* 原文作者： [Nicolas Bevacqua](http://www.smashingmagazine.com/author/nicolasbevacqua/)
* 原文：[AngularJS’ Internals In Depth](http://www.smashingmagazine.com/2015/01/angularjs-internals-in-depth/)

在AngularJS的代码库中呈现出了大量有趣的设计，最有趣的两个例子是scope的工作方式和directives(指令)的表现。

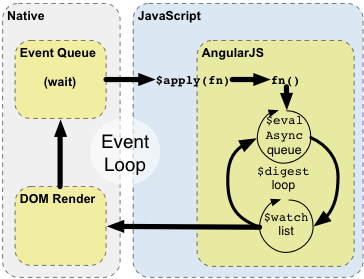
有的人第一次接触AngularJS时就被告知directives是和DOM交互，或供你随意操作DOM，就像jQuery. 这立马变得非常复杂，试想，scopes, directives 和controllers相互作用.

复杂的设置之后，你开始学习它先进的思想：the digest cycle，独立的scope、内嵌以及指令中不同的链接函数。这些已经非常负责，这篇文章不会涉及指令，但会在后续的文章中写到。

这篇文字将带你跳过AngularJS 的scopes和AngularJS应用生命周期的各种坑，同时提供有趣信息的深度阅读。

（门槛是高的，但scope是很难解释的。如果在此惨遭失败，至少我会抛出几个重要的点。）

如果这个图表看起来非常的费解，那么这篇文章很适合你。

[](http://p9.qhimg.com/d/inn/94482819/01-mindbender.png)

([Image credit](https://github.com/angular/angular.js/wiki/Understanding-Scopes)) ([View large version](https://www.w3ctech.com/topic/1611))

（免责声明：这篇文字是基于 [AngularJS version 1.3.0](https://github.com/angular/angular.js/tree/v1.3.0)）

AngularJS 用scopes分离指令和DOM的通信。scopes也存在于controller层。scopes 是普通的JavaScript对象，AngularJS没有过多的操作。只是添加了“一串”带有一个或两个$符号前缀的内部属性。其中以$$开头的常常不是必须的，经常用它们作代码气味，可以避免更深入的理解digest循环。

**哪种scope是我们要讨论的？**[**Link**](https://www.w3ctech.com/topic/1611#what-kind-of-scopes-are-we-talking-about)

在AngualrJS 俚语中，scope”不是你平时思考Javascript代码或编程的scope。scope通常适用于维持一段代码块的上下文、变量等。通常情况下，scope用来包含一段代码中的上下文、变量等。

（在大多数语言中，变量维持在一个用花括号（{}）或代码块定义的虚拟袋中，这就是大家所说的块级作用域.相比之下，Javascript 使用的是词法作用域，大致意思是使用函数或者全局对象定义虚拟袋而不是代码块。虚拟袋可以嵌套更多小的袋子。....）

下面用一个简单的例子来解释函数

function eat (thing) {

console.log('Eating a ' + thing);

}

function nuts (peanut) {

var hazelnut = 'hazelnut';

function seeds () {

var almond = 'almond';

eat(hazelnut); // I can reach into the nuts bag!

}

// Almonds are inaccessible here.

// Almonds are not nuts.

}

我不在赘述这个问题，因为这不是人们谈论AngularJS时的scope，如果你喜欢了解更多关于JavaScript语言上下文的scope请参考“[Where Does this Keyword Come From?](http://ponyfoo.com/articles/where-does-this-keyword-come-from)。

**AngularJS 中scope 的继承**[**Link**](https://www.w3ctech.com/topic/1611#scope-inheritance-in-angularjs)

在Angular的条款中，scope也是结合上下文的。AngularJS中，一个scope跟一个元素关联（以及所有它的子元素），而一个元素不是必须直接跟一个scope关联。元素通过以下三中方式被分配一个scope：

1、scope通过controller或者directive创建在一个element上（指令不总是引入新的scope）

<nav ng-controller='menuCtrl'>

2、如果一个scope不存在于元素上，那么它将继承它的父级scope

<nav ng-controller='menuCtrl'>

<a ng-click='navigate()'>Click Me!</a> <!-- also <nav>'s scope -->

</nav>

3、如果一个元素不是某个ng-app的一部分，那么它不属于任何scope。

<head>

<h1>Pony Deli App</h1>

</head>

<main ng-app='PonyDeli'>

<nav ng-controller='menuCtrl'>

<a ng-click='navigate()'>Click Me!</a>

</nav>

</main>

为了弄清楚元素的scope，试着由内到外递归我刚刚列出的3条元素规则？？？ 它创建了一个新的scope？那是它的scope，它有父级吗？检查它的父级是某个ng-app的一部分吗？。遗憾的是---没有scope

你可以用充满魔力的开发者工具轻松的找出元素的scope。

**拨开 AngularJS scope 内部属性**[**Link**](https://www.w3ctech.com/topic/1611#pulling-up-angularjs-internal-scope-properties)

在介绍digest如何工作以及内部如何表现前，我会剖析scope的一些属性来介绍某些概念。我也会让你知道我如何获取这些属性。首先，打开 Chrome 并导航到我正在使用的一个 angular应用程序。然后，我将审查一个元素并打开开发者工具。

（你知道[$0能让你获得最后一个选中的元素](https://developers.google.com/chrome-developer-tools/docs/commandline-api#0_-_4)吗？$1让你能访问前一个被选中的元素等。我想你会经常用到$0，尤其是使用 AngularJS工作时。）

对于每一个DOM元素， 用 angular.element 包装跟使用[jQuery](http://jquery.com/) 或 jqLite, jQuery 的 [迷你版](https://github.com/angular/angular.js/blob/caed2dfe4feeac5d19ecea2dbb1456b7fde21e6d/src/jqLite.js)是一样的。 一旦被包装， 你通过 scope() 函数得到的结果 —— 你猜对了！—— 就是跟元素关联的 AngularJS scope。结合$0,我发现自己经常使用下面的命令。

angular.element($0).scope()

（当然，如果你使用jQuery，那么 $($0).scope() 也会达到一样的效果。不管 jQuery 是否可用， angular.element 一直可用。）

然后我能审查该 scope， 确定该 scope 是我预期的， 确定属性的值是否跟我预期的匹配。让我们来看看一个典型的scope中可访问的特殊属性， 它们以一个或多个 $ 符号开头。

for(o in $($0).scope())o[0]=='$'&&console.log(o)

这就足够了，我将遍历所有的属性，

**深入 AngularJS Scope 的内部学习**[**Link**](https://www.w3ctech.com/topic/1611#examining-a-scopes-internals-in-angularjs)

下面，我列出了由该命令产生的属性，按功能分组。让我们从基本的开始，它仅仅提供scope导航。

* [$id](https://github.com/angular/angular.js/blob/v1.3.0/src/ng/rootScope.js#L127) scope 的唯一标识
* [$root](https://github.com/angular/angular.js/blob/v1.3.0/src/ng/rootScope.js#L131) 根scope
* [$parent](https://github.com/angular/angular.js/blob/v1.3.0/src/ng/rootScope.js#L217) 父级scope， 如果 scope == scope.$root 则为 null
* [$$childHead](https://github.com/angular/angular.js/blob/v1.3.0/src/ng/rootScope.js#L223) 第一个子 scope， 如果没有则为 null
* [$$childTail](https://github.com/angular/angular.js/blob/v1.3.0/src/ng/rootScope.js#L221) 最后一个子scope， 如果没有则为 null
* [$$prevSibling](https://github.com/angular/angular.js/blob/v1.3.0/src/ng/rootScope.js#L218) 前一个相邻节点 scope， 如果没有则为 null
* [$$nextSibling](https://github.com/angular/angular.js/blob/v1.3.0/src/ng/rootScope.js#L220) 下一个相邻节点 scope， 如果没有则为 null

这没什么惊喜， 浏览 scope 没什么意思。有时候访问 $parent 看起来是得当的， 但是有更好的、低耦合的方式来处理父级通信而不是紧紧的与人为的scope绑定在一起。 比如说 使用事件监听器，下面我们将讲到。

**AngularJS Scope 的事件模型**[**Link**](https://www.w3ctech.com/topic/1611#event-model-in-angularjs-scope)

下面介绍的属性允许我们发布事件和订阅事件。这个模式叫[发布/订阅](http://en.wikipedia.org/wiki/Publish%E2%80%93subscribe_pattern)。

* [$$listeners](https://github.com/angular/angular.js/blob/v1.3.0/src/ng/rootScope.js#L1092) 在scope上注册事件监听器。
* [$on(evt, fn)](https://github.com/angular/angular.js/blob/v1.3.0/src/ng/rootScope.js#L1089-L1109) 注册一个名为evt，监听器为fn的事件。
* [$emit(evt, args)](https://github.com/angular/angular.js/blob/v1.3.0/src/ng/rootScope.js#L1134-L1182) 发送事件 evt， 在scope 链上冒泡，在当前scope 以及所有的 $parents 上触发，包括 $rootScope。
* [$broadcast(evt, args)](https://github.com/angular/angular.js/blob/v1.3.0/src/ng/rootScope.js#L1206-L1258) 发送事件 evt， 在当前scope 以及它 所有的 children 上触发。

当事件触发的时候， 事件监听器传递 event 对象和其它参数到 $emit 或者 $broadcast 函数。有很多方式为 scope 上的事件传递值。

指令可以通过事件来告知一些重要的事情发生了。查看下面的指令示例，一个按钮被点击后，告知你喜欢吃什么类型的食物。

angular.module('PonyDeli').directive('food', function () {

return {

scope: { // I'll come back to directive scopes later

type: '=type'

},

template: 'I want to eat some {{type}}!',

link: function (scope, element, attrs) {

scope.eat = function () {

letThemHaveIt();

scope.$emit('food.order, scope.type, element);

};

function letThemHaveIt () {

// Do some fancy UI things

}

}

};

});

我为事件添加了命名空间，你也应该这么做。它可以防止命名冲突，你可以清楚的知道事件的来源或者你正在订阅的是什么事件。如果你对分析数据感兴趣或者想追踪 food 的元素点击，可以用 [Mixpanel](https://mixpanel.com/)。这确实很有意义，并且没有理由污染你的指令或者控制器。你可以用一个指令来处理 food 点击的分析和追踪，这是一个很好的自足的方式。

angular.module('PonyDeli').directive('foodTracker', function (mixpanelService) {

return {

link: function (scope, element, attrs) {

scope.$on('food.order, function (e, type) {

mixpanelService.track('food-eater', type);

});

}

};

});

这个服务的实现是不贴切的，因为它仅仅是对Mixpanel的客户端API的封装。 它的HTML看起来应该像下面的样子， 我用了一个 controller 来维护所有的实物类型。[ng-app](https://github.com/angular/angular.js/blob/v1.3.0/src/Angular.js#L1293)指令很好的帮助 AngularJS 自动启动我的程序。为了完善这个例子，我添加了一个 ng-repeat指令来渲染我所有的 food 而不是重复自身。这只是通过 foodTypes 循环，在 foodCtrl 的scope 中是可以访问的。

<ul ng-app='PonyDeli' ng-controller='foodCtrl' food-tracker>

<li food type='type' ng-repeat='type in foodTypes'></li>

</ul>

angular.module('PonyDeli').controller('foodCtrl', function ($scope) {

$scope.foodTypes = ['onion', 'cucumber', 'hazelnut'];

});

完整的项目例子[托管在 CodePen](http://codepen.io/bevacqua/pen/qmBGd)

表面上看是一个很好的例子，但你应该思考是否需要一个事件给其它人订阅。也许一个服务就行。这种情况。你可以用另一种方式。 你会问自己你需要事件因为你不知道谁会订阅food.order，这意味着使用事件更加面向未来。你也会说 food 追踪指令没有理由存在，因为它没有跟 DOM ，甚至是 scope 交互， 只是监听了一个事件，你可以使用 service 替换它。

在刚刚的情况下，这两种说法都是对的。当更多的组件需要 food.order-aware 时，才会感觉事件的做法更清晰。事实上，事件很有用，当你真的需要桥接两个 scope 的间隙时，其它的因素就不那么重要了。

正如我们所见，在本文即将到来的第二部分更加仔细的审查指令，在 scope 间通信时事件甚至不是必要的。一个子 scope 可以它的父scope读取，通过父scope绑定它，它也可以更新这些值。

(很少情况下通过事件来帮助子 scope 与 父 scope 更好的通信。)

同级之前往往很难相互通信， 经常通过它们共同的父级scope 来通信。通常从 $rootScope 开始广播，然后在你希望的相邻节点监听，如下：

<body ng-app='PonyDeli'>

<div ng-controller='foodCtrl'>

<ul food-tracker>

<li food type='type' ng-repeat='type in foodTypes'></li>

</ul>

<button ng-click='deliver()'>I want to eat that!</button>

</div>

<div ng-controller='deliveryCtrl'>

<span ng-show='received'>

A monkey has been dispatched. You shall eat soon.

</span>

</div>

</body>

angular.module('PonyDeli').controller('foodCtrl', function ($rootScope) {

$scope.foodTypes = ['onion', 'cucumber', 'hazelnut'];

$scope.deliver = function (req) {

$rootScope.$broadcast('delivery.request', req);

};

});

angular.module('PonyDeli').controller('deliveryCtrl', function ($scope) {

$scope.$on('delivery.request', function (e, req) {

$scope.received = true; // deal with the request

});

});

这个例子也[放在 CodePen](http://codepen.io/bevacqua/pen/CzGla)

渐渐的，你将对事件和服务越来越熟悉。我想告诉你，当你期望视图模块改变来响应事件，你应该使用 event， 当你不期望视图模块改变，你应该使用 service。有时候响应是这两种的混合：一个动作触发了一个事件，事件调用了一个 service， 或者 service 从 $rootScope广播了一个事件。这视情况而定，并且你应该这样分析，而不是随意使用一个方法。

如果你有两个组件通过 $rootScope 通信，你可能更喜欢使用 $rootScope.$emit （而不是 $broadcast）和 $rootScope.$on。这种方式下， 事件只会在 $rootScope.$$listeners 之间传播， 那些你知道没有该事件的监听器的后代的 $rootScope上，不会循环浪费时间。

angular.module('PonyDeli').factory("notificationService", function ($rootScope) {

function notify (data) {

$rootScope.$emit("notificationService.update", data);

}

function listen (fn) {

$rootScope.$on("notificationService.update", function (e, data) {

fn(data);

});

}

// Anything that might have a reason

// to emit events at later points in time

function load () {

setInterval(notify.bind(null, 'Something happened!'), 1000);

}

return {

subscribe: listen,

load: load

};

});

[CodePen](http://codepen.io/bevacqua/pen/HsBCa)

事件跟服务已经差不多了，让我们移步到一些其它的属性。

**Digest 变化**[**Link**](https://www.w3ctech.com/topic/1611#digesting-changesets)

了解这个恐怖的过程是认识 AngularJS的关键。

AngularJS 基于它的数据绑定的特性，通过循环脏检测来追踪变化并且在变化时触发事件。这比听起来简单。事实上， 它就是这样简单。让我们快速的浏览一下 $digest 循环的核心组件。 首先，有一个 scope.$digest 方法，通过递归检测scope 和它的后代们的变化。

1. [$digest()](https://github.com/angular/angular.js/blob/v1.3.0/src/ng/rootScope.js#L710)  
   执行 $digest 循环脏检测
2. [$$phase](https://github.com/angular/angular.js/blob/v1.3.0/src/ng/rootScope.js#L1271) digest 循环的当前阶段， [null, '$apply', '$digest'] 中的一个。

你需要小心触发 digest，因为当你已经在一个 digest 阶段而尝试这么做， 会因为一些无法解释的现象导致 AngularJS 出错。

让我们看看 [文档](http://docs.angularjs.org/api/ng.$rootScope.Scope#methods_$digest)里关于 $digest怎么说

Processes all of the [watchers](http://docs.angularjs.org/api/ng.$rootScope.Scope#methods_$watch) of the current scope and its children. Because a [watcher](http://docs.angularjs.org/api/ng.$rootScope.Scope#methods_$watch)’s listener can change the model, the [$digest()](http://docs.angularjs.org/api/ng.$rootScope.Scope#methods_$digest) keeps calling the [watchers](http://docs.angularjs.org/api/ng.$rootScope.Scope#methods_$watch) until no more listeners are firing. This means that it is possible to get into an infinite loop. This function will throw 'Maximum iteration limit exceeded.' if the number of iterations exceeds 10.

Usually, you don’t call [$digest()](http://docs.angularjs.org/api/ng.$rootScope.Scope#methods_$digest) directly in [controllers](http://docs.angularjs.org/api/ng.directive:ngController) or in [directives](http://docs.angularjs.org/api/ng.$compileProvider#methods_directive). Instead, you should call [$apply()](http://docs.angularjs.org/api/ng.$rootScope.Scope#methods_$apply) (typically from within a [directives](http://docs.angularjs.org/api/ng.$compileProvider#methods_directive)), which will force a [$digest()](http://docs.angularjs.org/api/ng.$rootScope.Scope#methods_$digest).

它处理当前scope 及其 后代们的所有的 [watchers](http://docs.angularjs.org/api/ng.$rootScope.Scope#methods_$watch)。因为 watcher 的监听器可以改变 model，[$digest()](http://docs.angularjs.org/api/ng.$rootScope.Scope#methods_$digest) 持续调用 watcher 直到没有监听器被触发。这意味着可能进入死循环。如果迭代超过10次，这个函数会抛出异常 'Maximum iteration limit exceeded'。

通常，我们在控制器或指令中不直接调用 $digest。相反，你应该调用[$apply()](http://docs.angularjs.org/api/ng.$rootScope.Scope#methods_$apply) (通常在一个指令里) 用来强制执行一个 $digest()。

所以，一个 $digest 处理所有的 watcher，处理这些 watcher时，这些watcher触发，直到没有别的触发 watcher。为了我们理解这个循环，仍然有两个问题需要解答。

* "watcher" 是什么鬼？！
* 什么触发 $digest？！

这两个问题的答案因复杂度差异很大，但是我会尽量为你解释清楚。我将介绍 watcher，让你有个自己的看法。

如果你读过这一步，你或许已经知道什么事 watcher了。你或许使用过 [scope.$watch](https://github.com/angular/angular.js/blob/v1.3.0/src/ng/rootScope.js#L356)，甚至用过[scope.$watchCollection](https://github.com/angular/angular.js/blob/v1.3.0/src/ng/rootScope.js#L530)。$$watchers 属性有用 scope 上所有的 watcher。

* [$watch(watchExp, listener, objectEquality)](https://github.com/angular/angular.js/blob/v1.3.0/src/ng/rootScope.js#L356) 为scope添加一个 watch 监听器
* [$watchCollection](https://github.com/angular/angular.js/blob/v1.3.0/src/ng/rootScope.js#L530) watch 数组元素或对象属性
* [$$watchers](https://github.com/angular/angular.js/blob/v1.3.0/src/ng/rootScope.js#L383) 保持所有的 watch 与 scope 的关联

watcher 是AngularJS的数据绑定功能的很重要的一面， 但是为了触发这些 watcher，AngularJS 需要我们的帮助。否则，它不能有效的更新数据绑定的变量为正确值。思考下面的例子：

<body ng-app='PonyDeli'>

<ul ng-controller='foodCtrl'>

<li ng-bind='prop'></li>

<li ng-bind='dependency'></li>

</ul>

</body>

angular.module('PonyDeli').controller('foodCtrl', function ($scope) {

$scope.prop = 'initial value';

$scope.dependency = 'nothing yet!';

$scope.$watch('prop', function (value) {

$scope.dependency = 'prop is "' + value + '"! such amaze';

});

setTimeout(function () {

$scope.prop = 'something else';

}, 1000);

});

我们有初始值 'initial value'，我们期望第二行 HTML 变为 'prop is "something else"！ 惊喜发生在1秒后，对吗？ 甚至更有趣的是，你至少期待第一行变为'something else'！为什么不是呢？这是个 watcher 吗？

实际上，你在 HTML 标签上的一系列操作都是以创建 watcher 结束。这种情况下，每个 [ng-bind 指令创建一个 watcher](https://github.com/angular/angular.js/blob/v1.3.0/src/ng/directive/ngBind.js#L54-L68),当 prop、 dependency 变化时它会更新 HTML <li>。

这样，你现在可以把你的代码想象成3个 watcher，每个 ng-bind 指令的watcher， 还有一个在控制器中的。AngularJS如何知道 timeout 执行后的属性变化？你应该想起 AngularJS 更新属性时，可以在 timeout 回调时添加一个手动的 digest。

setTimeout(function () {

$scope.prop = 'something else';

$scope.$digest();

}, 1000);

我放置了一个[没有 $digest 的例子](http://codepen.io/bevacqua/pen/lLbtI)在 CodePen，以及 timeout[有 $digest](http://codepen.io/bevacqua/pen/vwDoz)的。你可以用[$timeout service](https://github.com/angular/angular.js/blob/v1.3.0/src/ng/timeout.js#L35) 替换 setTimeout，它提供了一些错误处理，并且会执行 $apply()。

$timeout(function () {

$scope.prop = 'something else';

}, 1000);

* [$apply(expr)](https://github.com/angular/angular.js/blob/v1.3.0/src/ng/rootScope.js#L1018-L1033) 解析和计算一个表达式，然后在 $rootScope 上执行 $digest 循环

为了在每个 scope 上执行 digest， $apply 提供了很好的错误处理功能。如果你尝试调优性能， 使用 $digest 或许能够保证， 但是在我了解 AngularJS 内部工作原理而感觉良好之前， 我会远离它。实际上很少时候需要手动调用 $digest()；$apply 总是更好的选择。

现在我们回到第二个问题上来。

* 什么触发了 $digest？！

Digest的内部触发在AngularJS代码库中具有重要地位。它们的要么直接被触发，要么是调用 $apply() 触发，就像我们在 $timeout 服务里看到的。不管是AngularJS中的核心还是边缘的指令都会触发digest。 digest 触发你的 watcher， watcher更新你的 UI。这是基本的思路。

你可以从 AngularJS wiki里面找到关于好的实践资源，链接在文章底部。

我已经解释了 watcher 和 $digest 循环如何相互交互。下面，我将列出一些与 $digest 循环相关的属性，它们可以在 scope 上找到。 这些可以帮助你在 AngularJS 编译时解析文本表达式， 或者在 digest 循环的不同阶段执行一小段代码。

* [$eval(expression, locals)](https://github.com/angular/angular.js/blob/v1.3.0/src/ng/rootScope.js#L922-L924) 立刻解析和计算出一个 scope 表达式。
* [$evalAsync(expression)](https://github.com/angular/angular.js/blob/v1.3.0/src/ng/rootScope.js#L955-L967) 在稍后的时间里解析和计算一个表达式。
* [$$asyncQueue](https://github.com/angular/angular.js/blob/v1.3.0/src/ng/rootScope.js#L736-L744) 同步处理队列，会消耗每个 digest
* [$$postDigest(fn)](https://github.com/angular/angular.js/blob/v1.3.0/src/ng/rootScope.js#L969-L971) 在下一个 digest 周期后执行 fn
* [$$postDigestQueue](https://github.com/angular/angular.js/blob/v1.3.0/src/ng/rootScope.js#L970) 用 $$postDigest(fn) 注册方法

**scope已死！scope万岁！(The Scope Is Dead! Long Live the Scope!)**[**Link**](https://www.w3ctech.com/topic/1611#the-scope-is-dead-long-live-the-scope)

在最后的部分，来聊聊 scope 的性能。尽管有时候你可能要用 $new 声明自己的scope， 但它们使用内部手段处理 scope 的生命周期。

* [$$isolateBindings](https://github.com/angular/angular.js/blob/v1.3.0/src/ng/compile.js#L756) 独立 scope 绑定（例如：{ options: '@megaOptions' }）
* [$new(isolate)](https://github.com/angular/angular.js/blob/v1.3.0/src/ng/rootScope.js#L193) 创建一个子 scope 或者一个独立的 scope， 它不继承自它们的父级。
* [$destroy](https://github.com/angular/angular.js/blob/v1.3.0/src/ng/rootScope.js#L857) 从 scope 链里移除该 scope； scope 和后代们不会收到事件， watcher 也不再被触发。
* [$$destroyed](https://github.com/angular/angular.js/blob/v1.3.0/src/ng/rootScope.js#L863) scope 是否被销毁。

独立的scope？这是什么鬼？这个系列的第二部分将讨论指令，它包括独立的scope，嵌入，link 函数，编译器，指令控制器等。Wait for it!

**扩展阅读**

Here are some additional resources you can read to extend your comprehension of AngularJS.

* “[The Angular Way](http://ponyfoo.com/articles/the-angular-way),” Nicolas Bevacqua
* “[Anti-Patterns](https://github.com/angular/angular.js/wiki/Anti-Patterns),” AngularJS, GitHub
* “[Best Practices](https://github.com/angular/angular.js/wiki/Best-Practices),” AngularJS, GitHub
* [TodoMVC AngularJS example](http://todomvc.com/examples/angularjs/#/)
* [Egghead.io: Bite-Sized Video Training With AngularJS](https://egghead.io/), John Lindquist
* [ng-newsletter](http://www.ng-newsletter.com/)
* “[Using scope.$watch and scope.$apply](http://stackoverflow.com/a/15113029/389745),” StackOverflow