try {
      eval('1+2');
    } catch (e) {
      this.evilError = e.message;
    }
  };
});
```

protractor.js

```
var util, webdriver;

var incBtn = element(by.id('inc'));
var counter = element(by.id('counter'));
var evilBtn = element(by.id('evil'));
var evilError = element(by.id('evilError'));

function getAndClearSevereErrors() {
  return browser.manage().logs().get('browser').then(function(browserLog) {
    return browserLog.filter(function(logEntry) {
      return logEntry.level.value > webdriver.logging.Level.WARNING;
    });
  });
}

function clearErrors() {
  getAndClearSevereErrors();
}

function expectNoErrors() {
  getAndClearSevereErrors().then(function(filteredLog) {
```

```

        expect(filteredLog.length).toEqual(0);
        if (filteredLog.length) {
            console.log('browser console errors: ' + util.inspect(filteredLog));
        }
    });
}

function expectError(regex) {
    getAndClearSevereErrors().then(function(filteredLog) {
        var found = false;
        filteredLog.forEach(function(log) {
            if (log.message.match(regex)) {
                found = true;
            }
        });
        if (!found) {
            throw new Error('expected an error that matches ' + regex);
        }
    });
}

beforeEach(function() {
    util = require('util');
    webdriver = require('protractor/node_modules/selenium-webdriver');

    // For now, we only test on Chrome,
    // as Safari does not load the page with Protractor's injected script
    // and Firefox webdriver always disables content security policy (#1000)
    if (browser.params.browser !== 'chrome') {
        return;
    }

    it('should not report errors when the page is loaded', function() {
        // clear errors so we are not dependent on previous tests
        clearErrors();
        // Need to reload the page as the page is already loaded when
        // we come here
        browser.driver.getCurrentUrl().then(function(url) {
            browser.get(url);
        });
    });
});

```

```
    });  
    expectNoErrors();  
  });  
  
  it('should evaluate expressions', function() {  
    expect(counter.getText()).toEqual('0');  
    incBtn.click();  
    expect(counter.getText()).toEqual('1');  
    expectNoErrors();  
  });  
  
  it('should throw and report an error when using "eval"', function() {  
    evilBtn.click();  
    expect(evilError.getText()).toMatch(/Content Security Policy/);  
    expectError(/Content Security Policy/);  
  });
```



## ng-cut

- Angular@1.4.7
  - `ng` 模块中的指令
- 为剪切（cut）指定自定义行为。

## 指令信息

这个指令的执行优先级为0级

## 用法

用作属性：

```
<window, input, select, textarea, a
  ng-cut="expression">
...
</window, input, select, textarea, a>
```

## 参数

参数	形式	详细
ngCut	expression	通过剪切操作触发需要解析的表达式（事件对象可以通过 <code>\$event</code> 获得）

## 例子

index.html

```
<input ng-cut="cut=true" ng-init="cut=false; value='cut me'" ng-model="cut: {{cut}}"
```

## ng-paste

- Angular@1.4.7
- `ng` 模块中的指令

为粘贴（paste）指定自定义行为。

## 指令信息

这个指令的执行优先级为0级

## 用法

用作属性：

```
<window, input, select, textarea, a
  ng-paste="expression">
...
</window, input, select, textarea, a>
```

## 参数

参数	形式	详细
ngPaste	expression	通过粘贴操作触发需要解析的表达式（事件对象可以通过 \$event 获得）

## 例子

index.html

```
<input ng-paste="paste=true" ng-init="paste=false" placeholder='paste
pasted: {{paste}}'
```

## ng-hide

- Angular@1.4.7
- `ng` 模块中的指令

根据元素的 `ng-hide` 属性上表达式的解析式来控制元素的显示与隐藏。元素的显示与隐藏是根据元素上是否有 `.ng-hide` CSS 类来决定的。这个 CSS 类在 AngularJS 内部被预设好并设置了 `display` 样式为 `none`（使用了 `!important` 标志）。在 CSP 模式下请在你的文件中引入 `angular-csp.css`。（请看 `ngCsp`）

```
<!-- 当 $scope.myValue 值为真（元素被隐藏） -->
<div ng-hide="myValue" class="ng-hide"></div>

<!-- 当 $scope.myValue 值为假（元素则可见） -->
<div ng-hide="myValue"></div>
```

`ngHide` 的表达式被解析为真值后会为该元素添加一个 `.ng-hide` 的 CSS 类来使其隐藏。如果解析值为假，则会移除 `.ng-hide` CSS 类来让元素显示出来。

## 为什么要用 `!important` ？

你可能想知道为什么要在 `.ng-hide` CSS 类中使用 `!important`。这是因为 `.ng-hide` 选择器可以轻易被级别较高的选择器覆盖。举个例子，像改变一个 HTML 列表项目的 `display` 样式这样简单的事，就会将隐藏的元素显示出来。这更会为 CSS 架构带来更大的问题。

使用 `!important` 之后，显示和隐藏的行为就会按照预期实现，就算 CSS 选择器之前存在冲突也无妨（当然 `!important` 不能用于有冲突的样式上）。如果开发者想要去通过覆盖样式的方式来改变隐藏元素的方式，则在他们专属的 CSS 代码中使用 `!important` 就可以了。

## 覆盖 `.ng-hide`

默认的, `.ng-hide` CSS 类使用 `display: none!important` 来修饰元素。如果你希望改变 `ngShow/ngHide` 的隐藏方式, 那么你可以通过重设 `.ng-hide` CSS 类的样式来实现:

```
.ng-hide {  
  /* 这是隐藏元素的另一种方式 */  
  display: block!important;  
  position: absolute;  
  top: -9999px;  
  left: -9999px;  
}
```

## 关于使用 `ngHide` 动画需要注意的一点

`ngShow/ngHide` 动画的工作方式是, 当指令的表达式解析为 `true` 和 `false` 时, 随即触发显示和隐藏的事件。这个系统的工作方式就像 `ngClass` 的动画系统, 除非被添加或删除的 `.ng-hide` CSS 类被你自己的 CSS 类取代了。

```
/* 在页面底部可以找到例子 */  
  
.my-element.ng-hide-add, .my-element.ng-hide-remove {  
  
  transition: 0.5s linear all;  
}  
  
.my-element.ng-hide-add { ... }  
.my-element.ng-hide-add.ng-hide-add-active { ... }  
.my-element.ng-hide-remove { ... }  
.my-element.ng-hide-remove.ng-hide-remove-active { ... }
```

要记得, 截至 AngularJS 的 1.3.0-beta.11, 在动画进行的过程中那个, 不需要将显示属性更改为块 —— `ngAnimate` 将会为你触发样式的自动切换。

## 指令信息

这个指令的执行优先级为0级 这个指令可以被当做集合元素 (multiElement) 使用

## 用法

以元素属性的形式使用:

```
<ANY  
  ng-hide="expression">  
  ...  
</ANY>
```

## 动画

- 删除类 (removeClass) : `.ng-hide` - 将会在 `ngHide` 表达式解析为真之后并且在内容被设置为隐藏之前生效。
- 添加类 (addClass) : `.ngHide` - 将会在 `ngHide` 表达式解析为非真之后并且在内容被设置为可见之前生效。

点击这儿了解更多关于动画中的执行步骤 ( `$animate` ) 。

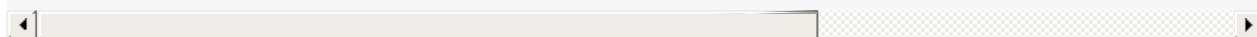
## 参数

参数	形式	详细
ngHide	expression	如果表达式解析值为真，元素则被隐藏。

## 例子

index.html

```
Click me: <input type="checkbox" ng-model="checked" aria-label="Toggle" />
<div>
  Show:
  <div class="check-element animate-hide" ng-show="checked">
    <span class="glyphicon glyphicon-thumbs-up"></span> I show up v
  </div>
</div>
<div>
  Hide:
  <div class="check-element animate-hide" ng-hide="checked">
    <span class="glyphicon glyphicon-thumbs-down"></span> I hide wh
  </div>
</div>
```



glyphicons.css

```
@import url(../../components/bootstrap-3.1.1/css/bootstrap.css);
```

animations.css

```
.animate-hide {  
  transition: all linear 0.5s;  
  line-height: 20px;  
  opacity: 1;  
  padding: 10px;  
  border: 1px solid black;  
  background: white;  
}  
  
.animate-hide.ng-hide {  
  line-height: 0;  
  opacity: 0;  
  padding: 0 10px;  
}  
  
.check-element {  
  padding: 10px;  
  border: 1px solid black;  
  background: white;  
}
```

#### protractor.js

```
var thumbsUp = element(by.css('span.glyphicon-thumbs-up'));  
var thumbsDown = element(by.css('span.glyphicon-thumbs-down'));  
  
it('should check ng-show / ng-hide', function() {  
  expect(thumbsUp.isDisplayed()).toBeFalsy();  
  expect(thumbsDown.isDisplayed()).toBeTruthy();  
  
  element(by.model('checked')).click();  
  
  expect(thumbsUp.isDisplayed()).toBeTruthy();  
  expect(thumbsDown.isDisplayed()).toBeFalsy();  
});
```

## ng-show

- Angular@1.4.7
- `ng` 模块中的指令

(译：与 `ngHide` 完全相反，只不过都是由 `.ng-hide` 这个 CSS 类来控制显隐的。)

根据元素的 `ng-show` 属性上表达式的解析式来控制元素的显示与隐藏。元素的显示与隐藏是根据元素上是否有 `.ng-hide` CSS 类来决定的。这个 CSS 类在 AngularJS 内部被预设好并设置了 `display` 样式为 `none` (使用了 `!important` 标志)。在 CSP 模式下请在你的文件中引入 `angular-csp.css`。(请看 `ngCsp`)

```
<!-- 当 $scope.myValue 值为真 (元素则可见) -->
<div ng-show="myValue"></div>

<!-- 当 $scope.myValue 值为假 (元素被隐藏) -->
<div ng-show="myValue" class="ng-hide"></div>
```

`ngshow` 的表达式被解析为假值后会为该元素添加一个 `.ng-hide` 的 CSS 类来使其隐藏。如果解析值为真，则会移除 `.ng-hide` CSS 类来让元素显示出来。

## 为什么要用 `!important` ?

你可能想知道为什么要在 `.ng-hide` CSS 类中使用 `!important`。这是因为 `.ng-hide` 选择器可以轻易被级别较高的选择器覆盖。举个例子，像改变一个 HTML 列表项目的 `display` 样式这样简单的事，就会将隐藏的元素显示出来。这更会为 CSS 架构带来更大的问题。

使用 `!important` 之后，显示和隐藏的行为就会按照预期实现，就算 CSS 选择器之前存在冲突也无妨 (当然 `!important` 不能用于有冲突的样式上)。如果开发者想要去通过覆盖样式的方式来改变隐藏元素的方式，则在他们专属的 CSS 代码中使用 `!important` 就可以了。

## 覆盖 `.ng-hide`



默认的, `.ng-hide` CSS 类使用 `display: none!important` 来修饰元素。如果你希望改变 `ngShow/ngshow` 的隐藏方式, 那么你可以通过重设 `.ng-show` CSS 类的样式来实现, 注意, 需要为选择器添加 `:not(.ng-hide-animate)` 伪类以确保扩展的动画类可以被正确的添加:

```
.ng-hide:not(.ng-hide-animate) {  
  /* 这是隐藏元素的另一种方式 */  
  display: block!important;  
  position: absolute;  
  top: -9999px;  
  left: -9999px;  
}
```

默认情况下你不需要在 CSS 中覆盖任何东西, 动画可以解决展示样式的问题。

## 关于使用 `ngShow` 动画需要注意的一点

`ngShow/ngHide` 动画的工作方式是, 当指令的表达式解析为 `true` 和 `false` 时, 随即触发显示和隐藏的事件。

这个系统的工作方式就像 `ngClass` 的动画系统, 除非你必须使用 `!important` 覆盖 `display` 属性, 以便在动画中演示出一个隐藏的行为。

```
/* 在页面底部可以找到例子 */  
  
.my-element.ng-show-add, .my-element.ng-show-remove {  
  
  transition: 0.5s linear all;  
}  
  
.my-element.ng-show-add { ... }  
.my-element.ng-show-add.ng-show-add-active { ... }  
.my-element.ng-show-remove { ... }  
.my-element.ng-show-remove.ng-show-remove-active { ... }
```

要记得, 截至 AngularJS 的 1.3.0-beta.11, 在动画进行的过程中那个, 不需要将显示属性更改为块 —— `ngAnimate` 将会为你触发样式的自动切换。

## 指令信息

这个指令的执行优先级为0级 这个指令可以被当做集合元素（multiElement）使用

## 用法

以元素属性的形式使用:

```
<ANY  
  ng-show="expression">  
  ...  
</ANY>
```

## 动画

- 删除类（removeClass）： `.ng-hide` - 将会在 `ngShow` 表达式解析为真之后并且在内容被设置为隐藏之前生效。
- 添加类（addClass）： `.ng-hide` - 将会在 `ngshow` 表达式解析为假之后并且在内容被设置为可见之前生效。

点击[这儿](#)了解更多关于动画中的执行步骤（`$animate`）。

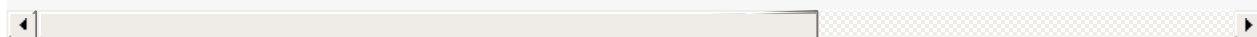
## 参数

参数	形式	详细
ngShow	<code>expression</code>	如果表达式解析值为假，元素则被隐藏。

## 例子

`index.html`

```
Click me: <input type="checkbox" ng-model="checked" aria-label="Toggle" />
<div>
  Show:
  <div class="check-element animate-show" ng-show="checked">
    <span class="glyphicon glyphicon-thumbs-up"></span> I show up v
  </div>
</div>
<div>
  Hide:
  <div class="check-element animate-show" ng-hide="checked">
    <span class="glyphicon glyphicon-thumbs-down"></span> I hide wh
  </div>
</div>
```



glyphicons.css

```
@import url(../../components/bootstrap-3.1.1/css/bootstrap.css);
```

animations.css

```
.animate-show {
  line-height: 20px;
  opacity: 1;
  padding: 10px;
  border: 1px solid black;
  background: white;
}

.animate-show.ng-hide-add, .animate-show.ng-hide-remove {
  transition: all linear 0.5s;
}

.animate-show.ng-hide {
  line-height: 0;
  opacity: 0;
  padding: 0 10px;
}

.check-element {
  padding: 10px;
  border: 1px solid black;
  background: white;
}
```

#### protractor.js

```
var thumbsUp = element(by.css('span.glyphicon-thumbs-up'));
var thumbsDown = element(by.css('span.glyphicon-thumbs-down'));

it('should check ng-show / ng-hide', function() {
  expect(thumbsUp.isDisplayed()).toBeFalsy();
  expect(thumbsDown.isDisplayed()).toBeTruthy();

  element(by.model('checked')).click();

  expect(thumbsUp.isDisplayed()).toBeTruthy();
  expect(thumbsDown.isDisplayed()).toBeFalsy();
});
```



## ng-href

- Angular@1.4.7
- `ng` 模块中的指令

在 `href` 属性中使用像 `{{hash}}` 这样的 Angular 标记时，如果用户在 Angular 替换掉 `{{hash}}` 之前点击了链接，那么链接则会走向错误的 URL 地址。直到 Angular 替换掉标记。这个链接将会被打破，并且很有可能将返回一个 404 错误。`ng-href` 指令将会解决这个问题。

错误的写法：

```
<a href="http://www.gravatar.com/avatar/{{hash}}">link1</a>
```

正确的写法：

```
<a ng-href="http://www.gravatar.com/avatar/{{hash}}">link1</a>
```

## 指令信息

这个指令的执行优先级为99级

## 用法

用作属性：

```
<A  
  ng-href="template">  
  ...  
</A>
```

## 参数

参数	形式	详细
ngHref	template	任何包含双花括号标记的字符串。

## 例子

这个例子展示了链接中 `href` , `ng-href` , `ng-click` 属性的各种组合使用以及他们的不同行为。

index.html

```
<input ng-model="value" /><br />
<a id="link-1" href ng-click="value = 1">link 1</a> (link, don't re
<a id="link-2" href="" ng-click="value = 2">link 2</a> (link, don't
<a id="link-3" ng-href="/{{'123'}}">link 3</a> (link, reload!)<br /
<a id="link-4" href="" name="xx" ng-click="value = 4">anchor</a> (
<a id="link-5" name="xxx" ng-click="value = 5">anchor</a> (no link
<a id="link-6" ng-href="{{value}}">link</a> (link, change location
```

protractor.js

```
it('should execute ng-click but not reload when href without value',
  element(by.id('link-1')).click();
  expect(element(by.model('value')).getAttribute('value')).toEqual('');
  expect(element(by.id('link-1')).getAttribute('href')).toBe('');
});

it('should execute ng-click but not reload when href empty string',
  element(by.id('link-2')).click();
  expect(element(by.model('value')).getAttribute('value')).toEqual('');
  expect(element(by.id('link-2')).getAttribute('href')).toBe('');
});

it('should execute ng-click and change url when ng-href specified',
  expect(element(by.id('link-3')).getAttribute('href')).toMatch(/\/123/);

  element(by.id('link-3')).click();
```

```
// At this point, we navigate away from an Angular page, so we need
// to use browser.driver to get the base webdriver.

browser.wait(function() {
    return browser.driver.getCurrentUrl().then(function(url) {
        return url.match(/\/123$/);
    });
}, 5000, 'page should navigate to /123');
});

it('should execute ng-click but not reload when href empty string and value is set', function() {
    element(by.id('link-4')).click();
    expect(element(by.model('value')).getAttribute('value')).toEqual('123');
    expect(element(by.id('link-4')).getAttribute('href')).toBe('');
});

it('should execute ng-click but not reload when no href but name specified', function() {
    element(by.id('link-5')).click();
    expect(element(by.model('value')).getAttribute('value')).toEqual('123');
    expect(element(by.id('link-5')).getAttribute('href')).toBe(null);
});

it('should only change url when only ng-href', function() {
    element(by.model('value')).clear();
    element(by.model('value')).sendKeys('6');
    expect(element(by.id('link-6')).getAttribute('href')).toMatch(/\/6$/);

    element(by.id('link-6')).click();

    // At this point, we navigate away from an Angular page, so we need
    // to use browser.driver to get the base webdriver.
    browser.wait(function() {
        return browser.driver.getCurrentUrl().then(function(url) {
            return url.match(/\/6$/);
        });
    }, 5000, 'page should navigate to /6');
});
```



## ng-src

- Angular@1.4.7
- `ng` 模块中的指令

在 `src` 属性中使用使用像 `{{hash}}` 这样的 Angular 标记时不能正确的工作：直到直到 Angular 替换掉标记之前，浏览器都会使用包含了 `{{hash}}` 的字面文本来获取源数据。`ng-src` 指令将会解决这个问题。

错误的写法：

```

```

正确的写法：

```

```

## 指令信息

这个指令的执行优先级为99级

## 用法

用作属性：

```
<IMG  
  ng-src="template">  
...  
</IMG>
```

## 参数

参数	形式	详细
ngSrc	expression	任何包含双花括号标记的字符串。

## ng-srcset

- Angular@1.4.7
- `ng` 模块中的指令

在 `srcset` 属性中使用使用像 `{{hash}}` 这样的 Angular 标记时不能正确的工作：直到直到 Angular 替换掉标记之前，浏览器都会使用包含了 `{{hash}}` 的字符串文本来获取源数据。`ng-srcset` 指令将会解决这个问题。

错误的写法：

```
<img srcset="http://www.gravatar.com/avatar/{{hash}} 2x" alt="Description" data-bbox="100 314 899 377"/>
```

正确的写法：

```
<img ng-srcset="http://www.gravatar.com/avatar/{{hash}} 2x" alt="Description" data-bbox="100 430 899 493"/>
```

## 指令信息

这个指令的执行优先级为99级

## 用法

用作属性：

```
<IMG
  ng-srcset="template">
...
</IMG>
```

## 参数

参数	形式	详细
ngSrcset	expression	任何包含双花括号标记的字符串。

## ng-if

- Angular@1.4.7
- ng 模块中的指令

ngIf 指令根据一个 { 表达式 } 的值在 DOM 树中删除或者重新创建某个部分。如果表达式链接的 ngIf 解析为 false 则元素则会从 DOM 中删除，否则（为 true ）元素的一个克隆会被插入到 DOM 中。

ngIf 与 ngShow 和 ngHide 的不同之处在于，ngIf 会将元素在 DOM 中完全的删除或者重建，而并不是通过 display 这个 css 属性来改变它的可见性。一个比较常见的例子是使用 CSS 选择器时要依赖于 DOM 中的某个元素的位置，如 :first-child 或者 :last-child 伪类，这个差异会变得很明显。

需要注意的是当使用 ngIf 删除一个元素时，这个元素的作用域（scope）也会被销毁，反之当元素恢复时一个新的作用域会被创建。被 ngIf 创建的作用域会使用原型继承的方式来从它的父级继承作用域。关于这点的一个重要意义是，如果 ngIf 同时使用 ngModel 绑定到父作用域上的一个 javascript 原始定义。在这种情况下，任何针对子作用域的修改可见性的方式都将会被父作用域的值覆盖。

同样地，ngIf 使用他们的编译状态来再现元素。这个行为的一个例子是，如果一个元素的 class 的属性会在该元素编译完成后被直接的改写，使用一些像 jQuery 的 .addClass() 这样的方法，在这之后删除了这个元素。当 ngIf 重新展现这个元素的时候被添加的 CSS 类会丢失掉，这是因为再生元素时使用的是原始的编译状态。

另外，你可以通过 ngAnimate 模块提供进入和离开的动画效果。

## 指令信息

这个指令会创建一个新的作用域 这个指令的执行优先级为600级 这个指令可以针对嵌套元素（multiElement）使用

## 用法

以元素属性的形式使用:

```
<ANY  
  ng-if="expression">  
  ...  
</ANY>
```

## 动画

**enter** - 仅在 `ngIf` 内容改变，新的 DOM 元素被创建并注入到 `ngIf` 容器之后生效。**leave** - 在 `ngIf` 的内容从 DOM 被删除之前生效。

点击[这儿](#)了解更多关于动画中的执行步骤（`$animate`）。

## 参数

参数	形式	详细
<code>ngIf</code>	<code>expression</code>	如果表达式解析值为假则元素会从 DOM 树中删除，反之向 DOM 树中添加一个编译元素的拷贝。

## 例子

index.html

```
<label>Click me: <input type="checkbox" ng-model="checked" ng-init=  
Show when checked:  
<span ng-if="checked" class="animate-if">  
  This is removed when the checkbox is unchecked.  
</span>
```

animations.css

```
.animate-if {  
  background:white;  
  border:1px solid black;  
  padding:10px;  
}  
  
.animate-if.ng-enter, .animate-if.ng-leave {  
  transition:all cubic-bezier(0.250, 0.460, 0.450, 0.940) 0.5s;  
}  
  
.animate-if.ng-enter,  
.animate-if.ng-leave.ng-leave-active {  
  opacity:0;  
}  
  
.animate-if.ng-leave,  
.animate-if.ng-enter.ng-enter-active {  
  opacity:1;  
}
```

## ng-switch



## ng-include

- Angular@1.4.7
- `ng` 模块中的指令

获取，编译，包含一个外部的 HTML 碎片。

默认情况下，模板 URL 被限定在与应用文件同一域和协议下。这样做是对其调用了 `$sce.getTrustedResourceUrl` 为了载入其他域或协议下的模板，你可以将它们加入你的白名单，或者将它们包装成受信任的值。参考 Angular 的 Strict Contextual Escaping. (SCE)（译：`$sce` 服务）

此外，浏览器的同源策略（Same Origin Policy）和跨域资源分享（Cross-Origin Resource Sharing : CORS）策略会进一步影响模板是否能成功加载，例如，`ngInclude` 不能工作在所有浏览器跨域环境的请求中，和访问某些浏览器的 `file://`。

## 指令信息

这个指令会创建一个新的作用域 这个指令的执行优先级为400级

## 用法

以元素属性的形式使用: (这个指令可以像自定义元素一样使用，但是要注意 IE 浏览器的限制。译：详见指引中的 IE 兼容性)

```
<ng-include
  src="string"
  [onload="string"]
  [autoscroll="string"]>
...
</ng-include>
```

以元素属性的形式使用:

```
<ANY
  ng-include="string"
  [onload="string"]
  [autoscroll="string"]>
...
</ANY>
```

用作 CSS 类

```
<ANY class="ng-include: string; [onload: string;] [autoscroll: str:
```

## 动画

**enter** - 新的内容被置入浏览器时的动画 **leave** - 存在的内容被移除时的动画

进入和离开的动画会同时发生。

点击[这儿](#)了解更多关于动画中的执行步骤（ `$animate` ）。

## 参数

参数	形式	详细
ngInclude / src	string	解析为 URL 的 angular 表达式，如果源是一个字符串内容，确保它被单引号包裹， 如： <code>src="'myPartialTemplate.html'"</code>
onload（可选）	string	新的部分载入后表达式被解析执行。
autoscroll（可选）	string	内容载入后是否调用 <code>\$anchorScroll</code> 来滚动视图  - 如果属性未设置，禁用滚动 - 如果设置了属性但没有赋值， 启用滚动 - 其他情况下在表达式解析为真值时启用滚动

## 事件

`$includeContentRequested`

每次 `ngInclude` 内容被请求时广播（Emit）这个事件。

类型：冒泡（emit）

目标：`ngInclude` 声明的作用域

## 参数

参数	形式	详细
<code>angularEvent</code>	<code>Object</code>	综合事件对象
<code>src</code>	<code>string</code>	载入内容的 URL

## `$includeContentLoaded`

每次 `ngInclude` 内容被重载时广播（Emit）这个事件。

类型：冒泡（emit）

目标：当前 `ngInclude` 的作用域

## 参数

参数	形式	详细
<code>angularEvent</code>	<code>Object</code>	综合事件对象
<code>src</code>	<code>string</code>	载入内容的 URL

## `$includeContentError`

当一个模板的 HTTP 请求返回错误响应时广播此事件（`status < 200 || status > 299`）

类型：冒泡（emit）

目标：`ngInclude` 声明的作用域

## 参数

参数	形式	详细
angularEvent	Object	综合事件对象
src	string	载入内容的 URL

## 例子

index.html

```
<div ng-controller="ExampleController">
  <select ng-model="template" ng-options="t.name for t in templates">
    <option value="">(blank)</option>
  </select>
  url of the template: <code>{{template.url}}</code>
  <hr/>
  <div class="slide-animate-container">
    <div class="slide-animate" ng-include="template.url"></div>
  </div>
</div>
```

script.js

```
angular.module('includeExample', ['ngAnimate'])
.controller('ExampleController', ['$scope', function($scope) {
  $scope.templates =
    [ { name: 'template1.html', url: 'template1.html'},
      { name: 'template2.html', url: 'template2.html'} ];
  $scope.template = $scope.templates[0];
}]);
```

template1.html

template1.html 的内容

template2.html

template2.html 的内容

animations.css

```
.slide-animate-container {  
  position:relative;  
  background:white;  
  border:1px solid black;  
  height:40px;  
  overflow:hidden;  
}  
  
.slide-animate {  
  padding:10px;  
}  
  
.slide-animate.ng-enter, .slide-animate.ng-leave {  
  transition:all cubic-bezier(0.250, 0.460, 0.450, 0.940) 0.5s;  
  
  position:absolute;  
  top:0;  
  left:0;  
  right:0;  
  bottom:0;  
  display:block;  
  padding:10px;  
}  
  
.slide-animate.ng-enter {  
  top:-50px;  
}  
.slide-animate.ng-enter.ng-enter-active {  
  top:0;  
}  
  
.slide-animate.ng-leave {  
  top:0;  
}  
.slide-animate.ng-leave.ng-leave-active {  
  top:50px;  
}
```

protractor.js

```
var templateSelect = element(by.model('template'));
var includeElem = element(by.css('[ng-include]'));

it('should load template1.html', function() {
  expect(includeElem.getText()).toMatch(/Content of template1.html/);
});

it('should load template2.html', function() {
  if (browser.params.browser == 'firefox') {
    // Firefox can't handle using selects
    // See https://github.com/angular/protractor/issues/480
    return;
  }
  templateSelect.click();
  templateSelect.all(by.css('option')).get(2).click();
  expect(includeElem.getText()).toMatch(/Content of template2.html/);
});

it('should change to blank', function() {
  if (browser.params.browser == 'firefox') {
    // Firefox can't handle using selects
    return;
  }
  templateSelect.click();
  templateSelect.all(by.css('option')).get(0).click();
  expect(includeElem.isPresent()).toBe(false);
});
```

## ng-init

- Angular@1.4.7
- `ng` 模块中的指令

`ngInit` 指令是你可以在当前作用域解析表达式。

这个指令可以随意将逻辑中非必须的东西添加进你的模板。只有少数情况下适合使用 `ngInit`，例如为 `ngRepeat` 的特定属性包装别名，你可以在下面的例子中看到；通过服务器端脚本注入数据。除了这些少数案例，在作用域使用控制器初始化值要比 `ngInit` 好得多。

注意：如果在 `ngInit` 中有一个使用了过滤器的定义，请确保你使用了括号包裹他们来保证正确的优先级：

```
<div ng-init="test1 = ($index | toString)"></div>
```

## 指令信息

这个指令的执行优先级为450级

## 用法

用作属性：

```
<ANY  
  ng-init="expression">  
  ...  
</ANY>
```

用作 CSS 类

```
<ANY class="ng-init: expression;" ... </ANY>
```

## 参数



参数	形式	详细
ngInit	expression	需要被转化的表达式

## 例子

index.html

```
<script>
  angular.module('initExample', [])
    .controller('ExampleController', ['$scope', function($scope) {
      $scope.list = [['a', 'b'], ['c', 'd']];
    }]);
</script>
<div ng-controller="ExampleController">
  <div ng-repeat="innerList in list" ng-init="outerIndex = $index">
    <div ng-repeat="value in innerList" ng-init="innerIndex = $index">
      <span class="example-init">list[ {{outerIndex}} ][ {{innerIndex}} ] = {{value}}
    </div>
  </div>
</div>
```

protractor.js

```
it('should alias index positions', function() {
  var elements = element.all(by.css('.example-init'));
  expect(elements.get(0).getText()).toBe('list[ 0 ][ 0 ] = a;');
  expect(elements.get(1).getText()).toBe('list[ 0 ][ 1 ] = b;');
  expect(elements.get(2).getText()).toBe('list[ 1 ][ 0 ] = c;');
  expect(elements.get(3).getText()).toBe('list[ 1 ][ 1 ] = d;');
});
```

## ng-jq

- Angular@1.4.7
- `ng` 模块中的指令

使用这个指令指定 `angular.element` 使用的库。要么是通过空的 `ng-jq` 来指定 `jqLite` 要么是在 `window` 下设置 `jquery` 的变量名（例如：`jQuery`）。

由于 Angular 在其载入后查询这个指令（不要等待 `DOMContentLoaded` 事件），它必须被放置到载入 Angular 的 `script` 标签前。而且，只有第一个 `ng-jq` 实例会被使用，其他则会被忽略。

## 指令信息

这个指令的执行优先级为0级

## 用法

用作属性：

```
<ANY  
  [ng-jq="string"]>  
...  
</ANY>
```

## 参数

参数	形式	详细
ngJq（可选）	<code>string</code>	被用于 <code>angular.element</code> 的库名，这个库可以在 <code>window</code> 下获取到。

## 例子

这个例子展示了如何使用 `ngJq` 指令来指定 `jqLite` 到 `html` 标签上。

```
<!doctype html>
<html ng-app ng-jq>
...
...
</html>
```

这个例子展示了如何以另一个不同的名字来使用 `jQuery` 库。这个库的名字必须可以在顶层（`window`）获取到。

```
<!doctype html>
<html ng-app ng-jq="jQueryLib">
...
...
</html>
```

## ng-keydown

- Angular@1.4.7
  - `ng` 模块中的指令
- 为 `keydown` 事件指定之定义行为。

## 指令信息

这个指令的执行优先级为0级

## 用法

用作属性：

```
<ANY ng-keydown="expression">
...
</ANY>
```

## 参数

参数	形式	详细
ngKeydown	<code>expression</code>	通过 <code>keydown</code> 操作触发需要解析的表达式（事件对象可以通过 <code>\$event</code> 获得并且有 <code>keyCode</code> ， <code>altKey</code> 等。

## 例子

```
<input ng-keydown="count = count + 1" ng-init="count=0">
key down count: {{count}}
```

## ng-keypress

- Angular@1.4.7
  - `ng` 模块中的指令
- 为 `keypress` 事件指定之定义行为。

### 指令信息

这个指令的执行优先级为0级

### 用法

用作属性：

```
<ANY ng-keypress="expression">
...
</ANY>
```

### 参数

参数	形式	详细
ngKeypress	expression	通过按键操作触发需要解析的表达式（事件对象可以通过 <code>\$event</code> 获得并且有 <code>keyCode</code> ， <code>altKey</code> 等。

### 例子

index.html

```
<input ng-keypress="count = count + 1" ng-init="count=0">
key press count: {{count}}
```

## ng-keyup

- Angular@1.4.7
  - `ng` 模块中的指令
- 为 `keyup` 事件指定之定义行为。

## 指令信息

这个指令的执行优先级为0级

## 用法

用作属性：

```
<ANY ng-keyup="expression">
...
</ANY>
```

## 参数

参数	形式	详细
ngKeyUp	expression	通过键弹起操作触发需要解析的表达式（事件对象可以通过 <code>\$event</code> 获得并且有 <code>keyCode</code> ， <code>altKey</code> 等

## 例子

index.html

```
<p>Typing in the input box below updates the key count</p>
<input ng-keyup="count = count + 1" ng-init="count=0"> key up count

<p>Typing in the input box below updates the keycode</p>
<input ng-keyup="event=$event">
<p>event keyCode: {{ event.keyCode }}</p>
<p>event altKey: {{ event.altKey }}</p>
```

## ng-list

- Angular@1.4.7
- `ng` 模块中的指令

分割字符串与数组字符串之间的文本转换。默认的分隔符是后面跟随一个空格的逗号 - 相当于 `ng-list=", "`。你可以为 `ngList` 属性指定一个自定义的值来作为分隔符 - 例如, `ng-list=" | "`。

这个指令的行为会受到 `ngTrim` 属性的影响。

如果 `ngTrim` 设置为 `false`, 分隔符旁的空白会被每个列表项遵守。这意味着指令的用户需要负责处理空白, 但仍允许你使用空白作为分隔符。如 `tab` 或换行字符。

反之, 当切分时忽略掉分隔符旁的空白 (就算它是组合列表项时别一起加入的也不例外), 并且在列表项被添加进 `model` 之前会剥离其周围的空白。

## 带验证的例子

app.js

```
angular.module('listExample', [])
.controller('ExampleController', ['$scope', function($scope) {
    $scope.names = ['morpheus', 'neo', 'trinity'];
}]);
```

index.html



```

<form name="myForm" ng-controller="ExampleController">
  <label>List: <input name="namesInput" ng-model="names" ng-list rec
  <span role="alert">
    <span class="error" ng-show="myForm.namesInput.$error.required">
      Required!</span>
    </span>
  <br>
  <tt>names = {{names}}</tt><br/>
  <tt>myForm.namesInput.$valid = {{myForm.namesInput.$valid}}</tt><br/>
  <tt>myForm.namesInput.$error = {{myForm.namesInput.$error}}</tt><br/>
  <tt>myForm.$valid = {{myForm.$valid}}</tt><br/>
  <tt>myForm.$error.required = {{!!myForm.$error.required}}</tt><br/>
</form>

```

protractor.js

```

var listInput = element(by.model('names'));
var names = element(by.exactBinding('names'));
var valid = element(by.binding('myForm.namesInput.$valid'));
var error = element(by.css('span.error'));

it('should initialize to model', function() {
  expect(names.getText()).toContain('["morpheus","neo","trinity"]');
  expect(valid.getText()).toContain('true');
  expect(error.getCssValue('display')).toBe('none');
});

it('should be invalid if empty', function() {
  listInput.clear();
  listInput.sendKeys('');

  expect(names.getText()).toContain('');
  expect(valid.getText()).toContain('false');
  expect(error.getCssValue('display')).not.toBe('none');
});

```

## Example - splitting on newline

index.html

```
<textarea ng-model="list" ng-list="\n" ng-trim="false"></textarea>
<pre>{{ list | json }}</pre>
```

protractor.js

```
it("should split the text by newlines", function() {
  var listInput = element(by.model('list'));
  var output = element(by.binding('list | json'));
  listInput.sendKeys('abc\ndef\nghi');
  expect(output.getText()).toContain(['\n  "abc",\n  "def",\n  "ghi"']);
});
```

## 指令信息

这个指令的执行优先级为0级

## 用法

用作属性：

```
<input
  [ng-list="string"]>
...
</input>
```

## 参数

参数	形式	详细
ngList（可选）	string	被用于分割值的分割符。

## **ng-model-options**

## ng-model

- Angular@1.4.7
- `ng` 模块中的指令

`ngModel` 指令将一个 `input` , `select` , `textarea` (或者自定义的表单控件) 绑定到使用了 `NgModelController` 的作用域的一个属性上, 这个属性是由这个指令创建并导出的。

`ngModel` 负责：

- 将视图 (view) 绑定到那些需要它的指令的模型 (model) 上, 如 `input` , `textarea` , `select` .
- 提供验证功能 (例如 : `required`, `number`, `email`, `url`) 。
- 保存控件的状态 (合法的/非法的 `valid/invalid`, 改变/未改变 `dirty/pristine`, 触摸过/未触摸 `touched/untouched`, 验证错误) 。
- 在元素上设置包含动画的相关的 CSS 类 (`ng-valid`, `ng-invalid`, `ng-dirty`, `ng-pristine`, `ng-touched`, `ng-untouched`)
- 在控件的父表 (form) 中注册。

注意：`ngModel` 将会试图绑定到当前作用域的属性上。这个属性是 `ngModel` 被赋值的表达式解析的结果, 但如果这个属性并不真实存在这个作用域中, 那么它会被隐性的创建并被添加到作用域中。

对于使用 `ngModel` 的最佳实践, 请看：

For best practices on using ngModel, see:

- [理解作用域 \(Scopes\)](#)

关于如何使用 `ngModel` 基础的例子, 请看 (译：下面的例子都在指令的 `ng.input` 中)：

- `input`
  - `text`
  - `checkbox`
  - `radio`
  - `number`
  - `email`
  - `url`

- date
- datetime-local
- time
- month
- week
- select
- textarea

## CSS 类

下面这些 CSS 类的增删是由 `input/select/textarea` 元素所依赖的模型值的合法性 (validity) 所决定的。

- `ng-valid` : 模型值是合法的
- `ng-invalid` : 模型值是非法的
- `ng-valid-[key]` : 由 `$setValidity` 添加的合法键
- `ng-invalid-[key]` : 由 `$setValidity` 添加的非法键
- `ng-pristine` : 控件还没有发生过交互行为
- `ng-dirty` : 控件发生过交互行为
- `ng-touched` : 控件已经失去焦点
- `ng-untouched` : 控件还没有失去焦点
- `ng-pending` : 任何异步的验证都没有完成 (`$asyncValidators`)

要记得, `ngAnimate` 在删除或添加这些 CSS 类时可以对它们中的每一项进行检测。

## 动画钩子

当依附在模型上的表单元素添加或删除了任意相关的 CSS 类时, 模型上的动画都将被触发。这些类有: `.ng-pristine`, `.ng-dirty`, `.ng-invalid`, `.ng-valid` 当然也包括模型执行的任何其他的验证。动画在 `ngModel` 被触发的方式与 `ngClass` 指令中的方式相似, 这些动画可以被钩嵌于 CSS 过渡, 关键帧动画, JS 动画。

下面展示了一个简单的 CSS 过渡动画的应用, 当表单元素被验证后将其渲染成非法的样式。

```
/* 要确保引入了 `ngAnimate` 模块以保证能钩嵌更优良的动画效果。 */

.my-input {
  transition:0.5s linear all;
  background: white;
}
.my-input.ng-invalid {
  background: red;
  color:white;
}
```

## 指令信息

这个指令的执行优先级为1级

## 用法

用作属性：

```
<input>
...
</input>
```

## 例子

index.html

```
<script>
angular.module('inputExample', [])
  .controller('ExampleController', ['$scope', function($scope) {
    $scope.val = '1';
  }]);
</script>
<style>
.my-input {
  transition:all linear 0.5s;
  background: transparent;
}
.my-input.ng-invalid {
  color:white;
  background: red;
}
</style>
<p id="inputDescription">
  Update input to see transitions when valid/invalid.
  Integer is a valid value.
</p>
<form name="testForm" ng-controller="ExampleController">
  <input ng-model="val" ng-pattern="/^\d+$/" name="anim" class="my-
    aria-describedby="inputDescription" />
</form>
```

## 绑定到属性获取器/属性设置器（Binding to a getter/setter）

有时候将 `ngModel` 绑定到 `getter/setter` 函数上是有好处的。当不为 `getter/setter` 传入参数时会返回改模型的表示，当传入唯一参数时会设置模型的内部状态，有事可以用来比较模型的内部表示 与导出到视图的模型有何不同。

最佳实践：要保证 ``getters`` 的速度，因为 Angular 相较于你代码的其他部分可能会

你可以为一个元素添加 `ng-model-options="{ getterSetter: true }"` 来使 `ng-model` 依附到 `getter/setter` 上。你也可以将 `ng-model-options="{ getterSetter: true }"` 添加到一个 `<form>` 标签上，使其内部的所有表单都拥有 `getter/setter`，查看 `ngModelOptions` 了解更多。

下面展示了如何使用 `getter/setter` 形式的 `ngModel`

index.html

```
<div ng-controller="ExampleController">
  <form name="userForm">
    <label>Name:
      <input type="text" name="userName"
        ng-model="user.name"
        ng-model-options="{ getterSetter: true }" />
    </label>
  </form>
  <pre>user.name = <span ng-bind="user.name()"></span></pre>
</div>
```

app.js

```
angular.module('getterSetterExample', [])
.controller('ExampleController', ['$scope', function($scope) {
  var _name = 'Brian';
  $scope.user = {
    name: function(newName) {
      // Note that newName can be undefined for two reasons:
      // 1. Because it is called as a getter and thus called with no
      // 2. Because the property should actually be set to undefined
      //    input is invalid
      return arguments.length ? (_name = newName) : _name;
    }
  };
}]);
```



## ng-mousedown

- Angular@1.4.7
- `ng` 模块中的指令

为 `mousedown` 事件指定之定义行为。

## 指令信息

这个指令的执行优先级为0级

## 用法

用作属性：

```
<ANY ng-mousedown="expression">
...
</ANY>
```

## 参数

参数	形式	详细
ngKeydown	<code>expression</code>	通过 <code>mousedown</code> 操作触发需要解析的表达式（事件对象可以通过 <code>\$event</code> 获得。

## 例子

```
<button ng-mousedown="count = count + 1" ng-init="count=0">
  Increment (on mouse down)
</button>
count: {{count}}
```

## ng-mouseenter

- Angular@1.4.7
  - `ng` 模块中的指令
- 为 `mouseenter` 事件指定之定义行为。

### 指令信息

这个指令的执行优先级为0级

### 用法

用作属性：

```
<ANY ng-mouseenter="expression">
...
</ANY>
```

### 参数

参数	形式	详细
ngMouseenter	<code>expression</code>	通过 <code>mouseenter</code> 操作触发需要解析的表达式（事件对象可以通过 <code>\$event</code> 获得）。

### 例子

```
<button ng-mouseenter="count = count + 1" ng-init="count=0">
  Increment (when mouse enters)
</button>
count: {{count}}
```

## ng-mouseleave

- Angular@1.4.7
- `ng` 模块中的指令

为 `mouseleave` 事件指定之定义行为。

### 指令信息

这个指令的执行优先级为0级

### 用法

用作属性：

```
<ANY ng-mouseleave="expression">
...
</ANY>
```

### 参数

参数	形式	详细
ngMouseleave	expression	通过 <code>mouseleave</code> 操作触发需要解析的表达式（事件对象可以通过 <code>\$event</code> 获得）。

### 例子

```
<button ng-mouseleave="count = count + 1" ng-init="count=0">
  Increment (when mouse leaves)
</button>
count: {{count}}
```

## ng-mousemove

- Angular@1.4.7
  - `ng` 模块中的指令
- 为 `mousemove` 事件指定之定义行为。

### 指令信息

这个指令的执行优先级为0级

### 用法

用作属性：

```
<ANY ng-mousemove="expression">
...
</ANY>
```

### 参数

参数	形式	详细
ngMousemove	<code>expression</code>	通过 <code>mousemove</code> 操作触发需要解析的表达式（事件对象可以通过 <code>\$event</code> 获得）。

### 例子

```
<button ng-mousemove="count = count + 1" ng-init="count=0">
  Increment (when mouse moves)
</button>
count: {{count}}
```

## ng-mouseover

- Angular@1.4.7
- `ng` 模块中的指令

为 `mouseover` 事件指定之定义行为。

## 指令信息

这个指令的执行优先级为0级

## 用法

用作属性：

```
<ANY ng-mouseover="expression">
...
</ANY>
```

## 参数

参数	形式	详细
ngMouseover	<code>expression</code>	通过 <code>mouseover</code> 操作触发需要解析的表达式（事件对象可以通过 <code>\$event</code> 获得）。

## 例子

```
<button ng-mouseover="count = count + 1" ng-init="count=0">
  Increment (when mouse is over)
</button>
count: {{count}}
```

## ng-mouseup

- Angular@1.4.7
  - `ng` 模块中的指令
- 为 `mouseup` 事件指定之定义行为。

## 指令信息

这个指令的执行优先级为0级

## 用法

用作属性：

```
<ANY ng-mouseup="expression">
...
</ANY>
```

## 参数

参数	形式	详细
ngMouseup	<code>expression</code>	通过 <code>mouseup</code> 操作触发需要解析的表达式（事件对象可以通过 <code>\$event</code> 获得）。

## 例子

```
<button ng-mouseup="count = count + 1" ng-init="count=0">
  Increment (on mouse up)
</button>
count: {{count}}
```

**<select>**

## ng-options



## ng-value

- Angular@1.4.7
- `ng` 模块中的指令

将表达式解析值绑定到 `<option>` 或者 `input[radio]` 的 `value` 属性上，以便在元素被选取时，这个元素上的 `ngModel` 可以设置绑定值。

当使用 `ngRepeat` 动态生成单选按钮时，`ngValue` 的作用会体现出来，下面有例子。

同样地，`ngValue` 也可用于生成 `select` 的 `option` 元素。然而这种情况下，`value` 只支持字符串，所以 `ngModel` 的结果要保证是字符串，`ngOptions` 指令则支付非字符串形式的 `value`。

## 指令信息

这个指令的执行优先级为0级

## 用法

用作属性：

```
<input  
  [ng-value="string"]>  
...  
</input>
```

## 参数

参数	形式	详细
ngValue (可选)	string	angular 表达式, 它的值会被绑定到表单元素的 <code>value</code> 属性上。

## 例子

index.html

```
<script>
  angular.module('valueExample', [])
    .controller('ExampleController', ['$scope', function($scope) {
      $scope.names = ['pizza', 'unicorns', 'robots'];
      $scope.my = { favorite: 'unicorns' };
    }]);
</script>
<form ng-controller="ExampleController">
  <h2>Which is your favorite?</h2>
  <label ng-repeat="name in names" for="{{name}}">
    {{name}}
    <input type="radio"
      ng-model="my.favorite"
      ng-value="name"
      id="{{name}}"
      name="favorite">
  </label>
  <div>You chose {{my.favorite}}</div>
</form>
```

protractor.js

```
var favorite = element(by.binding('my.favorite'));

it('should initialize to model', function() {
  expect(favorite.getText()).toContain('unicorns');
});

it('should bind the values to the inputs', function() {
  element.all(by.model('my.favorite')).get(0).click();
  expect(favorite.getText()).toContain('pizza');
});
```

## **ng-pluralize**

## ng-repeat

- Angular@1.4.7
- `ng` 模块中的指令

`ngRepeat` 指令会为集合中每一项实例化出一个模板。每个模板的实例有它自己的作用域（scope），在此作用域中会给出集合中当前项所对应的循环变量，和与该项目索引或键值对应的 `$index`。

会在每个模板实例所在的局部作用域中暴露一些特殊的属性。包括：

变量	形式	详细
<code>\$index</code>	<code>number</code>	重复元素的在迭代中的偏移（0.....length-1）
<code>\$first</code>	<code>boolean</code>	如果重复元素在迭代的第一位则为 <code>true</code>
<code>\$middle</code>	<code>boolean</code>	如果重复元素在迭代的中间位置则为 <code>true</code>
<code>\$last</code>	<code>boolean</code>	如果重复元素在迭代的最后一位则为 <code>true</code>
<code>\$even</code>	<code>boolean</code>	如果迭代位置 <code>\$index</code> 是偶数则为真
<code>\$odd</code>	<code>boolean</code>	如果迭代位置 <code>\$index</code> 是奇数则为真

可以使用 ``ngInit`` 为这些属性创建别名。可以用于实例化嵌套的 ``ngRepeat``。

## 迭代对象的属性

可以通过如下方式使用 `ngRepeat` 迭代对象的属性。

```
<div ng-repeat="(key, value) in myObj"> ... </div>
```

你需要知道 JavaScript 规范中没有定义返回的对象键的顺序（Angular 1.3 之后，`ng-repeat` 指令使用了字母对键进行排序）。

版本 1.4 移除了字母排序。当轮询 `myObj` 中的 `key` 时，我们默认使用的是浏览器返回的顺序。似乎是这样的，浏览器通常情况下遵循的策略是键被定义时提供的顺序。尽管如此，也有例外情况，就是在键被删除并恢复的时候，请

看[https://developer.mozilla.org/en-US/docs/Web/JavaScript/Reference/Operators/delete#Cross-browser\\_issues](https://developer.mozilla.org/en-US/docs/Web/JavaScript/Reference/Operators/delete#Cross-browser_issues)

如果这不是你所期望的，那么推荐你在把数据提供给 `ngRepeat` 之前将你的对象转化为排序过的数组。你可以通过使用像 `toArrayFilter` 这样的过滤器或者自行行为对象添加一个 `$watch` 来实现。

## 跟踪与重复（Tracking and Duplicates）

当集合的内容改变时，`ngRepeat` 也会针对 DOM 做出相应的改变：

- 添加一项，就会向 DOM 中添加这个模板的新的实例。
- 删除一项，从DOM 中删除它的模板实例。
- 当项目发生重排序的时候，它们各自的模板也会在 DOM 中重新排序。

默认情况下，`ngRepeat` 不允许数组中有重复的项。这是由于如果存在重复项的话，不可能维持集合项和 DOM 项之间一一映射的关系。

如果你想迭代重复的项，你可以使用你自己的 `track by` 表达式来替换默认的跟踪行为。

举个例子，你可以使用特殊的作用域属性 `$index` 作为索引来跟踪集合中的每一项。

```
<div ng-repeat="n in [42, 42, 43, 43] track by $index">
  {{n}}
</div>
```

你可以在 `track by` 中使用任意的表达式，包括在作用域中的自定义函数。

```
<div ng-repeat="n in [42, 42, 43, 43] track by myTrackingFunction(n)">
  {{n}}
</div>
```

如果你使用的对象拥有可以用来识别的属性，你可以跟踪这个识别码来代替跟踪整个对象。如果你稍后重载了你的数据，`ngRepeat` 也不会为已经渲染过的项重建 DOM 元素，即时集合中的 JavaScript 对象已经被新的东西替换。

```
<div ng-repeat="model in collection track by model.id">
  {{model.name}}
</div>
```

在没有为 `track by` 提供表达式的情况下就使用项的特性进行跟踪，相当于使用了内置的 `$id` 函数。

```
<div ng-repeat="obj in collection track by $id(obj)">
  {{obj.prop}}
</div>
```

注意：`track by` 必须总是位于表达式的最末尾的位置：

```
<div ng-repeat="model in collection | orderBy: 'id' as filtered_res">
  {{model.name}}
</div>
```

## 特殊的重复的起止点（Special repeat start and end points）

为了重复一系列的元素而不是一个父元素，`ngRepeat`（以及其他的 `ng` 指令）提供了可以扩展循环范围的工具，它分别使用 `ng-repeat-start` 和 `ng-repeat-end` 明确定义循环的起止点。`ng-repeat-start` 虽然指令与 `ng-repeat` 工作方式相同，但是它会重复出从它自身起（包括 `ng-repeat-start` 所在的标签）到 `ng-repeat-end` 所在标签为止，内部所有的 HTML 代码。

下面的例子使用了这个特性：

```
<header ng-repeat-start="item in items">
  Header {{ item }}
</header>
<div class="body">
  Body {{ item }}
</div>
<footer ng-repeat-end>
  Footer {{ item }}
</footer>
```

And with an input of `['A', 'B']` for the items variable in the example above, the output will evaluate to:

```
<header>
  Header A
</header>
<div class="body">
  Body A
</div>
<footer>
  Footer A
</footer>
<header>
  Header B
</header>
<div class="body">
  Body B
</div>
<footer>
  Footer B
</footer>
```

自定义 `ngRepeat` 起止点的这种方式还支持 AngularJS 中提供的其他所有 HTML 指令的句法风格。（如 **`data-ng-repeat-start`**, **`x-ng-repeat-start`** 和 **`ng:repeat-start`**）

## 指令信息

这个指令会创建一个新的作用域 这个指令的执行优先级为1000级 这个指令可以针对嵌套元素（multiElement）使用

## 用法

以元素属性的形式使用:

```
<ANY
  ng-repeat="repeat_expression">
  ...
</ANY>
```

## 动画

- .enter** - 当一个新的项被添加到列表中或者被过滤器保留了下来。
  - .leave** - 当一个新的项从列表中被移除或者被过滤器过滤掉。
  - .move** - 当一个邻项被过滤掉而引起了重排序，或者项目内容发生了重排序。
- 点击[这儿](#)了解更多关于动画中的执行步骤（`$animate`）。

## 参数

参数	形式	详细
		<p>这个表达式需要说明怎样去枚举一个集合，当前支持以下几种形式：</p> <ul style="list-style-type: none"><li>- <code>variable in expression</code> - <code>variable</code> 是用户定义的循环变量，<code>expression</code> 是用于集合枚举的一个作用域上的表达式。 例如：<code>album in artist.albums</code></li><li>- <code>(key, value) in expression</code> - <code>key</code> 和 <code>value</code> 是可以任意由定义 的识别符，<code>expression</code> 是用于集合枚举的一个作用域上的表达式。 例如：<code>(name, age) in {'adam':10, 'amalie':12}</code></li><li>- <code>variable in expression track by tracking_expression</code> - 你可以设置一个跟踪</li></ul>



ngRepeat	repeat_expression	<p>(tracking) 表达式用于将 DOM 元素和集合中的对象联系起来。如果没有提供跟踪用的表达式，ng-repeat 默认使用身份 (identity) 来连接元素。使用多个跟踪表达式来解析同一个键是会产生错误的。（这将意味着两个独立的对象映射到同一个 DOM 元素上，所以这是不可能的）要注意，跟踪表达式要写在最末尾的位置，在过滤器之后，在其他表达式之后。</p> <p>例如：item in items 与 item in items track by \$id(item) 是等效的，这表明是通过数组项的身份连接了DOM元素。</p> <p>例如：item in items track by \$id(item)。内置的 \$id() 函数可以用来为数组项分配唯一的 \$\$hashKey 属性。这个属性之后会作为一个键将 DOM 元素和相应的数组项连接起来。移动数组中相同的对象也会移动以同样的方式移动 DOM 中的元素。</p> <p>例如：item in items track by item.id 是一个来自数据库的项的一个典型的模式。在这种情况下，对象的身份就不重要了，两个对象只有 id 属性相同是才会被认为相等。</p> <p>例如：item in items 竖线 filter:searchText track by item.id 为项添加过滤去的跟组表达式。</p> <p>- variable in expression as alias_expression – 你可以提供一个别名表达式来存储过滤后结果。比较有代表性的用法当过滤器在循环体上生效时渲染一个特殊消息，但是过滤后的结果被置空。</p> <p>例如：</p> <pre>item in items 竖线 filter:x as results</pre> <p>将会在过滤进程完成后将 items 存储成 results</p> <p>请注意 as [variable name] 不是一个操作行为而是 ngRepeat 微语法的一部分。所以他只能用在最后（不能用作一个操作，而且要再表达式的内部）。</p> <p>例如：item in items 竖线 filter : x 竖线 orderBy : order 竖线 limitTo : limit as results .</p>
----------	-------------------	---

## 例子

这个例子，在作用域中初始化了一个名字列表，之后使用 ngRepeat 展示出了其中的每一个人。

index.html

```
<div ng-init="friends = [
  {name:'John', age:25, gender:'boy'},
  {name:'Jessie', age:30, gender:'girl'},
  {name:'Johanna', age:28, gender:'girl'},
  {name:'Joy', age:15, gender:'girl'},
  {name:'Mary', age:28, gender:'girl'},
  {name:'Peter', age:95, gender:'boy'},
  {name:'Sebastian', age:50, gender:'boy'},
  {name:'Erika', age:27, gender:'girl'},
  {name:'Patrick', age:40, gender:'boy'},
  {name:'Samantha', age:60, gender:'girl'}
]">
  I have {{friends.length}} friends. They are:
  <input type="search" ng-model="q" placeholder="filter friends..." />
  <ul class="example-animate-container">
    <li class="animate-repeat" ng-repeat="friend in friends | filter:q"
      [{{$index + 1}}] {{friend.name}} who is {{friend.age}} years
    </li>
    <li class="animate-repeat" ng-if="results.length == 0">
      <strong>No results found...</strong>
    </li>
  </ul>
</div>
```

animations.css

```
.example-animate-container {  
  background:white;  
  border:1px solid black;  
  list-style:none;  
  margin:0;  
  padding:0 10px;  
}  
  
.animate-repeat {  
  line-height:40px;  
  list-style:none;  
  box-sizing:border-box;  
}  
  
.animate-repeat.ng-move,  
.animate-repeat.ng-enter,  
.animate-repeat.ng-leave {  
  transition:all linear 0.5s;  
}  
  
.animate-repeat.ng-leave.ng-leave-active,  
.animate-repeat.ng-move,  
.animate-repeat.ng-enter {  
  opacity:0;  
  max-height:0;  
}  
  
.animate-repeat.ng-leave,  
.animate-repeat.ng-move.ng-move-active,  
.animate-repeat.ng-enter.ng-enter-active {  
  opacity:1;  
  max-height:40px;  
}
```

protractor.js

```
var friends = element.all(by.repeater('friend in friends'));

it('should render initial data set', function() {
  expect(friends.count()).toBe(10);
  expect(friends.get(0).getText()).toEqual('[1] John who is 25 years old');
  expect(friends.get(1).getText()).toEqual('[2] Jessie who is 30 years old');
  expect(friends.last().getText()).toEqual('[10] Samantha who is 60 years old');
  expect(element(by.binding('friends.length')).getText())
    .toMatch("I have 10 friends. They are:");
});

it('should update repeater when filter predicate changes', function() {
  expect(friends.count()).toBe(10);

  element(by.model('q')).sendKeys('ma');

  expect(friends.count()).toBe(2);
  expect(friends.get(0).getText()).toEqual('[1] Mary who is 28 years old');
  expect(friends.last().getText()).toEqual('[2] Samantha who is 60 years old');
});
```

## ng-style

- Angular@1.4.7
- `ng` 模块中的指令

`ngStyle` 指令允许你在 HTML 元素上设置 CSS 样式。

## 指令信息

这个指令的执行优先级为0级

## 用法

用作属性：

```
<ANY  
  ng-style="expression">  
  ...  
</ANY>
```

用作 CSS 类

```
<ANY class="ng-style: expression;"> ... </ANY>
```

## 参数

参数	形式	详细
ngStyle	expression	表达式解析成为一个对象，这个对象的键就是 CSS 的样式名，键对应的值就是样式的值 由于一些 CSS 的样式名对于一个对象来说不是一种合法的键名的表示，所以它们必须添加引号。请看下面关于 'background-color' 的例子

## 例子

## index.html

```
<input type="button" value="set color" ng-click="myStyle={color:'red'}">
<input type="button" value="set background" ng-click="myStyle={'background-color':'red'}">
<input type="button" value="clear" ng-click="myStyle={}">
<br/>
<span ng-style="myStyle">Sample Text</span>
<pre>myStyle={{myStyle}}</pre>
```

## style.css

```
span {
  color: black;
}
```

## protractor.js

```
var colorSpan = element(by.css('span'));

it('should check ng-style', function() {
  expect(colorSpan.getCssValue('color')).toBe('rgba(0, 0, 0, 1)');
  element(by.css('input[value=\'set color\']')).click();
  expect(colorSpan.getCssValue('color')).toBe('rgba(255, 0, 0, 1)');
  element(by.css('input[value=clear]')).click();
  expect(colorSpan.getCssValue('color')).toBe('rgba(0, 0, 0, 1)');
});
```

## ng-transclude

- Angular@1.4.7
- `ng` 模块中的指令

这个指令会在父级指令的内部标记一个插入点，当父级指令重新进行模板渲染时，会将标签内原有的 DOM 安放到插入点的位置。

元素内的内容会在被插入到特定的位置前删除掉。

## 指令信息

这个指令的执行优先级为0级

## 用法

用作元素：（这个指令可以当做自定义标签使用，但是要注意IE的兼容性）

```
<ng-transclude>
...
</ng-transclude>
```

用作属性：

```
<ANY>
...
</ANY>
```

用作 CSS 类

```
<ANY class=""> ... </ANY>
```

## 例子

index.html

```
<script>
angular.module('transcludeExample', [])
  .directive('pane', function(){
    return {
      restrict: 'E',
      transclude: true,
      scope: { title:'@' },
      template: '<div style="border: 1px solid black;">' +
        '<div style="background-color: gray">{{title}}</div>' +
        '<ng-transclude></ng-transclude>' +
        '</div>'
    };
  })
  .controller('ExampleController', ['$scope', function($scope) {
    $scope.title = 'Lorem Ipsum';
    $scope.text = 'Neque porro quisquam est qui dolorem ipsum quia
  }]);
</script>
<div ng-controller="ExampleController">
  <input ng-model="title" aria-label="title"> <br/>
  <textarea ng-model="text" aria-label="text"></textarea> <br/>
  <pane title="{{title}}">{{text}}</pane>
</div>
```

protractor.js



```
it('should have transcluded', function() {  
  var titleElement = element(by.model('title'));  
  titleElement.clear();  
  titleElement.sendKeys('TITLE');  
  var textElement = element(by.model('text'));  
  textElement.clear();  
  textElement.sendKeys('TEXT');  
  expect(element(by.binding('title')).getText()).toEqual('TITLE');  
  expect(element(by.binding('text')).getText()).toEqual('TEXT');  
});
```

## <script>

- Angular@1.4.7
- `ng` 模块中的指令

将 `<script>` 里的内容载入到 `$templateCache`，以便

`ngInclude`，`ngView` 或其他指令使用这个模板，`<script>` 的 `type` 属性必须设置成 `text/ng-template`，并且还要通过 `id` 属性为模板设置一个缓存名称，然后这个名称就可以用在指令的 `templateUrl` 属性中了。

## 指令信息

这个指令的执行优先级为0级

## 用法

像个元素（标签）一样使用

html

```
<script
  type="string"
  id="string">
  ...
</script>
```

## 参数

参数	形式	详细
type	string	必须设置为 'text/ng-template'.
id	string	模板的缓存名称

## 例子

index.html

```
<script type="text/ng-template" id="/tpl.html">
  Content of the template.
</script>

<a ng-click="currentTpl='/tpl.html'" id="tpl-link">Load inlined template</a>
<div id="tpl-content" ng-include src="currentTpl"></div>
```

protractor.js

```
it('should load template defined inside script tag', function() {
  element(by.css('#tpl-link')).click();
  expect(element(by.css('#tpl-content')).getText()).toMatch(/Content of the template./);
});
```

**ngMessage** **ng-message-exp**

**ngMessage** **ng-message**

**ngMessage** **ng-messages-include**

**ngMessage** **ng-messages**

**ngRoute** **ng-view**



**ngTouch** **ng-click**

## ngTouch **ng-swipe-left**

## ngTouch ng-swipe-right

# currency

- Angular@1.4.7
  - `ng` 模块中的过滤器
- 将数字格式化为货币（currency）（例如：\$1,234.56）。如果没有提供货币符号时，默认使用当前语言环境（locale）的符号。

## 用法

HTML 模板绑定中

```
{{ currency_expression | currency : symbol : fractionSize }}
```

JavaScript 中

```
$filter('currency')(amount, symbol, fractionSize)
```

## 参数

参数	形式	详细
amount	number	过滤器的输入
symbol(可选)	string	货币符号或者要展示的识别码
fractionSize(可选)	number	小数位的取舍量。默认为当前语言地区的最大量。

## 返回

`string` 格式化后的数值

## 例子

index.html

```
<script>
  angular.module('currencyExample', [])
    .controller('ExampleController', ['$scope', function($scope) {
      $scope.amount = 1234.56;
    }]);
</script>
<div ng-controller="ExampleController">
  <input type="number" ng-model="amount" aria-label="amount"> <br>

  default currency symbol ($):
  <span id="currency-default">{{amount | currency}}</span><br>

  custom currency identifier (USD$):
  <span id="currency-custom">{{amount | currency:"USD$"}}</span>

  no fractions (0):
  <span id="currency-no-fractions">{{amount | currency:"USD$":0}}</span>
</div>
```

protractor.js

```
it('should init with 1234.56', function() {
  expect(element(by.id('currency-default')).getText()).toBe('$1,234.56');
  expect(element(by.id('currency-custom')).getText()).toBe('USD$1,234.56');
  expect(element(by.id('currency-no-fractions')).getText()).toBe('$1,235');
});

it('should update', function() {
  if (browser.params.browser == 'safari') {
    // Safari 不识别减号键
    // 请看 https://github.com/angular/protractor/issues/481
    return;
  }
  element(by.model('amount')).clear();
  element(by.model('amount')).sendKeys('-1234');
  expect(element(by.id('currency-default')).getText()).toBe('- $1,234');
  expect(element(by.id('currency-custom')).getText()).toBe('- USD$1,234');
  expect(element(by.id('currency-no-fractions')).getText()).toBe('- $1,235');
});
```

## date

- Angular@1.4.7
- `ng` 模块中的过滤器

根据要求的格式将日期格式化为字符串。

格式化的字符串可以由以下原件组成：

- 'yyyy': 年份用4位数字表示(e.g. AD 1 => 0001, AD 2010 => 2010)
- 'yy': 年份用2位数字表示, 补全0 (00-99). (e.g. AD 2001 => 01, AD 2010 => 10)
- 'y': 年份用最少位数字表示, e.g. (AD 1 => 1, AD 199 => 199)
- 'MMMM': 月份 (January-December)
- 'MMM': 月份 (Jan-Dec)
- 'MM': 月份, 补全0 (01-12)
- 'M': 月份 (1-12)
- 'dd': 日期, 补全0 (01-31)
- 'd': 日期 (1-31)
- 'EEEE': 星期,(Sunday-Saturday)
- 'EEE': 星期, (Sun-Sat)
- 'HH': 小时, 补全0 (00-23)
- 'H': 补全0 (0-23)
- 'hh': AM/PM 表示的小时, 补全0 (01-12)
- 'h': AM/PM 表示的小时, (1-12)
- 'mm': 分钟, 补全0 (00-59)
- 'm': 分钟 (0-59)
- 'ss': 秒, 补全0 (00-59)
- 's': 秒 (0-59)
- 'sss': 毫秒, 补全0 (000-999)
- 'a': AM/PM 标记
- 'Z': 用4位表示时区的偏移 (-1200-+1200)
- 'ww': 周数, 补全0 (00-53). 01周是每年的包含第一个周四的周
- 'w': Week of year (0-53). 01周是每年的包含第一个周四的周
- 'G', 'GG', 'GGG': 时代的简写字符串 (e.g. 'AD')
- 'GGGG': 时代的完整字符串 (e.g. 'Anno Domini')

格式字符串还可以是下列预定义的本地化的格式之一：

- 'medium': en\_US 地区的形式, 等同于 'MMM d, y h:mm:ss a' (e.g. Sep 3, 2010 12:05:08 PM)
- 'short': en\_US 地区的形式, 等同于 'M/d/yy h:mm a' (e.g. 9/3/10 12:05 PM)
- 'fullDate': en\_US 地区的形式, 等同于 'EEEE, MMMM d, y' (e.g. Friday, September 3, 2010)
- 'longDate': en\_US 地区的形式, 等同于 'MMMM d, y' (e.g. September 3, 2010)
- 'mediumDate': en\_US 地区的形式, 等同于 'MMM d, y' (e.g. Sep 3, 2010)
- 'shortDate': en\_US 地区的形式, 等同于 'M/d/yy' (e.g. 9/3/10)
- 'mediumTime': en\_US 地区的形式, 等同于 'h:mm:ss a' (e.g. 12:05:08 PM)
- 'shortTime': en\_US 地区的形式, 等同于 'h:mm a' (e.g. 12:05 PM)

格式字符串可以包含文字。但是需要使用 `'` 包裹进行转义 (e.g. "h 'in the morning")。

如果想使用单引号,则需要转义 - 举个例子, 在一行里有两个单引号 (e.g. "h 'o'clock")。

## 用法

HTML 模板绑定中

```
{{ date_expression | date : format : timezone }}
```

JavaScript 中

```
$filter('date')(date, format, timezone)
```

## 参数



参数	形式	详细
date	number Date string	可以格式化的日期形式包括 Date 对象，毫秒数（字符串或者数字类型）以及各种 ISO 8601 标准的日期时间字符串（如：yyyy-MM-ddTHH:mm:ss.sssZ， 和它的简写形式，像 yyyy-MM-ddTHH:mmZ, yyyy-MM-dd or yyyyMMddTHHmmssZ）。如果没有指定时区，则时间会根据当地的时区显示
format(可选)	string	格式化的规则 (请看描述)。如果没有给定值，将会使用 mediumDate 。
timezone(可选)	number	用于格式化的时区设定。

参数识别 UTC/GMT 和美国大陆时区的缩写，但是一般情况会使用时区偏移，例如，'+0430'（格林威治子午线以东 4 小时 30 分）如果不指定，将使用浏览器使用的时区。|

返回

string 格式化后的字符串或者不能被识别为日期的输入。

例子

index.html

```
<span ng-non-bindable>{{1288323623006 | date:'medium'}}</span>:
  <span>{{1288323623006 | date:'medium'}}</span><br>
<span ng-non-bindable>{{1288323623006 | date:'yyyy-MM-dd HH:mm:ss Z'}}</span>:
  <span>{{1288323623006 | date:'yyyy-MM-dd HH:mm:ss Z'}}</span><br>
<span ng-non-bindable>{{1288323623006 | date:'MM/dd/yyyy @ h:mma'}}</span>:
  <span>{'1288323623006' | date:'MM/dd/yyyy @ h:mma'}}</span><br>
<span ng-non-bindable>{{1288323623006 | date:"MM/dd/yyyy 'at' h:mma"}}</span>:
  <span>{'1288323623006' | date:"MM/dd/yyyy 'at' h:mma"}}</span></pre>
```

```
it('should format date', function() {
  expect(element(by.binding("1288323623006 | date:'medium'")).getText())
    .toMatch(/Oct 2\d, 2010 \d{1,2}:\d{2}:\d{2} (AM|PM)/);
  expect(element(by.binding("1288323623006 | date:'yyyy-MM-dd HH:mm:ss'")).getText())
    .toMatch(/2010\-10\-2\d \d{2}:\d{2}:\d{2} (\-|\+)?\d{4}/);
  expect(element(by.binding("'1288323623006' | date:'MM/dd/yyyy @ h:mm:ss AM/PM'")).getText())
    .toMatch(/10\/2\d\/2010 @ \d{1,2}:\d{2}(AM|PM)/);
  expect(element(by.binding("'1288323623006' | date:'\''MM/dd/yyyy' at h:mm:ss AM/PM'")).getText())
    .toMatch(/10\/2\d\/2010 at \d{1,2}:\d{2}(AM|PM)/);
});
```

## filter

- Angular@1.4.7
- `ng` 模块中的过滤器

从输入数组中过滤出符合条件的子集，并返回这个子集的复制品。

## 用法

HTML 模板绑定中

```
{{ filter_expression | filter : expression : comparator }}
```

JavaScript 中

```
$filter('filter')(array, expression, comparator)
```

## 参数

参数	形式	详细
array	array	原数组
expression	string object function()	<p>用于选择的谓词（译：既条件）可以是以下之一：</p> <ul style="list-style-type: none"> <li>- string :字符串用来匹配数组的内容。所有包含它的字符串或者字符串属性包含它的对象会被返回。同样适用于嵌套的对象属性中。这个谓词可以通过在字符串前加 <code>!</code> 前缀进行取反。</li> <li>- Object :一个可以用于过滤包含在数组内对象的特定属性模式对象，例如：<code>{name:"M", phone:"1"}</code> 这个谓词将会返回一个数组，这个数组中的每个项目的 <code>name</code> 属性都会包含 <code>"M"</code>，而 <code>phone</code> 属性都会包含 <code>"1"</code>。 <code>\$</code> 这个特殊的属性名可以用于匹配对象的任何属性，或者它嵌套的对象属性。就相当于像用简单的子字符串匹配字符串那样。这个谓词可以通过在字符串前加 <code>!</code> 前缀进行取反。例如 谓词 <code>{name: "!M"}</code> 将会使返回值为一个由 <code>name</code> 属性不包含 <code>'M'</code> 的对象组成的数组</li> </ul> <p>注意：谓词一个属性仅仅会匹配相同层级的属性，但特殊属性 <code>\$</code> 会匹配相同层级和更深层级的属性。例如：一个数组 <code>{name: {first: 'John', last: 'Doe'}}</code> 不会与 <code>{name: 'John'}</code>，但是会与 <code>{\$: 'John'}</code> 匹配。</p> <ul style="list-style-type: none"> <li>- function(value, index, array) :一个谓词函数可以写出任意的过滤器。这个函数会被数组的每个元素调用，函数得到的参数依次为（元素，元素索引，整个数组）</li> </ul> <p>最后返回的结果是一个用谓词返回为真值的元素组成的数组。</p>
comparator	function(actual, expected) true undefined	<p>比较器，它用于确定预期值 (来自过滤的表达式) 和实际值 (来自数组中的对象) 是否匹配。可以为以下之一：</p> <ul style="list-style-type: none"> <li>- function(actual, expected) : 函数会将对象的值和谓词的值进行比较，如果两个值是相等的则返回 <code>true</code>。</li> <li>- true : function(actual, expected) { return angular.equals(actual, expected); } 的简写形式. 对预期值和实际值进行严格的比较。</li> <li>- false 或 undefined : 不区分大小写的方式来寻找子字符串的函数的简写形式。</li> </ul> <p>原始值被转化成字符串，对象不能与原始值相比较，除非他们自定义的 <code>toString</code> 方法（如：<code>Date</code> 对象）。</p>

## 返回

string 格式化后的数值

## 例子

index.html

```
<div ng-init="friends = [{name:'John', phone:'555-1276'},
                        {name:'Mary', phone:'800-BIG-MARY'},
                        {name:'Mike', phone:'555-4321'},
                        {name:'Adam', phone:'555-5678'},
                        {name:'Julie', phone:'555-8765'},
                        {name:'Juliette', phone:'555-5678'}]"></div>

<label>Search: <input ng-model="searchText"></label>
<table id="searchTextResults">
  <tr><th>Name</th><th>Phone</th></tr>
  <tr ng-repeat="friend in friends | filter:searchText">
    <td>{{friend.name}}</td>
    <td>{{friend.phone}}</td>
  </tr>
</table>
<hr>
<label>Any: <input ng-model="search.$"></label> <br>
<label>Name only <input ng-model="search.name"></label><br>
<label>Phone only <input ng-model="search.phone"></label><br>
<label>Equality <input type="checkbox" ng-model="strict"></label><br>
<table id="searchObjResults">
  <tr><th>Name</th><th>Phone</th></tr>
  <tr ng-repeat="friendObj in friends | filter:search:strict">
    <td>{{friendObj.name}}</td>
    <td>{{friendObj.phone}}</td>
  </tr>
</table>
```

protractor.js

```
var expectFriendNames = function(expectedNames, key) {
  element.all(by.repeater(key + ' in friends').column(key + '.name')).
    arr.forEach(function(wd, i) {
      expect(wd.getText()).toMatch(expectedNames[i]);
    });
});

it('should search across all fields when filtering with a string',
  var searchText = element(by.model('searchText'));
  searchText.clear();
  searchText.sendKeys('m');
  expectFriendNames(['Mary', 'Mike', 'Adam'], 'friend');

  searchText.clear();
  searchText.sendKeys('76');
  expectFriendNames(['John', 'Julie'], 'friend');
});

it('should search in specific fields when filtering with a predicate',
  var searchAny = element(by.model('search.$'));
  searchAny.clear();
  searchAny.sendKeys('i');
  expectFriendNames(['Mary', 'Mike', 'Julie', 'Juliette'], 'friendObj');
});

it('should use a equal comparison when comparator is true', function() {
  var searchName = element(by.model('search.name'));
  var strict = element(by.model('strict'));
  searchName.clear();
  searchName.sendKeys('Julie');
  strict.click();
  expectFriendNames(['Julie'], 'friendObj');
});
```

## json

- Angular@1.4.7
- `ng` 模块中的过滤器

允许你将一个 JavaScript 对象转化成一个 JSON 形式的字符串。

该过滤器多用于调试。当使用双花括号时，绑定将自动转化成 JSON。

## 用法

HTML 模板绑定中

```
{{ json_expression | json : spacing }}
```

JavaScript 中

```
$filter('json')(object, spacing)
```

## 参数

参数	形式	详细
object	*	需要过滤的任何 JavaScript 对象（包括数组和原始类型）
spacing(可选)	number	空格缩进的数量, 默认为两个 2.

## 返回

string JSON 的字符串

## 例子

index.html

```
<pre id="default-spacing">{{ {'name':'value'} | json }}</pre>
<pre id="custom-spacing">{{ {'name':'value'} | json:4 }}</pre>
```

protractor.js

```
it('should jsonify filtered objects', function() {
  expect(element(by.id('default-spacing')).getText()).toMatch(/\{\r
  expect(element(by.id('custom-spacing')).getText()).toMatch(/\{\n
});
```



## lowercase

- Angular@1.4.7
- `ng` 模块中的过滤器

将字符串转化为小写

## 用法

HTML 模板绑定中

```
{{ lowercase_expression | lowercase }}
```

JavaScript 中

```
$filter('lowercase')(input)
```

## 参数

参数	形式	详细
input	string	过滤器的输入

## 返回

string 格式化后的数值

## uppercase

- Angular@1.4.7
- `ng` 模块中的过滤器

将字符串转化为大写

## 用法

HTML 模板绑定中

```
{{ uppercase_expression | uppercase }}
```

JavaScript 中

```
$filter('uppercase')(input)
```

## 参数

参数	形式	详细
input	string	过滤器的输入

## 返回

string 格式化后的数值

## number

- Angular@1.4.7
- `ng` 模块中的过滤器

将数字格式化为字符串。

如果输入的是 `null` 或者 `undefined`，则会原封不动的返回。如果输入是无穷（`Infinity/-Infinity`），会返回无穷的符号 `'∞'`。如果输入不是数字则返回一个空字符串。

## 用法

HTML 模板绑定中

```
{{ number_expression | number : fractionSize }}
```

JavaScript 中

```
$filter('number')(number, fractionSize)
```

## 参数

参数	形式	详细
number	number string	需要格式化的数字
fractionSize(可选)	number string	小数位的取舍量。默认为当前语言地区的最大量。

## 返回

`string` 数字会依据小数数位取舍，并且每三个数字会使用 `,` 分隔开。

## 例子

[index.html](#)

```

<script>
  angular.module('numberFilterExample', [])
    .controller('ExampleController', ['$scope', function($scope) {
      $scope.val = 1234.56789;
    }]);
</script>
<div ng-controller="ExampleController">
  <label>Enter number: <input ng-model='val'></label><br>
  Default formatting: <span id='number-default'>{{val | number}}</span><br>
  No fractions: <span>{{val | number:0}}</span><br>
  Negative number: <span>{{-val | number:4}}</span>
</div>

```

prowtractor.js

```

it('should format numbers', function() {
  expect(element(by.id('number-default')).getText()).toBe('1,234.56789');
  expect(element(by.binding('val | number:0')).getText()).toBe('1,235');
  expect(element(by.binding('-val | number:4')).getText()).toBe('-1,234.56789');
});

it('should update', function() {
  element(by.model('val')).clear();
  element(by.model('val')).sendKeys('3374.333');
  expect(element(by.id('number-default')).getText()).toBe('3,374.333');
  expect(element(by.binding('val | number:0')).getText()).toBe('3,374');
  expect(element(by.binding('-val | number:4')).getText()).toBe('-3,374.333');
});

```

## limitTo

- Angular@1.4.7
- `ng` 模块中的过滤器

创建一个新的数组或者字符串来包含给定数量的元素。这些元素要么是从原数组（或字符串或数字）的头部开始截取要么是从尾部截取，这由你指定的值和符号（正或负）来决定。如果输入一个数字，则它将被转化成字符串。

## 用法

HTML 模板绑定中

```
{{ limitTo_expression | limitTo : limit : begin }}
```

JavaScript 中

```
$filter('limitTo')(input, limit, begin)
```

## 参数

参数	形式	详细
input	number array string	需要限制的源数字，数组，字符串
limit	string number	返回数组或者字符串的长度。如果截取的数字为正，会从原数组/字符串的开头进行复制。如果为负，会从结尾进行复制。截取在到达原数组/字符串长度后会被修剪。

如果 `limit` 为 `undefined`，返回值将是原数组/字符串| `begin(可选)`| `string` `number` | 截取起始位置的索引值。为负值时，说明从输入的尾部开始计算偏移。默认值为0。|

## 返回

`string` `array` 指定限制长度的一个新的子数组或者子字符串，如果输入的元素比截取数还要少，新的子数组或子字符串也会更短。

## 例子

index.html

```

<script>
  angular.module('limitToExample', [])
    .controller('ExampleController', ['$scope', function($scope) {
      $scope.numbers = [1,2,3,4,5,6,7,8,9];
      $scope.letters = "abcdefghi";
      $scope.longNumber = 2345432342;
      $scope.numLimit = 3;
      $scope.letterLimit = 3;
      $scope.longNumberLimit = 3;
    }]);
</script>
<div ng-controller="ExampleController">
  <label>
    Limit {{numbers}} to:
    <input type="number" step="1" ng-model="numLimit">
  </label>
  <p>Output numbers: {{ numbers | limitTo:numLimit }}</p>
  <label>
    Limit {{letters}} to:
    <input type="number" step="1" ng-model="letterLimit">
  </label>
  <p>Output letters: {{ letters | limitTo:letterLimit }}</p>
  <label>
    Limit {{longNumber}} to:
    <input type="number" step="1" ng-model="longNumberLimit">
  </label>
  <p>Output long number: {{ longNumber | limitTo:longNumberLimit }}</p>
</div>

```

protractor.js

```

var numLimitInput = element(by.model('numLimit'));
var letterLimitInput = element(by.model('letterLimit'));
var longNumberLimitInput = element(by.model('longNumberLimit'));

```

```
var limitedNumbers = element(by.binding('numbers | limitTo:numLimit'));
var limitedLetters = element(by.binding('letters | limitTo:letterLimit'));
var limitedLongNumber = element(by.binding('longNumber | limitTo:longNumberLimit'));

it('should limit the number array to first three items', function() {
    expect(numLimitInput.getAttribute('value')).toBe('3');
    expect(letterLimitInput.getAttribute('value')).toBe('3');
    expect(longNumberLimitInput.getAttribute('value')).toBe('3');
    expect(limitedNumbers.getText()).toEqual('Output numbers: [1,2,3]');
    expect(limitedLetters.getText()).toEqual('Output letters: abc');
    expect(limitedLongNumber.getText()).toEqual('Output long number: 123456789');
});

// There is a bug in safari and protractor that doesn't like the multiple it() calls
// it('should update the output when -3 is entered', function() {
//     numLimitInput.clear();
//     numLimitInput.sendKeys('-3');
//     letterLimitInput.clear();
//     letterLimitInput.sendKeys('-3');
//     longNumberLimitInput.clear();
//     longNumberLimitInput.sendKeys('-3');
//     expect(limitedNumbers.getText()).toEqual('Output numbers: [7,8,9]');
//     expect(limitedLetters.getText()).toEqual('Output letters: ghi');
//     expect(limitedLongNumber.getText()).toEqual('Output long number: 456789');
// });

it('should not exceed the maximum size of input array', function() {
    numLimitInput.clear();
    numLimitInput.sendKeys('100');
    letterLimitInput.clear();
    letterLimitInput.sendKeys('100');
    longNumberLimitInput.clear();
    longNumberLimitInput.sendKeys('100');
    expect(limitedNumbers.getText()).toEqual('Output numbers: [1,2,3]');
    expect(limitedLetters.getText()).toEqual('Output letters: abcdefg');
    expect(limitedLongNumber.getText()).toEqual('Output long number: 123456789');
});
```

## orderBy

- Angular@1.4.7
- `ng` 模块中的过滤器

使用表达式的谓词为数组排序。对字符串使用字母顺序进行排序，数字则按数字的大小排序。

注意：如果你发现数字不是按照你的预期进行排序的，请确定他们被保存的实际形式确实为数字而不是字符串。

## 用法

HTML 模板绑定中

```
{{ orderBy_expression | orderBy : expression : reverse }}
```

JavaScript 中

```
$filter('orderBy')(array, expression, reverse)
```

## 参数



参数	形式	详细
array	array	需要排序的数组
expression	string function(*) Array. <function(*)>= Array. <string>=	<p>用于比较来确定元素顺序的谓词。</p> <p>可以为以下之一：</p> <ul style="list-style-type: none"><li>- function : Getter 函数.这个函数返回的结果会使用 &lt; , === , &gt; 操作符进行排序。</li><li>- string : 一个 Angular 表达式。表达式解析用来比较元素。（举个例子，表达式为 name , 则以 name 属性进行排序，表达式为 name.substr(0, 3) , 则以 name 属性的值的前3个字符进行排序）。一个常量表达式解析的结果会被当做属性名而被用于比较（例如："special name"就会使用他们的 special name 属性进行排序）。一个表达式可以通过设置 + 或 - 前缀来控制排序（升序或降序）方式（例如：+name 或 -name ）。如果没有提供属性（例如：'+'; 译：仅仅规定了升序）则会用数组元素本身来进行排序。</li><li>- Array : 一个函数、字符串谓词组成的数组，第一个谓词用于排序，当两项相等时，则用下一个谓词排序。如果不传入谓词或者为空则默认为 '+'</li></ul>
reverse(可选)	boolean	翻转数组的顺序

返回

string 经过排序数组的复制品

例子

下面的例子展示了一个简单的 ngRepeat , 数据通过 age 来降序排列（谓词设置成了 -age ）。 翻转（reverse）没有设置，默认不生效（false）

index.html

```
<script>
angular.module('orderByExample', [])
.controller('ExampleController', ['$scope', function($scope) {
    $scope.friends =
        [{name: 'John', phone: '555-1212', age: 10},
         {name: 'Mary', phone: '555-9876', age: 19},
         {name: 'Mike', phone: '555-4321', age: 21},
         {name: 'Adam', phone: '555-5678', age: 35},
         {name: 'Julie', phone: '555-8765', age: 29}];
    }]);
</script>
<div ng-controller="ExampleController">
    <table class="friend">
        <tr>
            <th>Name</th>
            <th>Phone Number</th>
            <th>Age</th>
        </tr>
        <tr ng-repeat="friend in friends | orderBy:'-age'">
            <td>{{friend.name}}</td>
            <td>{{friend.phone}}</td>
            <td>{{friend.age}}</td>
        </tr>
    </table>
</div>
```

谓词和翻转参数可以通过作用域（scope）属性动态地控制。下面的例子将展示。

index.html

```
<script>
angular.module('orderByExample', [])
.controller('ExampleController', ['$scope', function($scope) {
    $scope.friends =
        [{name: 'John', phone: '555-1212', age: 10},
         {name: 'Mary', phone: '555-9876', age: 19},
         {name: 'Mike', phone: '555-4321', age: 21},
         {name: 'Adam', phone: '555-5678', age: 35},
```

```

        {name:'Julie', phone:'555-8765', age:29}]];
$scope.predicate = 'age';
$scope.reverse = true;
$scope.order = function(predicate) {
    $scope.reverse = ($scope.predicate === predicate) ? !$scope.reverse : $scope.reverse;
    $scope.predicate = predicate;
};
}]);
</script>
<style type="text/css">
    .sortorder:after {
        content: '\25b2';
    }
    .sortorder.reverse:after {
        content: '\25bc';
    }
</style>
<div ng-controller="ExampleController">
    <pre>Sorting predicate = {{predicate}}; reverse = {{reverse}}</pre>
    <hr/>
    [ <a href="" ng-click="predicate=''>unsorted</a> ]
    <table class="friend">
        <tr>
            <th>
                <a href="" ng-click="order('name')">Name</a>
                <span class="sortorder" ng-show="predicate === 'name'" ng-c
            </th>
            <th>
                <a href="" ng-click="order('phone')">Phone Number</a>
                <span class="sortorder" ng-show="predicate === 'phone'" ng-c
            </th>
            <th>
                <a href="" ng-click="order('age')">Age</a>
                <span class="sortorder" ng-show="predicate === 'age'" ng-c
            </th>
        </tr>
        <tr ng-repeat="friend in friends | orderBy:predicate:reverse">
            <td>{{friend.name}}</td>
            <td>{{friend.phone}}</td>
            <td>{{friend.age}}</td>
        </tr>
    </table>
</div>

```

```
    </tr>
  </table>
</div>
```

当然也可以手动地调用 `orderBy` 过滤器。注入 `$filter` 服务，使用如 `$filter('orderBy')` 这样来检索排序的过滤器，然后调用 `$filter('orderBy')` 返回的结果，并传入参数。

index.html

```
<div ng-controller="ExampleController">
  <table class="friend">
    <tr>
      <th><a href="" ng-click="reverse=false;order('name', false)">
        (<a href="" ng-click="order('-name',false)">^</a></th>
      <th><a href="" ng-click="reverse=!reverse;order('phone', reverse)">
      <th><a href="" ng-click="reverse=!reverse;order('age',reverse)">
    </tr>
    <tr ng-repeat="friend in friends">
      <td>{{friend.name}}</td>
      <td>{{friend.phone}}</td>
      <td>{{friend.age}}</td>
    </tr>
  </table>
</div>
```

script.js

```
angular.module('orderByExample', [])
.controller('ExampleController', ['$scope', '$filter', function($scope, $filter) {
    var orderBy = $filter('orderBy');
    $scope.friends = [
        { name: 'John',    phone: '555-1212',    age: 10 },
        { name: 'Mary',    phone: '555-9876',    age: 19 },
        { name: 'Mike',    phone: '555-4321',    age: 21 },
        { name: 'Adam',    phone: '555-5678',    age: 35 },
        { name: 'Julie',   phone: '555-8765',    age: 29 }
    ];
    $scope.order = function(predicate, reverse) {
        $scope.friends = orderBy($scope.friends, predicate, reverse);
    };
    $scope.order('-age', false);
}]);
```

# linky

- Angular@1.4.7
  - `ngSanitize` 模块中的过滤器
- 在文本输入中找到链接（links）并将他们转化为真实的 HTML 链接。支持 http/https/ftp/mailto 和普通的电子邮件地址的链接。

需要依赖 `ngSanitize` 模块。

## 用法

HTML 模板绑定中

```
<span ng-bind-html="linky_expression | linky"></span>
```

JavaScript 中

```
$filter('linky')(text, target)
```

## 参数

参数	形式	详细			
text	string	输入的内容			
target	string	Window (_blank	_self	_parent	_top) 或者用于打开连接的命名了的 frame

## 返回

string html化链接的文本。

## 例子

index.html

```

<script>
  angular.module('linkyExample', ['ngSanitize'])
    .controller('ExampleController', ['$scope', function($scope) {
      $scope.snippet =
        'Pretty text with some links:\n'+
        'http://angularjs.org/, \n'+
        'mailto:us@somewhere.org, \n'+
        'another@somewhere.org, \n'+
        'and one more: ftp://127.0.0.1/.';
      $scope.snippetWithTarget = 'http://angularjs.org/';
    }]);
</script>
<div ng-controller="ExampleController">
Snippet: <textarea ng-model="snippet" cols="60" rows="3"></textarea>
<table>
  <tr>
    <td>Filter</td>
    <td>Source</td>
    <td>Rendered</td>
  </tr>
  <tr id="linky-filter">
    <td>linky filter</td>
    <td>
      <pre>&lt;div ng-bind-html="snippet | linky"&gt;<br>&lt;/div&gt;
    </td>
    <td>
      <div ng-bind-html="snippet | linky"></div>
    </td>
  </tr>
  <tr id="linky-target">
    <td>linky target</td>
    <td>
      <pre>&lt;div ng-bind-html="snippetWithTarget | linky: '_blank'"
    </td>
    <td>
      <div ng-bind-html="snippetWithTarget | linky: '_blank'"></div>
    </td>
  </tr>
  <tr id="escaped-html">

```

```
<td>no filter</td>
<td><pre>&lt;div ng-bind="snippet"&gt;<br>&lt;/div&gt;</pre></td>
<td><div ng-bind="snippet"></div></td>
</tr>
</table>
```

protractor.js



```
it('should linkify the snippet with urls', function() {
  expect(element(by.id('linky-filter')).element(by.binding('snippet'))
    .toBe('Pretty text with some links: http://angularjs.org/, use@angularjs.org, and one more: ftp://127.0.0.1/.'));
  expect(element.all(by.css('#linky-filter a')).count()).toEqual(4);
});

it('should not linkify snippet without the linky filter', function() {
  expect(element(by.id('escaped-html')).element(by.binding('snippet'))
    .toBe('Pretty text with some links: http://angularjs.org/, make use of@angularjs.org, and one more: ftp://127.0.0.1/.'));
  expect(element.all(by.css('#escaped-html a')).count()).toEqual(0);
});

it('should update', function() {
  element(by.model('snippet')).clear();
  element(by.model('snippet')).sendKeys('new http://link.');
  expect(element(by.id('linky-filter')).element(by.binding('snippet'))
    .toBe('new http://link.'));
  expect(element.all(by.css('#linky-filter a')).count()).toEqual(1);
  expect(element(by.id('escaped-html')).element(by.binding('snippet'))
    .toBe('new http://link.'));
});

it('should work with the target property', function() {
  expect(element(by.id('linky-target')).
    element(by.binding("snippetWithTarget | linky:'_blank'")).getText()
    .toBe('http://angularjs.org/');
  expect(element(by.css('#linky-target a')).getAttribute('target')).
    toEqual('_blank');
});
```

使用 Angular 过程中遇到的问题和解决办法会放到这个部分





关于表单

[AngularJS form](#)表单验证

