[指令 1](#_Toc31824)

[基本概念 1](#_Toc29072)

[总结 1](#_Toc28500)

[www 1](#_Toc26027)

[基本概念 1](#_Toc23340)

[AngularJS 初始化加载流程 1](#_Toc15045)

[指令概述 1](#_Toc16023)

[replace: 3](#_Toc25215)

[replace:false 3](#_Toc25378)

[replace:true 3](#_Toc27861)

[replace:false 3](#_Toc31450)

[replace:true 3](#_Toc9629)

[restrict: 3](#_Toc25681)

[指定使用方式 3](#_Toc29419)

[restrict:EAMC 3](#_Toc4077)

[template: 4](#_Toc9129)

[模板内容：template[简单内容] 4](#_Toc6304)

[模板内容：templateUrl+replace[外部引用] 4](#_Toc6120)

[$templateCache[缓存] 4](#_Toc29563)

[transclude: 4](#_Toc14511)

[replace+transclude 4](#_Toc29882)

[:true 4](#_Toc9047)

[:'element' 4](#_Toc7731)

[注意: 5](#_Toc19638)

[compile: link: 5](#_Toc2647)

[scope: 6](#_Toc17059)

[scope = false 6](#_Toc8230)

[scope = true 6](#_Toc9881)

[scope = {} 6](#_Toc24510)

[简单对比： 6](#_Toc4743)

[独立scope的绑定策略 7](#_Toc21616)

[scope:全局scope与directive本地scope通信 7](#_Toc24850)

[说明 7](#_Toc24711)

[实例 8](#_Toc23751)

[指令的使用 8](#_Toc30747)

[指令和控制器结合使用 8](#_Toc13848)

[指令link中回调控制器con中定义方法 9](#_Toc19366)

[一个标签中调用多个指令[require] 9](#_Toc24094)

[指令实例---手风琴 10](#_Toc10304)

[影藏显示 10](#_Toc5227)

[手风琴 10](#_Toc17061)

[提示框 11](#_Toc10359)

[angular内置指令的使用 11](#_Toc2927)

[ng-class 11](#_Toc13165)

[ng-style 11](#_Toc24113)

[ng-switch 11](#_Toc30048)

[ng-repeat 11](#_Toc10833)

[ng-if 12](#_Toc6451)

[ng-show 13](#_Toc27608)

[ng-hide 13](#_Toc31931)

[ng-focus 13](#_Toc15244)

[ng-click 13](#_Toc18140)

[ng-class-even 13](#_Toc14609)

## 指令

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| 基本概念 | | | | |
| 总结 | 基础：关于restrict template templateUrl $templateCache replace transclude  angularjs内置的63个指令  关于compile link:操作元素，添加CSS样式，绑定事件（指令的运行原理）  指令与控制器，指令与指令的交互  scope的类型，scope绑定策略，scope独立作用域  Directive思想的起源和原理概述  其他：  编写指令的指导：ERP类型系统 与 互联网/电商类型系统必备UI组件  第三方指令库angular-ui | | | |
| www | 改善用户体验的 3 个 AngularJS 指令  <http://www.oschina.net/translate/better-ux-with-angularjs-directives>  你知道用AngularJs怎么定义指令吗？  <http://www.itnose.net/detail/6144038.html>  AngularJS 自定义指令  <http://blog.csdn.net/qq_20448859/article/details/52094380>  angular 指令详解，如何创建自定义指令？  <http://www.w3cscript.com/Angular/2014-11-26/5.html> | | | |
| 基本概念 | [directive:指令; <美>命令，训令，指令; 方针;]  replace:true|在生成HTML的过程总会不会将**<hello></hello>**删除  template配置项|待展示的div标签  restrict:'E'| 匹配模式，含义如下表：   |  |  |  | | --- | --- | --- | | 字母 | 声明风格 | 示例 | | E | 元素 | <my-menu title=products></my-menu> | | A | 属性-默认 | <div my-menu=product></my-menu> | | C | 样式类 | <div class=my-menu:products></div> | | M | 注释 | <!-- directive: my-menu product -->  <div></div> |  1. 指令执行的机制-关于compile link | | | |
| AngularJS 初始化加载流程 | 1、浏览器载入HTML，然后把它解析成DOM。 2、浏览器载入angular.js脚本。 3、AngularJS等到DOMContentLoaded事件触发。 4、AngularJS寻找ng-app指令，这个指令指示了应用的边界。 5、使用ng-app中指定的模块来配置注入器($injector)。 6、注入器($injector)是用来创建“编译服务($compile service)”和“根作用域($rootScope)”的。 7、编译服务($compile service)是用来编译DOM并把它链接到根作用域($rootScope)的。 8、ng-init指令将“World”赋给作用域里的name这个变量。 9、通过{{name}}的替换，整个表达式变成了“Hello World”。 | | | |
| 指令概述 | 概述  一个指令用来引入新的HTML语法。指令是DOM元素上的标记，使元素拥有特定的行为。举例来说，静态的HTML不知道如何来创建和展现一个日期选择器控件。  让HTML能识别这个语法，我们需要使用指令。指令通过某种方法来创建一个能够支持日期选择的元素。我们会循序渐进地介绍这是如何实现的。  如果你写过AngularJS的应用，那么你一定已经使用过指令，不管你有没有意识到。你肯定已经用过简单的指令，比如 ng-mode, ng-repeat, ng-show等。这些指令都赋予DOM元素特定的行为。例如，ng-repeat 重复特定的元素，ng-show 有条件地显示一个元素。如果你想让一个元素支持拖拽，你也需要创建一个指令来实现它。指令背后基本的想法很简单。它通过对元素绑定事件监听或者改变DOM而使HTML拥有真实的交互性。  jQuery视角  想象一下使用jQuery如何创建一个日期选择器。首先，我们在HTML中添加一个普通的输入框，然后通过jQuery调用 $(element).dataPicker() 来将它转变成一个日期选择器。但是，仔细想一下。当一个设计人员过来检查HTML标记的时候，他/她能否立刻猜到这个字段实际上表示的内容？这只是一个简单的输入框，或者一个日期选择器？你需要查看jQuery代码来确定这些。而Angular的方法是使用一个指令来扩展HTML。所以，一个日期选择器的指令可以是下面的形式：  <input type="text" />  这种创建UI组建的方式更加直接和清晰。你可以轻易地通过查看元素就明白这到底是什么。  创建自定义指令：  一个Angular指令可以有以下的四种表现形式：   1. 一个新的HTML元素（<data-picker></data-picker>） 2. 元素的属性（<input type=”text” data-picker/>）   3. CSS class（<input type=”text” class=”data-picker”/>）  4. 注释（<!–directive:data-picker –>）  当然，我们可以控制我们的指令在HTML中的表现形式。下面我们来看一下AngularJS中的一个典型的指令的写法。指令注册的方式与 controller 一样，但是它返回的是一个拥有指令配置属性的简单对象(指令定义对象) 。  angular.module('myapp', []).directive('helloWorld', function() {  return {  restrict: 'AE',  replace: 'true',  template: '<h3>Hello World!!</h3>'  };  });  .directive()方法在模块中注册了一个新的指令。这个方法的第一个参数是这个指令的名字。第二个参数是一个返回指令定义对象的函数。如果你的指令依赖于其他的对象或者服务，比如 $rootScope, $http, 或者$compile，他们可以在这个时间被注入。这个指令在HTML中以一个元素使用，如下：  <hello-world/>  <hello:world/>  <div hello-world></div>  <div hello:world/>  如果你想要符合HTML5的规范，你可以在元素前面添加 x- 或者 data-的前缀。所以下面的标记也会匹配 helloWorld 指令：  <div data-hello-world></div>  <div x-hello-world></div>  注意： 在匹配指令的时候，Angular会在元素或者属性的名字中剔除 x- 或者 data- 前缀。 然后将 – 或者 : 连接的字符串转换成驼峰(camelCase)表现形式，然后再与注册过的指令进行匹配。这是为什么，我们在HTML中以 hello-world 的方式使用 helloWorld 指令。其实，这跟HTML对标签和属性不区分大小写有关。 尽管上面的指令仅仅实现了静态文字的显示，但是这里还是有一些有趣的点值得我们去挖掘。  我们在指令定义过程中使用了三个属性来配置指令。我们来一一介绍他们的作用。  IMG_256  restrict  [vt.限制，限定; 约束，束缚;]  这个属性用来指定指令在HTML中如何使用（还记得之前说的，指令的四种表示方式吗）。在上面的例子中，我们使用了 ‘AE’。所以这个指令可以被当作新的HTML元素或者属性来使用。如果要允许指令被当作class来使用，我们将 restrict 设置成 ‘AEC’。  template  这个属性规定了指令被Angular编译和链接（link）后生成的HTML标记。这个属性值不一定要是简单的字符串。template 可以非常复杂，而且经常包含其他的指令，以及表达式({{ }})等。更多的情况下你可能会见到 templateUrl， 而不是 template。所以，理想情况下，你应该将模板放到一个特定的HTML文件中，然后将 templateUrl 属性指向它。  replace  这个属性指明生成的HTML内容是否会替换掉定义此指令的HTML元素。在我们的例子中，我们用 <hello-world></hello-world>的方式使用我们的指令，并且将 replace 设置成 true。所以，在指令被编译之后，生成的模板内容替换掉了 <hello-world></hello-world>。最终的输出是 <h3>Hello World!!</h3>。如果你将 replace 设置成 false，也就是默认值，那么生成的模板会被插入到定义指令的元素中。  打开这个 plunker，在”Hello World!!”右键检查元素内容，来更形象地明白这些。  link函数和Scope  指令生成出的模板其实没有太多意义，除非它在特定的scope下编译。默认情况下，指令并不会创建新的子scope。更多的，它使用父scope。也就是说，如果指令存在于一个controller下，它就会使用这个controller的scope。 如何运用scope，我们要用到一个叫做 link 的函数。它由指令定义对象中的link属性配置。让我们来改变一下我们的 helloWorld 指令，当用户在一个输入框中输入一种颜色的名称时，Hello World 文字的背景色自动发生变化。同时，当用户在 Hello World 文字上点击时，背景色变回白色。 相应的HTML标记如下：  <body ng-controller="MainCtrl">  <input type="text" ng-model="color" placeholder="Enter a color" />  <hello-world/>  </body>  修改后的 helloWorld 指令如下：  app.directive('helloWorld', function() {  return {  restrict: 'AE',  replace: true,  template: '<p style="background-color:{{color}}">Hello World',  link: function(scope, elem, attrs) {  elem.bind('click', function() {  elem.css('background-color', 'white');  scope.$apply(function() {  scope.color = "white";  });  });  elem.bind('mouseover', function() {  elem.css('cursor', 'pointer');  });  }  };  });  我们注意到指令定义中的 link 函数。 它有三个参数：  scope – 指令的scope。在我们的例子中，指令的scope就是父controller的scope。  elem – 指令的jQLite(jQuery的子集)包装DOM元素。如果你在引入AngularJS之前引入了jQuery，那么这个元素就是jQuery元素，而不是jQLite元素。由于这个元素已经被jQuery/jQLite包装了，所以我们就在进行DOM操作的时候就不需要再使用 $()来进行包装。  attr – 一个包含了指令所在元素的属性的标准化的参数对象。举个例子，你给一个HTML元素添加了一些属性：，那么可以在 link 函数中通过 attrs.someAttribute 来使用它。  link函数主要用来为DOM元素添加事件监听、监视模型属性变化、以及更新DOM。在上面的指令代码片段中，我们添加了两个事件， click，和 mouseover。click 处理函数用来重置 <p> 的背景色，而 mouseover 处理函数改变鼠标为 pointer。在模板中有一个表达式 {{color}}，当父scope中的 color 发生变化时，它用来改变 Hello World 文字的背景色。 这个 plunker 演示了这些概念。  compile函数  compile 函数在 link 函数被执行之前用来做一些DOM改造。它接收下面的参数：  tElement – 指令所在的元素  attrs – 元素上赋予的参数的标准化列表  要注意的是 compile 函数不能访问 scope，并且必须返回一个 link 函数。但是如果没有设置 compile 函数，你可以正常地配置 link 函数，（有了compile，就不能用link，link函数由compile返回）。compile函数可以写成如下的形式：  app.directive('test', function() {  return {  compile: function(tElem,attrs) {  //do optional DOM transformation here  return function(scope,elem,attrs) {  //linking function here  };  }  };  });  大多数的情况下，你只需要使用 link 函数。这是因为大部分的指令只需要考虑注册事件监听、监视模型、以及更新DOM等，这些都可以在 link 函数中完成。 但是对于像 ng-repeat 之类的指令，需要克隆和重复 DOM 元素多次，在 link 函数执行之前由 compile 函数来完成。这就带来了一个问题，为什么我们需要两个分开的函数来完成生成过程，为什么不能只使用一个？要回答好这个问题，我们需要理解指令在Angular中是如何被编译的！  指令是如何被编译的  当应用引导启动的时候，Angular开始使用 $compile 服务遍历DOM元素。这个服务基于注册过的指令在标记文本中搜索指令。一旦所有的指令都被识别后，Angular执行他们的 compile 方法。如前面所讲的，compile 方法返回一个 link 函数，被添加到稍后执行的 link 函数列表中。这被称为编译阶段。如果一个指令需要被克隆很多次（比如 ng-repeat），compile函数只在编译阶段被执行一次，复制这些模板，但是link 函数会针对每个被复制的实例被执行。所以分开处理，让我们在性能上有一定的提高。这也说明了为什么在 compile 函数中不能访问到scope对象。 在编译阶段之后，就开始了链接（linking）阶段。在这个阶段，所有收集的 link 函数将被一一执行。指令创造出来的模板会在正确的scope下被解析和处理，然后返回具有事件响应的真实的DOM节点。  改变指令的Scope  默认情况下，指令获取它父节点的controller的scope。但这并不适用于所有情况。如果将父controller的scope暴露给指令，那么他们可以随意地修改 scope 的属性。在某些情况下，你的指令希望能够添加一些仅限内部使用的属性和方法。如果我们在父的scope中添加，会污染父scope。 其实我们还有两种选择：  一个子scope – 这个scope原型继承子父scope。  一个隔离的scope – 一个孤立存在不继承自父scope的scope。  这样的scope可以通过指令定义对象中 scope 属性来配置。  下面的代码片段是一个例子：  app.directive('helloWorld', function() {  return {  scope: true, // use a child scope that inherits from parent  restrict: 'AE',  replace: 'true',  template: '<h3>Hello World!!</h3>'  };  });  上面的代码，让Angular给指令创建一个继承自父socpe的新的子scope。  另外一个选择，隔离的scope：  app.directive('helloWorld', function() {  return {  scope: {}, // use a new isolated scope  restrict: 'AE',  replace: 'true',  template: '<h3>Hello World!!</h3>'  };  });  这个指令使用了一个隔离的scope。隔离的scope在我们想要创建可重用的指令的时候是非常有好处的。通过使用隔离的scope，我们能够保证我们的指令是自包含的，可以被很容易的插入到HTML应用中。 它内部不能访问父的scope，所保证了父scope不被污染。  在我们的 helloWorld 指令例子中，如果我们将 scope 设置成 {}，那么上面的代码将不会工作。 它会创建一个新的隔离的scope，那么相应的表达式 {{color}} 会指向到这个新的scope中，它的值将是 undefined. 使用隔离的scope并不意味着我们完全不能访问父scope的属性。其实有一些技术可以允许我们访问父scope的属性，甚至监视他们的变化。我们会在指令这个系列的第二部分中讨论这些技术，以及一些更高级的概念，比如 Controller 函数。 第二部分也会和你一起使用Angular指令创建一个较为丰富的记事本应用。 所以，请保持关注。 | | | |
| replace: | | | | |
| replace:false | | | | replace:true |
| <hello>  <div>中间值</div>  <a href="baidu.com">baidu</a> </hello> <script type="text/javascript">  var mm=angular.module("app",[]);  mm.directive("hello",function(){  return {  restrict: "E",  template: "<div>hi syl</div>",  replace: false  }  }) </script> | | | | <hello>  <div>中间值</div>  <a href="baidu.com">baidu</a> </hello> <script type="text/javascript">  var mm=angular.module("app",[]);  mm.directive("hello",function(){  return {  restrict: "E",  template: "<div>hi syl</div>",  replace: true  }  }) </script> |
|  | | | |  |
| replace:false | | | | replace:true |
| <**hello**></**hello**> <**script** type="text/javascript">  **var mm**=**angular**.module(**"app"**,[]);  **mm**.directive(**"hello"**,**function**(){  **return** {  **restrict**: **"E"**,  **template**: **"<div>hi syl</div>"**,  **replace**: **false** }  }) </**script**> | | | | <**hello**></**hello**> <**script** type="text/javascript">  **var mm**=**angular**.module(**"app"**,[]);  **mm**.directive(**"hello"**,**function**(){  **return** {  **restrict**: **"E"**,  **template**: **"<div>hi syl</div>"**,  **replace**: **true** }  }) </**script**> |
| **效果：**    **生成：** | | | | **效果：**    **生成：** |
| restrict: | | | | |
| 指定使用方式  restrict:EAMC | <**hello**></**hello**> <**div** hello></**div**> <**div** class="hello"></**div**> **<!--directive:hello -->** <**div**></**div**>  <**script** type="text/javascript">  **var mm**=**angular**.module(**"app"**,[]);  **mm**.directive(**"hello"**,**function**(){  **return** {  **restrict**: **"EAMC"**,  **template**: **"<div>hi syl</div>"**,  **replace**: **true** }  }) </**script**> | | | |
| **效果：**    **生成：**    **说明：**  ---restrict:限制，限定; 约束，束缚;限制模式; 限制，约束; 禁止;  ---C：在注释内部最后需要加一个空格，不然会失效[坑]  ---A：默认值 | | | |
| template: | | | | |
| 模板内容：template[简单内容] | **mm**.directive(**"hello"**,**function**(){  **return** {  **restrict**: **"EAMC"**,  **template**: **"<div>adsf</div>"**,  **replace**: **true** } }) | | | |
| ---[template：模板; 样板; 型板;]  ---**template的值应该是一个标签包围的字符串；也可以是一个空字符串**  **---若template的值为一个单独的字符串："11111";则会出现报错：** | | | |
| 模板内容：templateUrl+replace[外部引用] | <**div**></**div**> <**script** type="text/javascript">  **var mm**=**angular**.module(**"app"**,[]);  **mm**.directive(**"hello"**,**function**(){  **return** {  **restrict**: **"EAMC"**,  **templateUrl**: **"hello.html"**,  **replace**: **true** }  }) </**script**> | | | |
| $templateCache[缓存] | // run : 当注射器加载完所有模块，此方法执行一次 // angular内置的$templateCache将需要使用的模板缓存起来：通过put()存储，get()获取  **mm**.run(**function**($templateCache){  $templateCache.put(**"hello.html"**,**"<div>hello everyone ! ! ! </div>"**)  $templateCache.put(**"other"**,**"<div>Hi Hi</div>"**)  })  **mm**.directive(**"hello"**,**function**($templateCache){  **return** {  **restrict**: **"EAMC"**,  **templateUrl**: $templateCache.get(**"hello.html"**),  **replace**: **true** }  }) | | | |
| transclude: | | | | |
| replace+transclude | ---使用template 与 replace的结合会将原来写在<hello></hello>指令中的内容替换掉;  ---使用template 与 transclude 可以保留编译前指令内部书写的内容，包括指令；这意味着可以实现指令的嵌套使用！  ---[transclude:嵌入]赋值：布尔值 | 字符"element" (mm:false) | | | |
| :true | transclude: true,  开启transclude后，就可以使用ng-transclude来指明：应该在**模板template**什么地方放置**指令元素**中的内容；4  其template中的值非必须 标签包裹 ;template中div[ng-transclude]标签中任何值都无展示效果。  另外当开启transclude，会创建一个新的transclude空间，并且继承了父作用域（即使Scope设置为隔离作用域），上面代码中的{{name}}是依赖于父作用域的，仍然能被渲染出来，就说明了这点。我们再看看生成的html为下图所示，可以发现文本“你看不见我”消失了，这是因为被transclude内容替换掉了。这里的transclude内容就是{{name}} | | | |
|  |  | | | |
|  | |  | |
|  |  | | | |
|  |  | | |
| :'element' | transclude:"element", replace:true,  与transclude：true区别在于嵌入的内容，以上面的例子来说:  当transclude：true时候，嵌入的内容为{{name}}，  当transclude：“element”时候，嵌入的内容为  <hello-world>  {{name}}  </hello-world>  没错，此时嵌入的内容为整个元素。  将上面代码transclude：true换成transclude："element"后，你会发现运行结果并不是和你想的一样，再次查看生成的html代码，你会发现指令绑定的元素被转换为了一个HTML注释。  关于这方面的疑问可以查看 transclude: 'element' is useless without replace:true 获取更多。解决方案是加上replace: true。   |  |  | | --- | --- | | **使用** | **模板template** | | <**handsome-me**>  {{name}} </**handsome-me**> | <**div**>  <**p**>111111</**p**>  <**div** ng-transclude>无用、看不见</**div**> </**div**> | | **效果** | | | transclude:true | <**div**>  <**p**>11111</**p**>  {{name}} </**div**> | | transclude:'element'  replace:true | <**div**>  <**p**>111111</**p**>  <**handsome-me**>  {{name}}  </**handsome-me**> </**div**> | | | | |
| 注意: | 1.在一个指令的模板template上只能申明一个ng-transclude。那么现在问题来了，如果我们想把嵌入部分多次放入我们的模板中要怎么办？则可以使用$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。。。  2.template中不必一定要被根标签包围  3.若template中存在"<div ng-transclude></div>"，则必须添加tansclude:true;若template中存在"<div ng-transclude></div>"，而transclude的值为false，或不存在transclude则会报错： | | | |

compile: link:

|  |  |
| --- | --- |
|  | angularjs指令中的compile与link函数详解  <http://www.jb51.net/article/58229.htm> |
|  | 在angularJs应用启动之前，它们是以HTML文本形式存在文本编辑器当中。应用启动会进行编译和链接，作用域会同HTML进行绑定。这个过程包含了两个阶段！  在编译的阶段，angularJs会遍历整个的文档并根据JavaScript中指令定义来处理页面上什么的指令。在遍历的过程中，有可能一层套着一层，一直延深处遍历。一但遍历和编译完毕就会返回一个叫做模板函数的函数。在这个函数没被返回（return）之前我们可以对编译后的DOM树进行修改。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。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。编译函数(compile)负责对模板DOM进行转换。 链接函数(link)负责将作用域和DOM进行链接。  1.compile选项可以返回一个对象或函数。compile函数自身不会经常去写，但是link函数经常被使用。请看下面的例子  IMG_256  我们写了一个DIV标签，定义了一个autohello的一个属性，属性值5.   我们想把 “小平你好的这几句话”输出5遍。我们定义个一个compile 的函数，里面进行了一些DOM操作。在angularJs里面内置了一个小型的jq，所以我们可以用jq的写法来操作。  页面输出如下：    IMG_257  在最后return一个闭包函数，其实就是我们的所说的link函数。平常我们这样定义的compile函数很少用，多种写法，因为写起来比较复杂。  有的同学还挺好奇的，你不是说最后返回的函数是link吗？我可以写多个link函数嘛？为了满足同学的好奇心，我们可以这样改写  IMG_258  然后输出：不起作用哦！！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。  IMG_259  那如果我们注释掉compile函数呢？结果会怎样呢？  IMG_260  结果得出：这个link起作用哦。因为compile被注释掉了 |
|  | 1. compile与link的区别   IMG_261 |
|  |  |
|  |  |

scope:

|  |  |  |
| --- | --- | --- |
|  | [AngularJS 指令的 Scope (作用域)](https://segmentfault.com/a/1190000002773689):  <https://segmentfault.com/a/1190000002773689>  每当一个指令被创建的时候，都会有这样一个选择，是继承自己的父作用域（一般是外部的Controller提供的作用域或者根作用域（$rootScope）），还是创建一个新的自己的作用域，当然AngularJS为我们指令的scope参数提供了三种选择，分别是：false,true,{}；默认情况下是false。 | |
| scope = false | <**style**>  **div** {  padding: 6px;  }  **div**.container {  border: 1px solid black;  }  **div**.my-info {  border: 1px solid blue;  }  **div**.my-directive {  border: 1px solid green;  } </**style**>  <**div** ng-app="MyApp">  <**div** class="container" ng-controller="MyController">  <**div** class="my-info">我的名字是：<**span** ng-bind="**name**"></**span**>  **<!-- 使用"ng-bind"防止网络状态不佳时出现没有被赋值表达式 -->** <**br**/>我的年龄是：<**span** ng-bind="age"></**span**>   </**div**>  **<!-- 使用属性声明指令 -->** <**div** class="my-directive" my-directive></**div**>  </**div**> </**div**> | |
|  | angular.module("MyApp", []).controller("MyController", **function** ($scope) {  $scope.name = "dreamapple";  $scope.age = 20;  $scope.changeAge = **function** () {  $scope.age = 22;  } }).directive("myDirective", **function** () {  **var** template = [  "<div class='my-directive'>",  " <h3>下面部分是我们创建的指令生成的</h3>",  " 我的名字是：<span ng-bind='name'></span><br/>",  " 我的年龄是：<span ng-bind='age'></span>",  " <input type='text' ng-model='name'>",  " </div>"  ];  **var** obj = {  restrict: "AE",  scope: **false**,  replace: **true**,  template: template  };  **return** obj; }); | |
|  | 在这种情况下，在指令模板中可以直接使用父作用域中的变量，函数；  因为我们将scope的属性设置为false所以，我们创建的指令继承了父作用域的一切属性和方法，这也使得在指令的模板中我们可以使用这些属性和方法。  注意：此时我们在输入框里改变名字，会发现上面的两个名字都发生了变化，你肯定会说，这肯定是这样啊，数据绑定嘛，好，我们接着往下走。 | |
| scope = true | 当把scope属性设置为true时，这表明我们创建的指令要创建一个新的作用域，这个作用域继承自我们的父作用域。  等等，刚才我们不是说了，当把scope属性值设置为false时，不也是继承我们的父作用域吗？表急，我们接着往下看。  修改上面的JS代码，将指令中的：scope:false修改为scope:true  然后我们再试着在我们的input输入框中写一些字符串，会发现，指令中的那个name发生了变化，但是指令外的那个name却没有发生变化，这说明了一个问题。  当我们将scope设置为true的时候，我们就新创建了一个作用域，只不过这个作用域是继承了我们的父作用域；我觉得可以这样理解，我们新创建的作用域是一个新的作用域，只不过在初始化的时候，用了父作用域的属性和方法去填充我们这个新的作用域。它和父作用域不是同一个作用域。  当我们将scope设置为false的时候,我们创建的指令和父作用域（其实是同一个作用域）共享同一个model模型，所以在指令中修改模型数据，它会反映到父作用域的模型中。 | |
| scope = {} | 当我们将scope的属性设置为{}时，我们可以做更多的事情。  AngularJS最强的大的地方之一就是它可以构建组建，无论放在哪里都是可以使用的； 这所以可以做到这些，不得不归功于指令的这个属性；当我们将scope设置为{}时，意味着我们创建的一个新的与父作用域隔离的新的作用域，这使我们在不知道外部环境的情况下，就可以正常工作，不依赖外部环境。 | |
| 简单对比： | 没有创建独立scope，会导致指令不能独立使用 | 添加独立scope：{}， |
| <**hello**></**hello**> <**hello**></**hello**> <**hello**></**hello**> <**hello**></**hello**> <**script** type="text/javascript">  **var mm**=**angular**.module(**"app"**,[]);  **mm**.directive(**'hello'**, **function**() {  **return** {  **restrict**: **'E'**,  **transclude**:**true**,  **template**: **'<input type="text" ng-model="username">{{username}}'** };  }); </**script**> | <**hello**></**hello**> <**hello**></**hello**> <**hello**></**hello**> <**hello**></**hello**> <**script** type="text/javascript">  **var mm**=**angular**.module(**"app"**,[]);  **mm**.directive(**'hello'**, **function**() {  **return** {  **restrict**: **'E'**,  **transclude**:**true**,  **scope**:{},  **template**: **'<input type="text" ng-model="username">{{username}}'** };  }); </**script**> |
|  |  |
|  | 我们修改了上述的JS代码和HTML代码 JS代码： | |
| <**div** ng-app="MyApp">  <**div** class="container" ng-controller="MyController">  <**div** class="my-info">我的名字是：<**span** ng-bind="**name**"></**span**>   <**br**/>我的年龄是：<**span** ng-bind="age"></**span**>  <**br**/>  </**div**>  <**div** class="my-directive" my-directive my-name="{{**name**}}" age="age" change-my-age="changeAge()"></**div**>  </**div**> </**div**> <**script**>  angular.module("MyApp", []).controller("MyController", **function** ($scope) {  $scope.name = "dreamapple";  $scope.age = 20;  $scope.changeAge = **function** () {  $scope.age = 0;  }  }).directive("myDirective", **function** () {  **var** template = [  "<div class='my-directive'>",  " <h3>下面部分是我们创建的指令生成的</h3>",  " 我的名字是：<span ng-bind='name'></span><br/>",  " 我的年龄是：<span ng-bind='age'></span><br/>",  " 在这里修改名字：<input type='text' ng-model='name'><br/>",  " <button ng-click='changeAge()'>修改年龄</button>",  " </div>"  ];  **var** obj = {  restrict: "AE",  scope: {  name: '@myName',  age: '=',  changeAge: '&changeMyAge'  },  replace: **true**,  template: template  };  **return** obj;  }); </**script**> | |
| 独立scope的绑定策略 | 我们使用了隔离的作用域，不代表我们不可以使用父作用域的属性和方法。  我们可以通过向scope的{}中传入特殊的前缀标识符（即prefix），来进行数据的绑定。  在创建了隔离的作用域，我们可以通过@,&,=引用应用指令的元素的属性，如上面的代码那样，我们可以在  <div class="my-directive" my-directive my-name="{{name}}" age="age" change-my-age="changeAge()"></div>  这个元素中，利用前缀标识符通过使用属性my-name,age,change-my-age来引用这些属性的值。  下面我们来看看如何使用这些前缀标识符： | |
|  | 具有隔离作用域的指令最主要的使用场景是创建可复用的组件，组件可以在未知上下文使用，并且可以避免污染所处的外部作用域或不经意地污染内部作用域。  直接为scope属性赋空对象可以使其数据完全与外部隔绝。但是，完全无数据的隔离作用域并不常见，我们可以在为对象添加属性与值以指定该隔离作用域内的变量如何与外部作用域联系。AngularJS提供了以下三种绑定策略，使我们能够将指令内部的隔离作用域同指令外部的作用域进行数据绑定。 | |
| @  这是一个单项绑定的前缀标识符 使用方法：在元素中使用属性，好比这样<div my-directive my-name="{{name}}"></div>，注意，属性的名字要用-将两个单词连接，因为是数据的单项绑定所以要通过使用{{}}来绑定数据。  将该隔离作用域的变量，同使用指令的DOM元素属性值相绑定；  把当前属性作为字符串传递，还可以绑定来自外层scope的值，在属性值中插入{{}}即可； | |
| =  这是一个双向数据绑定前缀标识符 使用方法：在元素中使用属性，好比这样<div my-directive age="age"></div>,注意，数据的双向绑定要通过=前缀标识符实现，所以不可以使用{{}}。  将该隔离作用域的变量同父级作用域的变量进行双向绑定:隔离作用域的name变量与父作用域的name变量进行双向绑定，它们会同步变化； | |
| &  这是一个绑定函数方法的前缀标识符 使用方法：在元素中使用属性，好比这样<div my-directive change-my-age="changeAge()"></div>，注意，属性的名字要用-将多个个单词连接。  将该隔离作用域的函数返回值同父级作用域的任何属性（包括数组，对象，函数）进行绑定；  当符号后面跟的属性名与变量名相同，可以略去。即name:”@”等同与name”@name”。 | |
|  | 注意：在新创建指令的作用域对象中，使用属性的名字进行绑定时，要使用驼峰命名标准，比如下面的代码。  scope: {  name: '@myName',// `myName` 就是原来元素中的`my-name`属性  age: '=',  changeAge: '&changeMyAge'// `changeMyAge`就是原来元素中的`change-my-age`属性 } | |
| 进一步说明，我们的指令是如何利用这些前缀标识符来寻找我们想要的属性或者函数的？  @ 当指令编译到模板的name时，就会到scope中寻找是否含有name的键值对，如果存在，就像上面那样，看到@就知道这是一个单向的数据绑定，然后寻找原来的那个使用指令的元素上（或者是指令元素本身）含有这个值的属性即my-name={{name}}，然后在父作用域查找{{name}}的值，得到之后传递给模板中的name。  =和&与@差不多，只不过=进行的是双向的数据绑定，不论模板还是父作用域上的属性的值发生改变都会使另一个值发生改变，而&是绑定函数而已。 | |
|  | 1. 在template中直接使用con中的值   <script>  mm.directive('hello2126', function() {  return {  restrict: 'AE',  transclude:true,  template: '{{name1}}'  };  }); </script> <div ng-controller="con2126" >  <hello2126 dataname="{{name1}}">1</hello2126> </div> <script type="text/javascript">  mm.controller("con2126",["$scope",function($scope){  $scope.name1="sylsyl";  }]) </script> | 1. 在link中处理来自con中的值，再赋值到template中   <script>  mm.directive('label2125', function() {  return {  restrict: 'AE',  transclude:true,  template: '{{name2}}',  link:function(scope,ele,attrs){  scope.name2=attrs.dataname+"change";  //此处的attrname最好是全小写英文字符串  }  };  }); </script> <div ng-controller="con2125" >  <label2125 dataname="{{name1}}">1</label2125> </div> <script type="text/javascript">  mm.controller("con2125",["$scope",function($scope){  $scope.name1="sylsyl";  }]) </script> |
| 1. 单向绑定 父（子）   <script type="text/javascript">  mm.directive('hello', function() {  return {  restrict: 'AE',  transclude:true,  scope:{  dataname:'@'  },  template: '{{dataname}}'  };  }); </script>  <div ng-controller="con" >  <hello dataname="{{name1}}">1</hello> </div> <script type="text/javascript">  mm.controller("con",["$scope",function($scope){  $scope.name1="sylsyl";  }]) </script> | 1. 双向绑定 父 = 子   <script type="text/javascript">  mm.directive('hello', function() {  return {  restrict: 'AE',  transclude:true,  scope:{  dataname:'='  },  template: '<input type="text" ng-model="dataname"/>'  };  }); </script>  <div ng-controller="con" >  <input type="text" ng-model="name1"/>  <hello dataname="name1">1</hello> </div> <script type="text/javascript">  mm.controller("con",function($scope){  $scope.name1="sylsyl";  }) </script> |
| 1. 传递函数   <script>  mm.directive('hello', function () {  return {  restrict: 'AE',  transclude: true,  scope: {  say: '&'  },  template: '<input type="text" ng-model="userName"/>' +  '<br/>' +  '<button ng-click="say({name:userName})">点击</button>'  };  }); </script>  <div ng-controller="con" >  <hello say="sayHello(name)">1</hello> </div> <script type="text/javascript">  var mm=angular.module("app",[]);  mm.controller("con",["$scope",function($scope){  $scope.sayHello=function(name){  log("sayHello",name);  };  }]) </script> |  |

scope:全局scope与directive本地scope通信

|  |  |  |  |
| --- | --- | --- | --- |
|  | <http://www.cnblogs.com/hubcarl/p/4202053.html>  在项目开发时，全局scope 和 directive本地scope使用范围不够清晰，全局scope与directive本地scope通信掌握的不够透彻，这里对全局scope 和 directive本地scope的使用做一个总结。 | | |
| 说明 | 一、scope作用域  1、AngularJS中，子作用域一般都会通过JavaScript原型继承机制继承其父作用域的属性和方法。如果我们在子作用域中访问一个父作用域中定义的属性，JavaScript首先在子作用域中寻找该属性，没找到再从原型链上的父作用域中寻找，如果还没找到会再往上一级原型链的父作用域寻找。在AngularJS中，作用域原型链的顶端是$rootScope，JavaScript寻找到$rootScope为止.  2、但有一个例外：在directive中使用scope: { ... }，这种方式创建的作用域是一个独立的"Isolate"作用域;  它也有父作用域，但父作用域不在其原型链上，不会对父作用域进行原型继承。  没有原型继承，这是创建可复用directive组件的最佳选择。  这种方式定义作用域通常用于构造可复用的directive组件，因为它不会直接访问/修改父作用域的属性，不会产生意外的副作用。    二、Isolate scope 引用修饰符  1、 = or =attr “Isolate”作用域的属性与父作用域的属性进行双向绑定，任何一方的修改均影响到对方，这是最常用的方式；  2、 @ or @attr “Isolate”作用域的属性与父作用域的属性进行单向绑定，即“Isolate”作用域只能读取父作用域的值，并且该值永远的String类型；  3、 & or &attr “Isolate”作用域把父作用域的属性包装成一个函数，从而以函数的方式读写父作用域的属性，包装方法是$parse    三、directive 与 controller 数据传递和通信  1、父controller监听全局scope(父scope)变量, 并广播事件给子scope(directive scope,每个directvie都有自己独立的scope作用域)  2、directive 定义本地scope,通过=、@、&(方法)字符显示引用全局scope  3、directive scope(子scope)通过parent[$scope.$parent.xxx]引用全局scope的属性  4、directive监听全局scope变量变化,可以通过$scope.$parent.$watch方法 | | |
| 实例 | <div ng-controller="MyCtrl">     <button ng-click="show=true">show</button>     <dialog title="Hello }"             visible="}"             on-cancel="show=false;"             on-ok="show=false;parentScope();">         <!--上面的on-cancel、on-ok，是在directive的isoloate scope中通过&引用的。         如果表达式中包含函数，那么需要将函数绑定在parent scope（当前是MyCtrl的scope）中-->         Body goes here: username:} , title:}.         <ul>             <!--这里还可以这么玩~names是parent scope的-->             <li ng-repeat="name in names">}</li>         </ul>         <div>             Email:<input type="text" ng-model="email" style="width: 200px;height:20px"/>         </div>         <div>             Count:<input type="text" ng-model="person.Count" style="width: 120px;height:20px"/>             <button ng-click="changeCount()">Count加1</button>         </div>         <p></p>     </dialog>  </div> | | |
|  | var app = angular.module("Dialog", []);     app.controller("MyCtrl", function ($scope) {         $scope.person = {             Count: 0         };         $scope.email = 'carl@126.com';         $scope.names = ["name1", "name2", "name3"];         $scope.show = false;         $scope.username = "carl";         $scope.title = "parent title";         $scope.parentScope = function () {             alert("scope里面通过&定义的东东，是在父scope中定义");         };             $scope.changeCount = function () {             $scope.person.Count = $scope.person.Count + 1;         }             // 监听controller count变更, 并发出事件广播,再directive 中 监听count CountStatusChange变更事件         $scope.$watch('person.Count', function (newVal, oldVal) {             console.log('>>>parent Count change:' + $scope.person.Count);             if (newVal != oldVal) {                 console.log('>>>parent $broadcast count change');                 $scope.$broadcast('CountStatusChange', {"val": newVal})             }         });         });         app.directive('dialog', function factory() {         return {             priority: 100,             template: ['<div ng-show="visible">',                 '    <h3>}</h3>',                 '    <div class="body" ng-transclude></div>',                 '    <div class="footer">',                 '        <button ng-click="onOk()">OK</button>',                 '        <button ng-click="onCancel()">Close</button>',                 '    </div>',                 '</div>'].join(""),             replace: false,             transclude: true,             restrict: 'E',             scope: {                 title: "@",//引用dialog标签title属性的值                 visible: "@",//引用dialog标签visible属性的值                 onOk: "&",//以wrapper function形式引用dialog标签的on-ok属性的内容                 onCancel: "&"//以wrapper function形式引用dialog标签的on-cancel属性的内容             },             controller: ['$scope', '$attrs', function ($scope, $attrs) {                         // directive scope title 通过@ 引用dialog标签title属性的值，所以这里能取到值                 console.log('>>>title:' + $scope.title);                 >>>title:Hello carl scope.html:85                     // 通过$parent直接获取父scope变量页可以                 console.log('>>>parent username:' + $scope.$parent.username);                 >>>parent username:carl                     // directive scope 没有定义username 变量,并且没有引用父scope username变量, 所以这里是undefined                 console.log('>>>child username:' + $scope.username);                 >>>username:undefined                         // 接收由父controller广播count变更事件                 $scope.$on('CountStatusChange', function (event, args) {                     console.log("child scope on(监听) recieve count Change event :" + args.val);                 });                     // watch 父 controller scope对象                 $scope.$parent.$watch('person.Count', function (newVal, oldVal) {                     console.log('>>>>>>>child watch parent scope[Count]:' + oldVal + ' newVal:' + newVal);                 });                 }]         };     }); | | |
| 指令的使用 | | | |
| 指令和控制器结合使用 | | <**script** type="text/javascript">  **var mm**=angular.module("app",[]);  **mm**.directive('hello', **function**() {  **return** {  restrict: 'E',  template: '<span>Hi there-----{{name}}</span>',  replace: **true** };  }); </**script**> <**div** ng-controller="con">  <**div** ng-repeat='thing in things'>  {{thing}}.<**hello**></**hello**>  </**div**> </**div**> <**script**>  **mm**.controller('con',**function**($scope) {  $scope.name="syl";  $scope.things = [1,2,3,4,5,6];  }); </**script**> | |
| [指令的本质其实是一个替换过程。分2个阶段:compile（编译）和link（连接）] | |
| 指令link中回调控制器con中定义方法  $parse为directive中传入的参数 | | // 写法1： // scope.loadDate1(); // scope.loadDate2(); // 此调用方法有作用范围； // 若只写scope.loadDate1()则会导致在点击con2的情况下出现报错 // 若只写scope.loadDate12()则会导致在点击con1的情况下出现报错 // 若写scope.loadDate1();scope.loadDate2();则loadDate2()会将loadDate1()覆盖,同样报错  // 写法2： // scope.$apply("loadDate1()"); // scope.$apply("loadDate2()"); // scope.$apply("loadDate2()"); // 此调用方法有作用范围； // 若只写scope.loadDate1();导致在点击con2的情况下出现无效果，但不会报错 // 若只写scope.loadDate2();导致在点击con1的情况下出现无效果，但不会报错 // 若写scope.$apply("loadDate1()");scope.$apply("loadDate2()"); OK OK,没问题！ **<!--定义指令-->** <**script** type="text/javascript">  **mm**.directive(**"hellow"**,**function**(){  **return** {  **restrict**: **"AE"**,  **transclude**:**true**,  **template**: **"<div>AAAA <div ng-transclude> BBBB </div> CCCC<div>"**,  link:**function**(scope,ele,attrs){  ele.bind(**"click"**,**function**(){  ele.text(**"0=0=0=0 点击后：：："**)  log(**"addEventListener success!!!"**);  **console**.log(**typeof** attrs.loadfun);  scope.$apply(**"loadDate1()"**);  scope.$apply(**"loadDate2()"**);  $parse(attrs.loadfun)(scope.$parent);  scope.$apply(attrs.loadfun+**"()"**);  })  }  }  }) </**script**> **<!--使用：视图view-->** <**div** ng-controller="con1">  <**hellow** loadfun="loadDate1">添加数据</**hellow**> </**div**> <**div** ng-controller="con2">  <**hellow** loadfun="loadDate2">tiajiashuju</**hellow**> </**div**>  typeof => string  获取的值为loadfun的赋值  // 写法3： // 本质与写法2类似 // scope.$apply(attrs.howtoload);  由方法3联想到方法2  // attrs.howtoload 中howtoload必须为小写  **<!--定义指令-->** <**script** type="text/javascript">  **mm**.directive(**"hellow"**,**function**(){  **return** {  **restrict**: **"AE"**,  **transclude**:**true**,  **template**: **"<div>AAAA <div ng-transclude> BBBB </div> CCCC<div>"**,  link:**function**(scope,ele,attrs){  ele.bind(**"click"**,**function**(){  ele.text(**"0=0=0=0 点击后：：："**)  log(**"addEventListener success!!!"**);  **console**.log(**typeof** attrs.loadfun);  scope.$apply(attrs.loadfun);  })  }  }  }) </**script**> **<!--使用：视图view-->** <**div** ng-controller="con1">  <**hellow** loadfun="loadDate1()">添加数据</**hellow**> </**div**> <**div** ng-controller="con2">  <**hellow** loadfun="loadDate2()">tiajiashuju</**hellow**> </**div**> **<!--使用：控制器con-->** <**script** type="text/javascript">  **mm**.controller(**"con1"**,[**'$scope'**,**function**($scope){  log(**"con 11 !!!"**)  $scope.loadDate1=**function**(){  log(**"load fun 1 :::"**);  }  }])  **mm**.controller(**"con2"**,**function**($scope){  log(**"con 22 !!!"**)  $scope.loadDate2=**function**(){  log(**"load fun 2 :::"**);  }  }) </**script**> | |
| 一个标签中调用多个指令[require] | | <**script** type="text/javascript">  **mm**.directive(**"superman"**,**function**(){  **return** {  **scope**:{}, //独立作用域  **restrict**:**"AE"**,  // 指令内部的controller，使指令暴露出一组方法给外部进行调用  controller:**function**($scope){  $scope.**abilities**=[];  **this**.addone=**function**(){  $scope.**abilities**.push(**"one"**);  },  **this**.addtwo=**function**(){  $scope.**abilities**.push(**"two"**);  },  **this**.addthree=**function**(){  $scope.**abilities**.push(**"three"**);  }  },  // 处理指令内部的一些事情：绑定事件、数据，或者进行dom操作，添加样式  link:**function**(scope,ele,attrs){  ele.bind(**"click"**,**function**(){  log(**"click click !!!"**)  log(scope.**abilities**)  })  }  }  })  **mm**.directive(**"one"**,**function**(){  **return**{  **require**:**"^superman"**, // 指定依赖关系  // 第四个参数关联到上面superman中controller暴露的一些方法  // 写了require之后才可以写第四个参数  link:**function**(scope,ele,attrs,con){  con.addone();  }  }  })  **mm**.directive(**"two"**,**function**(){  **return** {  **require**:**"^superman"**,  link:**function**(scope,ele,attrs,con){  con.addtwo();  }  }  })  **mm**.directive(**"three"**,**function**(){  **return** {  **require**:**"^superman"**,  link:**function**(scope,ele,attrs,con){  con.addthree();  }  }  }) </**script**>  <**superman** one>one</**superman**> <**superman** one two>two</**superman**> <**superman** one two three>three</**superman**> | |
| 指令实例---手风琴 | | | |
|  | | | 《AngularJS》5个实例详解Directive（指令）机制  <http://damoqiongqiu.iteye.com/blog/1965167>  3个可以改善用户体验的AngularJS指令介绍  <http://www.jb51.net/article/68109.htm> |
| 影藏显示 | | | **<!------------------------------------------------------------------------【辅助样式】-->** <**style**>  .expander {  border: 1px solid black;  width: 250px;  }   .expander > .title {  background-color: black;  color: white;  padding: .1em .3em;  cursor: pointer;  }   .expander > .body {  padding: .1em .3em;  } </**style**> **<!------------------------------------------------------------------------【指令定义】-->** <**script** type="text/javascript">  **var mm** = angular.module("app", []);  **mm**.directive('expander', **function** () {  **return** {  restrict: 'EA',  replace: **true**,  transclude: **true**,  scope: {  title: '=expanderTitle'  },  template: '<div>' +  '<div class="title" ng-click="toggle()">{{title}}</div>' +  '<div class="body" ng-show="showMe" ng-transclude></div>' +  '</div>',  link: **function** (scope, element, attrs) {  scope.showMe = **false**;  scope.toggle = **function** () {  scope.showMe = !scope.showMe;  }  }  }  }); </**script**> **<!------------------------------------------------------------------------【使用】-->** <**div** ng-controller="con">  <**expander** class='expander' expander-title='title'>  {{text}}  </**expander**> </**div**> <**script**>  **mm**.controller('con', **function** ($scope) {  $scope.title = '点击展开';  $scope.text = '这里是内部的内容。';  }); </**script**> |
|  |
| 手风琴 | | | <**style**>  .expander {  border: 1px solid black;  width: 250px;  }   .expander > .title {  background-color: black;  color: white;  padding: .1em .3em;  cursor: pointer;  }   .expander > .body {  padding: .1em .3em;  } </**style**>  <**script** type="text/javascript">  **var mm** = angular.module("app", []);  **mm**.directive('accordion', **function** () {  **return** {  restrict: 'EA',  replace: **true**,  transclude: **true**,  template: '<div ng-transclude></div>',  controller: **function** () {  **var** expanders = [];  **this**.gotOpened = **function** (selectedExpander) {  angular.forEach(expanders, **function** (expander) {  **if** (selectedExpander != expander) {  expander.showMe = **false**;  }  });  }  **this**.addExpander = **function** (scope) {  expanders.push(scope);  }  }  }  });  **mm**.directive('expander', **function** () {  **return** {  restrict: 'EA',  replace: **true**,  transclude: **true**,  require: '^?accordion',  scope: {  title: '=expanderTitle'  },  template: '<div>' +  '<div class="title" ng-click="toggle()">{{title}}</div>' +  '<div class="body" ng-show="showMe" ng-transclude></div>' +  '</div>',  link: **function** (scope, element, attrs, accordionController) {  scope.showMe = **false**;  accordionController.addExpander(scope);  scope.toggle = **function** () {  scope.showMe = !scope.showMe;  accordionController.gotOpened(scope);  }  }  }  }); </**script**>  <**div** ng-controller="con">  <**accordion**>  <**expander** class='expander' ng-repeat='expander in expanders' expander-title='expander.title'>  {{expander.text}}  </**expander**>  </**accordion**> </**div**>  <**script**>  **mm**.controller('con', **function** ($scope) {  $scope.expanders = [{  title: 'Click me to expand',  text: 'Hi there folks, I am the content that was hidden but is now shown.'  }, {  title: 'Click this',  text: 'I am even better text than you have seen previously'  }, {  title: 'Test',  text: 'test'  }];  }); </**script**> |
|  |
| 提示框 | | | <**div** ng-app="Dialog" ng-controller="MyCtrl">  <**button** ng-click="**show**=true">show</**button**>  <**dialogs** title="title" visible="{{**show**}}" on-cancel="show=false;" on-ok="show=false;methodInParentScope();">  **<!--上面的on-cancel、on-ok，是在directive的isoloate scope中通过&引用的。  如果表达式中包含函数，那么需要将函数绑定在parent scope（当前是MyCtrl的scope）中-->  <!--这里还可以这么玩~names是parent scope的-->** <**ul**>  <**li** ng-repeat="**name** in **names**">+++</**li**>  </**ul**>  </**dialogs**> </**div**> <**script** type="text/javascript">  **var** **myModule** = angular.module("Dialog", []);  **myModule**.controller("MyCtrl", **function** ($scope) {  $scope.names = ["name1", "name2", "name3"];  $scope.show = **false**;  $scope.username = "carl";  $scope.title = "parent title";  $scope.methodInParentScope = **function**() {  **alert**("scope里面通过&定义的东东，是在父scope中定义！！。。。");  };  });  **myModule**.directive('dialogs', **function** **factory**() {  **return** {  priority:100,  template:['<div ng-show="{{visible}}">',  '    <h3>title</h3>',  '    <div class="body" ng-transclude></div>',  '    <div class="footer">',  '        <button ng-click="onOk()">OK</button>',  '        <button ng-click="onCancel()">Close</button>',  '    </div>',  '</div>'].join(""),  replace:**false**,  transclude: **true**,  restrict:'E',  scope:{  title:"@",//引用dialog标签title属性的值  onOk:"&",//以wrapper function形式引用dialog标签的on-ok属性的内容  onCancel:"&",//以wrapper function形式引用dialog标签的on-cancel属性的内容  visible:"@"//引用dialog标签visible属性的值  }  };  }); </**script**> |

angular内置指令的使用

|  |  |
| --- | --- |
|  | <http://www.cnblogs.com/wushangjue/p/4500158.html> |
| ng-class | ngClass指令允许在HTML标签上通过绑定一个代表所有将被添加的类的表达式去动态的设定CSS类名。  该指令操作有三种不同的方式，根据三种类型的表达式计算结果为：  1、如果表达式计算的结果是个字符串，那么这个字符串就是一个或多个以空格分隔开的类名；  2、如果表达式计算的结果是个数组，而且这个数组的元素都是字符串，那么这个数组的每个元素就是一个或多个以空格分隔开的类名；  3、如果表达式的计算结果为一个对象，那么对象的每个键值对的相对应的有真实存在的值将会被当作类名。  格式：ng-calss=“value”   ng-calss=“{true：’class-one’，false：“classs-two”}[value]”  value：表达式。 |
| <**style** type="text/css">  .red{color:red}  .blue{color:blue}  .yellow{color:yellow}  .border{  border: 1px solid #f08080;  } </**style**> |
| <**div** ng-class="['red','border']">中华人名共和国</**div**> |
| <**div** ng-class='{"red":false,"blue":true}'>中华人名共和国</**div**>  <**div** ng-class="{'red':true,'blue':false,'yellow':true}">中华</**div**> |
| <**input** type="button" ng-click="isChecked = !isChecked" value="toggle" /> <**span** ng-class="{red:isChecked}">Hello World !!!</**span**> <**span** ng-class="{true:'red'}[isChecked]">Hello World !!!</**span**>  <**div** ng-class="{true: 'active', false: 'inactive'}[isActive]"> </**div**> |
| ng-style | ngStyle指令允许在HTML元素上动态设置css样式，这个指令需要和ngClass区分下，ngClass是动态设置css类名，ngStyle是动态设置css样式。  格式：ng-style=“value”    class=”ng-style:value” |
| <**p** ng-style="{color:'red'}">Hello World</**p**> <**p** class="ng-style:{color:'red'}">Hello World</**p**> |
| ng-switch | <div ng-switch on="1+1">     <p ng-switch-default>0</p>     <p ng-switch-when="1">1</p>     <p ng-switch-when="2">2</p>     <p ng-switch-when="3">3</p> </div> |
| ng-repeat | 1. 基础使用+作用域分析   <**div** ng-controller="directUse" >  <**input** type="button" ng-repeat="button in buttons" id="btn{{$index}}" value="{{button}}" ng-click="clickHandler(button);"/>  </**div**> <**script** type="text/javascript">  angular.module("myApp", []).controller("directUse", **function** ($scope) {  $scope.buttons = ["button1","button2","button3"];  $scope.clickHandler = **function**(value){  console.log(value);  **/\*  \* ng-controller指令会创建一个新的作用域scope，测试代码如下：  \* 我们可以使用angular.element(domElement).scope()方法来获得某一个DOM元素相关联的作用域。  \* \*/  var** scope0 = angular.element(document.getElementById("btn0")).scope();  **var** scope1 = angular.element(document.getElementById("btn1")).scope();  console.log(scope0 == scope1);//输出false  console.log(scope0.$parent === $scope);//true  console.log(scope1.$parent === $scope);//true  };  }) </**script**>    可以看到ng-repeat指令会新建作用域，而且是为循环中的每个dom元素新建一个作用域。通过F12调试，可以看到scope0和scope1中都有一个buttons属性，这个属性就是从父作用域下继承得到的，很类似于javascript的原型链。 |
| 1. 关于作用域的分析   <**input** type="button" value="method1" ng-click="method1();">   <**input** type="button" value="method2" ng-click="method2();">   <**input** type="button" value="method3" ng-click="method3();">  $scope.method1 = **function**(){  **var** scope0 = angular.element(**$**("#btn0")[0]).scope();  scope0.buttons = ["a1","b1","c1"]; };  $scope.method2 = **function**(){  **var** scope0 = angular.element(**$**("#btn0")[0]).scope();  scope0.$parent.buttons = ["a2","b2","c2"]; };  $scope.method3 = **function**(){  **var** scope0 = angular.element(**$**("#btn0")[0]).scope();  scope0.buttons[0] = "a3";  scope0.buttons[1] = "b3";  scope0.buttons[2] = "c3"; };  当点击method1、method2、method3的时候，我们希望将按钮button1、button2、button3替换掉。运行上面的代码可以发现：method2和method3都能成功达到目的，但是method1不能达到目的。这其实很类似C语言中传值传引用的问题。   |  | | --- | | 1. 传值传引用的问题 | | **function** **wrongChangeName**(obj){  obj = {"name":"hehe"}; } **function** **rightChangeName**(obj){  obj.name="hehe"; }  **var** **obj** = {"name":"aty"};  **wrongChangeName**(**obj**); alert(**obj**.name);//仍然是aty    **rightChangeName**(**obj**); alert(**obj**.name);//hehe    wrongChangeName就类似于我们上面的method1，而rightChangeName类似于上面的method3。也就是说如果我们想在childScope中修改parentScope中某个属性的值，那么该属性一定不能是javascript基本数据类型，一定要是对象类型。而且不能直接通过=进行赋值修改，必须是调用对象的方法来修改。 | | Javascript的基于原型的继承和基于类的继承在一些细微的方面是不同的。通常这不是问题，但是差别往往会在使用$scope时出现。在AngularJS中每一个$scope都从它的父$scope继承过来，最高层是$rootScope。（$scope在指令中表现的有些不同，指令中的隔离作用域仅继承那些显式声明的属性。）  从父级那里分享数据对于原型继承来说并不重要。不过如果不小心的话，会遮蔽父级$scope的属性。  我们想在导航栏上呈现一个用户名，然后进入登陆表单。  <div ng-controller="navCtrl">     <span>{{user}}</span>     <div ng-controller="loginCtrl">          <span>{{user}}</span>          <input ng-model="user"></input>     </div></div>    考你下：当用户在设置了ngModel的文本框中输入了值，哪个模板会被更新？是navCtrl，loginCtrl还是两者？  如果你选loginCtrl，那么你可能对原型继承的机理比较了解了。当寻找字面值时，原型链并没有被涉及。如果navCtrl要被更新的话，那么查找原型链是必要的。当一个值时对象的时候就会发生这些。（记住在Javascript中，函数、数组合对象都算作对象）  所以想要获得期望的效果就需要在navCtrl上创建一个对象可以被loginCtrl引用。  <div ng-controller="navCtrl">     <span>{{user.name}}</span>     <div ng-controller="loginCtrl">          <span>{{user.name}}</span>          <input ng-model="user.name"></input>     </div></div>  现在既然user是一个对象了，原型链会被考虑进去，navCtrl的模板和$scope也会随着loginCtrl更新。  这可能看上去像一个设计好的例子，但当涉及到像ngRepeat那样会创建子$scope的时候问题就会出现。 | | 1. 继承的问题 | | @flag1  function F1() {  this.name = "f1";  }  function F2() {}  f1 = new F1();  F2.prototype = f1;  f2 = new F2();  f2.name ="f2";  console.log(f1.name)  console.log(f2.name)      我把scalar 译为“纯量”，就是指this.name这样的，反例就是this.person = {name : "f1"}     F2的原型是f1,如果f2中没有name属性，那么由于原型继承就会去f1中寻找。如果你给f2赋予name属性并赋值。那么就相当于新建了一个值为“f2”的name属性,并且与f1无关,互相不会有影响  @flag2  对于这个例子，一开始子scope没有foo属性，所以子scope显示的时候会由于原型继承去父scope中寻找，找到了就显示。此时你改变父scope的输入，子scope也会跟随变化。但是一旦你在子scope输入值，子scope自己就会新建一个foo属性并赋予输入的值。此时由于foo是纯量，那么此时子scope与父scope就完全无关了，不管哪一方变化都不会影响对方了。如果换做对象，因为是引用类型，就不会有这问题了。  @flag3  function F1() {  this.person = {  name : "f1"  };  }  function F2() {}  f1 = new F1();  F2.prototype = f1;  f2 = new F2();  f2.person = {  name : "f2"  }  // f2.person.name = "f2";  console.log(f2.person.name)  console.log(f1.person.name) | |
| 1. 奇偶样式:ngClassEven/ngClassOdd   ngClassEven和ngClassOdd 指令和ngClass工作原理一样，除此之外，他们在ng-repeat里分别对偶（奇）数行有对应的影响。  格式： ng-calss-even=“value”    ng-class-odd=“value”  <**ol** ng-init="items = ['A','B','C','D']">  <**li** ng-repeat="i in items"><**span** ng-class-odd="'red'" ng-class-even="'blue'">{{i}}</**span**></**li**> </**ol**> |
| ng-if | ng-if会根据表达式的值来决定当前这个DOM元素的创建与销毁。当表达式为返回true时，创建，反之，则销毁。销毁就意味着这个DOM元素从页面中删除了。  ng-if指令可以根据表达式的值在DOM中生成或移除一个元素。如果赋值给ng-if的表达式的值是false，那对应的元素将会从DOM中移除，否则生成一个新的元素插入DOM中。ng-if同no-show和ng-hide指令最本质的区别是，它不是通过CSS显示或隐藏DOM节点，而是删除或者新增结点。  ng-if重新创建元素时用的是它们编译后的状态。如果ng-if内部的代码加载之后被jQuery修改过（例如用.addClass），那么当ng-if的表达式值为false时，这个DOM元素会被移除，表达式再次成为true时这个元素及其内部的子元素会被重新插入DOM，此时这些元素的状态会是它们的原始状态，而不是它们上次被移除时的状态。也就是说无论用jQuery的.addClass添加了什么类都不会存在了。而ng-show和ng-hide则可以保留dom元素上次修改后的状态。  当一个元素被ng-if从DOM中移除，同它关联的作用域也会被销毁。而且当它重新加入DOM中时，会通过原型继承从它的父作用域生成一个新的作用域。也就是说ng-if会新建作用域，而ng-show和ng-hide则不会。   |  | | --- | | 1.ng-if will remove elements from DOM. This means that all your handlers or anything else attached to those elements will be lost. For example, if you bound a click handler to one of child elements, when ng-if evaluates to false, that element will be removed from DOM and your click handler will not work any more, even after ng-if later evaluates to true and displays the element. You will need to reattach the handler.  2.ng-show/ng-hide does not remove the elements from DOM. It uses CSS styles to hide/show elements (note: you might need to add your own classes). This way your handlers that were attached to children will not be lost.  3.ng-if creates a child scope while ng-show/ng-hide does not  Elements that are not in the DOM have less performance impact and your web app might appear to be faster when using ng-if compared to ng-show/ng-hide. In my experience, the difference is negligible. Animations are possible when using both ng-show/ng-hide and ng-if, with examples for both in the Angular documentation. | |
| ng-show  ng-hide | ng-show和ng-hide根据所给表达式的值来显示或隐藏HTML元素。当赋值给ng-show指令的值为false时元素会被隐藏，值为true时元素会显示。ng-hide功能类似，使用方式相反。元素的显示或隐藏是通过改变CSS的display属性值来实现的。 |
| ng-focus | <input ng-focus="count = count + 1" ng-init="count=0" /> <h1>{{count}}</h1> |
|  |  |
| ng-click | 在Angular中已经对一些ng事件如ngClick,ngBlur,ngCopy,ngCut,ngDblclick…中加入了一个变量叫做$event,.Angular在执行事件函数时候传入了一个名叫$event的常量，该常量即代表当前event对象，如果你在Angular之前引入了jQuery那么这就是jQuery的event.  所以我们可以利用event的stopPropagation来阻止事件的冒泡：如下代码 |
|  | <**div** ng-click="click('parent',$event)">  given some text for click  <**hr**>  <**input** type="checkbox" ng-model="stopPropagation" />Stop Propagation  <**hr**>  <**button** type="button" ng-click="click('button',$event)">button</**button**> </**div**> |
|  | $scope.click = **function**(name,$event){  console.log(name +" -----called");  **if**($scope.stopPropagation){  $event.stopPropagation();  } }; |
| ng-class-even | 为表格的偶数行设置 class="striped":  <tr ng-repeat="x in records" ng-class-even="'striped'">   <td>{{x.Name}}</td>   <td>{{x.Country}}</td>  </tr> |