窗体顶端

窗体底端

**angularjs directive 实例 详解**

张映 发表于 2014-03-13

分类目录： [js/jquery](http://blog.51yip.com/category/jsjquery)

标签：[angular](http://blog.51yip.com/tag/angular), [angularjs](http://blog.51yip.com/tag/angularjs), [directive](http://blog.51yip.com/tag/directive), [js](http://blog.51yip.com/tag/js), [例子](http://blog.51yip.com/tag/%e4%be%8b%e5%ad%90)

前面提到了angularjs的factory,service,provider，这个可以理解成php的model，这种model是 不带html的，今天所说的directive,也可以理解成php的model，也可以理解成插件，只不过这种model是带html的，例如：php 的分页函数。

**一，angularjs directive的常用格式，以及参数说明**

**1，直接return**

[查看复制打印?](http://blog.51yip.com/jsjquery/1607.html)

1. var phonecatDirectives = angular.module('phonecatDirectives', []);
3. phonecatDirectives.directive('directiveName', function($inject) {
5. return {
6. template: '<div></div>',
8. replace: false,
10. transclude: true,
12. restrict: 'E',
14. scope: { ... },
16. controller: function($scope, $element){ .... },
18. compile: function(tElement, tAttrs, transclude) {
19. return {
20. pre: function preLink(scope, iElement, iAttrs, controller) { ... },
21. post: function postLink(scope, iElement, iAttrs, controller) { ... }
22. }
23. },
25. link: function(scope, iElement, iAttrs) { ... }
26. };
27. });

**2，定义一个js的域**

[查看复制打印?](http://blog.51yip.com/jsjquery/1607.html)

1. var phonecatDirectives = angular.module('phonecatDirectives', []);
3. phonecatDirectives.directive('directiveName', function($inject) {
5. var mydi = {
6. template: '<div></div>',
7. ..................... ,
8. link: function(scope, iElement, iAttrs) { ... }
9. };
11. return mydi;
12. });

**3，angularjs directive 对像参数说明**

name - 当前scope的名称，注册时可以使用默认值（不填）。

priority（优先级）- 当有多个directive定义在同一个DOM元素时，有时需要明确它们的执行顺序。这属性用于在directive的compile function调用之前进行排序。如果优先级相同，则执行顺序是不确定的（经初步试验，优先级高的先执行，同级时按照类似栈的“后绑定先执行”。另外， 测试时有点不小心，在定义directive的时候，两次定义了一个相同名称的directive，但执行结果发现，两个compile或者link function都会执行）。

terminal（最后一组）- 如果设置为”true”，则表示当前的priority将会成为最后一组执行的directive。任何directive与当前的优先级相同的话，他们 依然会执行，但顺序是不确定的（虽然顺序不确定，但基本上与priority的顺序一致。当前优先级执行完毕后，更低优先级的将不会再执行）。

scope - 如果设置为：

true - 将为这个directive创建一个新的scope。如果在同一个元素中有多个directive需要新的scope的话，它还是只会创建一个 scope。新的作用域规则不适用于根模版（root of the template），因此根模版往往会获得一个新的scope。

{}(object hash) - 将创建一个新的、独立(isolate)的scope。”isolate” scope与一般的scope的区别在于它不是通过原型继承于父scope的。这对于创建可复用的组件是很有帮助的，可以有效防止读取或者修改父级 scope的数据。这个独立的scope会创建一个拥有一组来源于父scope的本地scope属性(local scope properties)的object hash。这些local properties对于为模版创建值的别名很有帮助（useful for aliasing values for templates -\_-!）。本地的定义是对其来源的一组本地scope property的hash映射(Locals definition is a hash of local scope property to its source #&)$&@#)($&@#\_)：

@或@attr - 建立一个local scope property到DOM属性的绑定。因为属性值总是String类型，所以这个值总是返回一个字符串。如果没有通过@attr指定属性名称，那么本地名 称将与DOM属性的名称一直。例如<widget my-attr=”hello {{name}}”>，widget的scope定义为：{localName:’@myAttr’}。那么，widget scope property的localName会映射出”hello {{name}}"转换后的真实值。name属性值改变后，widget scope的localName属性也会相应地改变（仅仅单向，与下面的”=”不同）。name属性是在父scope读取的（不是组件scope）

=或=expression（这里也许是attr） - 在本地scope属性与parent scope属性之间设置双向的绑定。如果没有指定attr名称，那么本地名称将与属性名称一致。例如<widget my-attr=”parentModel”>，widget定义的scope为：{localModel:’=myAttr’}，那么 widget scope property “localName”将会映射父scope的“parentModel”。如果parentModel发生任何改变，localModel也会发生改 变，反之亦然。（双向绑定）

&或&attr - 提供一个在父scope上下文中执行一个表达式的途径。如果没有指定attr的名称，那么local name将与属性名称一致。例如<widget my-attr=”count = count + value”>，widget的scope定义为：{localFn:’increment()’}，那么isolate scope property “localFn”会指向一个包裹着increment()表达式的function。一般来说，我们希望通过一个表达式，将数据从isolate scope传到parent scope中。这可以通过传送一个本地变量键值的映射到表达式的wrapper函数中来完成。例如，如果表达式是increment(amount)，那 么我们可以通过localFn({amount:22})的方式调用localFn以指定amount的值（上面的例子真的没看懂，&跑哪去 了？）。

controller - controller 构造函数。controller会在pre-linking步骤之前进行初始化，并允许其他directive通过指定名称的require进行共享（看 下面的require属性）。这将允许directive之间相互沟通，增强相互之间的行为。controller默认注入了以下本地对象：

$scope - 与当前元素结合的scope

$element - 当前的元素

$attrs - 当前元素的属性对象

$transclude - 一个预先绑定到当前转置scope的转置linking function :function(cloneLinkingFn)。(A transclude linking function pre-bound to the correct transclusion scope)

require - 请求另外的controller，传入当前directive的linking function中。require需要传入一个directive controller的名称。如果找不到这个名称对应的controller，那么将会抛出一个error。名称可以加入以下前缀：

? - 不要抛出异常。这使这个依赖变为一个可选项。

^ - 允许查找父元素的controller

restrict - EACM的子集的字符串，它限制directive为指定的声明方式。如果省略的话，directive将仅仅允许通过属性声明：

E - 元素名称： <my-directive></my-directive>

A - 属性名： <div my-directive=”exp”></div>

C - class名： <div class=”my-directive:exp;”></div>

M - 注释 ： <!-- directive: my-directive exp -->

template - 如果replace 为true，则将模版内容替换当前的HTML元素，并将原来元素的属性、class一并迁移；如果为false，则将模版元素当作当前元素的子元素处理。 想了解更多的话，请查看“Creating Widgets”章节（在哪啊。。。Creating Components就有。。。）

templateUrl - 与template基本一致，但模版通过指定的url进行加载。因为模版加载是异步的，所以compilation、linking都会暂停，等待加载完毕后再执行。

replace - 如果设置为true，那么模版将会替换当前元素，而不是作为子元素添加到当前元素中。（注：为true时，模版必须有一个根节点）

transclude - 编译元素的内容，使它能够被directive所用。需要(在模版中)配合ngTransclude使用(引用)。transclusion的优点是 linking function能够得到一个预先与当前scope绑定的transclusion function。一般地，建立一个widget，创建isolate scope，transclusion不是子级的，而是isolate scope的兄弟。这将使得widget拥有私有的状态，transclusion会被绑定到父级（pre-isolate）scope中。（上面那段话 没看懂。但实际实验中，如果通过<any my-directive>{{name}}</any my-directive>调用myDirective，而transclude设置为true或者字符串且template中包 含<sometag ng-transclude>的时候，将会将{{name}}的编译结果插入到sometag的内容中。如果any的内容没有被标签包裹，那么结果 sometag中将会多了一个span。如果本来有其他东西包裹的话，将维持原状。但如果transclude设置为’element’的话，any的整 体内容会出现在sometag中，且被p包裹）

true - 转换这个directive的内容。（这个感觉上，是直接将内容编译后搬入指定地方）

‘element’ - 转换整个元素，包括其他优先级较低的directive。（像将整体内容编译后，当作一个整体（外面再包裹p），插入到指定地方）

compile - 这里是compile function，将在下面实例详细讲解

link - 这里是link function ，将在下面实例详细讲解。这个属性仅仅是在compile属性没有定义的情况下使用。

**三，angularjs directive 实例讲解**

下面的实例都围绕着，上面的参数来展开的

**1，directive声明方式实例**

[查看复制打印?](http://blog.51yip.com/jsjquery/1607.html)

1. //directive文件directives.js中定义一个myTest
2. 'use strict';
4. var phonecatDirectives = angular.module('phonecatDirectives', []);
6. phonecatDirectives.directive('myTest', function() {
7. return {
8. restrict: 'ACEM',
9. require: '^ngModel',
10. scope: {
11. ngModel: '='
12. },
13. template: '<div><h4>Weather for {{ngModel}}</h4></div>'
14. }
15. });
17. //controller文件controller.js中定义directive1
18. 'use strict';
20. var dtControllers = angular.module('dtControllers', []);
22. dtControllers.controller('directive1',['$scope',
23. function($scope) {
24. $scope.name = 'this is tank test';
25. }
26. ]);
28. //在app文件app.js中整合controller,directive
30. 'use strict';
32. var phonecatApp = angular.module('phonecatApp', [
33. 'phonecatDirectives',
34. 'dtControllers'
35. ]);
37. //html文件
38. <script src="../lib/angular/angular.js"></script>
39. <script src="../js/app.js"></script>
40. <script src="../js/controller.js"></script>
41. <script src="../js/directives.js"></script>
42. <body ng-app="phonecatApp">
43. <div ng-controller="directive1">
44. <input type="text" ng-model="city" placeholder="Enter a city" />
46. <my-test ng-model="city" ></my-test>
47. <span my-test="exp" ng-model="city"></span>
48. <!-- directive: my-test exp -->
49. <span ng-model="city"></span>
50. </div>
51. </body>

**上例结果：<!-- directive: my-test exp -->这个不起作用，不知道为什么，尝试了好多方法都不起作用。**

**2，template和templateUrl区别与联系**

**templateUrl其实根template功能是一样的，只不过templateUrl加载一个html文件，上例中，我们也能发现问题，template后面根的是html的标签，如果标签很多呢，那就比较不爽了。可以将上例中的，template改一下。**

[查看复制打印?](http://blog.51yip.com/jsjquery/1607.html)

1. phonecatDirectives.directive('myTest', function() {
2. return {
3. restrict: 'ACEM',
4. require: '^ngModel',
5. scope: {
6. ngModel: '='
7. },
8. templateUrl:'../partials/tem1.html'   //tem1.html中的内容就是上例中template的内容。
9. }
10. });

**3，scope重定义**

[查看复制打印?](http://blog.51yip.com/jsjquery/1607.html)

1. //directives.js中定义myAttr
2. phonecatDirectives.directive('myAttr', function() {
3. return {
4. restrict: 'E',
5. scope: {
6. customerInfo: '=info'
7. },
8. template: 'Name: {{customerInfo.name}} Address: {{customerInfo.address}}<br>' +
9. 'Name: {{vojta.name}} Address: {{vojta.address}}'
10. };
11. });
13. //controller.js中定义attrtest
14. dtControllers.controller('attrtest',['$scope',
15. function($scope) {
16. $scope.naomi = { name: 'Naomi', address: '1600 Amphitheatre' };
17. $scope.vojta = { name: 'Vojta', address: '3456 Somewhere Else' };
18. }
19. ]);
21. //html中
22. <body ng-app="phonecatApp">
23. <div ng-controller="attrtest">
24. <my-attr info="naomi"></my-attr>
25. </div>
26. </body>

运行结果：

[查看复制打印?](http://blog.51yip.com/jsjquery/1607.html)

1. Name: Naomi Address: 1600 Amphitheatre      //有值,因为customerInfo定义过的
2. Name: Address:       //没值 ，因为scope重定义后，vojta是没有定义的

可能把上面的directive简单改一下，

[查看复制打印?](http://blog.51yip.com/jsjquery/1607.html)

1. phonecatDirectives.directive('myAttr', function() {
2. return {
3. restrict: 'E',
4. template: 'Name: {{customerInfo.name}} Address: {{customerInfo.address}}<br>' +
5. 'Name: {{vojta.name}} Address: {{vojta.address}}'
6. };
7. });

运行结果如下：根上面正好相反

1. Name: Address:
2. Name: Vojta Address: 3456 Somewhere Else

**4，transclude的使用**

**transclude的用法，有点像jquery里面的$().html()功能**

[查看复制打印?](http://blog.51yip.com/jsjquery/1607.html)

1. //directives.js增加myEvent
2. phonecatDirectives.directive('myEvent', function() {
3. return {
4. restrict: 'E',
5. transclude: true,
6. scope: {
7. 'close': '&onClick'     //根html中的on-click="hideDialog()"有关联关系
8. },
9. templateUrl: '../partials/event\_part.html'
10. };
11. });
13. //controller.js增加eventtest
14. dtControllers.controller('eventtest',['$scope','$timeout',
15. function($scope, $timeout) {
16. $scope.name = 'Tobias';
17. $scope.hideDialog = function () {
18. $scope.dialogIsHidden = true;
19. $timeout(function () {
20. $scope.dialogIsHidden = false;
21. }, 2000);
22. };
23. }
24. ]);
26. //event.html
27. <body ng-app="phonecatApp">
28. <div ng-controller="eventtest">
29. <my-event ng-hide="dialogIsHidden" on-click="hideDialog()">
30. Check out the contents, {{name}}!
31. </my-event>
32. </div>
33. </body>
35. //event\_part.html
36. <div>
37. <a href ng-click="close()">×</a>
38. <div ng-transclude></div>
39. </div>

**5，controller，link，compile有什么不同**

[查看复制打印?](http://blog.51yip.com/jsjquery/1607.html)

1. //directives.js增加exampleDirective
2. phonecatDirectives.directive('exampleDirective', function() {
3. return {
4. restrict: 'E',
5. template: '<p>Hello {{number}}!</p>',
6. controller: function($scope, $element){
7. $scope.number = $scope.number + "22222 ";
8. },
9. link: function(scope, el, attr) {
10. scope.number = scope.number + "33333 ";
11. },
12. compile: function(element, attributes) {
13. return {
14. pre: function preLink(scope, element, attributes) {
15. scope.number = scope.number + "44444 ";
16. },
17. post: function postLink(scope, element, attributes) {
18. scope.number = scope.number + "55555 ";
19. }
20. };
21. }
22. }
23. });
25. //controller.js添加
26. dtControllers.controller('directive2',['$scope',
27. function($scope) {
28. $scope.number = '1111 ';
29. }
30. ]);
32. //html
33. <body ng-app="phonecatApp">
34. <div ng-controller="directive2">
35. <example-directive></example-directive>
36. </div>
37. </body>

运行结果：

1. Hello 1111 22222 44444 55555 !

**由结果可以看出来，controller先运行，compile后运行，link不运行。**

将上例中的compile注释掉

[查看复制打印?](http://blog.51yip.com/jsjquery/1607.html)

1. //        compile: function(element, attributes) {
2. //            return {
3. //                pre: function preLink(scope, element, attributes) {
4. //                    scope.number = scope.number + "44444 ";
5. //                },
6. //                post: function postLink(scope, element, attributes) {
7. //                    scope.number = scope.number + "55555 ";
8. //                }
9. //            };
10. //        }

运行结果：

1. Hello 1111 22222 33333 !

**由结果可以看出来，controller先运行，link后运行，link和compile不兼容。**

随笔- 118  文章- 0  评论- 386

# [指令<AngularJs>](http://www.cnblogs.com/rohelm/p/4051437.html)

对于指令，可以把它简单的理解成在特定DOM元素上运行的函数，指令可以扩展这个元素的功能。

首先来看个完整的参数示例再来详细的介绍各个参数的作用及用法：

[复制代码](javascript:void(0);)

angular.module('myApp', [])

.directive('myDirective', function() {

return {

restrict: String,

priority: Number,

terminal: Boolean,

template: String or Template Function:

function(tElement, tAttrs) {...},

templateUrl: String,

replace: Boolean or String,

scope: Boolean or Object,

transclude: Boolean,

controller: String or

function(scope, element, attrs, transclude, otherInjectables) { ... },

controllerAs: String,

require: String,

link: function(scope, iElement, iAttrs) { ... },

compile: // 返回一个对象或连接函数，如下所示：

function(tElement, tAttrs, transclude) {

return {

pre: function(scope, iElement, iAttrs, controller) { ... },

post: function(scope, iElement, iAttrs, controller) { ... }

}

return function postLink(...) { ... }

}

};

});

[复制代码](javascript:void(0);)

### restrict[string]

restrict是一个可选的参数。用于指定该指令在DOM中以何种形式被声明。默认值是A，即以属性的形式来进行声明。  
可选值如下：  
E（元素）

<my-directive></my-directive>

A（属性，默认值）

<div my-directive="expression"></div>

C（类名）

<div class="my-directive:expression;"></div>

M（注释）

<--directive:my-directive expression-->

一般考虑到浏览器的兼容性，强烈建议使用默认的属性就可以即即以属性的形式来进行声明。最后一种方式建议再不要求逼格指数的时候千万不要用。

Code:

[复制代码](javascript:void(0);)

angular.module('app',[])

.directive('myDirective', function () {

return {

restrict: 'E',

template: '<a href="http://www.baidu.com">百度</a>'

};

})  
**HtmlCode:**  
 <my-directive></my-directive>

[复制代码](javascript:void(0);)

**效果：**http://images.cnitblog.com/blog/360406/201410/261004339809191.png

### priority[int]

大多数指令会忽略这个参数，使用默认值0，但也有些场景设置高优先级是非常重要甚至是必须的。例如，ngRepeat将这个参数设置为1000，这样就可以保证在同一元素上，它总是在其他指令之前被调用。

### ****terminal[bool]****

这个参数用来**停止运行当前元素上比本指令优先级低的指令**。但同当前指令优先级相同的指令还是会被执行。  
例如：ngIf的优先级略高于ngView（它们操控的实际就是**terminal参数**），如果ngIf的表达式值为true，ngView就可以被正常执行，但如果ngIf表达式的值为false，由于ngView的优先级较低就不会被执行。

### template[string or function]

template参数是可选的，必须被设置为以下两种形式之一：

* 一段HTML文本；
* 一个可以接受两个参数的函数，参数为tElement和tAttrs，并返回一个代表模板的字符串。tElement和tAttrs中的t代表template，是相对于instance的。

首先演示下第二种用法：

[复制代码](javascript:void(0);)

angular.module('app',[])

.directive('myDirective', function () {

return {

restrict: 'EAC',

template: function (elem, attr) {

return "<a href='" + attr.value + "'>" + attr.text + "</a>";

}

};

})

[复制代码](javascript:void(0);)

HtmlCode:（效果同上，不做演示了）

<my-directive value="http://www.baidu.com" text="百度"></my-directive>

<div my-directive

value="http://www.baidu.com"

text="百度"></div>

### templateUrl[string or function]

templateUrl是可选的参数，可以是以下类型：

* 一个代表外部HTML文件路径的字符串；
* 一个可以接受两个参数的函数，参数为tElement和tAttrs，并返回一个外部HTML文件路径的字符串。

无论哪种方式，模板的URL都将通过ng内置的安全层，特别是$getTrustedResourceUrl，这样可以保护模板不会被不信任的源加载。 默认情况下，调用指令时会在后台通过Ajax来请求HTML模板文件。加载大量的模板将严重拖慢一个客户端应用的速度。为了避免延迟，可以在部署应用之前对HTML模板进行缓存。

Code:

[复制代码](javascript:void(0);)

angular.module('app',[])

.directive('myDirective', function () {

return {

restrict: 'AEC',

templateUrl: function (elem, attr) {

return attr.value + ".html"; //当然这里我们可以直接指定路径，同时在模板中可以包含表达式

}

};

})

[复制代码](javascript:void(0);)

### replace[bool]

replace是一个可选参数，如果设置了这个参数，值必须为true，因为默认值为false。默认值意味着模板会被当作子元素插入到调用此指令的元素内部,  
例如上面的示例默认值情况下，生成的html代码如下：

<my-directive value="http://www.baidu.com" text="百度"><a href="http://www.baidu.com">百度</a></my-directive>

如果设置replace=true

<a href="http://www.baidu.com" value="http://www.baidu.com" text="百度">百度</a>

据我观察，这种效果只有设置restrict="E"的情况下，才会表现出实际效果。

介绍完基本的指令参数后，就要涉及到更重要的作用域参数了...

### scope参数[bool or object]

 scope参数是可选的，可以被设置为true或一个对象。默认值是false。

如果**一个元素上有多个指令使用了隔离作用域，其中只有一个可以生效。只有指令模板中的根元素可以获得一个新的作用域**。因此，对于这些对象来说scope默认被设置为true。内置指令ng-controller的作用，就是从父级作用域继承并创建一个新的子作用域。它会创建一个新的从父作用域继承而来的子作用域。这里的继承就不在赘述，和面向对象中的继承基本是一直的。

 首先我们来分析一段代码：

[复制代码](javascript:void(0);)

<div ng-app="app" ng-init="name= '祖父'">

<div ng-init="name='父亲'">

第一代：{{ name }}

<div ng-init="name= '儿子'" ng-controller="SomeController">

第二代： {{ name }}

<div ng-init="name='孙子'">

第三代： {{ name }}

</div>

</div>

</div>

</div>

[复制代码](javascript:void(0);)

我们发现第一代，我们初始化name为父亲,但是第二代和第三代其实是一个作用域，那么他们的name其实是一个对象，因此出现的效果如下：

第一代：父亲

第二代： 孙子

第三代： 孙子

我们在修改一下代码，把第三代隔离开来再看看效果：

[复制代码](javascript:void(0);)

<div ng-app="app"ng-init="name= '祖父'">

<div ng-init="name='父亲'">

第一代：{{ name }}

<div ng-init="name= '儿子'" ng-controller="SomeController">

第二代： {{ name }}

<div ng-init="name='孙子'" ng-controller="SecondController">

第三代： {{ name }}

</div>

</div>

</div>

</div>

[复制代码](javascript:void(0);)

JsCode:

[复制代码](javascript:void(0);)

angular.module('app', [])

.controller('SomeController',function($scope) {

})

.controller('SecondController', function ($scope) {

})

[复制代码](javascript:void(0);)

效果如下：

第一代：父亲

第二代： 儿子

第三代： 孙子

在修改下代码来看看继承：

[复制代码](javascript:void(0);)

<div ng-app="app"ng-init="name= '祖父的吻'">

<div>

第一代：{{ name }}

<div ng-controller="SomeController">

第二代： {{ name }}

<div ng-controller="SecondController">

第三代： {{ name }}

</div>

</div>

</div>

</div>

[复制代码](javascript:void(0);)

效果如下：

第一代：祖父的吻

第二代： 祖父的吻

第三代： 祖父的吻

如果要创建一个能够从外部原型继承作用域的指令，将scope属性设置为true，简单来说就是可继承的隔离，即不能反向影响父作用域。

 再来看个例子：

[复制代码](javascript:void(0);)

angular.module('myApp', [])

.controller('MainController', function ($scope) {

})

.directive('myDirective', function () {

return {

restrict: 'A',

scope:false,//切换为{}，true测试

priority: 100,

template: '<div>内部:{{ myProperty }}<input ng-model="myProperty"/></div>'

};

});

[复制代码](javascript:void(0);)

Html代码：

<div ng-controller='MainController' ng-init="myProperty='Hello World!'">

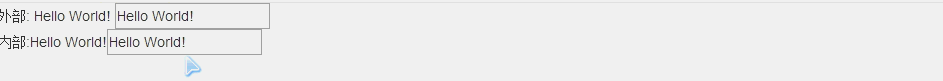
外部: {{ myProperty}}

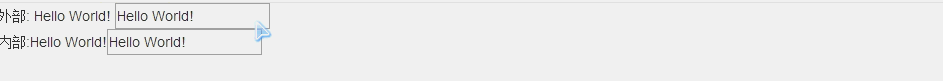
<input ng-model="myProperty" />

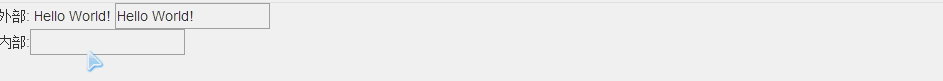
<div my-directive></div>

</div>

 当我们改变scope的值我们会发现

**false:**继承但不隔离

**true:**继承并隔离

**{}：**隔离且不继承 

### transclude

transclude是一个可选的参数。默认值是false。嵌入通常用来创建可复用的组件，典型的例子是模态对话框或导航栏。我们可以将整个模板，包括其中的指令通过嵌入全部传入一个指令中。**指令的内部可以访问外部指令的作用域，并且模板也可以访问外部的作用域对象**。为了将作用域传递进去，scope参数的值必须通过{}或true设置成隔离作用域。如果没有设置scope参数，那么指令内部的作用域将被设置为传入模板的作用域。

只有当你希望创建一个可以包含任意内容的指令时，才使用transclude: true。

 我们来看两个例子-导航栏：

[复制代码](javascript:void(0);)

<div side-box title="TagCloud">

<div class="tagcloud">

<a href="">Graphics</a>

<a href="">ng</a>

<a href="">D3</a>

<a href="">Front-end</a>

<a href="">Startup</a>

</div>

</div>

[复制代码](javascript:void(0);)

JsCode:

[复制代码](javascript:void(0);)

angular.module('myApp', [])

.directive('sideBox', function() {

return {

restrict: 'EA',

scope: {

title: '@'

},

transclude: true,

template: '<div class="sidebox"><div class="content"><h2 class="header">' +

'{{ title }}</h2><span class="content" **ng-transclude**></span></div></div>'

};

});

[复制代码](javascript:void(0);)

这段代码告诉ng编译器，将它从DOM元素中获取的内容放到它发现ng-transclude指令的地方。

再来你看个官网的例子：

[复制代码](javascript:void(0);)

angular.module('docsIsoFnBindExample', [])

.controller('Controller', ['$scope', '$timeout', function($scope, $timeout) {

$scope.name = 'Tobias';

$scope.hideDialog = function () {

$scope.dialogIsHidden = true;

$timeout(function () {

$scope.dialogIsHidden = false;

}, 2000);

};

}])

.directive('myDialog', function() {

return {

restrict: 'E',

transclude: true,

scope: {

'close': '&onClose'

},

templateUrl: 'my-dialog-close.html'

};

});

[复制代码](javascript:void(0);)

my-dialog-close.html

<div class="alert">

<a href class="close" ng-click="close()">&times;</a>

<div ng-transclude></div>

</div>

index.html

<div ng-controller="Controller">

<my-dialog ng-hide="dialogIsHidden" on-close="hideDialog()">

Check out the contents, {{name}}!

</my-dialog>

</div>

如果指令使用了transclude参数，那么在控制器无法正常监听数据模型的变化了。建议在链接函数里使用$watch服务。

### controller[string or function]

 controller参数可以是一个字符串或一个函数。当设置为字符串时，会以字符串的值为名字，来查找注册在应用中的控制器的构造函数.

angular.module('myApp', [])

.directive('myDirective', function() {

restrict: 'A',

controller: 'SomeController'

})

可以在指令内部通过匿名构造函数的方式来定义一个内联的控制器

[复制代码](javascript:void(0);)

angular.module('myApp',[])

.directive('myDirective', function() {

restrict: 'A',

controller:

function($scope, $element, $attrs, $transclude) {

// 控制器逻辑放在这里

}

});

[复制代码](javascript:void(0);)

我们可以将任意可以被注入的ng服务注入到控制器中，便可以在指令中使用它了。控制器中也有一些特殊的服务可以被注入到指令当中。这些服务有：

**1. $scope**

与指令元素相关联的当前作用域。  
**2. $element**  
当前指令对应的元素。  
**3. $attrs**  
由当前元素的属性组成的对象。

<div id="aDiv"class="box"></div>

具有如下的属性对象：

{

id: "aDiv",

class: "box"

}

**4. $transclude**  
嵌入链接函数会与对应的嵌入作用域进行预绑定。transclude链接函数是实际被执行用来克隆元素和操作DOM的函数。

[复制代码](javascript:void(0);)

angular.module('myApp',[])

.directive('myLink', function () {

return {

restrict: 'EA',

transclude: true,

controller:

function ($scope, $element,$attrs,$transclude) {

$transclude(function (clone) {

var a = angular.element('<a>');

a.attr('href', $attrs.value);

a.text(clone.text());

$element.append(a);

});

}

};

});

[复制代码](javascript:void(0);)

html

<my-link value="http://www.baidu.com">百度</my-link>

<div my-link value="http://www.google.com">谷歌</div>

仅在compile参数中使用transcludeFn是推荐的做法。**link函数可以将指令互相隔离开来，而controller则定义可复用的行为**。如果我们希望将当前指令的API暴露给其他指令使用，可以使用controller参数，否则可以使用link来构造当前指令元素的功能性（即内部功能）。如果我们使用了scope.$watch()或者想要与DOM元素做实时的交互，使用链接会是更好的选择。使用了嵌入，控制器中的作用域所反映的作用域可能与我们所期望的不一样，这种情况下，$scope对象无法保证可以被正常更新。当想要同当前屏幕上的作用域交互时，可以使用传入到link函数中的scope参数。

### controllerAs[string]

controllerAs参数用来设置控制器的别名，这样就可以在视图中引用控制器甚至无需注入$scope。

<div ng-controller="MainController as main">

<input type="text" ng-model="main.name" />

<span>{{ main.name }}</span>

</div>

JsCode:

angular.module('myApp',[])

.controller('MainController', function () {

this.name = "Halower";

});

控制器的别名使路由和指令具有创建匿名控制器的强大能力。这种能力可以将动态的对象创建成为控制器，并且这个对象是隔离的、易于测试。

### require[string or string[]]

**require**为字符串代表另外一个指令的名字。**require会将控制器注入到其所指定的指令中，并作为当前指令的链接函数的第四个参数**。字符串或数组元素的值是**会在当前指令的作用域中使用的指令名称**。在任何情况下，ng编译器在查找子控制器时都会参考当前指令的模板。

* 如果不使用^前缀，指令只会在自身的元素上查找控制器。指令定义只会查找定义在指令作当前用域中的ng-model=""
* 如果使用?前缀,在当前指令中没有找到所需要的控制器，会将null作为传给link函数的第四个参数。
* 如果添加了^前缀，指令会在上游的指令链中查找require参数所指定的控制器。
* 如果添加了?^ 将前面两个选项的行为组合起来，我们可选择地加载需要的指令并在父指令链中进行查找
* 如果没有任何前缀，指令将会在自身所提供的控制器中进行查找，如果没有找到任何控制器（或具有指定名字的指令）就抛出一个错误

### compile【object or function】

compile选项本身并不会被频繁使用，但是link函数则会被经常使用。本质上，当我们设置了link选项，实际上是创建了一个postLink() 链接函数，以便compile() 函数可以定义链接函数。通常情况下，如果设置了compile函数，说明我们希望在指令和实时数据被放到DOM中之前进行DOM操作，在这个函数中进行诸如添加和删除节点等DOM操作是安全的。

compile和link选项是互斥的。如果同时设置了这两个选项，那么会把compile所返回的函数当作链接函数，而link选项本身则会被忽略。

**编译函数负责对模板DOM进行转换。链接函数负责将作用域和DOM进行链接**。 在作用域同DOM链接之前可以手动操作DOM。在实践中，编写自定义指令时这种操作是非常罕见的，但有几个内置指令提供了这样的功能。

### ****link****

compile: function(tEle, tAttrs, transcludeFn) {

//todo:

return function(scope, ele, attrs) {

// 链接函数

};

链接函数是可选的。**如果定义了编译函数，它会返回链接函数**，因此当两个函数都定义时，编译函数会重载链接函数。如果我们的指令很简单，并且不需要额外的设置，可以从工厂函数（回调函数）返回一个函数来代替对象。如果这样做了，这个函数就是链接函数。

### ngModel

它提供更底层的API来处理控制器内的数据,这个API用来处理数据绑定、验证、 CSS更新等不实际操作DOM的事情,ngModel 控制器会随 ngModel 被一直注入到指令中，其中包含了一些方法。为了访问ngModelController必须使用require设置.

**ngModelController常用的元素如下：**

* **1.为了设置作用域中的视图值，需要调用 ngModel.$setViewValue() 函数。**

$setViewValue() 方法适合于在自定义指令中监听自定义事件（比如使用具有回调函数的jQuery插件），我们会希望在回调时设置$viewValue并执行digest循环。

[复制代码](javascript:void(0);)

复制代码

angular.module('myApp')

.directive('myDirective', function() {

return {

require: '?ngModel',

link: function(scope, ele, attrs, ngModel) {

if (!ngModel) return;

$(function() {

ele.datepicker({  
 //回调函数

onSelect: function(date) {

// 设置视图和调用 apply

scope.$apply(function() {

ngModel.$setViewValue(date);

});

}

});

});

}

};

});

复制代码

[复制代码](javascript:void(0);)

* + **2.$render方法可以定义视图具体的渲染方式**
  + **3.属性**  
    1. **$viewValue**  
    $viewValue属性保存着更新视图所需的实际字符串。  
    2. **$modelValue**  
    $modelValue由数据模型持有。 $modelValue和$viewValue可能是不同的，取决于$parser流水线是否对其进行了操作。  
    3. **$parsers**  
    $parsers 的值是一个由函数组成的数组，其中的函数会以流水线的形式被逐一调用。ngModel 从DOM中读取的值会被传入$parsers中的函数，并依次被其中的解析器处理。  
    4. **$formatters**  
    $formatters的值是一个由函数组成的数组，其中的函数会以流水线的形式在数据模型的值发生变化时被逐一调用。它和$parser流水线互不影响，用来对值进行格式化和转换，以便在绑定了这个值的控件中显示。  
    5. **$viewChangeListeners**  
    $viewChangeListeners的值是一个由函数组成的数组，其中的函数会以流水线的形式在视图中的值发生变化时被逐一调用。通过$viewChangeListeners，可以在无需使用$watch的情况下实现类似的行为。由于返回值会被忽略，因此这些函数不需要返回值。  
    6. **$error**  
    $error对象中保存着没有通过验证的验证器名称以及对应的错误信息。  
    7. **$pristine**  
    $pristine的值是布尔型的，可以告诉我们用户是否对控件进行了修改。  
    8. **$dirty**  
    $dirty的值和$pristine相反，可以告诉我们用户是否和控件进行过交互。  
    9. **$valid**  
    $valid值可以告诉我们当前的控件中是否有错误。当有错误时值为false， 没有错误时值为true。  
    10. **$invalid**  
    $invalid值可以告诉我们当前控件中是否存在至少一个错误，它的值和$valid相反。

# [你知道用AngularJs怎么定义指令吗？](http://www.itnose.net/detail/6144038.html)

2014-11-04 10:20

前言

最近学习了下angularjs指令的相关知识，也参考了前人的一些文章，在此总结下。

欢迎批评指出错误的地方。

Angularjs指令定义的API

AngularJs的指令定义大致如下

angular.module("app",[]).directive("directiveName",function(){

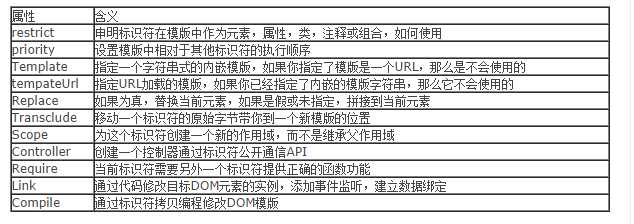
return{

//通过设置项来定义

};

})

其中return返回的对象包含很多参数，下面一一说明

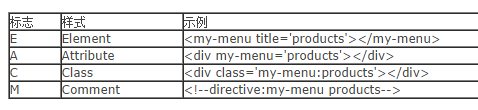


1.restrict

（字符串）可选参数，指明指令在DOM里面以什么形式被声明；

取值有：E(元素),A(属性),C(类),M(注释)，其中默认值为A；

E(元素)：<directiveName></directiveName>  
A(属性)：<div directiveName='expression'></div>  
C(类)：   <div class='directiveName'></div>  
M(注释)：<--directive:directiveName expression-->



例如restrict：‘EA’ 则表示指令在DOM里面可用元素形式和属性形式被声明；

一般来说，当你创建一个有自己模板的组件的时候，需要使用元素名，如果仅仅是为为已有元素添加功能的话，就使用属性名

注意：如果想支持IE8，则最好使用属性和类形式来定义。 另外Angular从1.3.x开始, 已经放弃支持IE8了.

2.priority

(数字)，可选参数，指明指令的优先级，若在单个DOM上有多个指令，则优先级高的先执行；

设置指令的优先级算是不常用的

比较特殊的的例子是，angularjs内置指令的ng-repeat的优先级为1000，ng-init的优先级为450；

3.terminal

（布尔型），可选参数，可以被设置为true或false，若设置为true，则优先级低于此指令的其他指令则无效，不会被调用(优先级相同的还是会执行)

4.template（字符串或者函数）可选参数，可以是：

（1）一段HTML文本

angular.module("app",[]).directive("hello",function(){

return{

restrict:'EA',

template:"<div><h3>hello world</h3></div>"

};

})

HTML代码为：<hello></hello>

结果渲染后的HTML为：<hello>

<div><h3>hello world</h3></div>

</hello>

（2）一个函数，可接受两个参数tElement和tAttrs

其中tElement是指使用此指令的元素，而tAttrs则实例的属性，它是一个由元素上所有的属性组成的集合（对象）形如：

{

title：‘aaaa’,

name:'leifeng'

}

下面让我们看看template是一个函数时候的情况

angular.module("app",[]).directive("directitle",function(){

return{

restrict:'EAC',

template: function(tElement,tAttrs){

var \_html = '';

\_html += '<div>'+tAttrs.title+'</div>';

return \_html;

}

};

})

HTML代码：<directitle title='biaoti'></directitle>

渲染之后的HTML：<div>biaoti</div>

因为一段HTML文本，阅读跟维护起来都是很麻烦的，所用通常会使用templateUrl这个。

5.templateUrl（字符串或者函数），可选参数，可以是

（1）一个代表HTML文件路径的字符串

（2）一个函数，可接受两个参数tElement和tAttrs（大致同上）

注意：在本地开发时候，需要运行一个服务器，不然使用templateUrl会报错 Cross Origin Request Script（CORS）错误

由于加载html模板是通过异步加载的，若加载大量的模板会拖慢网站的速度，这里有个技巧，就是先缓存模板

你可以再你的index页面加载好的，将下列代码作为你页面的一部分包含在里面。

<script type='text/ng-template' id='woshimuban.html'>

<div>我是模板内容</div>

</script>

这里的id属性就是被设置在templateUrl上用的。

另一种办法缓存是:

angular.module("template.html", []).run(["$templateCache", function($templateCache) {

$templateCache.put("template.html",

"<div>wo shi mu ban</div>");

}]);

 6.replace

（布尔值），默认值为false，设置为true时候，我们再来看看下面的例子（对比下在template时候举的例子）

angular.module("app",[]).directive("hello",function(){

return{

restrict:'EA',

replace：true,

template:"<div><h3>hello world</h3></div>"

};

})

HTML代码为：

<hello></hello>

渲染之后的代码：<div><h3>hello world</h3></div>

对比下没有开启replace时候的渲染出来的HTML。发现<hello></hello>不见了。

另外当模板为纯文本（即template:"wo shi wen ben"）的时候，渲染之后的html代码默认的为文本用span包含。

7.scope

可选参数，（布尔值或者对象）默认值为false，可能取值：

（1）默认值false。

表示继承父作用域;

（2）true

表示继承父作用域，并创建自己的作用域（子作用域）;

（3）{}

表示创建一个全新的隔离作用域；

7.1首先我们先来了解下scope的继承机制。我们用ng-controller这个指令举例，

我们都知道ng-controller（内置指令）可以从父作用域中继承并且创建一个新的子作用域。如下：

<!doctype html>

<html ng-app="myApp">

<head>

<script src="http://cdn.staticfile.org/angular.js/1.2.10/angular.min.js"></script>

</head>

<body>

<div ng-init="aaa='父亲'">

parentNode：{{aaa}}

<div ng-controller="myController">

chrildNode： {{aaa}}

</div>

</div>

<script>

angular.module('myApp', [])

.controller('myController',function($scope){

$scope.aaa = '儿子'

})

</script>

</body>

</html>

这时页面显示是

parentNode：父亲

chrildNode： 儿子

若去掉

$scope.aaa = '儿子'

则显示

parentNode：父亲

chrildNode： 父亲

注意：

1）若一个元素上有多个指令，使用了隔离作用域，则只有其中一个可以生效；

2）只有指令模板中的根元素才能获得一个新的作用域，这时候，scope就被设置为true了；

<!doctype html>

<html ng-app="myApp">

<head>

<script src="http://cdn.staticfile.org/angular.js/1.2.10/angular.min.js"></script>

</head>

<body>

<div ng-init="aaa='父亲'">

parentNode：{{aaa}}

<div class='one' ng-controller="myController">

chrildNode： {{aaa}}

<div class='two' ng-controller='myController2'>

{{aaa}}

</div>

</div>

</div>

<script>

angular.module('myApp', [])

.controller('myController',function($scope){

$scope.aaa = '儿子';

})

.controller('myController2',function($scope){

$scope.aaa = '孙女';

})

</script>

</body>

</html>

页面显示为：

parentNode：父亲

chrildNode： cunjieliu

孙女

上面中class为one那个div获得了指令ng-controller=’myController‘所创建的新的作用域；

而class为two那个div获得了指令ng-controller=’myController2‘所创建的新的作用域；

这就是“只有指令模板中的根元素才能获得一个新的作用域”；

接下来我们通过一个简单明了的例子来说明scope取值不同的差别

<!doctype html>

<html ng-app="myApp">

<head>

<script src="http://cdn.staticfile.org/angular.js/1.2.10/angular.min.js"></script>

</head>

<body>

<div ng-controller='MainController'>

父亲: {{name}}

<input ng-model="name" />

<div my-directive></div>

</div>

<script>

angular.module('myApp', [])

.controller('MainController', function ($scope) {

$scope.name = 'leifeng';

})

.directive('myDirective', function () {

return {

restrict: 'EA',

scope:false,//改变此处的取值,看看有什么不同

template: '<div>儿子:{{ name }}<input ng-model="name"/></div>'

};

});

</script>

</body>

</html>

依次设置scope的值false，true，{}，结果发现（大家别偷懒，动手试试哈）

当为false时候，儿子继承父亲的值，改变父亲的值，儿子的值也随之变化，反之亦如此。（继承不隔离）

当为true时候，儿子继承父亲的值，改变父亲的值，儿子的值随之变化，但是改变儿子的值，父亲的值不变。（继承隔离）

当为{}时候，没有继承父亲的值，所以儿子的值为空，改变任何一方的值均不能影响另一方的值。（不继承隔离）

tip：当你想要创建一个可重用的组件时隔离作用域是一个很好的选择，通过隔离作用域我们确保指令是‘独立’的,并可以轻松地插入到任何HTML app中，并且这种做法防止了父作用域被污染；

7.2隔离作用域可以通过绑定策略来访问父作用域的属性。

下面看一个例子

<!doctype html>

<html ng-app="myApp">

<head>

<script src="http://cdn.staticfile.org/angular.js/1.2.10/angular.min.js"></script>

</head>

<body>

<div ng-controller='MainController'>

<input type="text" ng-model="color" placeholder="Enter a color"/>

<hello-world></hello-world>

</div>

<script>

var app = angular.module('myApp',[]);

app.controller('MainController',function(){});

app.directive('helloWorld',function(){

return {

scope: false,

restrict: 'AE',

replace: true,

template: '<p style="background-color:{{color}}">Hello World</p>'

}

});

</script>

</body>

</html>

运行代码，并在input中输入颜色值，结果为



但是，但我们将scope设置为{}时候，再次运行上面的代码可以发现页面并不能成功完整显示！

原因在于，这里我们将scope设置为{}，产生了隔离作用域。

所以在template模板中{{color}}变成了依赖于自己的作用域，而不是依赖于父作用域。

因此我们需要一些办法来让隔离作用域能读取父作用域的属性，就是绑定策略。

下面我们就来探索设置这种绑定的几种方法

方法一：使用@（@attr）来进行单向文本（字符串）绑定

<!doctype html>

<html ng-app="myApp">

<head>

<script src="http://cdn.staticfile.org/angular.js/1.2.10/angular.min.js"></script>

</head>

<body>

<div ng-controller='MainController'>

<input type="text" ng-model="color" placeholder="Enter a color"/>

<hello-world color-attr='{{color}}'></hello-world> //注意这里设置了color-attr属性，绑定了{{color}}

</div>

<script>

var app = angular.module('myApp',[]);

app.controller('MainController',function(){});

app.directive('helloWorld',function(){

return {

scope: {color:'@colorAttr'}, //指明了隔离作用域中的属性color应该绑定到属性colorAttr

restrict: 'AE',

replace: true,

template: '<p style="background-color:{{color}}">Hello World</p>'

}

});

</script>

</body>

</html>

这种办法只能单向，通过在运行的指令的那个html标签上设置color-attr属性，并且采用{{}}绑定某个模型值。

注意，你也可以再这里直接绑定字符串的颜色值,如：color-attr=“red”；

然后你可以看到表达式{{color}}被赋值给了color-attr。

当表达式的值发生变化时，属性color-attr也会发生变化，所以也改变了隔离作用域中的属性color。

tips：如果绑定的隔离作用域属性名与元素的属性名相同，则可以采取缺省写法。

html：

<hello-world color="{{color}}"/>

js定义指令的片段：

app.directive('helloWorld',function(){

return {

scope: {

color: '@'

},

...

//配置的余下部分

}

});

方法二：使用=（=attr）进行双向绑定

<!doctype html>

<html ng-app="myApp">

<head>

<script src="http://cdn.staticfile.org/angular.js/1.2.10/angular.min.js"></script>

</head>

<body>

<div ng-controller='MainController'>

<input type="text" ng-model="color" placeholder="Enter a color"/>

{{color}}

<hello-world color='color'></hello-world> //注意这里的写法

</div>

<script>

var app = angular.module('myApp',[]);

app.controller('MainController',function(){});

app.directive('helloWorld',function(){

return {

scope:{color:'='},

restrict: 'AE',

replace: true,

template: '<div style="background-color:{{color}}">Hello World<div><input type="text" ng-model="color"></div></div>'

}

});

</script>

</body>

</html>

此处也类似上面采用了缺省的写法。

这里需要注意的是，我们要直接在指令运行的那个元素上设置属性时候，是直接将 实际的作用域模型 赋值给该属性（这里就是color）

这样一个双向绑定被建立了，改变任何一个input都会改变另一个值。



方法三：使用&来调用父作用域中的函数

<!doctype html>

<html ng-app="myApp">

<head>

<script src="http://cdn.staticfile.org/angular.js/1.2.10/angular.min.js"></script>

</head>

<body>

<div ng-controller='MainController'>

<input type="text" ng-model="name" placeholder="Enter a color"/>

{{name}}

<hello-world saysomething999="say();" name="liucunjie"></hello-world> //注意这里

</div>

<script>

var app = angular.module('myApp',[]);

app.controller('MainController',function($scope){

$scope.say = function(){

alert('hello');

}

$scope.name = 'leifeng';

});

app.directive('helloWorld',function(){

return {

scope:{

saysomething:'&saysomething999',

name:'@'

},

restrict: 'AE',

replace: true,

template: '<button type="button" ng-bind="name" ng-init="saysomething();"></button>'

}

});

</script>

</body>

</html>

运行之后，弹出alert框。

8.transclude

（布尔值或者字符‘element’），默认值为false；

这个配置选项可以让我们提取包含在指令那个元素里面的内容，再将它放置在指令模板的特定位置。

当你开启transclude后，你就可以使用ng-transclude来指明了应该在什么地方放置transcluded内容

<!doctype html>

<html ng-app="myApp">

<head>

<script src="http://cdn.staticfile.org/angular.js/1.2.10/angular.min.js"></script>

</head>

<body>

<div ng-controller='MainController'>

<div class='a'>

<p>china</p>

<hello-world>

{{name}}

</hello-world>

</div>

</div>

<script>

var app = angular.module('myApp',[]);

app.controller('MainController',function($scope){

$scope.name = 'leifeng';

});

app.directive('helloWorld',function(){

return {

scope:{},

restrict: 'AE',

transclude: true,

template: '<div class="b"><div ng-transclude>你看不见我</div></div>'

}

});

</script>

</body>

</html>

运行上面的代码，输出

china

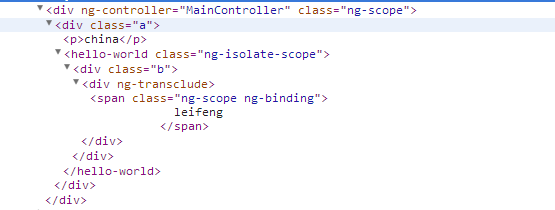
leifeng

另外当开启transclude，会创建一个新的transclude空间，并且继承了父作用域（即使Scope设置为隔离作用域），

上面代码中的{{name}}是依赖于父作用域的，仍然能被渲染出来，就说明了这点。

我们再看看生成的html为下图所示，可以发现文本“你看不见我”消失了，这是因为被transclude内容替换掉了。

这里的transclude内容就是{{name}}



接下来再来看看transclude的另一个取值transclude：“element”

那transclude：“element”与transclude：true有什么区别呢？

区别在于嵌入的内容，以上面的例子来说，

当transclude：true时候，嵌入的内容为{{name}}，

而当transclude：“element”时候，嵌入的内容为

<hello-world>

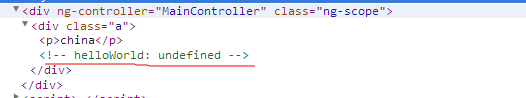
{{name}}

</hello-world>

没错，此时嵌入的内容为整个元素。

将上面代码transclude：true换成transclude：true后，再运行，你会发现结果并不是和你想的一样

再次查看生成的html代码

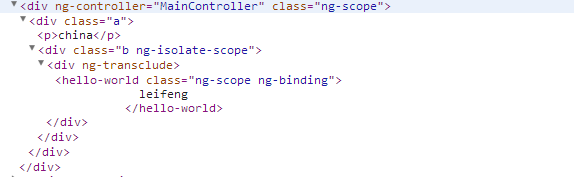


你会发现指令绑定的元素被转换为了一个HTML注释

关于这方面的疑问可以查看 [transclude: 'element' is useless without replace:true](https://github.com/angular/angular.js/issues/3368) 获取更多

解决方案是加上replace: true，就正常了

这时再查看HTML代码



 注意:在一个指令的模板template上只能申明一个ng-transclude。

OK，那么现在问题来了，如果我们想把嵌入部分多次放入我们的模板中要怎么办？

则可以使用$transclude(后面再controller选项中会讲)

或者可以使用compile函数，里面有个transcludeFn参数（后面会讲）

或者使用link链接函数。。。

9.controller

可以是一个字符串或者函数。

若是为字符串，则将字符串当做是控制器的名字，来查找注册在应用中的控制器的构造函数

angular.module('myApp', [])

.directive('myDirective', function() {

restrict: 'A', // 始终需要

controller: 'SomeController'

})

// 应用中其他的地方，可以是同一个文件或被index.html包含的另一个文件

angular.module('myApp')

.controller('SomeController', function($scope, $element, $attrs, $transclude) {

// 控制器逻辑放在这里

});

也可以直接在指令内部的定义为匿名函数，同样我们可以再这里注入任何服务（$log,$timeout等等）

angular.module('myApp',[])

.directive('myDirective', function() {

restrict: 'A',

controller:

function($scope, $element, $attrs, $transclude) {

// 控制器逻辑放在这里

}

});

另外还有一些特殊的服务（参数）可以注入

（1）$scope，与指令元素相关联的作用域

（2）$element，当前指令对应的 元素

（3）$attrs，由当前元素的属性组成的对象

（4）$transclude，嵌入链接函数，实际被执行用来克隆元素和操作DOM的函数

注意： 除非是用来定义一些可复用的行为，一般不推荐在这使用。

         指令的控制器和link函数（后面会讲）可以进行互换。区别在于，控制器主要是用来提供可在指令间复用的行为但link链接函数只能在当前内部指令中定义行为，且无法再指令间复用。

html代码: <my-site site="http://www.cnblogs.com/cunjieliu">雷锋叔叔的博客</my-site>

js代码：

<script>

angular.module('myApp',[]).directive('mySite', function () {

return {

restrict: 'EA',

transclude: true, //注意此处必须设置为true

controller:

function ($scope, $element,$attrs,$transclude,$log) { //在这里你可以注入你想注入的服务

$transclude(function (clone) {

var a = angular.element('<a>');

a.attr('href', $attrs.site);

a.text(clone.text());

$element.append(a);

});

$log.info("hello everyone");

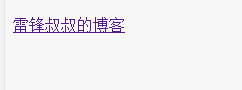
}

};

});

</script>

运行上面的代码就是



并且在控制台下输出hello everyone

让我们看看$transclude();在这里，它可以接收两个参数，第一个是$scope，作用域，第二个是带有参数clone的回调函数。

而这个clone实际上就是嵌入的内容（经过jquery包装），可以在它上做很多DOM操作。

它还有最简单的用法就是

<script>

angular.module('myApp',[]).directive('mySite', function () {

return {

restrict: 'EA',

transclude: true,

controller:

function ($scope, $element,$attrs,$transclude,$log) {

var a = $transclude(); //$transclude()就是嵌入的内容

$element.append(a);

}

};

});

</script>

注意：使用$transclude会生成一个新的作用域。

默认情况下，如果我们简单实用$transclude()，那么默认的其作用域就是$transclude生成的作用域

但是如果我们实用$transclude($scope,function(clone){}),那么作用域就是directive的作用域了

那么问题又来了。如果我们想实用父作用域呢

可以使用$scope.$parent

<div ng-controller='parentctrl'>

<div ng-controller='sonctrl'>

<my-site site="http://www.cnblogs.com/cunjieliu"><div>雷锋叔叔的博客</div></my-site>

</div>

</div>

<script>

var app = angular.module('myApp',[]);

app.controller('sonctrl',function($scope){

$scope.title = 'hello son';

});

app.controller('parentctrl',function($scope){

$scope.title = 'hello parent';

});

app.directive('mySite', function () {

return {

restrict: 'EA',

transclude: true,

controller:

function ($scope, $element,$attrs,$transclude,$log) {

var a = $transclude();

$element.append(a);

$log.info($scope.title);

$log.info($scope.$parent.title);

}

};

});

</script>

同理想要一个新的作用域也可以使用$scope.$parent.new()；

10.controllerAs

这个选项的作用是可以设置你的控制器的别名

一般以前我们经常用这样方式来写代码：

angular.module("app",[])

.controller("demoController",["$scope",function($scope){

$scope.title = "angualr";

}])

<div ng-app="app" ng-controller="demoController">

{{title}}

</div>

后来angularjs1.2给我们带来新语法糖，所以我们可以这样写

angular.module("app",[])

.controller("demoController",[function(){

this.title = "angualr";

}])

<div ng-app="app" ng-controller="demoController as demo">

{{demo.title}}

</div>

同样的我们也可以再指令里面也这样写

<script>

angular.module('myApp',[]).directive('mySite', function () {

return {

restrict: 'EA',

transclude: true,

controller:'someController',

controllerAs:'mainController'

//..其他配置

};

});

</script>

11.require(字符串或者数组)

字符串代表另一个指令的名字，它将会作为link函数的第四个参数

具体用法我们可以举个例子说明

假设现在我们要编写两个指令，两个指令中的link链接函数中（link函数后面会讲）存在有很多重合的方法，

这时候我们就可以将这些重复的方法写在第三个指令的controller中（上面也讲到controller经常用来提供指令间的复用行为）

然后在这两个指令中，require这个拥有controller字段的的指令（第三个指令），

最后通过link链接函数的第四个参数就可以引用这些重合的方法了。

<!doctype html>

<html ng-app="myApp">

<head>

<script src="http://cdn.staticfile.org/angular.js/1.2.10/angular.min.js"></script>

</head>

<body>

<outer-directive>

<inner-directive></inner-directive>

<inner-directive2></inner-directive2>

</outer-directive>

<script>

var app = angular.module('myApp', []);

app.directive('outerDirective', function() {

return {

scope: {},

restrict: 'AE',

controller: function($scope) {

this.say = function(someDirective) {

console.log('Got:' + someDirective.message);

};

}

};

});

app.directive('innerDirective', function() {

return {

scope: {},

restrict: 'AE',

require: '^outerDirective',

link: function(scope, elem, attrs, controllerInstance) {

scope.message = "Hi,leifeng";

controllerInstance.say(scope);

}

};

});

app.directive('innerDirective2', function() {

return {

scope: {},

restrict: 'AE',

require: '^outerDirective',

link: function(scope, elem, attrs, controllerInstance) {

scope.message = "Hi,shushu";

controllerInstance.say(scope);

}

};

});

</script>

</body>

</html>

上面例子中的指令innerDirective和指令innerDirective2复用了定义在指令outerDirective的controller中的方法

也进一步说明了，指令中的controller是用来让不同指令间通信用的。

另外我们可以在require的参数值加上下面的某个前缀，这会改变查找控制器的行为：

（1）没有前缀，指令会在自身提供的控制器中进行查找，如果找不到任何控制器，则会抛出一个error

（2）？如果在当前的指令没有找到所需的控制器，则会将null传给link连接函数的第四个参数

（3）^如果在当前的指令没有找到所需的控制器，则会查找父元素的控制器

（4）?^组合

12.Anguar的指令编译过程

首先加载angularjs库，查找到ng-app指令，从而找到应用的边界，

根据ng-app划定的作用域来调用$compile服务进行编译，

angularjs会遍历整个HTML文档，并根据js中指令的定义来处理在页面上声明的各个指令

按照指令的优先级(priority)排列，根据指令中的配置参数(template，place，transclude等)转换DOM

然后就开始按顺序执行各指令的compile函数（如果指令上有定义compile函数）对模板自身进行转换

注意：此处的compile函数是我们指令中配置的，跟上面说的$compile服务不一样。

每个compile函数执行完后都会返回一个link函数，所有的link函数会合成一个大的link函数

然后这个大的link函数就会被执行，主要做数据绑定，通过在DOM上注册监听器来动态修改scope中的数据，

或者是使用$watchs监听 scope中的变量来修改DOM，从而建立双向绑定等等。

若我们的指令中没有配置compile函数，那我们配置的link函数就会运行，

她做的事情大致跟上面complie返回之后所有的link函数合成的的大的link函数差不多。

所以：在指令中compile与link选项是互斥的，如果同时设置了这两个选项，

         那么就会把compile所返回的函数当做是链接函数，而link选项本身就会被忽略掉

13.compile编译函数和link链接函数

13.1compile编译函数选项

compile选项可以返回一个对象或者函数

在这里我们可以在指令和实时数据被放到DOM中之前进行DOM操作，

比如我们可以在这里进行添加或者删除节点的DOM的操作。

所以编译函数是负责对模板的DOM进行转换，并且仅仅只会运行一次。

//compile函数的语法

compile:function compile(tElement,tAttrs,transclude){

return{

pre:function preLink(scope,iElement,iAttrs,controller){},

post:function postLink(scope,iElement,iAttrs,controller){}

}

}

对于我们编写的大部分的指令来说，并不需要对模板进行转换，所以大部分情况只要编写link函数就可以了。

tips：preLink函数会在编译阶段之后，指令链接到子元素之前执行

        类似的，postLink会在指令链接到子元素之后执行

        这意味着，为了不破坏绑定过程，如果你需要修改DOM结构，你应该在postLink函数中来做这件事。

13.2link链接函数选项

链接函数负责将作用域和DOM进行链接。

//link链接函数

link:function postLink(scope,iElement,iAttrs){}

若指令中定义有require选项，则link函数会有第四个参数，代表控制器或者所依赖的指令的控制器（上面require选项例子已有例子）

14.ngModel

 偷个懒，这个重要的东西就留着下次再讲吧