**Angular JS**

**directive 指令**

指令的语法：

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

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

  return {

      restrict: 'AE',

      replace: true,

      template: '<h3>Hello World!!</h3>'

  };

});

* **Restrict**

**restrict 的选项 : 必须大写， 小写无效**  


**<my-element> 对应 directive(“myElement”, function(){}) 使用驼峰命名法**

属性的合并原则

template: '<div class="user1 user2" col="yescom" col="comment" newcol="test" rowid="rowid" head="sohu yahoo">Hello World</div>'

<mytag class="user1" col="nocom" col="name" rowid="rowid" head="sohu yahoo"></mytag>

替换结果是:

<div head="sohu yahoo" rowid="rowid" newcol="test" col="nocom yescom" class="user1 user1 user2">Hello World</div>

合并原则：

1) 同名属性只有单一值， 如果名字相同这保留唯一性， 如 rowid = "rowid"

2) 同名属性多个值， 如果值相同则保留唯一性， 如 head="sohu yahoo"

3) 同名属性多个值， 如果只要有一个值不同则不 去重复叠加：如 head="sohu yahoo sohu yahoo1"

4) 同名属性的值不同，则不去重复的叠加 , 如: class="user1 user1 user2"

4) 同名属性在元素里有多个，则取第一个，后面的属性将被忽略， 如 col="nocom" col="yescom"

合并后 col = "nocom yescom"

例： html style 不会被重复； directive style 会被重复

app.directive("mytag", function () {

return {

restrict: 'AEC',

replace: true,

transclude: false,

template: '<div style="font-size:36px; color:red;" newcol="test" col="comment">Hello World</div>'

}

});

<mytag col="name" rowid="rowid" style="text-decoration: underline;"></mytag>

替换结果是:

<div newcol="test" col="name comment" style="text-decoration: underline;;font-size:36px; color:red;;font-size:36px; color:red;" rowid="rowid">Hello World</div>

style = "text-decoration:underline; font-size:36px; color: red; font-size:36px; color:red"

html style 不会被重复； directive style 会被重复(不知道何原因导致重复，也许是bug)

以上例子是替换restrict: E

restrict: A

app.directive("myattr", function() {

return {

restrict: "A",

replace: true,

transclude: false,

template: '<span class="user2" col="yescom" col="name" rowid="rowid" head="sohu yahoo">Good Friday</span>'

}

});

1) 替换元素， 属性会按照属性的合并原则附加到被替换的元素上。myattr="hello"

<select myattr="hello" class="user1" col="nocom" col="name" rowid="rowid" head="sohu yahoo" />

<option value="a">AAA</option>

<option value="b">BBB</option>

</select>

<span head="sohu yahoo" rowid="rowid" col="nocom yescom" class="user1 user2" myattr="hello">Good Friday</span>

2) tag 可以是任何自定义的

<liu myattr="hello">Eastern Day</liu>

<span head="sohu yahoo" rowid="rowid" col="yescom" class="user2" myattr="hello">Good Friday</span>

只要是带有 myattr属性 都会被替换掉， 不管是 <select> <input> <div> 或者 非法的 <myother> 等都替换成span

3) 不能有两个 directive 作用于同一个HTML DOM 上

<mytag myattr class="hello"></mytag>

但是需要注意： 如果是 <mytag myattr class="hello"></mytag> mytag 已经被定义成 directive 则出错。

也就是说， 不能有两个 directive 作用于同一个HTML DOM 上。

4) 不能做指令二次迭代：

app.directive("mytag", function () {

return {

restrict: 'E',

replace: true,

transclude: false,

template: '<div myattr>Hello World</div>' - 注意 mytag 里有 myattr 属性

}

}).directive("myattr", function() { - myattr 也是指令

return {

restrict: "A",

replace: true,

transclude: false,

template: '<span class="user2" col="yescom" col="name" rowid="rowid" head="sohu yahoo">Good Friday</span>'

}

});

如果我们定义HTML :

<mytag>You are bet</mytag> 将转成 <div myattr>Hello world</div> 接着在转成 myattr 指令, 这样是出错的

restrict: C

app.directive("myattr", function() {

return {

restrict: "C",

replace: true,

transclude: false,

template: '<input type="text" class="user1" ng-model="data1" />'

}

});

<liu class="myattr user1"></liu>

<input type="text" ng-model="data1" class="myattr user1 user1 ng-pristine ng-untouched ng-valid">

class的合并原则:

1) 是叠加合并

2) 不能进行二次迭代

template: '<input type="text" class="myattr user1 user2" ng-model="data1" />'

<liu class="myattr user1"></liu>

app.directive("myattr", function() {

return {

restrict: "C",

replace: true,

transclude: false,

template: '<input type="text" myattr="good" class="user1 user2" ng-model="data1" />'

}

});

<input type="text" ng-model="data1" class="myattr user1 user1 user2 ng-pristine ng-untouched ng-valid" myattr="undefined good">

如下则造成迭代：出错

app.directive("myattr", function() {

return {

restrict: "AC",

replace: true,

transclude: false,s

template: '<input type="text" myattr="good" class="user1 user2" ng-model="data1" />'

}

});

<liu class="myattr user1"></liu>

restrict: M

app.directive("myattr", function() {

return {

restrict: "M",

replace: true,

transclude: false,

template: '<input type="text" ng-model="data1" />'

}

});

<!-- directive: myattr --> 必须有空格否则无效

<input type="text" ng-model="data1" myattr="" class="ng-pristine ng-untouched ng-valid">

myattr 变成了属性

<!-- directive: myattr rid -->

<input type="text" ng-model="data1" myattr="rid" class="ng-pristine ng-untouched ng-valid">

无效的写法：

<!--<span style="font-size:36px;"> directive: myattr rid </span> -->

<!-- good morning: directive: myattr rid -->

注意 ：

1) M 不能和 A 同时使用， 会引起迭代

因为 M : myattr 最终会生成属性 myattr="" , 而属性 myattr 又属于A， 引发二次指令迭代。

2) M 使用replace : false 没有意义 ， 否则内容为空

3) M 使用 transclude : true / false 都一样

* **replace**

replace: false 默认

app.directive("myattr", function() {

return {

restrict: "AEC",

replace: false,

transclude: false,

template: '<input type="text" ng-model="data1" />'

}

});

E: <myattr class="user1" ng-model="name">Hello E</myattr>

<br />

A: <liu myattr="myaaa mybbb" ng-model="name">Hello A</liu>

<br />

C: <liu class="myattr user1" ng-model="name">Hello C</liu>

<br />

E:

<myattr class="user1 ng-pristine ng-untouched ng-valid" ng-model="name">

<input class="ng-pristine ng-untouched ng-valid" type="text" ng-model="data1">

</myattr>

A:

<liu class="ng-pristine ng-untouched ng-valid" ng-model="name" myattr="myaaa mybbb">

<input class="ng-pristine ng-untouched ng-valid" type="text" ng-model="data1">

</liu>

C:

<liu class="myattr user1 ng-pristine ng-untouched ng-valid" ng-model="name">

<input class="ng-pristine ng-untouched ng-valid" type="text" ng-model="data1">

</liu>

总结： 把 template 的内容插入到 html 的tag 里，template 不继承合并任何属性, 因为外层仍然保持原来的tag, 只是将 template 插入其中而已。

app.directive("mytag", function () {

return {

restrict: "AEC",

replace: false,

transclude: true,

template:

'<input uid="200" ng-model="first\_name"> :

{{first\_name}} - Include:

<span ng-transclude><input value="good morning" /></span>'

}

});

<mytag rid=300><b>Hello World</b></mytag>

<mytag class="ng-binding" rid="300">

<input class="ng-pristine ng-untouched ng-valid" ng-model="first\_name" uid="200">

: - Include:

<span ng-transclude="">

<b class="ng-scope">Hello World</b>

</span>

</mytag>

注意： replace : false 即使多个并列DOM元素，也是插入到指令里面，外层仍热不变， 所以里面的指令不继承属性和做属性合并的动作。

replace: true

上面同样的例子： 把 replace 改为true , 则出错

app.directive("mytag", function () {

return {

restrict: "AEC",

replace: true,

transclude: true,

template:'<input uid=200 ng-model="first\_name"> : {{first\_name}} - Include: <span ng-transclude><input value="good morning" /></span>'

}

});

<mytag rid=300><b>Hello World</b></mytag>

注意：因为需要替换， template 的 DOM 是多个并列的元素组成， 所以出错。 解决方法是加一个外层包裹。

app.directive("mytag", function () {

return {

restrict: "AEC",

replace: true,

transclude: true,

template:'<div rid=500><input uid=200 ng-model="first\_name"> : {{first\_name}} - Include: <span ng-transclude><input value="good morning" /></span></div>'

}

});

<mytag rid=300><b>Hello World</b></mytag>

<div class="ng-binding" rid="300 500">

<input class="ng-pristine ng-untouched ng-valid" ng-model="first\_name" uid="200">

: - Include:

<span ng-transclude="">

<b class="ng-scope">Hello World</b>

</span>

</div>

注意：以上 <mytag> 被 <div> 替换掉， 而属性 rid 做合并

* transclude 默认是 false

transclude: false

app.directive("mytag", function () {

return {

restrict: "AEC",

replace: true,

transclude: false,

template:'<div rid=500><input uid=200 ng-model="first\_name"> : <span style="color:red">{{first\_name}}</span></div>'

}

});

<mytag rid=300><b>Hello World</b></mytag> - 里面内容将完全被覆盖掉

<div rid="300 500">

<input class="ng-valid ng-dirty ng-valid-parse ng-touched" ng-model="first\_name" uid="200">

:

<span class="ng-binding" style="color:red">dfadfasdfdsaf</span>

</div>

transclude: true

需要使用 ng-transclude 属性来继承内容

注意：

1) transclude : false 使用 ng-transclude 则出错， 只能在 true 的情况下使用

2) <span class="ng-scope">xxxx</span> 这是自动生成的用来包裹原来的内容

app.directive("mytag", function () {

return {

restrict: "AEC",

replace: true,

transclude: true,

template:'<div rid=500><input uid=200 ng-model="first\_name">:{{first\_name}} <span style="color:red" ng-transclude>OKOKOK</span></div>' - ng-transclude 里面的内容会被覆盖掉

}

});

<mytag rid=300><b>Hello World</b></mytag>

<div class="ng-binding" rid="300 500">

<input class="ng-valid ng-dirty ng-valid-parse ng-touched" ng-model="first\_name" uid="200">

:343

<span ng-transclude="" style="color:red">

<b class="ng-scope">Hello World</b>

</span>

</div>

transclude: true

注意：

1) transclude : true 对于ng-transclude 里面的数据模型是一个独立的全新的子$scope 继承自父 $scope

，并非是 directive controller 这个scope。 和它是彼此独立的。

app.controller("cust1", function ($scope) {

$scope.fname = "Cust 111 Fname";

$scope.change = function () {

console.log("Cust Change Function :" + $scope.fname);

}

});

app.directive("mytag", function () {

return {

scope: true,

restrict: "AEC",

replace: true,

transclude: true,

template: '<span>

Scope: <input ng-model="fname" /> : {{fname}}<br>

<span ng-transclude></span>

</span>',

controller: function ($scope, $element, $rootScope) {

$scope.new\_name = "mytab new name";

$scope.fname = "mytab fname";

$scope.change = function () {

console.log("My Tag: " + 200 + " : " + $scope.fname);

}

}

}

});

<body ng-app="myApp" style="font-size: 16px;">

Root: <input type="text" ng-model="fname" /><br />

<div ng-controller="cust0">

Cust0: <input type="text" ng-model="fname" /><br />

<div ng-controller="cust1">

Cust1: <input type="text" ng-model="fname" /><br />

MyTag: <mytag>

Trans: <input type="text" ng-model="fname" />

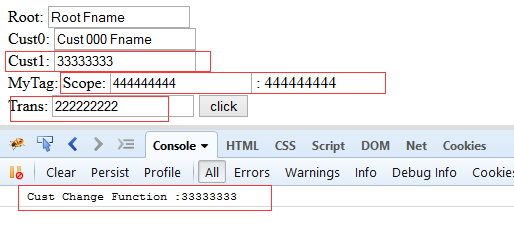
<input type="button" value="click" ng-click="change()" />

</mytag>

</div>

</div>

</body>



所以我们可以看到有三个scope:

1. 父域 ng-controller="cust1" 所在的 scope

2. scope: true directive controller 所在的scope

3. ng-transclude 所创建的继承自父域的另外一个新的子域

在 ng-click="change()"，（方法是prototype ? 相当于静态的？) 因为是父域上的方法

所以输出的 $scope.fname 也是父域上的值。

input type="button" value="click" ng-click="change()" />

2) 可以在 controller 里重载 $transclude 来覆盖或者重写 ng-transclude的内容

如果我们重载 $transclude($scope,function(clone){}), 那么作用域就是directive的作用域了

app.directive("mytag", function () {

return {

scope: {},

restrict: "AEC",

replace: true,

transclude: true,

template: '<div rid=500>

<input uid=200 ng-model="lname">:{{lname}}

<span ng-transclude></span>

</div>',

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

$scope.fname = "mytag fname";

$scope.lname = "mytag lname";

$transclude($scope, function (clone) {

console.log(clone.length);

$(clone).first().css("color", "blue");

$(clone).last().html($scope.fname + " ~ " + $scope.lname);

$element.append(clone);

});

}

}

});

<mytag rid=300><input ng-model="fname" value="Hello World" />Hello World</mytag>

<div class="ng-binding ng-isolate-scope" rid="300 500">

<input class="ng-pristine ng-untouched ng-valid" ng-model="lname" uid="200">

:mytag lname

-- below $scope is copy from parent $scope

<span ng-transclude="">

<input class="ng-pristine ng-untouched ng-valid ng-scope" value="Hello World" ng-model="fname">

<span class="ng-scope">Hello World</span>

</span>

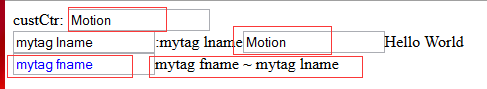
<br>

-- below append by $transclude, so $scope is directive scope

<input class="ng-pristine ng-untouched ng-valid ng-scope" value="Hello World" ng- model="fname" style="color: blue;">

<span class="ng-scope">mytag fname ~ mytag lname</span>

</div>



我们可以在 controller 里重写 $transclude

如果我们重载 $transclude($scope,function(clone){}), 那么作用域就是directive的作用域了

3) 可以在 controller 里重载 $transclude 来覆盖或者重写 ng-transclude的内容

也可以使用如下方式：

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

$scope.fname = "mytag fname";

$scope.lname = "mytag lname";

var a = $transclude();

console.log(a.length);

a.css("color", "red");

},

* scope

scope : true; false; 以及 { "@", "=", "&" } 的区别：

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;

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）

（2）true

表示继承父作用域，并创建自己的作用域（子作用域）;

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）

如果是先改变儿子的值， 则父对子的改变将终止。

（3）{}

表示创建一个全新的隔离作用域；

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） , 但是可以通过属性来建立映射关系。

注意如何跨 scope 访问：

1) 在controller 里访问：

注入 $rootScope , 如果不注入$rootScope， 也可以这样写：$scope.$root.color 来访问 $rootScope

通过：$scope.$parent $scope.$root

$scope.$parent.color 不能使用 $parentScope ， $childScope

controller: function ($scope, $rootScope) {

$scope.fname = "[" +

$rootScope.fname + " ~ " +

$scope.$parent.fname + "~" +

$scope.$root.fname + "]";

}

2) directive 指令里使用: 注意directive是相对 $scope 的。 所以不能写 $rootScope

只能用 $root ; $parent

'<input type="text"

current="{{color}}"

parent="{{$parent.color}}"

rootVal="{{$root.color}}"

/>'

scope : false

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

app.controller("cust0", function ($scope, $rootScope) {

$scope.fname = "Cust 000 Fname";

$rootScope.fname = "Root Fname";

});

app.controller("cust1", function ($scope) {

$scope.fname = "Cust 111";

});

app.directive("mytag", function () {

return {

scope: true,

restrict: "AEC",

replace: true,

transclude: true,

template: '<span>

<input ng-model="fname"><br>

Current: {{fname}} <br>

Parent: {{$parent.fname}} <br>

Root: {{$root.fname}}</span>',

controller: function ($scope, $rootScope) {

$scope.fname = "mytab fname";

}

}

});

<body ng-app="myApp" style="font-size: 16px;">

Root: <input type="text" ng-model="fname" /><br />

<div ng-controller="cust0">

Cust0: <input type="text" ng-model="fname" /><br />

<div ng-controller="cust1">

Cust1: <input type="text" ng-model="fname" /><br />

MyTag: <mytag rid=300>

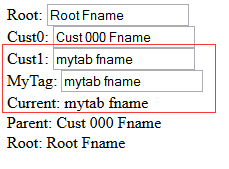
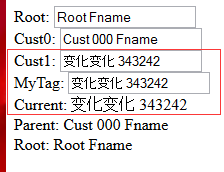
<input ng-model="fname" value="Hello World" />

</mytag><br />

</div>

</div>

</body>

$scope 是的范围和 ng-controller="cust1" 是同一个 $scope

所以变化都是修改同一个变量。

需要特别注意： transclude : true

app.directive("mytag", function () {

return {

scope: false,

restrict: "AEC",

replace: true,

transclude: true,

template: '<span><span ng-transclude></span><br>

Current: {{fname}} <br>

Parent: {{$parent.fname}} <br>

Root: {{$root.fname}}</span>',

controller: function ($scope, $rootScope) {

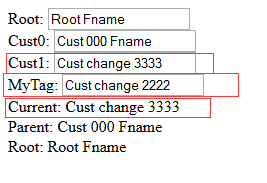
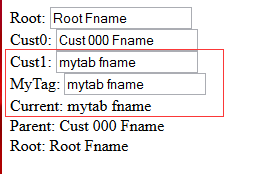
$scope.fname = "mytab fname";

}

}

});

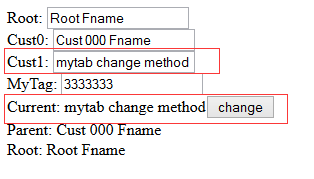
如 transclude 所介绍： transclude: true 会创建新的子$scope继承自父域$scope, 父可以改变子（单向）， 子不能改变父， 父对子的联系也会根据子的修改而终止。



子$scope.fname 的修改会影响到所有

其后父的修改可以影响到所有子

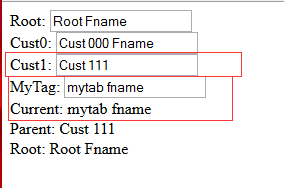
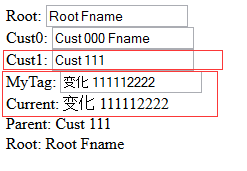
父与Current 是同一个； transclude 是子



Current 与父是同一个如上图

scope : true

使用一个子 scope 来继承来自 父亲scope

app.directive("mytag", function () {

return {

scope: true,

restrict: "AEC",

replace: true,

transclude: true,

template: '<span>

<input ng-model="fname"><br>

Current: {{fname}} <br>

Parent: {{$parent.fname}} <br>

Root: {{$root.fname}}</span>',

controller: function ($scope, $rootScope) {

$scope.fname = "mytab fname";

}

}

});

因为在 controller 里对 $scope.fname 有赋值所以子$scope 的值被覆盖了，

但是不影响父$scope的值， 同时父对子的联系被打断了。

如果我们不对子$scope.fname进行赋值操作

app.directive("mytag", function () {

return {

scope: true,

restrict: "AEC",

replace: true,

transclude: true,

template: '<span><input ng-model="fname"><br>

Current: {{fname}} <br>

Parent: {{$parent.fname}} <br>

Root: {{$root.fname}}</span>',

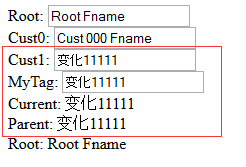
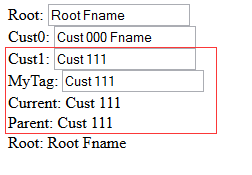
controller: function ($scope, $rootScope) {

//$scope.fname = "mytab fname";

}

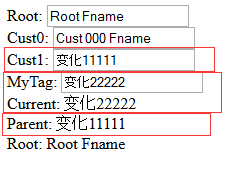
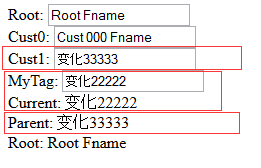
}

});



如果我们首先只改变 Cust1 : 即父 $scope.fname 则父子都跟着变化

如果我们现在改变子 $scope.fname

则父$scope 不会发生变化， 只影响子$scope，但是此时 父对子的单向联系被终止了

注意如果有多个变量， 只终止此变量， 不影响到其他变量， 其他关系如何仍旧如何。

$scope.fname; $scope.lname

需要特别注意： transclude : true

app.directive("mytag", function () {

return {

scope: true,

restrict: "AEC",

replace: true,

transclude: true,

template: '<span><span ng-transclude></span><br>

Current: {{fname}}<input type="button" ng-click="change()" value="change" /> <br>

Parent: {{$parent.fname}} <br>

Root: {{$root.fname}}</span>',

controller: function ($scope, $rootScope) {

$scope.fname = "mytab fname";

$scope.change = function () {

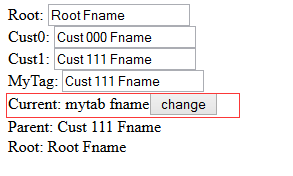
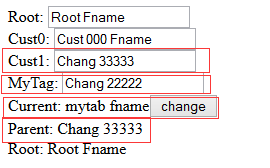
$scope.fname = "mytab change method";

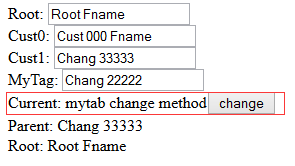
}

}

}

});

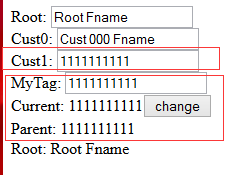
 如图transclude 的子域与 current directive 子域是两个独立域

我们可以看到

1） transclude 的子域继承自父域 ，

2）另外current directive 的子域是另外一个子域继承自父域，并且子域名的值被修改

3） 如果没有动过任何子域， 修改父域的值，会单向修改所有子域的值。

 如图所有子域都被修改

4） transclude 的子域与 current directive 子域的是两个独立的子域。 所以他们的修改彼此隔离不影响。

scope : {}

创建隔离的scope

情况一： transclude: true,

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

app.controller("cust0", function ($scope, $rootScope) {

$scope.fname = "Cust 000 Fname";

$rootScope.fname = "Root Fname";

});

app.controller("cust1", function ($scope) {

$scope.fname = "Cust 111 Fname";

$scope.change = function(){

console.log("Cust 111 :" + $scope.fname);

}

});

app.directive("mytag", function () {

return {

scope: {},

restrict: "AEC",

replace: true,

transclude: true,

template: '<div ng-click="change()" test="{{fname}} good" ng-transclude></div>',

controller: function ($scope, $element, $rootScope) {

$scope.fname = "mytab fname";

$scope.change = function () {

console.log( "My Tag: " +

$scope.fname + ":" +

$("input", $element).val()

);

//$scope.fname = "mytab change method";

}

}

}

});

<body ng-app="myApp" style="font-size: 16px;">

Root: <input type="text" ng-model="fname" /><br />

<div ng-controller="cust0">

Cust0: <input type="text" ng-model="fname" /><br />

<div ng-controller="cust1">

Cust1: <input type="text" ng-model="fname" /><br />

MyTag: <mytag ng-click="change()"

mytest="{{fname}}"

test="{{fname}}"

rid=300>

<input id="hi100"

ng-model="fname"

value="Hello World" /> :

<span ng-bind="fname"></span>

</mytag>

<br />

</div>

</div>

</body>

</html>

关注：

1） scope: {} 是创建了新的隔离的scope，而且不继承父的scope, 是全新的scope

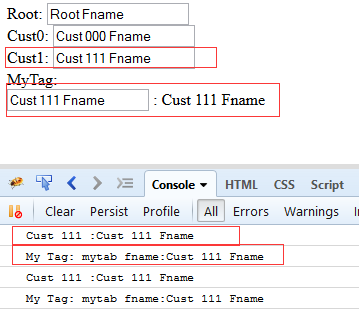
2) transclude: true, 在html tag 里定义了绑定的事件，绑定的数据，被放到directive里，会创建一

个新的子scope继承自父scope "ng-controller=cust1" 里的数据模型，而并非是使用全新的隔离的

scope

3) 在 directive里template 定义绑定的数据，是属于全新scope里controller定义的数据模型。

4) 对于覆盖定义的 ng-click会重复叠加。 符合属性叠加原则：



<div class="ng-isolate-scope" ng-transclude=""

test="Cust 111 Fname Cust 111 Fname good"  - 属性重复叠加，符合属性叠加原则, 属于父域

ng-click="change()" rid="300"

mytest="Cust 111 Fname"> - 属于父域

<input id="hi100" class="ng-pristine ng-valid ng-scope ng-touched"

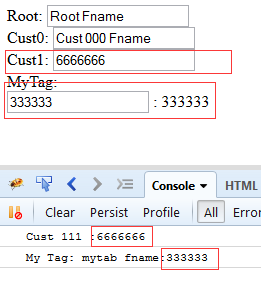
value="Hello World"

ng-model="fname"> - 属于子域继承

<span class="ng-scope"> : </span>

<span class="ng-binding ng-scope" ng-bind="fname">Cust 111 Fname</span> - 属于子域继承

</div>



<div class="ng-isolate-scope" ng-transclude=""

test="6666666 6666666 good" - 属于父域

 ng-click="change()" rid="300"

mytest="6666666"> - 属于父域

<input id="hi100" class="ng-valid ng-scope ng-touched ng-dirty ng-valid-parse"

value="Hello World"  - 虽然静止不变, 但是 $("input").val() 可以获取正确值

ng-model="fname"> - 属于子域继承

<span class="ng-scope"> : </span>

<span class="ng-binding ng-scope" ng-bind="fname">333333</span> - 属于子域继承

</div>

情况二: ng-click 事件

app.directive("mytag", function () {

return {

scope: {},

restrict: "AEC",

replace: true,

transclude: false,

template: '<input ng-click="change(200);" test="{{fname}} good" />',

controller: function ($scope, $element, $rootScope) {

$scope.fname = "mytab fname";

$scope.lname = "mytab lname";

$scope.change = function (id) {

console.log("My Tag: " + id + " : " + $scope.fname);

}

}

}

});

</script>

</head>

<body ng-app="myApp" style="font-size: 16px;">

Root: <input type="text" ng-model="fname" /><br />

<div ng-controller="cust0">

Cust0: <input type="text" ng-model="fname" /><br />

<div ng-controller="cust1">

Cust1: <input type="text" ng-model="fname" /><br />

MyTag: <mytag ng-click="change(300);"

ng-model="fname"

mytest="{{fname}}"

test="{{fname}}" rid=300>

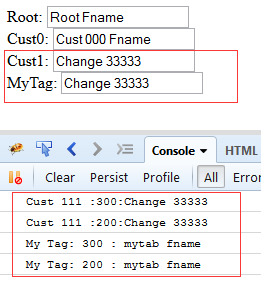
</mytag><br />

</div>

</div>

</body>

</html>



<div class="ng-scope" ng-controller="cust1">

Cust1:

<input class="ng-valid ng-dirty ng-touched ng-valid-parse" type="text" ng-model="fname">

<br>

MyTag:

<input class="ng-isolate-scope ng-valid ng-dirty ng-touched"  - 属于父域

test="Change 33333 Change 33333 good"  - 属于父域

ng-click="change(300); change(200);"  - 属于合并，而且都执行两遍

rid="300"

mytest="Change 33333" - 属于父域

ng-model="fname">

<br>

</div>

-----------------------------------------------------------------------

template: '<input ng-click="change(300); change(200);" test="{{fname}} good" />',



---------------------------------------------------------------------------------------

<mytag

ng-click="change(300); change(200);"

ng-model="fname"

mytest="{{fname}}"

test="{{fname}}"

rid=300

></mytag>



------------------------------------------------------------------------

情况三: ng-model 数据绑定

app.directive("mytag", function () {

return {

scope: {},

restrict: "AEC",

replace: true,

transclude: false,

template: '<input ng-model="fname" test="{{fname}}" />',- 此情况，进行数据绑定，属于隔离的域

controller: function ($scope, $element, $rootScope) {

$scope.change = function (id) {

console.log("My Tag: " + id + " : " + $scope.fname);

}

}

}

});

<body ng-app="myApp" style="font-size: 16px;">

Root: <input type="text" ng-model="fname" /><br />

<div ng-controller="cust0">

Cust0: <input type="text" ng-model="fname" /><br />

<div ng-controller="cust1">

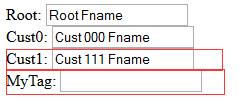
Cust1: <input type="text" ng-model="fname" /><br />

MyTag: <mytag mytest="{{fname}}" rid=300></mytag><br /> - 没有ng-model数据绑定

</div>

</div>

</body>



<input class="ng-pristine ng-untouched ng-valid ng-isolate-scope"

test=""  -属于隔离的域

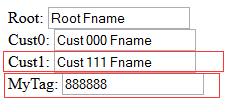
ng-model="fname" -属于隔离的域

rid="300"

mytest="Cust 111 Fname"-属于父域

>

修改：



<input class="ng-valid ng-isolate-scope ng-dirty ng-valid-parse ng-touched"

test="888888"

ng-model="fname" rid="300"

mytest="Cust 111 Fname">

情况四: ng-model 数据绑定： 不支持属性合并，提示多个绑定

app.directive("mytag", function () {

return {

scope: {},

restrict: "AEC",

replace: true,

transclude: false,

template: '<input ng-model="fname" test="{{fname}}" />',

controller: function ($scope, $element, $rootScope) {

$scope.change = function (id) {

console.log("My Tag: " + id + " : " + $scope.fname);

}

}

}

});

MyTag: <mytag ng-model="fname" mytest="{{fname}}" rid=300></mytag>



你可以看到出错信息： 因为属性合并 fname fname

同样，如果是

MyTag: <mytag ng-model="lname" mytest="{{fname}}" rid=300></mytag>

<input test="{{fname}}"

ng-model="lname fname"  - 不管名字是否相同， 属性合并造成错误：绑定多个

rid="300" mytest="{{fname}}">

情况五: transclude:flase 的隔离scope

app.directive("mytag", function () {

return {

scope: {},

restrict: "AEC",

replace: true,

transclude: false,

template: '<span><input ng-model="fname" test="{{fname}}" />{{fname}}</span>',

controller: function ($scope, $element, $rootScope) {

$scope.change = function (id) {

console.log("My Tag: " + id + " : " + $scope.fname);

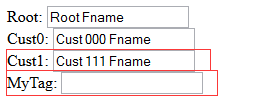
}

}

}

});

MyTag: <mytag ng-model="fname" mytest="{{fname}}" rid=300></mytag>



<span class="ng-binding ng-isolate-scope ng-valid"

rid="300"

mytest="Cust 111 Fname"

ng-model="fname"> - 由span 使用了ng-model，

<input class="ng-pristine ng-untouched ng-valid"

test="" " - 这是全新隔离域，值为undefined

ng-model="fname" - 这是全新隔离域isolateScope的ng-model， 所以为undefined

>

</span>

情况六： transclude: true, 创建继承父域的子域

app.directive("mytag", function () {

return {

scope: {},

restrict: "AEC",

replace: true,

transclude: true,

template: '<span>

<input ng-model="fname" test="{{fname}}" />{{fname}}<br>

Trans:<span ng-transclude></span>

</span>',

controller: function ($scope, $element, $rootScope) {

$scope.change = function (id) {

console.log("My Tag: " + id + " : " + $scope.fname);

}

}

}

});

<body ng-app="myApp" style="font-size: 16px;">

Root: <input type="text" ng-model="fname" /><br />

<div ng-controller="cust0">

Cust0: <input type="text" ng-model="fname" /><br />

<div ng-controller="cust1">

Cust1: <input type="text" ng-model="fname" /><br />

MyTag: <mytag ng-model="fname" mytest="{{fname}}" rid=300>

<input id="hi100" ng-model="fname" value="Hello World" />

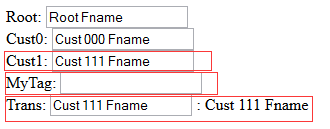
: <span ng-bind="fname"></span>

</mytag><br />

</div>

</div>

</body>



<span class="ng-binding ng-isolate-scope ng-valid" rid="300" mytest="Cust 111 Fname" ng-model="fname">

<input class="ng-pristine ng-untouched ng-valid"

test=""

 ng-model="fname"> - 这是全新隔离域

<br>

Trans:

<span ng-transclude=""> - transclude包裹的是继承父域的子域

<input id="hi100"

class="ng-pristine ng-untouched ng-valid ng-scope"

value="Hello World"

ng-model="fname">

<span class="ng-scope"> : </span>

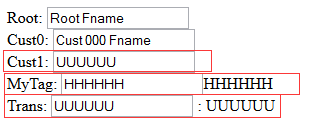
<span class="ng-binding ng-scope" ng-bind="fname">

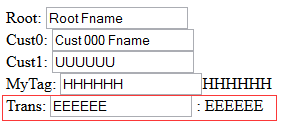
Cust 111 Fname

</span>

</span>

</span>





* scope: {"@", "=", "&"} 的使用

注意：

1） scope : { “@”, “=” , “&” } 前提是必须是 html tag 的属性, 属性名不存在则出错

2）ng-click = “event()” 因为是函数所以不能省略双括号， 否则不出错但是不工作

3）我们使用“&” 的时候， 放在属性里调用， 也是函数调用， 也不能省略双括号。

4） & 和 ng-click 是属于安全的绑定，如果没有定义此函数，顶多不执行，但不会出错，

不会象javascript 那样提示说函数未定义

a) 如何实现与父域关联：

app.directive("mytag", function () {

return {

scope: {

fname: "=ngModel" - 注意我们用回原来名字 fname, 当然也可以用回其他名字

},

restrict: "AEC",

replace: true,

transclude: true,

template: '<span>

<input ng-model="fname" test="{{fname}}" />{{fname}}<br>

- 上面的 fname其实已经是 "=ngModel" 双向绑定到父域ng-model上

Trans:<span ng-transclude></span>

</span>',

controller: function ($scope, $element, $rootScope) {

$scope.change = function (id) {

console.log("My Tag: " + id + " : " + $scope.fname);

}

}

}

});

<body ng-app="myApp" style="font-size: 16px;">

Root: <input type="text" ng-model="fname" /><br />

<div ng-controller="cust0">

Cust0: <input type="text" ng-model="fname" /><br />

<div ng-controller="cust1">

Cust1: <input type="text" ng-model="fname" /><br />

MyTag: <mytag ng-model="fname" mytest="fname" rid=300>

<input id="hi100" ng-model="fname" value="Hello World" /> :

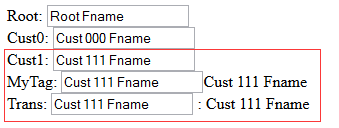
<span ng-bind="fname"></span>

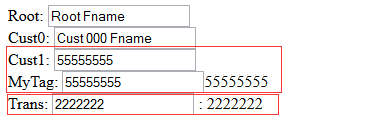
</mytag><br />

</div>

</div>

</body>





<span class="ng-binding ng-isolate-scope ng-valid" rid="300" mytest="fname" ng-model="fname">

<input class="ng-valid ng-dirty ng-touched"

test="55555555"

ng-model="fname">

55555555

- 上面已经被映射到父域的 ng-model上

<br>

Trans:

<span ng-transclude=""> - transclude包裹的是继承父域的子域

<input id="hi100" class="ng-valid ng-scope ng-dirty ng-valid-parse ng-touched"

value="Hello World"

ng-model="fname">

<span class="ng-scope"> : </span>

<span class="ng-binding ng-scope" ng-bind="fname">2222222</span>

</span>

</span>

b) 可以使用新的名字

app.directive("mytag", function () {

return {

scope: {

new\_name:"=ngModel" - 也可以使用其他名字

},

restrict: "AEC",

replace: true,

transclude: true,

template: '<span>

<input ng-model="new\_name" test="{{fname}}" />{{new\_name}}<br>

Trans:<span ng-transclude></span>

</span>',

controller: function ($scope, $element, $rootScope) {

$scope.new\_name = "mytab new name";

$scope.fname = "mytab fname";

$scope.change = function (id) {

console.log("My Tag: " + id + " : " + $scope.fname);

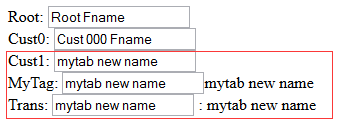
}

}

}

});

所以上面controller更改值，实际直接更改父域的值， 从而父对子域也联动修改。



c) 如果映射的属性不存在：即没有ng-model

scope: {

new\_name:"=ngModel" - 注意html tag 没有ng-model属性

},

template: '<span>

<input ng-model="new\_name" test="{{fname}}" />{{new\_name}}<br>

Trans:<span ng-transclude></span>

</span>',

MyTag: <mytag mytest="fname" rid=300>

<input id="hi100" ng-model="fname" value="Hello World" /> :

<span ng-bind="fname"></span>

</mytag><br />

提示出错：没有指定



d) 可以是任何自定义的属性：关于数据绑定不一定非要在 html tag 使用 ng-model 属性

app.directive("mytag", function () {

return {

scope: {

new\_name:"=mytest" - 使用自定义属性

},

restrict: "AEC",

replace: true,

transclude: true,

template: '<span>

<input ng-model="new\_name" test="{{fname}}" />{{new\_name}}<br>

Trans:<span ng-transclude></span>

</span>',

controller: function ($scope, $element, $rootScope) {

$scope.new\_name = "mytab new name";

$scope.fname = "mytab fname";

$scope.change = function (id) {

console.log("My Tag: " + id + " : " + $scope.fname);

}

}

}

});

<body ng-app="myApp" style="font-size: 16px;">

Root: <input type="text" ng-model="fname" /><br />

<div ng-controller="cust0">

Cust0: <input type="text" ng-model="fname" /><br />

<div ng-controller="cust1">

Cust1: <input type="text" ng-model="fname" /><br />

MyTag: <mytag mytest="fname" rid=300>

<input id="hi100" ng-model="fname" value="Hello World" /> :

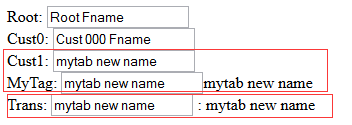
<span ng-bind="fname"></span>

</mytag><br />

</div>

</div>

</body>

记住transclude是继承父域的子域

d) 也可以是新的变量：

<body ng-app="myApp" style="font-size: 16px;">

Root: <input type="text" ng-model="fname" /><br />

<div ng-controller="cust0">

Cust0: <input type="text" ng-model="fname" /><br />

<div ng-controller="cust1">

Cust1: <input type="text" ng-model="home" /><br />

MyTag: <mytag mytest="home" rid=300>

<input id="hi100" ng-model="fname" value="Hello World" /> :

<span ng-bind="fname"></span>

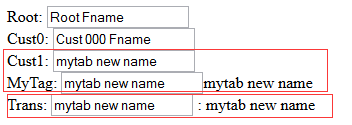
</mytag><br />

</div>

</div>

</body>

结果同上：因为在父域新建了一个变量 home , 通过关联映射随后值被修改



e) 容易犯的错误：

app.directive("mytag", function () {

return {

scope: {

new\_name:"=mytest"

},

restrict: "AEC",

replace: true,

transclude: true,

template: '<span>

<input ng-model="new\_name" test="{{fname}}" />{{new\_name}}<br>

Trans:<span ng-transclude></span>

</span>',

controller: function ($scope, $element, $rootScope) {

$scope.new\_name = "mytab new name";

$scope.fname = "mytab fname";

$scope.change = function (id) {

console.log("My Tag: " + id + " : " + $scope.fname);

}

}

}

});

<body ng-app="myApp" style="font-size: 16px;">

Root: <input type="text" ng-model="fname" /><br />

<div ng-controller="cust0">

Cust0: <input type="text" ng-model="fname" /><br />

<div ng-controller="cust1">

Cust1: <input type="text" ng-model="fname" /><br />

MyTag: <mytag mytest="{{fname}}" rid=300>

<input id="hi100" ng-model="fname" value="Hello World" /> :

<span ng-bind="fname"></span>

</mytag><br />

</div>

</div>

</body>

如果是用于关联， 不需要通过 {{}} 获取值， 只需要指定变量名即可



另外属性名不支持下划线

app.directive("mytag", function () {

return {

scope: {

new\_name:"=new\_name"

},

restrict: "AEC",

replace: true,

transclude: true,

template: '<span>

<input ng-model="new\_name" test="{{fname}}" />{{new\_name}}<br>

Trans:<span ng-transclude></span>

</span>',

controller: function ($scope, $element, $rootScope) {

$scope.new\_name = "mytab new name";

$scope.fname = "mytab fname";

$scope.change = function (id) {

console.log("My Tag: " + id + " : " + $scope.fname);

}

}

}

});

<mytag new\_name="fname" mytest="fname" rid=300>



支持 - \_ ,

new\_name , new-name 对应 newName

scope: {

new\_name:"=newName"

},

f) scope : { abc : "@abc"} ： 获取字符串值

如果只是想获取字符串值， 则可以如此处理

app.directive("mytag", function () {

return {

scope: {

new\_name:"@mytest"

},

restrict: "AEC",

replace: true,

transclude: true,

template: '<span>

<input ng-model="new\_name" test="{{fname}}" />{{new\_name}}<br>

Trans:<span ng-transclude></span>

</span>',

controller: function ($scope, $element, $rootScope) {

$scope.new\_name = "mytab new name";

$scope.fname = "mytab fname";

$scope.change = function (id) {

console.log("My Tag: " + id + " : " + $scope.fname);

}

}

}

});

<body ng-app="myApp" style="font-size: 16px;">

Root: <input type="text" ng-model="fname" /><br />

<div ng-controller="cust0">

Cust0: <input type="text" ng-model="fname" /><br />

<div ng-controller="cust1">

Cust1: <input type="text" ng-model="fname" /><br />

MyTag: <mytag mytest="{{fname}}" rid=300>

<input id="hi100" ng-model="fname" value="Hello World" /> :

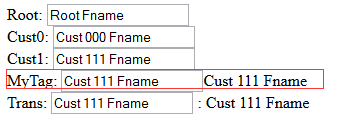
<span ng-bind="fname"></span>

</mytag><br />

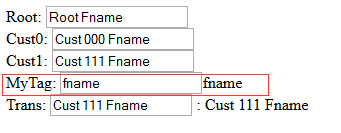
</div>

</div>

</body>



如果是 <mytag mytest="fname" rid=300> 结果则是 字符串 "fname"



h) scope : { abc : "&abc"} ： 用于关联函数

app.directive("mytag", function () {

return {

scope: {

myclick:"&" - 关联同名父域属性，也可以使用其他属性名称，不一定同名

myclick : "&otherAttr"

},

restrict: "AEC",

replace: true,

transclude: true,

template: '<span>

<input type="button" ng-click="myclick()" value="click" />{{new\_name}}<br>

Trans:<span ng-transclude></span>

</span>',

- 使用函数时一定要带（） - myclick()

controller: function ($scope, $element, $rootScope) {

$scope.new\_name = "mytab new name";

$scope.fname = "mytab fname";

//$scope.lname = "mytab lname";

$scope.change = function (id) {

console.log("My Tag: " + id + " : " + $scope.fname);

//$scope.fname = "mytab change method";

}

}

}

});

<body ng-app="myApp" style="font-size: 16px;">

Root: <input type="text" ng-model="fname" /><br />

<div ng-controller="cust0">

Cust0: <input type="text" ng-model="fname" /><br />

<div ng-controller="cust1">

Cust1: <input type="text" ng-model="fname" /><br />

MyTag: <mytag myclick="change()" mytest="fname" rid=300> - 使用函数时一定要带（）

<input id="hi100" ng-model="fname" value="Hello World" /> :

<span ng-bind="fname"></span>

</mytag><br />

</div>

</div>

</body>

例一：可以

app.directive("mytag", function () {

return {

scope: {

myclick:"&ngClick"

},

restrict: "AEC",

replace: true,

transclude: false,

template: '<span><input type="button" ng-click="myclick()" value="click" /></span>'

}

}

});

MyTag: <mytag ng-click="change()" mytest="fname" rid=300></mytag><br />

因为 template 有<span> 包裹， 所以可以， 但是 <span> 也会有click 事件

例二：出错不可以：

app.directive("mytag", function () {

return {

scope: {

myclick:"&ngClick"

},

restrict: "AEC",

replace: true,

transclude: false,

template: '<input type="button" ng-click="myclick()" value="click" />'

}

});

MyTag: <mytag ng-click="change()" mytest="fname" rid=300></mytag><br />

原因主要是： 合并 ng-click 属性出错

<input class="ng-isolate-scope" type="button" value="click"

ng-click="change() myclick()"  - 调用两函数

rid="300" mytest="fname">

例三：可以：

app.directive("mytag", function () {

return {

scope: {

myclick:"&ngClick"

},

restrict: "AEC",

replace: true,

transclude: false,

template: '<input type="button" ng-click="myclick();" value="click" />',

controller: function ($scope, $element, $rootScope) {

$scope.new\_name = "mytab new name";

$scope.fname = "mytab fname";

$scope.change = function () {

console.log("My Tag: " + 200 + " : " + $scope.fname);

}

}

}

});

<body ng-app="myApp" style="font-size: 16px;">

Root: <input type="text" ng-model="fname" /><br />

<div ng-controller="cust0">

Cust0: <input type="text" ng-model="fname" /><br />

<div ng-controller="cust1">

Cust1: <input type="text" ng-model="fname" /><br />

MyTag: <mytag ng-click="change();" mytest="fname" rid=300></mytag><br />

</div>

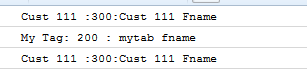
</div>

</body>

<input class="ng-isolate-scope" type="button" value="click"

ng-click="change(); myclick();"  - 执行两个函数

rid="300" mytest="fname">



例四：可以：

app.directive("mytag", function () {

return {

scope: {

myclick: "&ngClick"

},

restrict: "AEC",

replace: true,

transclude: false,

template: '<input type="button" ng-click="myclick();" value="click" />',

controller: function ($scope, $element, $rootScope) {

$scope.new\_name = "mytab new name";

$scope.fname = "mytab fname";

$scope.myclick = function () { - 会覆盖引用的函数

console.log("mytag myclick");

}

$scope.change = function () {

console.log("My Tag: " + 200 + " : " + $scope.fname);

}

}

}

});

<body ng-app="myApp" style="font-size: 16px;">

Root: <input type="text" ng-model="fname" /><br />

<div ng-controller="cust0">

Cust0: <input type="text" ng-model="fname" /><br />

<div ng-controller="cust1">

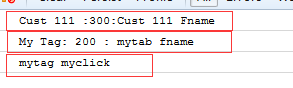
Cust1: <input type="text" ng-model="fname" /><br />

MyTag: <mytag ng-click="change();" mytest="fname" rid=300></mytag><br />

</div>

</div>

</body>



例五：可以：

app.directive("mytag", function () {

return {

scope: {

myclick: "&okclick"

},

restrict: "AEC",

replace: true,

transclude: false,

template: '<input type="button" ng-click="myclick()" value="click" />',

controller: function ($scope, $element, $rootScope) {

$scope.new\_name = "mytab new name";

$scope.fname = "mytab fname";

$scope.myclick = function () { - 会覆盖引用的函数

console.log("mytag myclick");

}

$scope.change = function () {

console.log("My Tag: " + 200 + " : " + $scope.fname);

}

}

}

});

<body ng-app="myApp" style="font-size: 16px;">

Root: <input type="text" ng-model="fname" /><br />

<div ng-controller="cust0">

Cust0: <input type="text" ng-model="fname" /><br />

<div ng-controller="cust1">

Cust1: <input type="text" ng-model="fname" /><br />

MyTag: <mytag okclick="change()" mytest="fname" rid=300></mytag><br />

</div>

</div>

</body>



如果没有在controller里定义 myclick 函数，则执行父域的函数



例六：如何设计带参数的：

app.controller("cust1", function ($scope) {

$scope.fname = "Cust 111 Fname";

$scope.change = function (id) {

console.log("Cust Change Function :" + id);

}

});

app.directive("mytag", function () {

return {

scope: {

myclick: "&okclick",

new\_name: "=param"

},

restrict: "AEC",

replace: true,

transclude: false,

template: '<input type="button" ng-click="myclick(new\_name)" value="click" />',

controller: function ($scope, $element, $rootScope) {

$scope.new\_name = "mytab new name";

$scope.fname = "mytab fname";

$scope.change = function () {

console.log("My Tag: " + 200 + " : " + $scope.fname);

}

}

}

});

<body ng-app="myApp" style="font-size: 16px;">

Root: <input type="text" ng-model="fname" /><br />

<div ng-controller="cust0">

Cust0: <input type="text" ng-model="fname" /><br />

<div ng-controller="cust1">

Cust1: <input type="text" ng-model="fname" /><br />

MyTag: <mytag okclick="change(fname)" param="fname" mytest="fname" rid=300></mytag>

</div>

</div>

</body>

例七：如何设计带参数的：隔离域的函数调用

app.directive("mytag", function () {

return {

scope: {

myclick: "&okclick"

},

restrict: "AEC",

replace: true,

transclude: false,

template: '<span>

<input type="button" ng-click="myclick(myparam)" value="click" />

<input ng-model="myparam">

</span>',

controller: function ($scope, $element, $rootScope) {

$scope.new\_name = "mytab new name";

$scope.fname = "mytab fname";

$scope.myclick = function (id) { - 会覆盖引用的函数

console.log("mytag myclick : " + id);

}

$scope.change = function () {

console.log("My Tag: " + 200 + " : " + $scope.fname);

}

}

}

});

<body ng-app="myApp" style="font-size: 16px;">

Root: <input type="text" ng-model="fname" /><br />

<div ng-controller="cust0">

Cust0: <input type="text" ng-model="fname" /><br />

<div ng-controller="cust1">

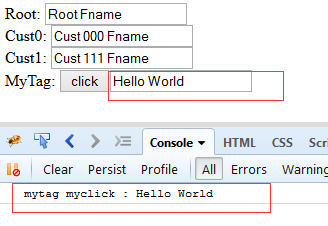
Cust1: <input type="text" ng-model="fname" /><br />

MyTag: <mytag okclick="change()" rid=300></mytag><br />

</div>

</div>

</body>



例八：如何设计带参数的：调用父域的带参数的函数

app.directive("mytag", function () {

return {

scope: {

myclick: "&okclick", - 映射函数

myparam: "=param" - 映射参数

},

restrict: "AEC",

replace: true,

transclude: false,

template: '<span>

<input type="button" ng-click="myclick(myparam)" value="click" />

<input ng-model="myparam">

</span>',

controller: function ($scope, $element, $rootScope) {

$scope.new\_name = "mytab new name";

$scope.fname = "mytab fname";

//$scope.myclick = function (id) { - 不能定义同名函数， 否则覆盖

// console.log("mytag myclick : " + id);

//}

$scope.change = function () {

console.log("My Tag: " + 200 + " : " + $scope.fname);

}

}

}

});

<body ng-app="myApp" style="font-size: 16px;">

Root: <input type="text" ng-model="fname" /><br />

<div ng-controller="cust0">

Cust0: <input type="text" ng-model="fname" /><br />

<div ng-controller="cust1">

Cust1: <input type="text" ng-model="p1" /><br />

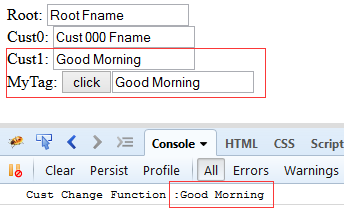
MyTag: <mytag okclick="change(p1)" param="p1"></mytag><br />

- 需要定义一个 $scope变量来作为函数的参数， 并且需要directive映射到此变量

</div>

</div>

</body>



如果这样会出错

MyTag: <mytag myclick="change({{fname}})" mytest="fname" rid=300></mytag><br />



我们讨论 scope : true 和 {} 的区别

1） scope : true 是创建新的子scope , 其值继承于 父scope. 父值单向改变子值， 子值不改变父值

controller($scope) 修改子值，子值如果被其他方式修改则， 父值对子值的单向联系终止。

2) scope : {} 是创建isolateScope , 不继承父，取决于 controller($scope) 的定义， 没有定义则 undefined

* **controller:**

controller: function controllerConstructor($scope, $element, $attrs, $transclude)

controller: function controllerConstructor($scope, $rootScope)

比较常见可以带四个注入项目: 名称不能写错，是注入，而不是javascript 函数参数。顺序可以随便

它是一个构造器函数，将来可以构造出一个实例传给引用它的指令。为什么叫controller（控制器）呢？其实就是告诉引用它的指令，你可以控制 我。至于可以控制那些东西呢，就需要在函数体中进行定义了。先看controller可以使用的参数，作用域、节点、节点的属性、节点内容的迁移，这些都 可以通过依赖注入被传进来，所以你可以根据需要只写要用的参数。关于如何对外暴露接口，我们在下面的例子来说明。

controller: 在 compile 执行完之后， 在link函数执行之前执行。

app.directive("mytag", function () {

return {

scope: true,

restrict: "AEC",

replace: true,

transclude: true,

template: '<span>Scope: <input ng-model="fname" /> : {{fname}}<br><span ng-transclude></span></span>',

controller: function ($rootScope, $scope, $element, $attrs) {

$attrs.uid = 1200;

$attrs.$set("uid", $rootScope.fname);

$("\*", $element).attr("eeid", "").attr("eeid", $attrs.uid);

$scope.new\_name = "mytab new name";

$scope.fname = "mytab fname";

$scope.change = function () {

console.log("My Tag: " + 200 + " : " + $scope.fname);

}

}

}

});

MyTag: <mytag sid=800 uid=990>

Trans: <input type="text" ng-model="fname" />

<input type="button" value="click" ng-click="change()" />

</mytag>

<div class="ng-scope ng-binding" uid="Root Fname" sid="800">

Scope:

<input class="ng-pristine ng-untouched ng-valid" ng-model="fname" eeid="Root Fname">

: mytab fname

<br eeid="Root Fname">

<span ng-transclude="" eeid="Root Fname">

<span class="ng-scope"> Trans: </span>

<input class="ng-pristine ng-untouched ng-valid ng-scope" type="text" ng-model="fname">

<input class="ng-scope" type="button" ng-click="change()" value="click">

</span>

</div>

* **compile: function(element, attributes)**

**compile后运行，link不运行(link就是compile中的postLink)**

**compile( element, attributes) 没有 scope**

**主要的目的:**

1. 可以在compile 函数里操作数据绑定，修改数据绑定, 修改属性和CSS 等
2. 但是由于不能操作**scope,** 所以也就不能对数据进行操作

app.directive("mytext", function(){

return {

restrict: "AEC",

replace: true,

transclude: false,

templateUrl: "mylink.ui",

compile : function(el, atr) {

$("textarea", el).attr("ng-model", "area"); 更改数据绑定

$("input", el).attr("ng-model", "hotel");

$("input", el).css("color", "red"); 修改 css

$(el).bind("mouseenter", function(ev) { - 操作DOM元素事件

$(this).css("background-color", "orange");

}).bind("mouseout", function(ev){

$(this).css("background-color", "white");

});

$(el).append("Compile coolel: <input ng-model='coolel'>{{coolel}}");-新增元素

},

link: function(sc, el, atr) { - 一旦定义compile 函数, link函数则不被执行

$("textarea", el).val("Hello world");

$("textarea", el).attr("ng-model", "area");

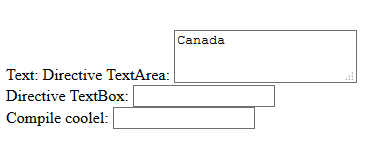
alert("link :" + $scope.area);

}

}

});

Text: <mytext></mytext>



<span class="ng-binding">

Directive TextArea:

<textarea class="ng-pristine ng-untouched ng-valid" ng-model="area"></textarea>

<br>

Directive TextBox:

<input class="ng-pristine ng-untouched ng-valid" type="text" ng-model="hotel" style="color: red;">

<br>

Compile coolel:

<input class="ng-pristine ng-untouched ng-valid" ng-model="coolel">

</span>

**directive link: function( scope, element, attributes )**

**主要的目的:**

1. 在link函数里可以修改属性和CSS ，绑定事件
2. 虽然可以修改 ng-model 属性, 但是对数据绑定将无效

app.directive("mytext", function(){

return {

restrict: "AEC",

replace: true,

transclude: false,

templateUrl: "mylink.ui",

link: function(sc, el, atr) {

$("textarea", el).attr("ng-model", "area"); 虽然属性被改变了, 但是并没有实现双向绑定

$("textarea", el).val("Hello world"); - 可以赋值

$("textarea", el).attr("ng-model", "area");

$("textarea", el).css("color", "orange");

$(el).bind("mouseenter", function(ev){ - 绑定事件

$(this).css("background-color", "blue");

}).bind("mouseout", function(ev) {

$(this).css("background-color", "orange");

});

alert("link :" + sc.area); - 可以操作scope

}

}

});

**directive compile & link:同时运行**

1. 只能在compile 函数里 返回 link函数; 来执行link

app.directive("mytext", function(){

return {

restrict: "AEC",

replace: true,

transclude: false,

templateUrl: "mylink.ui",

compile: function(el, atr) {

$("textarea", el).attr("ng-model", "area"); - 双向数据绑定

return function(sc, el, atr) {

$("textarea", el).val("Hello world"); - 赋值无效,有数据绑定

$(el).bind("mouseenter", function(ev){ - 绑定事件

$(this).css("background-color", "blue");

}).bind("mouseout", function(ev) {

$(this).css("background-color", "orange");

});

$("input", el).val(sc.schname); - 可以操作scope

sc.area = "Greate Vancouver"; - 可以更改 scope.area 并反馈到绑定元素

}

}

}

<span class="ng-binding" style="background-color: orange;">

Directive TextArea:

<textarea class="ng-valid ng-dirty ng-valid-parse ng-touched" ng-model="area"></textarea>

<br>

Directive TextBox:

<input type="text">

</span>

对于操作：

attrs : 获取值 attrs["attribute\_name"] , 设置值 attrs.$set("attr\_name", value);

element: 实际对应的是 template, 或者是 templateURL 的内容， 不包括 ng-transclude的嵌入内容

注意： replace: false/true element 的内容会不同，主要是外层是否被替换掉。

--------------------------------------------------------------------------------------------------------------------------

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)

--------------------------------------------------------------------------------------------------------------------------

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）

**directive compile 细分两个阶段函数: pre post**

compile: function(element, attributes) {

            return {

                pre: function preLink(scope, element, attributes) {

                },

                post: function postLink(scope, element, attributes) {

                }

            };

        }

app.directive("mytext", function(){

return {

restrict: "AEC",

replace: true,

transclude: false,

templateUrl: "mylink.ui",

compile: function(el, atr) {

$("textarea", el).attr("ng-model", "area");

return {

pre: function (sc, el, atr) {

$("textarea", el).val("Hello world");

$(el).bind("mouseenter", function(ev){

$(this).css("background-color", "blue");

}).bind("mouseout", function(ev) {

$(this).css("background-color", "orange");

});

$("input", el).val(sc.schname);

sc.area = "Greate Vancouver";

console.log("Pre: " + sc.area);

},

post: function (sc, el, atr) {

sc.area = "Greate Of China";

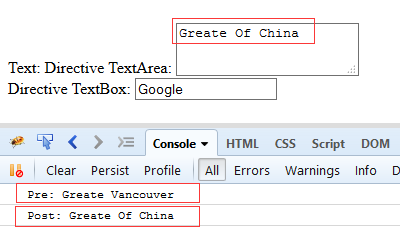
console.log("Post: " + sc.area);

}

}

}

}



**directive require :**

**使用controller和require进行指令间通信**

使用指令来定义一个ui组件是个不错的想法，首先使用起来方便，只需要一个标签或者属性就可以了，其次是可复用性高，通过controller 可以动态控制ui组件的内容，而且拥有双向绑定的能力。当我们想做的组件稍微复杂一点，就不是一个指令可以搞定的了，就需要指令与指令的协作才可以完成， 这就需要进行指令间通信。

想一下我们进行模块化开发的时候的原理，一个模块暴露（exports）对外的接口，另外一个模块引用（require）它，便可以使用它所提供的服务了。ng的指令间协作也是这个原理，这也正是自定义指令时controller参数和require参数的作用。

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

controller参数用于定义指令对外提供的接口，它的写法如下：

controller: function controllerConstructor($scope, $element, $attrs, $transclude)

它是一个构造器函数，将来可以构造出一个实例传给引用它的指令。为什么叫controller（控制器）呢？其实就是告诉引用它的指令，你可以控制 我。至于可以控制那些东西呢，就需要在函数体中进行定义了。先看controller可以使用的参数，作用域、节点、节点的属性、节点内容的迁移，这些都 可以通过依赖注入被传进来，所以你可以根据需要只写要用的参数。关于如何对外暴露接口，我们在下面的例子来说明。

require参数便是用来指明需要依赖的其他指令，它的值是一个字符串，就是所依赖的指令的名字，这样框架就能按照你指定的名字来从对应的指 令上面寻找定义好的controller了。不过还稍稍有点特别的地方，为了让框架寻找的时候更轻松些，我们可以在名字前面加个小小的前缀：^，表示从父 节点上寻找，使用起来像这样：require : ‘^directiveName’，如果不加，$compile服务只会从节点本身寻找。另外还可以使用前缀：？，此前缀将告诉$compile服务，如 果所需的controller没找到，不要抛出异常。

所需要了解的知识点就这些，接下来是例子时间，依旧是从书上抄来的一个例子，我们要做的是一个手风琴菜单，就是多个折叠菜单并列在一起，此例子用来展示指令间的通信再合适不过。

首先我们需要定义外层的一个结构，起名为accordion，代码如下：

http://img1.tuicool.com/7JJJry.gif

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

return {

restrict : 'E',

template : '<div ng-transclude></div>',

replace : true,

transclude : true,

controller :function(){

var expanders = [];

this.gotOpended = function(selectedExpander){

angular.forEach(expanders,function(e){

if(selectedExpander != e){

e.showText = false;

}

});

}

this.addExpander = function(e){

expanders.push(e);

}

}

}

});

View Code

需要解释的只有controller中的代码，我们定义了一个折叠菜单数组expanders，并且通过this关键字来对外暴露接口，提供两 个方法。gotOpended接受一个selectExpander参数用来修改数组中对应expander的showText属性值，从而实现对各个子 菜单的显隐控制。addExpander方法对外提供向expanders数组增加元素的接口，这样在子菜单的指令中，便可以调用它把自身加入到 accordion中。

看一下我们的expander需要做怎样的修改呢：

http://img1.tuicool.com/7JJJry.gif

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

return {

restrict : 'E',

templateUrl : 'expanderTemp.html',

replace : true,

transclude : true,

require : '^?accordion',

scope : {

title : '=etitle'

},

link : function(scope,element,attris,accordionController){

scope.showText = false;

accordionController.addExpander(scope);

scope.toggleText = function(){

scope.showText = ! scope.showText;

accordionController.gotOpended(scope);

}

}

};

});

View Code

首先使用require参数引入所需的accordion指令，添加?^前缀表示从父节点查找并且失败后不抛出异常。然后便可以在link函数 中使用已经注入好的accordionController了，调用addExpander方法将自己的作用域作为参数传入，以供 accordionController访问其属性。然后在toggleText方法中，除了要把自己的showText修改以外，还要调用 accordionController的gotOpended方法通知父层指令把其他菜单给收缩起来。

指令定义好后，我们就可以使用了，使用起来如下：

<accordion>

　　<expander ng-repeat="expander in expanders" etitle="expander.title">{{expander.text}}</expander>

</accordion>

外层使用了accordion指令，内层使用expander指令，并且在expander上用ng-repeat循环输出子菜单。请注意这里 遍历的数组expanders可不是accordion中定义的那个expanders，如果你这么认为了，说明还是对作用域不够了解。此 expanders是ng-repeat的值，它是在外层controller中的，所以，在testC中，我们需要添加如下数据：

http://img1.tuicool.com/7JJJry.gif

$scope.expanders = [

{title: '个人简介',

text: '大家好，我是一名前端工程师，我正在研究AngularJs，欢迎大家与我交流，Email:lvxiaobao\_fbi@163.com'},

{title: '我的爱好',

text: '运动类：篮球、足球、乒乓球。 电脑类：前端技术、打DOTA。 其他类：欣赏美女'},

{title: '性格及工作',

text: '追求完美主义的处女座极品男人就是我啦~严重的代码洁癖以及对垃圾代码的零容忍！希望通过自己的努力进入理想的公司工作。'}

];

View Code

这下就都全乎了，试一下我们的accordion组件是不是可以正常使用了呢：

{{expander.text}}

理解了其中的道理之后，使用起来就可以得心应手了，我也将在以后的实践中尝试编写更加复杂的组件，此小例子就当是抛砖引玉了~

**总结**

又到了总结时间，到此为止自定义指令的学习就告一段落了，但我相信相关的知识肯定远远不止这些，真正要将指令在项目中用好，还需要理解指令与ng的其他机制如何相互作用，还需更加深入的了解ng的指令机制等。所以学与用的转变还需要实践的检验。

app.directive("mycon", function () {

return {

scope: {},

transclude: "hell",

replace: false,

template: '<input ng-model="fname" />{{fname}} - {{$parent.fname}}- <span ng-transclude></span>',

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

console.log("controller");

},

compile: function (el, attrs, trans) {

console.log("compile");

return function (sc, el, atr) {

console.log("link");

}

}

}

});

执行先后顺序： compile controller link

1. 如果想实现父与子指令,子与子指令之间的通信. 理解继承原理

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

app.controller("cust0", function ($scope, $rootScope) {

$scope.fname = "Cust 000 Fname";

$rootScope.fname = "Root Fname";

$scope.change = function () {

console.log("Cust0 Change Function :" + $scope.fname);

}

});

app.controller("cust1", function ($scope) {

$scope.fname = "Cust 111 Fname";

$scope.change = function () {

console.log("Cust1 Change Function :" + $scope.fname);

}

});

app.directive("mydiv", function () {

return {

scope: {},

controller: function ($scope) {

$scope.fname = "mydiv first name";

$scope.set = function (fn) {

console.log("Before : " + $scope.fname);

$scope.fname = fn;

console.log("After : " + $scope.fname);

}

$scope.get = function () {

alert("mydiv : " + $scope.fname);

}

}

}

});

app.directive("mytag", function ($timeout) {

return {

**require**: "^mydiv",

scope: true,

restrict: "AEC",

replace: true,

transclude: true,

template: '<div>

Scope: <input ng-model="fname" /> : {{fname}}

<input type="button" value="Set" ng-click="change(fname)" />

<input type="button" value="Get" ng-click="Get()" /><br>

<span ng-transclude></span></div>',

controller: function ($rootScope, $scope, $transclude, $element, $attrs) {

}

}

});

</script>

</head>

<body ng-app="myApp" style="font-size: 16px;">

Root: <input type="text" ng-model="fname" /><br />

<div ng-controller="cust0">

Cust0: <input type="text" ng-model="fname" /><br />

<div ng-controller="cust1">

Cust1: <input type="text" ng-model="fname" /><br />

<mydiv>

MyDiv: <input type="text" ng-model="fname" /><br />

MyTag: <mytag sid=800 uid=990>

Trans: <input ng-model="fname" type="text" value="hello world" />

<input type="button" value="click" ng-click="change(fname)" />

</mytag>

<mytag sid=800 uid=990></mytag>

<mytag sid=800 uid=990></mytag>

</mydiv>

</div>

</div>

</body>

</html>

分析: 重要的原则：

1. 所有指令的scope都是相对于在 html 上最接近的 ng-controller 所定义的 $scope, 没有继续往上找
2. 所有指令里 ng-tranclude 里面的内容都会创建一个新的继承自 ng-controller的 $scope, 记住不是继承自directive 里controller的 $scope
3. 记住上面的原则是很重要的， 如果是父子指令，父指令有ng-transclude，子指令也有ng-transclude

子域仍然是 ng-controller的 $scope，但是这就要取决于子指令scope:true/false/{}

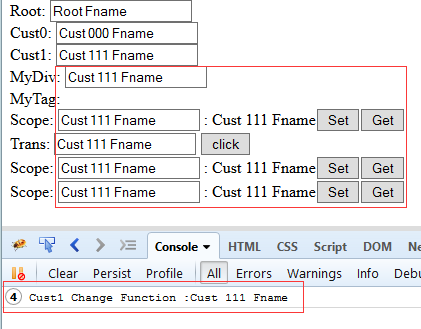
对于子指令：true/{} 因为是独立, 所以基本大家都是独立的

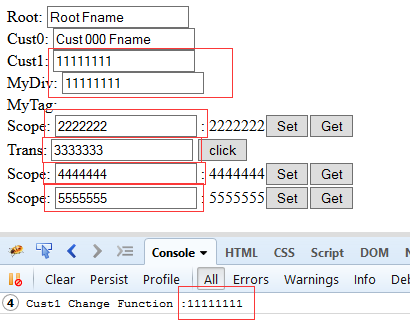
但是子指令: false; 那么prototype 由于父ng-tranclude的原因为子指令创建新的子域

所以子指令的prototype的指向这个新的子域，也就是相对于子指令的新子域了。

1. <mytag>-scope:true 虽然 <mydiv>-scope:{} 但是他所对应的scope应该是

<div ng-controller="cust1"> 所以<mytag>是继承自ng-controller="cust1"的scope的子域





看上面都是各自独立的子域

我们看看把 <mytag> 改成 scope: {}

app.directive("mytag", function ($timeout) {

return {

**require**: "^mydiv",

scope: {},

restrict: "AEC",

replace: true,

transclude: true,

template: '<div>Scope: <input ng-model="fname" /> : {{fname}}<input type="button" value="Set" ng-click="change(fname)" /><input type="button" value="Get" ng-click="Get()" /><br><span ng-transclude></span></div>',

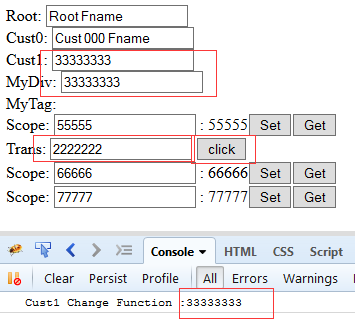
controller: function ($rootScope, $scope, $transclude, $element, $attrs) {

//$scope.fname = "mytag first name";

}

}

});



分析:

1. 只有 ng-transclude 是继承自<div ng-controller="cust1">
2. 其他都是隔离的子域名， 所以 ng-click 没有事件绑定, $scope.name也是undefined
3. 如果想实现父与子指令,子与子指令之间的通信. 理解继承原理

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

app.controller("cust0", function ($scope, $rootScope) {

$scope.fname = "Cust 000 Fname";

$rootScope.fname = "Root Fname";

$scope.change = function () {

console.log("Cust0 Change Function :" + $scope.fname);

}

});

app.controller("cust1", function ($scope) {

$scope.fname = "Cust 111 Fname";

$scope.change = function () {

console.log("Cust1 Change Function :" + $scope.fname);

}

});

app.directive("mydiv", function () {

return {

scope: {},

replace: true,

transclude:true,

template: '<div>

<input ng-model="fname">:{{fname}}

<input type="button" ng-click="change(fname)" value="change" /><br>

<span ng-transclude></span></div>',

controller: function ($scope) {

$scope.fname = "mydiv first name";

$scope.set = function (fn) {

console.log("Before : " + $scope.fname);

$scope.fname = fn;

console.log("After : " + $scope.fname);

}

$scope.get = function () {

alert("mydiv : " + $scope.fname);

}

$scope.change = function () {

console.log("mydiv Change Function :" + $scope.fname);

}

}

}

});

app.directive("mytag", function ($timeout) {

return {

**require**: "^mydiv",

scope: true,

restrict: "AEC",

replace: true,

transclude: true,

template: '<div>

Scope: <input ng-model="fname" /> : {{fname}}

<input type="button" value="Change" ng-click="change(fname)" />

<input type="button" value="Get" ng-click="Get()" /><br>

<span ng-transclude></span>

</div>',

controller: function ($rootScope, $scope, $transclude, $element, $attrs) {

//$scope.fname = "mytag first name";

}

}

});

<body ng-app="myApp" style="font-size: 16px;">

Root: <input type="text" ng-model="fname" /><br />

<div ng-controller="cust0">

Cust0: <input type="text" ng-model="fname" /><br />

<div ng-controller="cust1">

Cust1: <input type="text" ng-model="fname" /><br />

MyDiv: <mydiv>

MyTag: <mytag sid=800 uid=990>

Trans: <input ng-model="fname" type="text" />

<input type="button" value="change" ng-click="change(fname)" />

</mytag>

<mytag sid=800 uid=990></mytag>

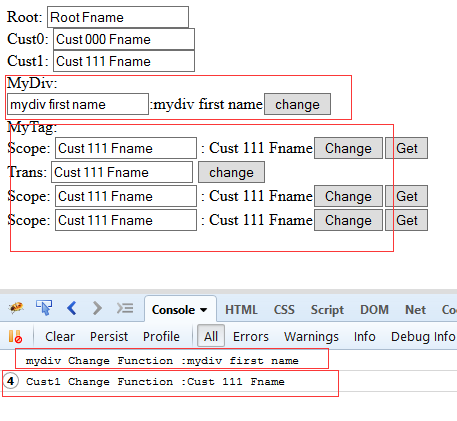
<mytag sid=800 uid=990></mytag>

</mydiv>

</div>

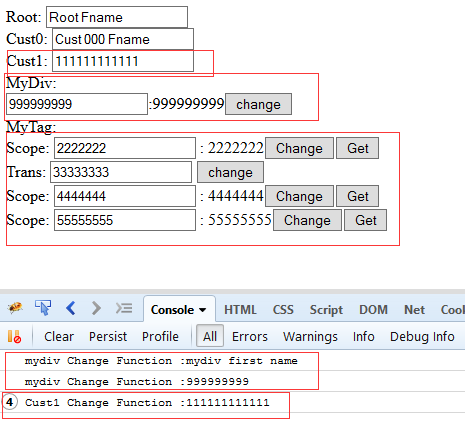
</div>

</body>



分析：

1. 虽然<mytag>的 父DOM 是<mydiv>, DOM 结构来说是父子关系, 但是 scope 的关系却不是父子关系<mytag>与<mydiv> 的scope 都属于 <div ng-controller="cust1"> 的scope之下， 他们都处于同一 个controller之下。
2. directive指令里定义的controller只是针对本指令的，默认情况下不对外公布。而他的$scope则,取决于 scope:false/true/{}
3. 所以 <mydiv> scope: {}, 他有自己的隔离域, 有自己的函数和自己的变量.
4. 所以 <mytag> scope: true, 继承子域.



5）对于<mytag> 的ng-transclude的内容所创建的子域，因为是html , 所以也是相对于

<div ng-controller="cust1">的

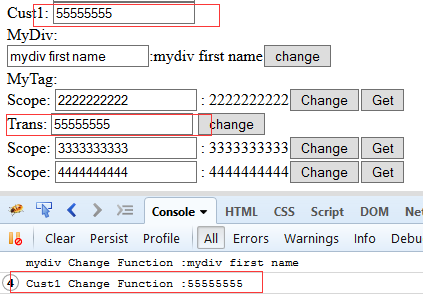
MyTag: <mytag sid=800 uid=990>

Trans: <input ng-model="fname" type="text" />

<input type="button" value="change" ng-click="change(fname)" />

</mytag>

如下图操作所示:



app.directive("mytag", function ($timeout) {

return {

**require**: "^mydiv",

scope: true,

restrict: "AEC",

replace: true,

transclude: true,

template: '<div>

Scope: <input ng-model="fname" /> : {{fname}}

<input type="button" value="Change" ng-click="change(fname)" />

<input type="button" value="Get" ng-click="Get()" /><br>

<span ng-transclude></span>

</div>',

controller: function ($rootScope, $scope, $transclude, $element, $attrs) {

$scope.change = function () {

console.log("mytag Change Function :" + $scope.fname);

}

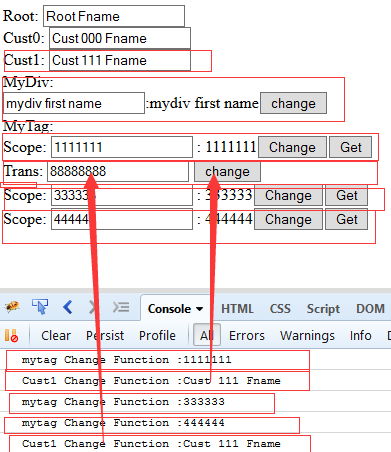
}

}

});

如果我们覆盖change函数, 函数属于子域, 不在属于父域(prototype) , 所以$scope.fname也子域的。

这里需要理解javascript this与prototype的继承原则



如果我们将 scope: false 这将会是很有趣的

app.directive("mytag", function ($timeout) {

return {

**require**: "^mydiv",

scope: false,

restrict: "AEC",

replace: true,

transclude: true,

template: '<div>

Scope: <input ng-model="fname" /> : {{fname}}

<input type="button" value="Change" ng-click="change(fname)" />

<input type="button" value="Get" ng-click="Get()" /><br>

<span ng-transclude></span>

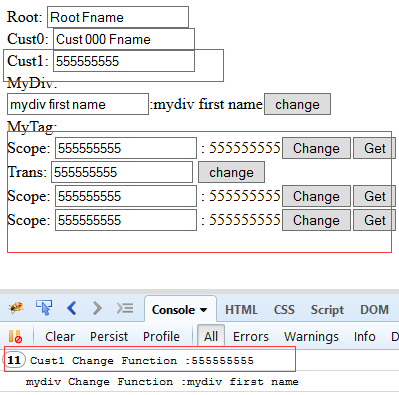
</div>',

controller: function ($rootScope, $scope, $transclude, $element, $attrs) {

}

}

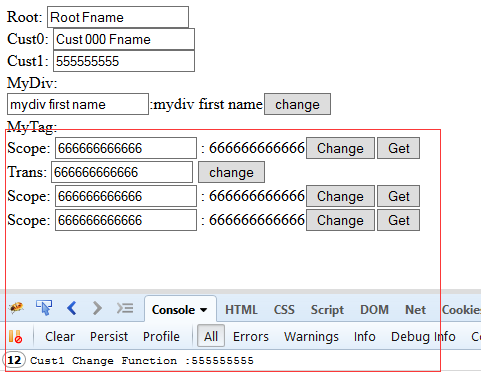
});



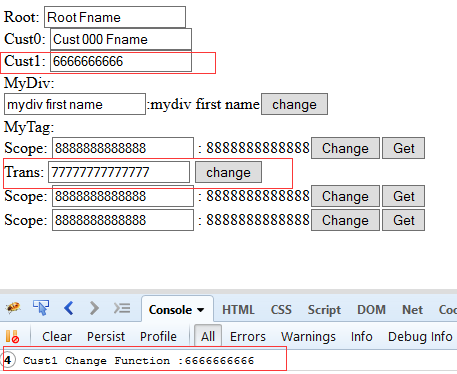
分析:

1. 因为<mytag>是scope : false 即不创建子域, 但是<mydiv> - transclude: true 所有ng-transclude里的内容都将创建子域， 该子域是继承自<div ng-controller="cust1">

如下图操作所示：



1. 由于$scope.fname 被修改，父子关系打断了， 函数change则没有修改过，所以还是使用父的

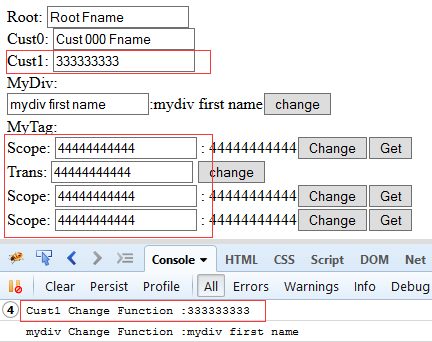


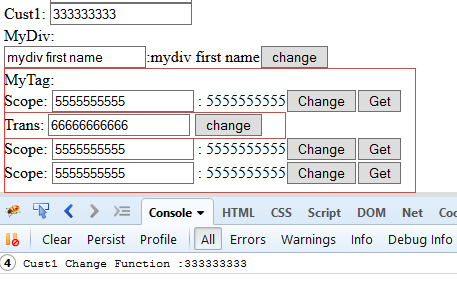
3）对于<mytag> scope:false 的ng-transclude的内容所创建的子域, 是相对于<mytag>的scope

这点跟 scope:true 不同, 如何理解呢:

scope:true 都各自创建独立的继承子域。

scope:false 理论不创建子域，但是由于transclude:true 所以就创建一个子域 给里面的内容共享.





3)如果想实现父与子指令,子与子指令之间的通信. 理解继承原理

app.controller("cust1", function ($scope) {

$scope.fname = "Cust 111 Fname";

$scope.change = function () {

console.log("Cust1 Change Function :" + $scope.fname);

}

});

app.directive("mydiv", function () {

return {

scope: false,

replace: true,

transclude: true,

template: '<div>

<input ng-model="fname">:{{fname}}

<input type="button" ng-click="change(fname)" value="change" /><br>  
<span ng-transclude></span>

</div>',

controller: function ($scope) {

$scope.fname = "mydiv first name";

$scope.set = function (fn) {

console.log("Before : " + $scope.fname);

$scope.fname = fn;

console.log("After : " + $scope.fname);

}

$scope.get = function () {

alert("mydiv : " + $scope.fname);

}

$scope.change = function () {

console.log("mydiv Change Function :" + $scope.fname);

}

}

}

});

app.directive("mydiv1", function () {

return {

scope: false,

replace: true,

transclude: true,

template: '<div>

<input ng-model="fname">:{{fname}}

<input type="button" ng-click="change(fname)" value="change" /><br>

<span ng-transclude></span>

</div>',

controller: function ($scope) {

$scope.fname = "mydiv111 first name";

$scope.set = function (fn) {

console.log("Before : " + $scope.fname);

$scope.fname = fn;

console.log("After : " + $scope.fname);

}

$scope.get = function () {

alert("mydiv : " + $scope.fname);

}

$scope.change = function () {

console.log("mydiv11 Change Function :" + $scope.fname);

}

}

}

});

<div ng-controller="cust1">

Cust1: <input type="text" ng-model="fname" />

<input type="button" ng-click="change(fname)" value="change" /><br />

<mydiv1></mydiv1>

<mydiv></mydiv>

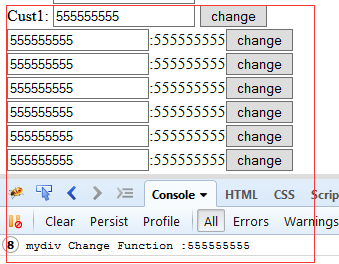
<mydiv></mydiv>

<mydiv1></mydiv1>

<mydiv1></mydiv1>

<mydiv1></mydiv1>

</div>



分析:

1. 所有共享同一个$scope, 函数与变量的覆盖，以最后一个定义为主，最后的指令会最后执行controller

4)如果想实现父与子指令,子与子指令之间的通信. 理解继承原理

* **require 与 link(**scope,element,attris,requreController**)**

实现指令之间的灵活通讯：

app.directive("mydiv", function () {

return {

scope: {},

replace: true,

transclude: true,

template: '<div>

<input ng-model="fname">:{{fname}}

<input type="button" ng-click="set(fname)" value="Set" /><br>

<span ng-transclude></span>

</div>',

controller: function ($scope) {

$scope.fname = "init value";

this.fname = "init value";

this.set = function (fn) {

console.log("This Before : " + this.fname);

this.fname = fn;

console.log("This After : " + this.fname);

console.log("Scope Before : " + $scope.fname);

$scope.fname = fn;

console.log("Scope After : " + $scope.fname);

}

this.get = function () {

alert("This mydiv : " + this.fname);

alert("Scope mydiv : " + $scope.fname);

}

}

}

});

app.directive("mytag", function ($timeout) {

return {

**require**: "^mydiv",

scope: {},

restrict: "AEC",

replace: true,

transclude: true,

template: '<div>

Scope: <input ng-model="fname" /> : {{fname}}

<input type="button" value="Set" ng-click="set(fname)" />

<input type="button" value="Get" ng-click="get()" /><br>

<span ng-transclude></span>

</div>',

controller: function ($rootScope, $scope, $transclude, $element, $attrs) {

$scope.fname = "mytag first name";

},

link: function (sc, el, attrs, pCtr) {

sc.set = pCtr.set;

sc.get = pCtr.get;

}

}

});

<body ng-app="myApp" style="font-size: 16px;">

Root: <input type="text" ng-model="fname" /><br />

<div ng-controller="cust0">

Cust0: <input type="text" ng-model="fname" /><br />

<div ng-controller="cust1">

Cust1: <input type="text" ng-model="fname" /><br />

MyDiv: <mydiv>

MyTag: <mytag sid=800 uid=990>

Trans: <input ng-model="fname" type="text" />

<input type="button" value="Set" ng-click="set(fname)" />

<input type="button" value="Get" ng-click="get(fname)" />

</mytag>

<mytag sid=800 uid=990></mytag>

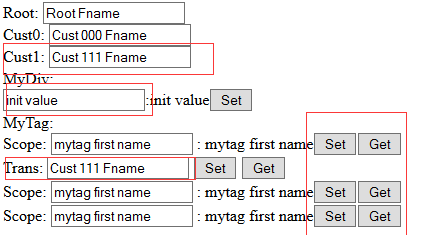
<mytag sid=800 uid=990></mytag>

</mydiv>

</div>

</div>

</body>



分析:

1. require 指令以后, 该指令的controller 就是一个class 暴露给 link 函数每个指令都初始化一个controller的实例。
2. 在controller实例里， this.xxxx 就是实例的变量与方法， 和javascript是一样的
3. 在controller实例里，$scope.xxx 变量是全局变量, 各个实例间可以共享
4. $scope.xxx = function() {} 函数是不会暴露给实例使用, 在实例里也看不到这些方法, 如下输出实例

只能看见 this.xxx 定义的东西。



1. 我们可以借助 this 来操作 $scope
2. 各个实例的保存各自的 this.xxxx 的值, 各个实例共享使用 $scope.xxxx 的值

* **controller 没有注入$scope**

<script language="javascript" type="text/javascript">

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

app.controller("customer", mycontrl);

function mycontrl() {

this.qty = 0;

this.cost = 0;

this.total = function total() {

return this.qty \* this.cost;

}

this.pay = function pay1() {

alert(this.total());

}

this.init = function (a, b) {

this.qty = a;

this.cost = b;

}

}

mycontrl.prototype.name = "William Liu";

</script>

</head>

<body ng-app="myApp" style="font-size: 16px;">

<div ng-controller="customer as cust" ng-init="cust.init(20,30)">

Scope: <input ng-model="name" />:{{name}}<br />

<input ng-model="cust.name" />:{{cust.name}}<br />

<input ng-model="cust.qty" />:{{cust.qty}}<br />

<input ng-model="cust.cost" />:{{cust.cost}}<br />

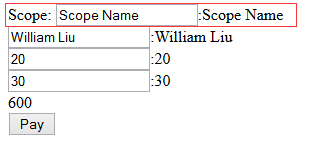
{{cust.total()}}

<br />

<button ng-click="cust.pay()">Pay</button>

</div>

</body>



分析：

1）ng-controller="customer as cust" 相当于 var cust = new customer();

2）可以定义一个构造函数给对象赋初值：ng-init="cust.init(20,30)"

3) 接下来所有的变量和方法都必须是对象： cust.qty cust.cost cust.pay()

4）虽然没有注入$scope, 如果不指定对象名, 默认是$scope

Scope: <input ng-model="name" />:{{name}}

function mycontrl($scope) { -- 但是还是需要注入$scope 否则无法访问

this.pay = function pay1() {

alert(this.total() + " name:" + $scope.name);

}

}

* **service, factory, provider**

**factory 方式与 javascript 工厂方式创建对象是一样的。**

factory()让我们通过返回一个包含service方法和数据的对象来定义一个service。在service方法里面我们可以注入services，比如 $http 和 $q等。

* **什么时候使用factory()方法**

在service里面当我们仅仅需要的是一个方法和数据的集合且不需要处理复杂的逻辑的时候，factory()是一个非常不错的选择。**注意：需要使用.config()来配置service的时候不能使用factory()方法**

app.factory("myfact", function () {

var name = "var\_name myfact";

var service = {

name: "myfact service",

get0: function () {

return name;

},

get1: function () {

return service.name;

},

get2: function () {

return this.name;

},

show: function () {

return ("[Factory var]: " + name + " [Service]:" + service.name + " [This]: " + this.name);

}

};

return service;

});

app.controller("customer", ["$scope", "myfact", function ($scope, mytt) {

$scope.get0 = mytt.get0;

$scope.get1 = mytt.get1;

$scope.get2 = mytt.get2;

$scope.show = mytt.show;

$scope.show1 = function () {

return mytt.get2();

}

} ]);

<div ng-controller="customer">

Scope: <input ng-model="name" />:{{name}}<br />

Var : {{ get0() }}<br />

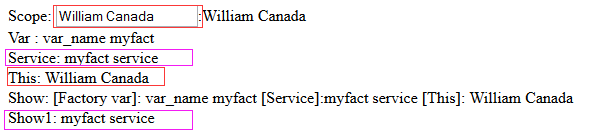
Service: {{ get1() }}<br />

This: {{ get2() }}<br />

Show: {{ show() }}<br />

Show1: {{ show1() }}<br />

</div>



分析：

1）理解 函数里面 this 的含义， this是根据函数所处的上下文对象而确定的。

例如： $scope.get2 = mytt.get2;

get2: function () {

return this.name;

}

This: {{ get2() }}<br />

此时的this 是指 $scope, 而this.name是指 $scope.name

2) 在html里的指令所对应的业务逻辑只和它所对应的controller联系， 而且是和对应controller默认的，或者注入

的$scope 进行交互。 如果controller注入了其他对象如factory， 也是必须通过$scope来间接交互。

$scope.get0 = mytt.get0;

$scope.get1 = mytt.get1;

$scope.get2 = mytt.get2;

$scope.show = mytt.show;

$scope.show1 = function () {

return mytt.get2();

}

也可以做很多事情：

$scope.name = mytt.name;

mytt.show2 = function () {

return $scope.name;

}

$scope.show2 = mytt.show2;

// html

Show2: {{show2()}}<br />

mytt.name 如此赋值 this.name 和 mytt.name 是相同的

mytt.show2 = function () {

return mytt.name = "mytt+" + $scope.name;

}

如此处理会进入死循环， 最终出错

mytt.show2 = function () {

return this.name = "mytt+" + $scope.name;

}

$scope.show2 = mytt.show2;

因为 this.name = $scope.name $scope.name = "mytt+" + $scope.name;

3) service = {} return service 这是对象， 只创建一个对象，当被注入到不同的controller时， 是属于传引用， 而不是传递值。

所以如果考虑controller之间共享数据， 可以考虑使用属于对象的属性如：service.name

app.factory("myfact", function () {

var name = "var\_name myfact";

var service = {

name: "myfact service",

get0: function () {

return name;

},

get1: function () {

return service.name;

},

get2: function () {

return this.name;

},

show: function () {

return ("[Factory var]: " + name + " [Service]:" + service.name + " [This]: " + this.name);

},

set: function (a) {

service.name = a; -- 此处使用对象变量来共享信息

}

};

return service;

app.controller("customer", ["$scope", "myfact", "$rootScope", function ($scope, mytt, rt) {

$scope.get0 = mytt.get0;

$scope.get1 = mytt.get1;

$scope.get2 = mytt.get2;

$scope.show = mytt.show;

$scope.set = mytt.set;

} ]);

app.controller("sale", ["$scope", "myfact", function (sc, myss) {

sc.get0 = myss.get0;

sc.get1 = myss.get1;

sc.get2 = myss.get2;

sc.show = myss.show;

sc.set = myss.set;

} ]);

<body ng-app="myApp" ng-controller="newone" style="font-size: 16px;">

<div ng-controller="customer">

Scope: <input ng-model="name" />:{{name}}<br />

Var : {{get0()}}<br />

Service: {{get1()}}<br />

This: {{get2()}}<br />

Show: {{show()}}<br />

<br />

<button ng-click="pay()">Pay</button>

<button ng-click="set('OK' + name)">Set</button>

<button ng-click="show2()">show</button>

</div>

<br />

<div ng-controller="sale">

Scope: <input ng-model="name" />:{{name}}<br />

Var : {{get0()}}<br />

Service: {{get1()}}<br />

This: {{get2()}}<br />

Show: {{show()}}<br />

<br />

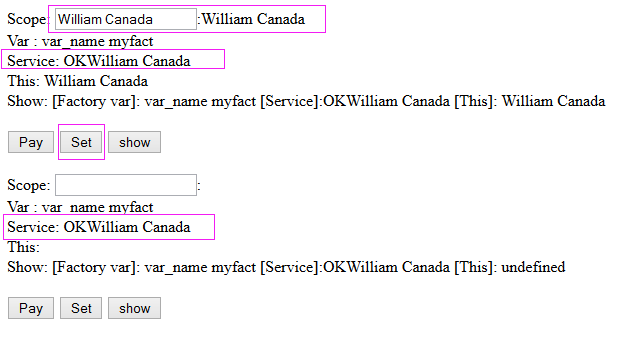
<button ng-click="pay()">Pay</button>

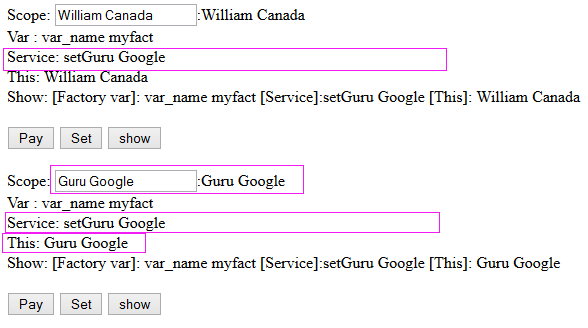
<button ng-click="set('set' + name)">Set</button>

<button ng-click="show2()">show</button>

</div>

</body>





同样的道理： 也可以使用在 factory里定义的公共变量 var name

app.factory("myfact", function () {

var name = "var\_name myfact";

var service = {

name: "myfact service",

get0: function () {

return name;

},

get1: function () {

return service.name;

},

get2: function () {

return this.name;

},

show: function () {

return ("[Factory var]: " + name + " [Service]:" + service.name + " [This]: " + this.name);

},

set: function (a) {

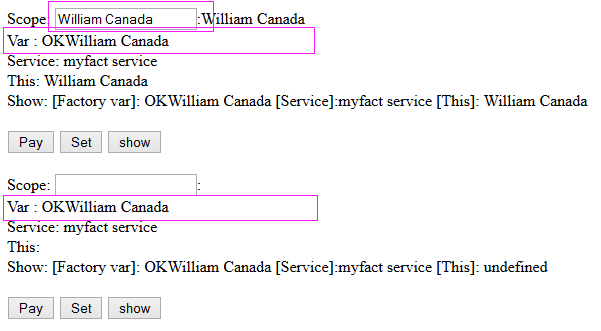
name = a; -- 此处使用公共变量来共享信息

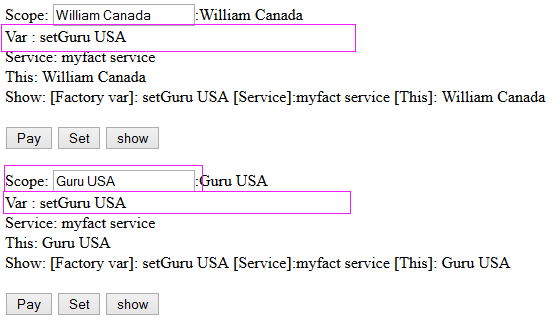
}

};

return service;

});





4) factory语法上可以灵活使用不同写法。 如下

app.factory("myfact", function () {

var name = "var\_name myfact";

var service = {};

service.name = "myfact service";

service.get0 = function () {

return name;

}

service.get1 = function () {

return service.name;

};

service.get2 = function () {

return this.name;

};

service.show = function () {

return ("[Factory var]: " + name + " [Service]:" + service.name + " [This]: " + this.name);

}

service.set = function (a) {

name = a;

}

return service; - 关键是要返回对象，这是实实在在的对象， 不需要 new 初始化

});

是大家共享这个对象， 所以 service.xxxx 变量和 factory var 定义的变量都可以用来共享使用。

需要注意的是： 函数里面的 this 关键字，是指向不同的对象， 并不一定是指向service， 有可能指向其他的对象。 就看函数指针是给哪个对象调用. 如：

$scope.get0 = mytt.get0;

$scope.get1 = mytt.get1;

$scope.get2 = mytt.get2;

$scope.show = mytt.show;

$scope.set = mytt.set;

另外函数里，是可以使用 factory var 和 service.xxxx 变量的。函数并不是复制给对象， 而且相当于 call() apply(); 指针调用。

**service 方式与 javascript 构造函数创建对象是一样的。**

* **什么时候适合使用service()方法**

service()方法很适合使用在功能控制比较多的service里面

**注意：需要使用.config()来配置service的时候不能使用service()方法**

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

app.controller("newone", function ($scope, $rootScope) {

$rootScope.name = "root name";

});

app.service("mysrv", function () {

var name = "var\_name myfact";

this.name = "myfact service";

this.get0 = function () {

return name;

}

this.get1 = function () {

return this.name;

};

this.set = function (a) {

name = a;

}

this.set1 = function (a) {

this.name = a;

}

});

app.controller("customer", ["$scope", "mysrv", "$rootScope", function ($scope, mytt, rt) {

mytt.name = "customer1111 name";

$scope.get0 = mytt.get0;

$scope.get1 = mytt.get1;

$scope.show = function () {

return "{scope}: " + $scope.name + " {service}: " + mytt.name + " {srv var}: " + mytt.get0();

};

$scope.set = mytt.set;

$scope.set1 = mytt.set1;

$scope.set2 = function (a) {

mytt.name = a;

}

} ]);

app.controller("sale", ["$scope", "mysrv", function (sc, myss) {

myss.name = "customer222 name";

sc.get0 = myss.get0;

sc.get1 = myss.get1;

sc.show = function () {

return "{scope}: " + sc.name + " {service}: " + myss.name + " {srv var}: " + myss.get0();

};

sc.set = myss.set;

sc.set1 = myss.set1;

sc.set2 = function (a) {

myss.name = a;

}

} ]);

<body ng-app="myApp" ng-controller="newone" style="font-size: 16px;">

<div ng-controller="sale">

Scope: <input ng-model="name" />:{{name}}<br />

Var : {{get0()}}<br />

this: {{get1()}}<br />

Show: {{show()}}<br />

<br />

<button ng-click="set('[Var0] ' + name)">Set Var</button>

<button ng-click="set1('[This0] ' + name)">Set1 This</button>

<button ng-click="set2('[Srv0] ' + name)">Set2 Service</button>

</div>

<div ng-controller="customer">

Scope: <input ng-model="name" />:{{name}}<br />

Var : {{get0()}}<br />

this: {{get1()}}<br />

Show: {{show()}}<br />

<br />

<button ng-click="set('[Var1] ' + name)">Set Var</button>

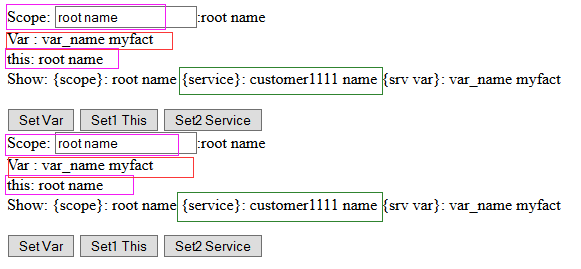
<button ng-click="set1('[This1] ' + name)">Set1 This</button>

<button ng-click="set2('[Srv1] ' + name)">Set2 Service</button>

</div>

<br />

</body>



分析：

1. sevice是通过构造函数来自动“new” 一个服务对象的， 而且只是new一个服务对象给所有的注入对象使用。

所以在service里通过 var 定义的变量， 通过 this.xxx 定义的变量都是共享使用的。

1. 理解 函数里面 this 的含义， this是根据函数所处的上下文对象而确定的。

$scope.get1 = mytt.get1;

this.get1 = function () {

return this.name; -- this是指对象$scope

};

1. 而service里的属性 this.name 则是service对象的属性，因为对象只有一个， 所以该属性是给各个controller共享使用的

service:

var name

this.set = function (a) {

name = a; -- 这是指全局变量 var name

}

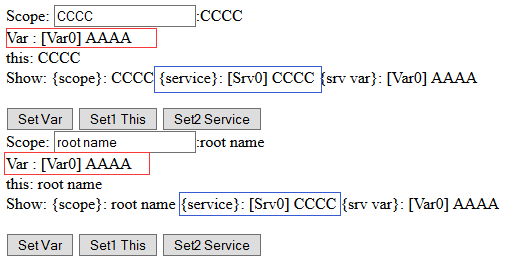
controller:

sc.set2 = function (a) {

myss.name = a; -- 这是指向对象变量 this.name

}

他们都是共享使用的， 如下图： 修改值，则会影响到所有



1. 我们可以通过将 service 对象的函数和变量 赋值给 $scope

甚至也可以直接执行 service 对象的函数和变量

需要清楚 service.func() { this.xxx } this指的是哪个对象

需要清楚 service对象的属性： myservice.xxx 指的是共用属性。

需要清楚 service里通过 var 定义的私有变量 如 var name 是不对外公开的。要想对外公开

必须在service函数里定义对这个私有变量进行操作：如

var name

this.set = function (a) {

name = a; -- 这是指全局变量 var name

}

1. 使用其他变量名来保留this的引用

service 的为了在函数里区别 this.XXX 和 service.XXX 的变量

函数里的 this 是不确定的。 为了避免混淆。可以使用 var self = this； 来保存引用

<script language="javascript" type="text/javascript">

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

app.service("iuser", function () {

var self = this;

this.name = "William Liu"; - this.name 其实就是 self.name

this.show = function () {

return "this: " + this.name + " self: " + self.name;

}

this.set = function (a) {

self.name = a;

}

});

app.controller("customer", function ($scope, iuser) {

$scope.name = iuser.name;

$scope.show = iuser.show;

$scope.set = iuser.set;

});

</script>

</head>

<body ng-app="myApp" style="font-size: 16px;">

<div ng-controller="customer">

Scope: <input ng-model="name" />{{ name }}<br />

Show: {{ show() }}<br />

<button ng-click="set(name)">Set</button>

</div>

<div ng-controller="customer">

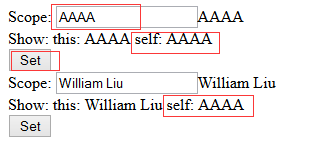
Scope: <input ng-model="name" />{{ name }}<br />

Show: {{ show() }}<br />

<button ng-click="set(name)">Set</button>

</div>

</body>



使用 self.name 即可共享给各个注入, 其结果和下面的方法是一样的。 但是封装性更好

app.controller("customer", function ($scope, iuser) {

$scope.name = iuser.name;

$scope.show = iuser.show;

$scope.set = iuser.set;

$scope.set1 = function (a) {

iuser.name = a;

}

});

例如：

function tt() {

this.t1 = "Hello";

var self = this;

var \_construct = function () {

alert("[this]: " + this.t1 + " [self]: " + self.t1);

}

\_construct();

}

var ttt = new tt();

输出：



服务可以注入其他服务：

app.service("iuser", function () {

var self = this;

this.name = "William Liu";

this.show = function () {

return "this: " + this.name + " self: " + self.name;

}

this.set = function (a) {

self.name = a;

}

});

app.service("juser", function (iuser) {

this.name = iuser.name;

this.show = iuser.show;

this.set = iuser.set;

this.set1 = function(a) {

iuser.name = a

}

});

app.controller("customer", function ($scope, juser) {

$scope.name = juser.name;

$scope.show = juser.show;

$scope.set = juser.set;

$scope.set1 = juser.set1;

});

Service 也可以这样写 ：

app.service("iuser", iuserSrv);

function iuserSrv() {

var name = "Var\_Name"; - var name 和下面的 name变量没有任何关系

}

iuserSrv.prototype.name = "Google Tom";

iuserSrv.prototype.show = function () {

return "Var: " + name + " this: " + this.name + " self: " + iuserSrv.prototype.name;

}

iuserSrv.prototype.set = function (a) {

iuserSrv.prototype.name = a;

}

**可以同时注入多个service ， factory**

app.controller("customer", ["$scope", "mysrv", "myfact", function ($scope, mytt, myff) {

mytt.name = "customer1111 name";

$scope.get0 = mytt.get0;

$scope.get1 = mytt.get1;

$scope.get1 = myff.get1;

$scope.show = function () {

return "{scope}: " + $scope.name + " {service}: " + myff.name + " {srv var}: " + mytt.get0();

};

$scope.set = myff.set;

$scope.set1 = mytt.set1;

$scope.set2 = function (a) {

mytt.name = a;

}

} ]);

**service ， factory 的区别：**

1. factory 需要直接返回服务对象， 也是共用的对象， 为所有的注入共享一个对象。
2. service 是使用 this 来定义， 不需要return对象, 第一次注入时将会 “new”一个对象，以后这个对象就可以给所有的注入使用。 是大家共享这个对象， 不会为每个注入创建独立的对象。

3）其实就是同javascript定义对象的方法一样： 工厂方式和构造函数方式。 一样的

我们要从Dependency Injection(依赖注入)的对象中获取的数据或者功能，都是Injector给的。

Injector会创建两种对象：**服务** 或 **专用对象**

Injector要知道如何创建这些对象，就要用户自行去“注册”。

有五种注册方法：

Provider, Value, Factory, Service 和 Constant

* **provider**

provider()是创建service最底层的方式，这也是唯一一个可以使用.config()方法配置创建service的方法

不像上面提到的方法那样，我们在定义的this.$get()方法里面进行依赖注入

* **什么时候使用provider()方法**

1. 当我们希望在应用开始前对service进行配置的时候就需要使用到provider()。比如，我们需要配置services在不同的部署环境里面（开发，演示，生产）使用不同的后端处理的时候就可以使用到了
2. 当我们打算发布开源provider()也是首选创建service的方法，这样就可以使用配置的方式来配置services而不是将配置数据硬编码写到代码里面。

app.config(function (**puserProvider**) { - 根据名称约定 puser + Provider(必须加上)

puserProvider.country = "JSON";

});

app.config(["puserProvider", function (pu) { - 可以使用匿名

pu.country = "JSON";

} ]);

app.provider("**puser**", function () {

var self = this;

var country = "America";

this.country = "Canada";

- 只要是使用 this定义的属性方法，就可以在config里的注入provider对象使用

- var定义的变量，是私有的变量, 在config里注入的provider对象是不可见的

- this是个环境变量，为了保证原有的引用，所以需要使用别名 var self = this

this.$get = function () { -- 定义在 this.$get函数里

---这部分是service的对象定义, 必须是工厂方式，必须return返回service对象-----------

var country = "Mexico";

var service = {

country: "China",

show: function () {

return "Country:

[this: " + this.country + "]

[self: " + self.country + "]

[var: " + country + "]";

},

set: function (a) {

this.country = a;

},

set1: function (a) {

self.country = a;

},

set2: function (a) {

country = a;

}

}

return service;

---------------------------------------------------------------------------------

}

});

<div ng-controller="sale">

Scope: <input ng-model="country" />{{ country }}<br />

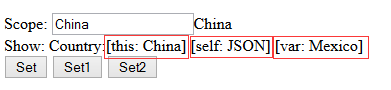
Show: {{ show() }}<br />

<button ng-click="set(country)">Set</button>

<button ng-click="set1(country)">Set1</button>

<button ng-click="set2(country)">Set2</button>

</div>



分析：

1) provider 本身是对象函数， 分为两部分组成：

第一部分: 用于给app.config 的provider对象使用

实际代码相当于：

var **puser** = function() {

var self = this;

var country = "America";

this.country = "Canada";

}

所以只有 this.xxx 的对象方法会暴露给对象。

第二部分: 用于定义service 注入，使用 this.$get 定义的函数就是用来返回service注入对象的。

必须是对象 return serviceObj

关于写法可以是多种形式的：

1) 外面定义， provider里实例化成对象，然后return

app.config(function (**puserProvider**) {

puserProvider.country = "JSON";

});

app.provider("puser", function () {

var self = this;

var country = "America";

this.country = "Canada";

this.$get = function () {

var country = "Mexico";

var service = new iService(this.country); - 必须是返回对象

-- this.country 对应的是puser的this.country所以可以被 config的provider对象访问修改

return service; - 必须return 对象

}

});

var iService = function (a) {

this.country = a;

}

iService.prototype.show = function () {

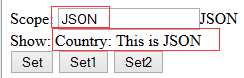
return "Country: This is " + this.country;

}

iService.prototype.init = function (a) {

this.country = a;

}



2） 直接使用 this来定义， 直接return this；这样return的就是 puser 整个对象

app.provider("puser", function () {

var self = this;

var country = "America";

this.country = "Canada";

this.city = "Burnaby";

this.showw = function () {

return "showw : " + this.country;

}

this.$get = function () {

var country = "Mexico";

this.country = "China";

this.show = function () {

return "Country:[this: " + this.country + "] [self: " + self.country + "] [var: " + country + "]";

}

this.set = function (a) {

this.country = a;

}

this.set1 = function (a) {

self.country = a;

}

this.set2 = function (a) {

country = a;

}

return this;

}

});

app.controller("sale", function ($scope, puser) {

$scope.country = puser.country;

$scope.city = puser.city;

$scope.show = puser.show;

$scope.showw = puser.showw;

$scope.set = puser.set;

$scope.set1 = puser.set1;

$scope.set2 = puser.set2;

});

<div ng-controller="sale">

Country: <input ng-model="country" />{{ country }}<br />

City<input ng-model="city" />{{ city }}<br />

Show: {{ show() }}<br />

Showw: {{ showw() }}<br />

<button ng-click="set(country)">Set</button>

<button ng-click="set1(country)">Set1</button>

<button ng-click="set2(country)">Set2</button>

</div>



分析：1） 我们可以看到 showw 函数和city 定义在 this.$get 外面， 但是我们return this；

所以我们可以使用此函数和 city

2) 我们看看country的取值优先级别：

puser 里定义的 this.country 会被 config puserProvider.country 覆盖掉

如果在 this.$get里修改的值， 将会是优先级最高

app.config(function (puserProvider) {

puserProvider.country = "JSON";

});

app.provider("puser", function () {

var self = this;

var country = "America";

this.country = "Canada";

this.city = "Burnaby";

this.showw = function () {

return "showw : " + this.country;

}

this.$get = function () {

var country = "Mexico";

this.country = "Self Change";

var service = {

country: "China",

show: function () {

return "Country:[this: " + this.country + "] [self: " + self.country + "] [var: " + country + "]";

},

set: function (a) {

this.country = a;

},

set1: function (a) {

self.country = a;

},

set2: function (a) {

country = a;

}

}

service.country = "This Change";

return service;

}

});



注意：1） this.$get 里的修改拥有最高的优先权， 其次是 userProvider， 最后才是puser

3) 也可以直接注入service或者factory

app.provider("puser", function () {

var self = this;

var country = "America";

this.country = "Canada";

this.city = "Burnaby";

this.showw = function () {

return "showw : " + this.country;

}

this.$get = function (ssuser) {

ssuser.country = this.country;

return ssuser;

}

});

app.service("ssuser", function () {

var country = "Mexico";

var self = this;

this.country = "Self Change";

this.show = function () {

return "Country:

[this: " + this.country + "]

[self: " + self.country + "]

[var: " + country + "]";

}

this.set = function (a) {

this.country = a;

}

this.set1 = function (a) {

self.country = a;

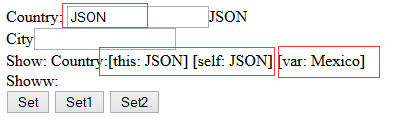
}

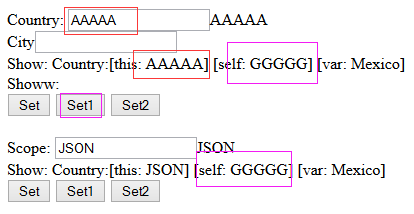
this.set2 = function (a) {

country = a;

}

});





4) 可以同时注入多个对象：但是对象只能return一个

this.$get = ["ssuser","fuser","$http", function (ssu, user, htpPost) {

user.city = "Burnaby BBY";

ssu.country = this.country;

return ssu;

}]

app.config(["puserProvider","saleProvider", function (pu, sa) {

pu.country = "JSON";

sa.address = "xxxxxx";

} ]);

* **filter过滤器**

AngularJS为我们提供了9个内建的过滤器

分别是currency, date, filter, json, limitTo, uppercase， lowercase, number, orderBy

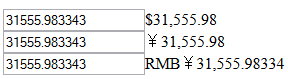
* {{ currency\_expression | currency : "symbol" : fractionSize}}

$filter('currency')(amount, symbol, fractionSize)

<input ng-model="amt" />{{ amt|currency }}<br />

<input ng-model="amt" />{{ amt|currency:"￥" }}<br />

<input ng-model="amt" />{{ amt|currency:"RMB￥":5 }}<br />



alert( $filter("currency")($scope.amt, "CAD$", 2) );

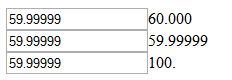
* {{ number\_expression | number : fractionSize}} 小数部分默认是 3

$filter('number')(number, fractionSize)

<input ng-model="amt" />{{ amt|number}}<br />

<input ng-model="amt" />{{ amt|number: 5 }}<br />

<input ng-model="amt" />{{ amt|number: -2 }}<br />

 -2 是对整数的四舍五入的， 40 = 0 445 = 400 1359 = 1400

* {{ date\_expression | date : format : timezone}}

$filter('date')(date, format, timezone)

<input ng-model="sdate" />{{ sdate | date : "yyyy-MM-dd h:mm:ssa":"-0700"}}<br />

<input ng-model="sdate" />{{ sdate | date : "yyyy-MM-dd h:mm:ssa":"-0900"}}<br />



alert( $filter("date")(1429900264 \* 1000, "y-M-d h:mm:ssa", "+0400") );

* {{ filter\_expression | filter : expression : comparator}}

$filter('filter')(array, expression, comparator)

只适用于数组， 过滤后返回新数值

expression: string, object , function

string - 是在数组元素上， 对所有的列和嵌套内容进行查找, 不区分大小写

app.controller("sale", function ($scope, $filter, puser) {

$scope.peoples = [

{ name: "Johnanh", phone: "604-888-1234",

com: [

{ title: "Yahoo", desc: "internet"},

{ title: "Sohu", desc: "Website"},

{ title: "Google", desc: "Application"}

]

},

{ name: "Johnson", phone: "604-888-5678" },

{ name: "Liuweihui", phone: "778-888-1298" },

{ name: "WilliamLiu", phone: "778-888-1276", com: "Yahoo" },

{ name: "Jason", phone: "604-555-6666" },

{ name: "Thomas", phone: "778-888-1276" },

{ name: "Willson778", phone: "604-888-5688" },

{ name: "Tomas778", phone: "604-888-5677" }

]

});

<div ng-controller="sale">

Search: <input ng-model="searchText" />

<table border="1">

<tr>

<td>Name<input ng-model="search.name" /></td>

<td>Phone<input ng-model="search.phone" /></td>

</tr>

<tr ng-repeat="people in peoples|filter:searchText" />

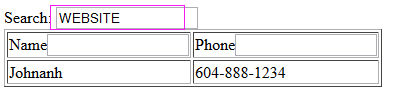
<td>{{people.name}}</td>

<td>{{people.phone}}</td>

</tr>

</table>

</div>



object - 可以使用对象来对特定的列搜索： {name:"john"} 默认也是模糊不区分大小写

我们也可以使用：$ 来匹配所有的列 如： { $ : "john" } 效果等于上面使用 string 来搜索

那么为什么我们还要有 $ ， 这是有它特殊用途的

<div ng-controller="sale">

Search: <input ng-model="search.$" />

<table border="1">

<tr>

<td>Name<input ng-model="search.name" /></td>

<td>Phone<input ng-model="search.phone" /></td>

</tr>

<tr ng-repeat="people in peoples|filter:search" />

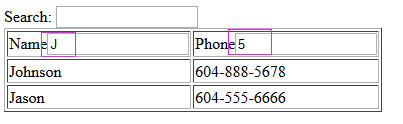
<td>{{people.name}}</td>

<td>{{people.phone}}</td>

</tr>

</table>

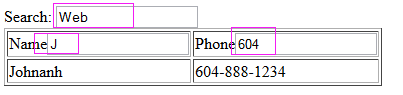
</div>



分析：

1） filter : search 此时 search是一个搜索对象 { name : "J", phone : "5" }

2) Search 我们使用的是 ng-model="search.$" 那么搜索对象就是：{ $: "Web", name : "J", phone : "5" }



3) { $: "string" } 的效果等同于 string 搜索，我们为什么还用， 如下很好说明

<div ng-controller="sale">

Search: <input ng-model="search" /> - 注意没有带 $

<table border="1">

<tr>

<td>Name<input ng-model="search.name" /></td>

<td>Phone<input ng-model="search.phone" /></td>

</tr>

<tr ng-repeat="people in peoples|filter:search" />

<td>{{people.name}}</td>

<td>{{people.phone}}</td>

</tr>

</table>

</div>



此时 search 是字符串类型， 所以Name不让输入任何内容， phone也不让输入内容，必须清理Search

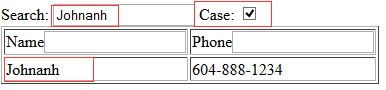
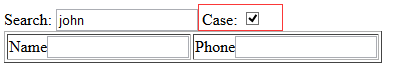


当我们录入到name， phone ， search就变成object， 所有数据绑定search就成了上面的错误提示。

comparator： (false|undefined) / true / function

false|undefined - 模糊查找，而且忽略大小写。

true - strict 严格查找， 值必须严格相同， 而且大小写也敏感。



* {{ json\_expression | json : spacing}}

$filter('json')(object, spacing)

angular.toJson(obj, pretty);

将 JSON Object 返回JSON字符串

<pre>{{peoples|json:5}}</pre>

[

{

"name": "Johnanh",

"phone": "604-888-1234",

"com": [

......................................

{

"name": "Tomas778",

"phone": "604-888-5677"

}

]

* {{ limitTo\_expression | limitTo : limit : begin}}

限制数组，字符，数字的个数

* {{ lowercase\_expression | lowercase}}

$filter('lowercase')( expression)

* {{ uppercase\_expression | uppercase}}

$filter('uppercase')( expression)

alert($filter("lowercase")("ABC"));

* {{ orderBy\_expression | orderBy : expression : reverse}}

$filter('orderBy')(array, expression, reverse)

1. 通过数据绑定来实现， 表格，点击题目来变化排序

<div ng-controller="sale">

Search: <input ng-model="orderby" />

Reverse: <input type="checkbox" ng-model="ordersn" value="0" />

<table border="1">

<tr>

<td><a href="" ng-click="orderby='name'; ordersn=!ordersn">Name</a></td>

<td><a href="" ng-click="orderby='phone';ordersn=!ordersn">Phone</a></td>

</tr>

<tr ng-repeat="people in peoples|orderBy:orderby:ordersn" />

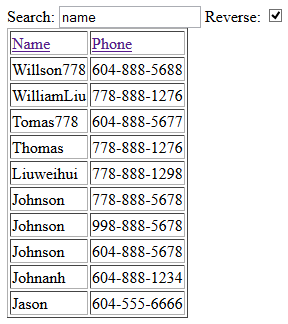
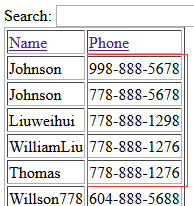
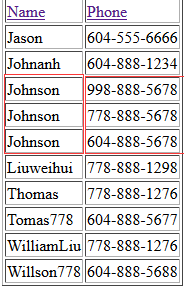
<td>{{people.name}}</td>

<td>{{people.phone}}</td>

</tr>

</table>

</div>

 2. 3.

2. 可以使用减号来代表 reverse = true

<tr ng-repeat="people in peoples|orderBy:'-phone'" />

<td>{{people.name}}</td>

<td>{{people.phone}}</td>

</tr>

3.可以通过数组来组合排序：

<tr ng-repeat="people in peoples|orderBy:['name','-phone']" />

<td>{{people.name}}</td>

<td>{{people.phone}}</td>

</tr>

4. 排序可以是表达式: 不一定非得是列名称。

<tr ng-repeat="people in peoples|orderBy:['name[2]','-phone']: true" />

<td>{{people.name}}</td>

<td>{{people.phone}}</td>

</tr>

按名字的第三个字符排序，而且是从大到小， 按电话从小到大， 因为 "-” + true 负负等正

* **filter 自定义过滤器**

app.filter("myfilter", function () {

var ff = function (input, p1, p2) {

if (input) {

return input.replace(/China/gi, "Great Contry(" + p1 + "," + p2 + ")");

} else {

return "N/A";

}

}

return ff;

});

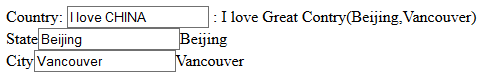
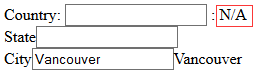
<div ng-controller="customer">

Country: <input ng-model="country" /> : {{ country | myfilter:state:city }}<br />

State<input ng-model="state" />{{ state }}<br />

City<input ng-model="city" />{{ city }}<br />

</div>



分析：

1. 过滤器就是一个函数， 该函数的第一个参数就是输入的值， 其次就是参数
2. 如果初始时没有定义 $scope.country 则在函数里要对input进行判断，否则undefined做操作会出错
3. 如果过滤器参数个数不定，也可以使用以下方法: arguments 是 javascript 的默认传递参数数组

app.filter("myfilter", function () {

return function () {

var input = arguments[0]?arguments[0]:"No Value Defined!";

var p1 = arguments[1];

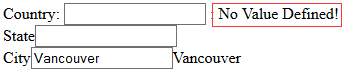
var p2 = arguments[2];

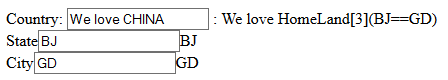
var plen = arguments.length;

return input.replace(/China/gi, "HomeLand[" + plen + "](" + p1 + "==" + p2 + ")");

}

});





1. 也可以对集合或者数组定义自己的过滤器：

app.filter("myfilter", function ($filter) {

return function () {

var input = arguments[0] ? arguments[0] : "No Value Defined!";

var p1 = arguments[1];

var p2 = arguments[2];

var plen = arguments.length;

input = $.grep(input, function (e, i) { - 这是数组，取序号为偶数的

e["name"] = p1 + ":" + i;

return (i % 2 == 0)

});

input = $filter("orderBy")(input, "name");

return input;

}

});

Name: <input ng-model="name" /><br />

<table border="1">

<tr>

<td>Name</td>

<td>Phone</td>

</tr>

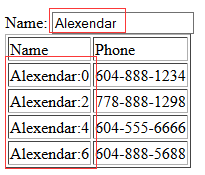
<tr ng-repeat="people in peoples|myfilter:name">

<td>{{ people.name }}</td>

<td>{{ people.phone }}</td>

</tr>

</table>



* **ngRoute 路由功能**

前提：

1）需要加载：<script type="text/javascript" src="angular-route.min.js"></script>

2）注入 var app = angular.module("myApp", ['ngRoute']);

3) 只支持单个视图ng-view，支持多个视图需要自定义或者其他第三方路由组件

4) 需要在config里注入 $routeProvider

5) 有几个注入与路由有关： $route, $routeParams, $location

6) 还有自定义注入： 通过 resolve : { key: value/object/service/factory/provider... }

7) 参数之间尽量不要留空格：{{name}},\_\_\_{{age}} HTML里使用多个空格 href="#cust/({{name}}, {{age}})"

<body ng-app="myApp" style="font-size: 16px;">

<div ng-controller="sales">

Name: <input ng-model="name" />:{{name}}<br />

Age: <input ng-model="age" />:{{age}}<br />

Country: <input ng-model="country" />:{{country}}<br />

City: <input ng-model="city" />:{{city}}<br />

<br />

<a href="#cust/({{name}},{{age}})">Customer Information</a><br />

<a href="#address/({{country}},{{city}})">Address Information</a><br />

</div>

<br />

<div ng-view></div>

</div>

<script language="javascript" type="text/javascript">

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

app.controller("sales", function ($scope) {

});

app.config(function ($routeProvider) {

$routeProvider

.when("/cust/(:name,:age)",

{

template: 'Customer Info:<br>Name: {{name}}<br>Age: {{age}}<br>URL: {{url}}',

controller: ["$scope", "$routeParams", "$location", function ($scope, $rp, $loc) {

$scope.name = $rp.name;

$scope.age = $rp.age;

$scope.url = $loc.path();

} ]

})

.when("/address/(:country1,:city1)",

{

template: 'Address Info:<br>Country: {{country}}<br>City: {{city}}<br>URL: {{url}}',

controller: "addrCtrl"

})

.otherwise({

redirectTo: '/'

});

});

app.controller("addrCtrl",

["$scope", "$routeParams", "$location",

function ($scope, $routeParams, $location) {

$scope.country = $routeParams.country1;

$scope.city = $routeParams.city1;

$scope.url = $location.path();

}

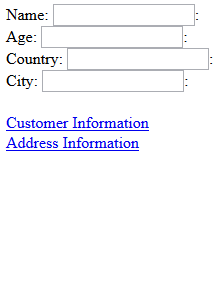
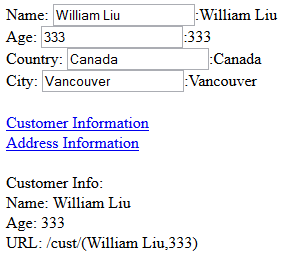
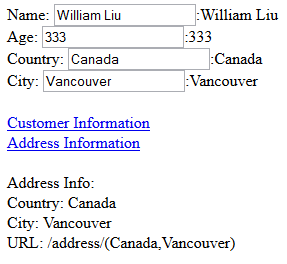
]);

</script>

运行结果：

/index1.html#/cust/(William Liu,333) - 如果参数是空 (,) 则不匹配路由，输出内容会为空白

/index1.html#/address/(Canada,Vancouver)

resolve 自定义注入，执行时机是在controller之前

.when("/cust/(:name,:age)",

{

template: 'Customer Info:<br>Name: {{name}}<br>Age: {{age}}<br>URL: {{url}}<br>YNN: {{ynn}}',

controller: [

"$scope",

"$routeParams",

"$location",

"yourname", function ($scope, $rp, $loc, yn) {

$scope.name = $rp.name;

$scope.age = $rp.age;

$scope.url = $loc.path();

$scope.ynn = yn;

} ],

resolve: {

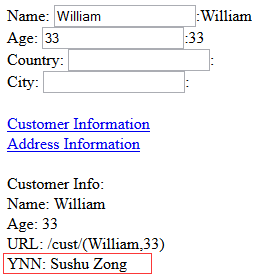
yourname: function() {

return "**Sushu Zong**";

}

}

})



以上例子是注入一个字符串， 可以注入服务，工厂，对象等复杂类型

$location 在路由里是很重要的对象。

$location服务下有如下方法：

1.path();获取当前页面的路径，也可以设置一个新的路径path("/")，这样就修改为“/”的路由了

$location.path();//获取

$location.path("/");//设置

2.replace()  
如果你希望跳转后用户不能点击后退按钮（对于登录之后的跳转这种发生在某个跳转之后的再次跳转很有用），AngularJS提供了replace()方法来实现这个功能：  
$location.path('/home');   
$location.replace();   
// 或者  
$location.path('/home').replace();

8. search()  
search()方法用来获取URL中的查询串：  
$location.search();   
我们可以向这个方法中传入新的查询参数，来修改URL中的查询串部分：  
// 用对象设置查询  
$location.search({name: 'Ari', username: 'auser'});   
// 用字符串设置查询  
$location.search('name=Ari&username=auser');   
search方法可以接受两个参数。  
search（可选，字符串或对象）  
这个参数代表新的查询参数。hash对象的值可以是数组。  
paramValue（可选，字符串）  
如果search参数的类型是字符串，那么paramValue会做为该参数的值覆盖URL当中的对应  
值。如果paramValue的值是null，对应的参数会被移除掉。

9. url()  
url()方法用来获取当前页面的URL：  
$location.url(); // 该URL的字符串  
如果调用url()方法时传了参数，会设置并修改当前的URL，这会同时修改URL中的路径、  
查询串和hash，并返回$location。  
// 设置新的URL   
$location.url('/home?name=Ari#hashthing');   
url()方法可以接受两个参数。  
url（可选，字符串）  
新的URL的基础的前缀。

replace（可选，字符串）  
想要修改成的路径。

如下给出例子：

app.controller("sales", function ($scope, $location) {

$scope.change = function () {

$location.path("customer/address/('China','Lechang')").replace();

}

$scope.loc = $location;

});

Location: <button ng-click="change()">Location</button><br />

absUrl: {{loc.absUrl()}}<br /> get only

url: {{loc.url()}}<br /> get / set

path: {{loc.path()}}<br /> get / set

protocol: {{loc.protocol()}}<br /> get only

host: {{loc.host()}}<br /> get only

port: {{loc.port()}}<br /> get only

state: {{loc.state()}}<br /> get / set

hash: {{loc.hash()}}<br /> get / set

path( url\_string ) & url( url\_string ) 直接重定向

1) url\_string只能是相对路径，不能是绝对路径:"http://www.google.com/address/('China','Lechang')"

因为只支持相对路径，而且自动加上 "/" 开头，即结果如下：

"file:///D:/Temp4/Temp4/index1.html#/http://www.google.com/address/%28%27China%27,%27Lechang%27%29"

2）url\_string 必须是 "/" 开头， 如果没有则会自动加上

**"file:///D:/Temp4/Temp4/index1.html#/customer/address/%28%27China%27,%27Lechang%27%29"**

**search , path, url**

app.controller("sales", function ($scope, $location) {

$scope.change = function () {

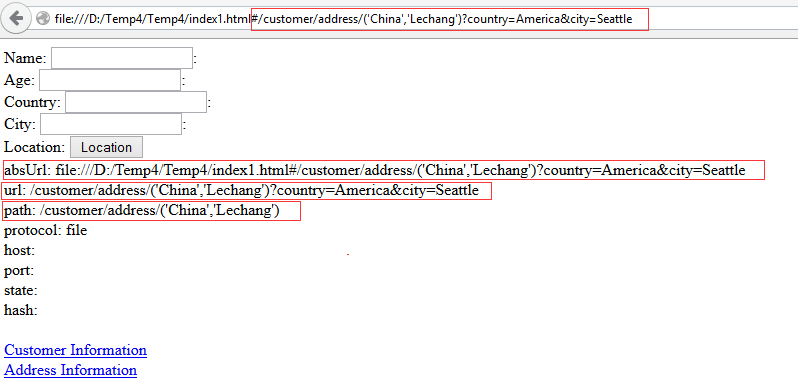
$location.path("customer/address/('China','Lechang')");

$location.search({ country: "America", city: "Seattle" });

}

});

Location: <button ng-click="change()">Location</button><br />

****分析：

a） path() 和 url() 不一样

$location.url("customer/address/('China','Lechang')?id=100&age=300");

****

$location.path("customer/address/('China','Lechang')?id=100&age=300");

****

注意：如果在 path里使用 ?id=100&age=300会当成值的一部分

1) path() 方法与 search() 方法

.when("/address/:country/:city",

{

template: 'Address Info:<br>

Country: {{country}}<br>

City: {{city}}<br>

URL: {{url}}<br>

Path: {{path}}<br>

ID: {{id}}',

controller: "addrCtrl"

})

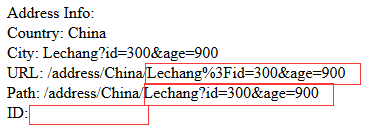
app.controller("sales", function ($scope, $location) {

$scope.change = function () {

$location.path("address/China/Lechang?id=300&age=900");

}

});

****

分析：

a) ?id=300&age=900 使用 path() 来设置参数， 将不会被当成参数。

b） 使用 $location.search({ id: 999, age: "Seattle" })

或者

$location.search("id=999&age=Seattle")

app.controller("sales", function ($scope, $location) {

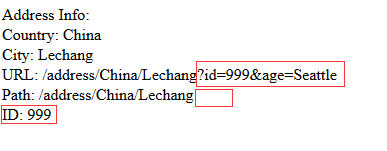
$scope.change = function () {

$location.path("address/China/Lechang");

$location.search({ id: 999, age: "Seattle" });

}

});



2) url() 方法与 search() 方法

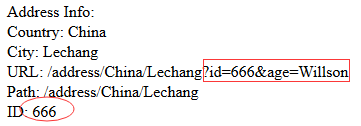
app.controller("sales", function ($scope, $location) {

$scope.change = function () {

$location.url("address/China/Lechang?id=666&age=Willson");

}

});



分析：

a) URL可以带?号的参数的

b) 参数优先级别的问题：$route path 里的参数优先级别最高，而且不会别？的参数所覆盖

？所带的参数是完全后者覆盖删除前者。

app.controller("sales", function ($scope, $location) {

$scope.change = function () {

$location.url("address/China/Lechang?id=666&city=Willson");

path的参数优先级别最高 ?的参数不会影响path的参数

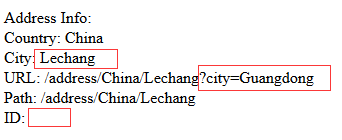
$location.search({ city: "Guangdong" }); - 完全删除前面 ? 的参数 id=666

}

});

**file:///D:/Temp4/Temp4/index1.html#/address/China/Lechang?city=Guangdong**

**url 的参数 ？id=666 被后者删除覆盖**

****

**与路由有关的事件：**

app.run(['$rootScope', '$window', '$location', '$log', function ($rootScope, $window, $location, $log) {

    var locationChangeStartOff = $rootScope.$on('$locationChangeStart', locationChangeStart);

    var locationChangeSuccessOff = $rootScope.$on('$locationChangeSuccess', locationChangeSuccess);

    var routeChangeStartOff = $rootScope.$on('$routeChangeStart', routeChangeStart);

    var routeChangeSuccessOff = $rootScope.$on('$routeChangeSuccess', routeChangeSuccess);

    function locationChangeStart(event) {

        $log.log('locationChangeStart');

        $log.log(arguments);

    }

    function locationChangeSuccess(event) {

        $log.log('locationChangeSuccess');

        $log.log(arguments);

    }

    function routeChangeStart(event) {

        $log.log('routeChangeStart');

        $log.log(arguments);

    }

    function routeChangeSuccess(event) {

        $log.log('routeChangeSuccess');

        $log.log(arguments);

    }

}]);

function locationChangeStart(event, newUrl) {

        $log.log('locationChangeStart');

        $log.log(arguments);

        var ret = $window.confirm('Are you sure to give it up? ');

        if (ret) {

            locationChangeStartOff(); //Stop listening for location changes or you can do others

            $location.path(newUrl);

            return;

        }

        event.preventDefault();

        return;

    }

我们来总结下：

我们利用location改变事件来实现了在提交Location之前做了定制化，除了location事件，还有route事件，利用route事件我们可以做‘loading..’效果。

# AngularJS的学习--$on、$emit和$broadcast的使用 - 疯狂的原始人

*原文*  [http://www.cnblogs.com/CraryPrimitiveMan/p/3679552.html](http://www.cnblogs.com/CraryPrimitiveMan/p/3679552.html?utm_source=tuicool)

* $emit只能向parent controller传递event与data. 也叫 “冒泡”
* $broadcast只能向child controller传递event与data 也叫“传播”
* $on用于接收event与data

注意：

a）$broadcast 与 $emit 必须是针对同名函数。也就是说在各个层里寻找到相同的函数， 如果没有则不执行。

b) event.stopPropagation(); 只适用于使用 $emit 发出的事件, 对于 $broadcast 发出的事件会出现错误：



所以如果不知道是什么发出的事情， 可以使用安全方式调用： (event.stopPropagation || angular.noop)();

c) event.preventDefault(); 是用来阻止默认的行为： 如果 <a href="xxxx" />, submit button 提交表单

例子如下：

$(function () {

$("a").bind("click", function (event) {

console.log("a click");

});

$("input:submit").bind("click", function (event) {

console.log("submit click");

});

})

<a href="#address/hello">Call Me</a><br />

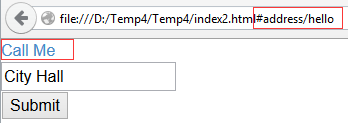
<form action="#address" method="get">

<input type="text" name="city" value="City Hall" /><br />

<input type="submit" value="Submit" />

</form>





$(function () {

$("a").bind("click", function (event) {

console.log("a click");

event.preventDefault();

});

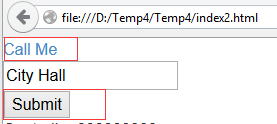
$("input:submit").bind("click", function (event) {

console.log("submit click");

event.preventDefault();

});

})



如果加上：event.preventDefault(); 则执行完后即停止执行默认的行为。

前提是保证 event 对象可用。

例子如下： html代码

<div ng-controller="ParentCtrl"> <!--父级-->

<div ng-controller="SelfCtrl"> <!--自己-->

<a ng-click="click()">click me</a>

<div ng-controller="ChildCtrl"></div> <!--子级-->

</div>

<div ng-controller="BroCtrl"></div> <!--平级-->

</div>

js代码

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

$scope.click = function () {

$scope.$broadcast('to-child', 'child');

$scope.$emit('to-parent', 'parent');

}

});

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

$scope.$on('to-parent', function(event,data) {

console.log('ParentCtrl', data); //父级能得到值

});

$scope.$on('to-child', function(event,data) {

console.log('ParentCtrl', data); //子级得不到值

});

});

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

$scope.$on('to-child', function(event,data) {

console.log('ChildCtrl', data); //子级能得到值

});

$scope.$on('to-parent', function(event,data) {

console.log('ChildCtrl', data); //父级得不到值

});

});

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

$scope.$on('to-parent', function(event,data) {

console.log('BroCtrl', data); //平级得不到值

});

$scope.$on('to-child', function(event,data) {

console.log('BroCtrl', data); //平级得不到值

});

});

最终结果

ParentCtrl child

ChildCtrl parent

$emit和$broadcast可以传多个参数，$on也可以接收多个参数。

在$on的方法中的event事件参数，其对象的属性和方法如下

| **事件属性** | **目的** |
| --- | --- |
| event.targetScope | 发出或者传播原始事件的作用域 |
| event.currentScope | 目前正在处理的事件的作用域 |
| event.name | 事件名称 |
| event.stopPropagation() | 一个防止事件进一步传播(冒泡/捕获)的函数(这只适用于使用`$emit`发出的事件) |
| event.preventDefault() | 这个方法实际上不会做什么事，但是会设置`defaultPrevented`为true。直到事件监听器的实现者采取行动之前它才会检查`defaultPrevented`的值。 |
| event.defaultPrevented | 如果调用了`preventDefault`则为true |

感觉比service在不同controller中通信要方便许多~~

<script language="javascript" type="text/javascript">

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

app.controller("ctr0", function ($scope) {

$scope.name = "ctr0 name";

$scope.$on("call", function (event, data, data1) {

console.log("Ctr0:" + event.name +

":" + event.targetScope.name +

":" + event.currentScope.name +

":[" + data + ":" + data1 + "]");

});

$scope.callme = function () {

$scope.$broadcast("call", " broadcase from ctr0", 1000);

}

})

.controller("ctr1", function ($scope) {

$scope.name = "ctr1 name";

$scope.$on("call", function (event, data, data1) {

console.log("Ctr1:" + event.name +

":" + event.targetScope.name +

":" + event.currentScope.name +

":[" + data + ":" + data1 + "]");

})

})

.controller("ctr2", function ($scope) {

$scope.$on("call", function (event, data, data1) {

console.log("Ctr2:" + event.name +

":" + event.targetScope.name +

":" + event.currentScope.name +

":[" + data + ":" + data1 + "]");

(event.stopPropagation || angular.noop)(); - 安全的阻止冒泡。到此为止结束

})

})

.controller("ctr3", function ($scope) {

$scope.$on("call", function (event, data) {

console.log("Ctr3:" + event.name +

":" + event.targetScope.name +

":" + event.currentScope.name +

":[" + data + "]");

})

})

.controller("ctr4", function ($scope) {

$scope.$on("call", function (event, data) {

for (var key in arguments) {

console.log("key: " + key + " val:" + arguments[key]);

}

console.log("Ctr4:" + event.name +

":" + event.targetScope.name +

":" + event.currentScope.name +

":[" + arguments.length + "]");

event.preventDefault(); - $on 里调用此方法不会阻止执行默认的行为

});

$scope.callme = function (event) {

console.log("length: " + arguments.length + ":" + event);

$scope.$emit("call", " emit from ctr4", 9999, 2000, 3000);

}

})

</script>

</head>

<body ng-app="myApp" style="font-size: 16px;">

<div ng-controller="ctr0">

Controller 000000000<br />

<button ng-click="callme()">Call Me</button><br />

<div ng-controller="ctr1">

Controller 11111111<br />

<div ng-controller="ctr2">

controller 22222222<br />

<div ng-controller="ctr3">

controller 3333333<br />

<div ng-controller="ctr4">

controller 44444444<br />

<a href="#address/hello" ng-click="callme()">Call Me</a><br />

</div>

</div>

</div>

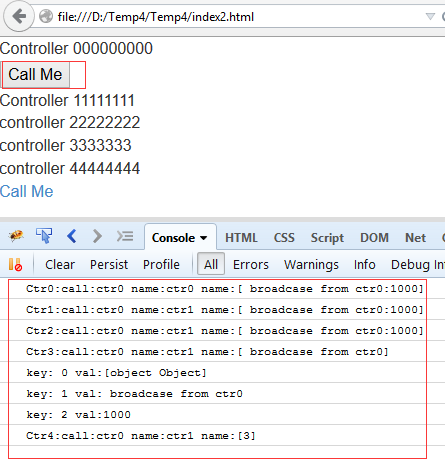
</div>

</div>

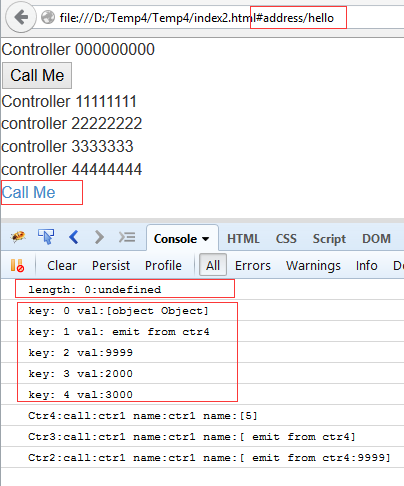
</body>

</html>

**执行向子广播：**

****

**执行向父冒泡:**

****

分析：

1） <a href="#address/hello" ng-click="callme()">Call Me</a>

$scope.$emit 冒泡

event.preventDefault(); - 阻止默认行为失败， 无效

(event.stopPropagation || angular.noop)(); - 安全的阻止冒泡。到此为止结束

所以你看不到 "ctr1" 和 "ctr0" 的方法被执行。

2）$scope.$emit("call", " emit from ctr4", 9999, 2000, 3000); 支持传递多个参数，

在 $on 接受方，第一个参数永远是 event， 其后则按顺序对应

for (var key in arguments) {

console.log("key: " + key + " val:" + arguments[key]);

}

arguments[0] - event object

event.name

event.targetScope -发出或者传播原始事件的作用域

event.currentScope -目前正在处理的事件的作用域

event.stopPropagation()

event.preventDefault()

3) 因为操作的对象都是 $scope , 而$scope涉及到嵌套层次，则遵守其继承原则：

即 currentScope 优先级高，没有定义往上父级parentScope查找，直到根 $rootScope

4) 如果想在 ng-click事件里传递event 可以使用$event ng-click="callme($event)"

* **$watch** angularJS使用watch监控数据模型的变化

使用$watch监控数据模型的变化

在scope 内置的所有函数中，用得最多的可能就是$watch 函数了，当你的数据模型中某一部分发生变化时，$watch 函数可以向你发出通知。你可以监控单个对象的属性，也可以监控需要经过计算的结果（函数），实际上只要能够被当作属性访问到，或者可以当作一个[Java](http://www.2cto.com/kf/ware/Java/)Script 函数被计算出来，就可以被$watch 函数监控。它的函数签名为

  $watch(watchFn, watchAction, deepWatch)

其中每个参数的详细含义如下。

watchFn

该参数是一个带有Angular 表达式或者函数的字符串，它会返回被监控的数据模型的当前值。这个表达式将会被执行很多次，所以你要保证它不会产生其他副作用。也就是说，要保证它可以被 调用很多次而不会改变状态。基于同样的原因，监控表达式应该很容易被计算出来。如果你使用字符串传递了一个Angular 表达式，那么它将会针对调用它的那个作用域中的对象而执行。

watchAction

这是一个函数或者表达式，当watchFn 发生变化时会被调用。如果是函数的形式，它将会接收到watchFn 的新旧两个值，以及作用域对象的引用。其函数签名为function(newValue, oldValue, scope)。

deepWatch

如果设置为true，这个可选的布尔型参数将会命令Angular 去检查被监控对象的每个属性是否发生了变化。如果你想要监控数组中的元素，或者对象上的所有属性，而不只是监控一个简单的值，你就可以使用这个参数。由于 Angular 需要遍历数组或者对象，如果集合比较大，那么运算负担就会比较重。

$watch 函数会返回一个函数，当你不再需要接收变更通知时，可以用这个返回的函数注销监控器。

如果我们需要监控一个属性，然后接着注销监控，我们可以使用以下代码：

  var dereg = $scope.$watch('someModel.someProperty', callbackOnChange());

…

dereg();

例如：

Price: <input ng-model="price" ng-model-options="{debounce: 1000}" />{{ price | currency:"$"}}

Quantity: <input ng-model="qty" ng-model-options="{debounce: 1000}" />{{ qty | number:0 }}<br />

Amount: {{ total() | currency:"$" }}<br />

<a href="#address/hello" ng-click="callme()">Cancel Watch</a><br />

.controller("ctr4", function ($scope) {

$scope.total = function () { return $scope.price \* $scope.qty; }

var mywatch = $scope.$watch($scope.total, function (nval, oval, sc) {

console.log("nval: " + nval + " oval: " + oval + " sc: " + sc.total());

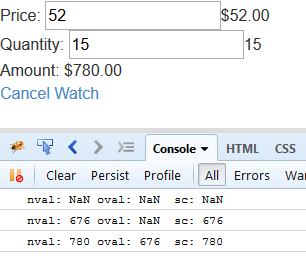
}, true);

$scope.callme = function (event) {

mywatch(); - 取消 watch

}

})

****

**AngularJs之表单及表单验证**

## Angular 的表单属性 $valid, $invalid, $pristine, $dirty

Angular 提供了有关表单的属性来帮助我们验证表单. 他们给我们提供了各种有关一个表单及其输入的信息，并且应用到了表单和输入.

|  |  |  |
| --- | --- | --- |
| 属性类 |  | 描述 |
| $valid | ng-valid | Boolean 告诉我们这一项当前基于你设定的规则是否验证通过 |
| $invalid | ng-invalid | Boolean 告诉我们这一项当前基于你设定的规则是否验证未通过 |
| $pristine | ng-pristine | Boolean 如果表单或者输入框没有使用则为True |
| $dirty | ng-dirty | Boolean 如果表单或者输入框有使用到则为True |

Angular 也提供了有关表单及其输入框的类，以便你能够依据每一个状态设置其样式.

### 访问表单属性

* **方位表单:** <form name>.<angular property>
* **访问一个输入框:** <form name>.<input name>.<angular property>

例如：

app.controller("ctr0", function ($scope) {

$scope.email = "william";

}

<div ng-controller="ctr0">

<form name="myform" novalidate>

Name: <input name="name"

ng-model="name"

ng-trim="false"

ng-minlength="3"

ng-maxlength="50" required />\*{{name.length}}\*<br /><br />

Email: <input type="email"

id="email"

name="email"

ng-minlength="5"

ng-maxlength="10" ng-model="email" required />{{email}}<br /><br />

URL: <input type="url" name="url" ng-model="url" required />

\*{{url.length}}\* - {{ myform.url.$error }} <br />

<button ng-disabled="myform.$invalid">Submit</button><br /><br />

<input type="button" onclick="$('#email').val('willson@hotmail.com')" value="Change" /><br />

</form>

<br />

<br />

Form Verify:<br />

valid: {{myform.$valid}}<br />

invalid: {{myform.$invalid}}<br />

pristine: {{myform.$pristine}}<br />

dirty: {{myform.$dirty}}<br />

error: <pre>{{myform.$error}}</pre><br />

<br />

Email Verify: <br />

valid: {{myform.email.$valid}}<br />

invalid: {{myform.email.$invalid}}<br />

pristine: {{myform.email.$pristine}}<br />

dirty: {{myform.email.$dirty}}<br />

Error: {{myform.email.$error}}

<br />

$valid , $invalid, $pristine, $dirty 是布尔值 true/false 用于条件判断

$error - 是对象{ }， 放在 form 的层级上， 是表单里的错误集合

{"required": [

{

"$validators":{},

"$asyncValidators":{},

"$parsers":[],

"$formatters":[null],

"$viewChangeListeners":[],

"$untouched":true,

"$touched":false,

"$pristine":true,

"$dirty":false,

"$valid":false,

"$invalid":true,

"$error":{"required":true},

"$name":"name",

"$options":null

},

{

"$validators":{},

"$asyncValidators":{},

"$parsers":[],

"$formatters":[null],

"$viewChangeListeners":[],

"$untouched":true,

"$touched":false,

"$pristine":true,

"$dirty":false,

"$valid":false,

"$invalid":true,

"$error":{"required":true},

"$name":"url",

"$options":null

}

],

"email":[

{

"$viewValue":"william",

"$validators":{},

"$asyncValidators":{},

"$parsers":[],

"$formatters":[null],

"$viewChangeListeners":[],

"$untouched":true,

"$touched":false,

"$pristine":true,

"$dirty":false,

"$valid":false,

"$invalid":true,

"$error":{"email":true},

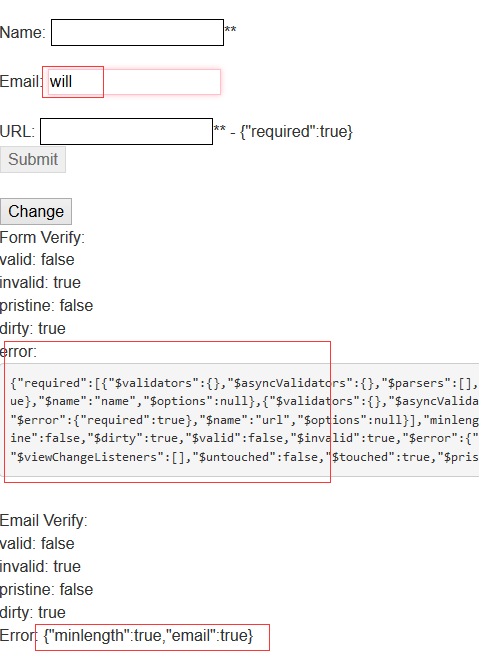
"$name":"email",

"$options":null

}

]

}



如果我把 myform.email 输出则是整个关于email验证的对象

Error:

{

"$viewValue":"wil",

"$validators":{},

"$asyncValidators":{},

"$parsers":[],

"$formatters":[null],

"$viewChangeListeners":[],

"$untouched":false,

"$touched":true,

"$pristine":false,

"$dirty":true,

"$valid":false,

"$invalid":true,

"$error":{"minlength":true,"email":true},

"$name":"email",

"$options":null

}

URL: <input type="url" name="url" ng-model="url" />\*{{url.length}}\* - {{ myform.url.$error }} <br/>

Error: {{myform.url}}

{

"$validators":{},

"$asyncValidators":{},

"$parsers":[],

"$formatters":[null],

"$viewChangeListeners":[],

"$untouched":true,

"$touched":false,

"$pristine":true,

"$dirty":false,

"$valid":true,

"$invalid":false,

"$error":{},

"$name":"url",

"$options":null

}

输入： http://

{

"$viewValue":"http://",

"$validators":{},

"$asyncValidators":{},

"$parsers":[],

"$formatters":[null],

"$viewChangeListeners":[],

"$untouched":false,

"$touched":true,

"$pristine":false,

"$dirty":true,

"$valid":false,

"$invalid":true,

"$error":{"url":true},

"$name":"url",

"$options":null

}

我们可以使用directive 来自定义错误信息：

app.directive("errmsg", function () {

return {

restrict: "E",

scope: {

field: "=col"

},

replace: true,

template: ['<span style="color:red; font-size:10px; padding-left:5px;" ng-show="field.$invalid">',

'<span ng-show="field.$error.required">Required; </span>',

'<span ng-show="field.$error.minlength">Less Chars; </span>',

'<span ng-show="field.$error.maxlength">More Chars; </span>',

'<span ng-show="field.$error.pattern">Invalid Chars; </span>',

'<span ng-show="field.$error.email">Invalid Email; </span>',

'<span ng-show="field.$error.url">Invalid URL; </span>',

'</span>'

].join("")

}

})

<style>

input.ng-invalid {

border: 1px solid pink;

box-shadow: 0px 0px 5px pink;

}

input.ng-pristine {

border: 1px solid black;

box-shadow: none;

}

</style>

<form name="myform" novalidate>

Name: <input name="name" ng-model="name"

ng-trim="true" - 默认设置， 会自动把ng-model的值剪切空白

pattern="abc\w+"

ng-minlength="5"

ng-maxlength="50" required /><errMsg col="myform.name"></errMsg><br /><br />

Email: <input type="email" id="email" name="email"

ng-minlength="5"

ng-maxlength="10"

ng-model="email" required

ng-model-options="{updateOn:'default blur', debounce:{default: 500, blur:10}}" />

<errMsg col="myform.email"></errMsg><br /><br />

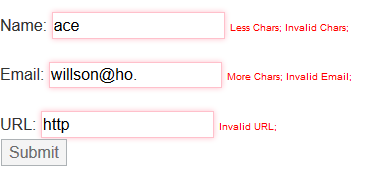
URL: <input type="url" name="url" ng-model="url" />

<errMsg col="myform.url"></errMsg></errMsg> <br />

<button ng-disabled="myform.$invalid">Submit</button><br /><br />

<input type="button" onclick="$('#email').val('willson@hotmail.com')" value="Change" /><br />

</form>



　form提供的属性都是用来表示表单的验证状态的，包括：$pristine(表单没有填写记录)、$dirty(表单有填写记 录)、$valid(通过验证)、$invalid(未通过验证)、$error(验证错误信息)。除$error外，前四个的值为true或false 表示相应的状态。$error的值为一个js对象，包含了以下验证内容的状态：

* email
* max
* maxlength
* min
* minlength
* number
* pattern
* required
* url

以上的值也是 true/false 来表示是否有错误， 其实如果没有错误是 undefined 不会有该键值对存在

如： $error.email = true 表示邮件格式有错误

$error.max = true 表示数值大于最大值

　CSS代码，为不同的状态设置不同的背景色：

input.ng-pristine

{

background-color: white;

}

input.ng-dirty

{

background-color: lightyellow;

}

input.ng-valid

{

background-color: lightgreen;

}

input.ng-invalid

{

background-color: pink;

}

**自定义表单验证：**

**主要使用是**ngModel.NgModelController

提供了很多方法：

**$setDirty(); $setUntouched(); $setPristine(); 等用于改变状态的**

**$setViewValue(value, trigger);**

**$commitViewValue();**

**$validate();**

**$rollbackViewValue(); 用于那些有时差修改， 想放回到上一次未提交前变化的值。**

### $viewValue， $modelValue

### $parsers

### $formatters

### $validators

### $asyncValidators

1) 表单对象，可以在controller里被阅读：

<form name="myform" novalidate>

Email: <input type="email"

name="email"

ng-model="email1" -- 注意在controller里并不是使用ng-model 来设置验证有关的事情

ng-minlength="5" ng-maxlength="10"

lwh required

ng-model-options="{updateOn:'default blur',allowInvalid:true, debounce:{default: 5000, blur:6000}}" />

<errMsg col="myform.email"></errMsg><br /><br />

</form>

app.controller("ctr0", function ($scope) {

$scope.email = "william";

// 在这里自定义验证，会出错，因为 DOM 还没有准备好 $scope.myform.email undefined

$scope.myform.email.$parsers.push(function (viewValue) {

$scope.myform.email.$setValidity("lwh", false);

return viewValue;

});

$scope.callme = function (evt) {

//alert(evt);

console.log($scope.myform.email.$viewValue + ":" + $scope.myform.email.$modelValue);

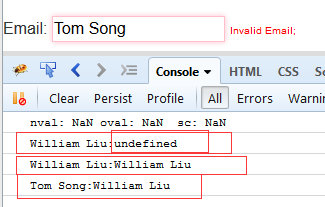
}

})

2) 由于设置的延时更新 allowInvalid:true 是关键， 影响到 $modelValue

如果allowInvalid:false , 则验证不通过，$modelValue = undefined

所以当我们输出 $viewValue 和 $modelValue 值时， 会有短暂的不同



3) 通过属性指令实现自定义验证 $parsers： $parsers 验证逻辑不会阻止 $viewValue 写向 $modelValue

即使： ng-model-options="{allowInvalid:false}" 也不会阻止

app.directive("errmsg", function () {

return {

restrict: "E",

scope: {

field: "=col"

},

replace: true,

template: ['<span style="color:red;

font-size:10px;

padding-left:5px;"

ng-show="field.$invalid">',

'<span ng-show="field.$error.required">Required; </span>',

'<span ng-show="field.$error.minlength">Less Chars; </span>',

'<span ng-show="field.$error.maxlength">More Chars; </span>',

'<span ng-show="field.$error.pattern">Invalid Chars; </span>',

'<span ng-show="field.$error.email">Invalid Email; </span>',

'<span ng-show="field.$error.url">Invalid URL; </span>',

'<span ng-show="field.$error.min">Less than Min; </span>',

'<span ng-show="field.$error.max">Large than Max; </span>',

'<span ng-show="field.$error.lwh">My Error; </span>',

'</span>'

].join("")

}

})

app.directive("evennum", function () {

return {

restrict: "AE",

**require**: 'ngModel', - 必须引用 ngModelController对象

link: function (sc, el, attrs, ctrl) {

ctrl.$parsers.push(function (viewValue) {

if (viewValue % 2 == 0) {

ctrl.$setValidity("lwh", true); - true 表示通过验证 valid

} else {

ctrl.$setValidity("lwh", false); - false 表示 invalid

}

return viewValue;

})

}

}

});

Number: <input type="number"

name="num"

ng-model="num\_of\_member"

min="10"

max="60"

evennum />

{{myform.num.$error}}

<errMsg col="myform.num"></errMsg></errMsg> <br />







<form name="yourform" novalidate> - 屏蔽html5 的验证功能，否则angularjs的验证不工作

Name: <input ng-model="user.name" /><br />

Age: <input ng-model="user.age" name="age" evennum />{{yourform.age.$error}}<br />

<button ng-click="show()">SHOW</button>

</form>

3) 通过属性指令实现自定义验证 $validators

app.directive("evennum", function () {

return {

restrict: "AE",

**require**: 'ngModel',

link: function (sc, el, attrs, ctrl) {

ctrl.$validators.odd = (function (mv, vv) {

return vv % 2 == 1; - true 表示通过验证{}， false表示验证失败 {odd:true}

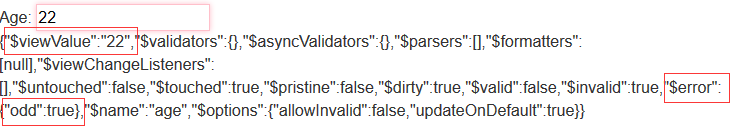
})

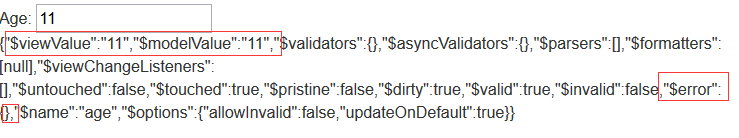
}

}

});

Age: <input ng-model="user.age" name="age" evennum odd ng-model-options="{allowInvalid:false}" />{{yourform.age}}<br />

****

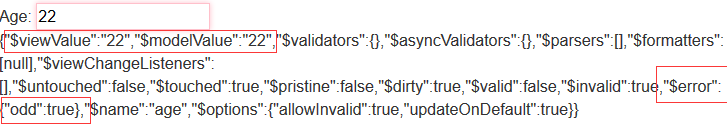
**我们看到没有 $modelValue**

**reture true; $error = {} 即验证通过， 我们看到 $modelValue = "11"**

4) ng-model-options="{allowInvalid:false}" 默认是false

Age:<input ng-model="user.age" name="age" evennum odd ng-model-options="{allowInvalid:true}" /><br />

{{yourform.age}}<br /><br />

****

**我们看到，即使验证不通过， 也允许写入 $modelValue**

### $asyncValidators

app.directive("prime", function($q, $timeout) {

var isPrime = function(n) {

if (n < 2) {

return false;

}

for (var i = 2; i <= Math.sqrt(n); i++) {

if (n % i == 0) {

return false;

}

}

return true;

};

return {

restrict: "A",

require: "ngModel",

link: function(scope, element, attributes, ngModel) {

ngModel.$asyncValidators.prime = function(modelValue) {

var defer = $q.defer();

$timeout(function(){

if(isPrime(modelValue)) {

defer.resolve();

} else {

defer.reject();

}

}, 2000);

return defer.promise;

}

}

};

});

4) $formatters 在验证中的使用: 和$parsers的方向相反， 这是从$modelValue 的修改触发去修改$viewValue

app.directive("evennum", function () {

return {

restrict: "AE",

**require**: 'ngModel',

link: function (sc, el, attrs, ctrl) {

ctrl.$formatters.push(function (viewValue) {

if (viewValue)

return viewValue.toUpperCase();

else

return "ABC";

})

}

}

});

<form name="yourform" novalidate>

Name: <input ng-model="user.name" /><br /><br />

View: <input ng-model="user.age" name="age" evennum ng-model-options="{updateOn:'blur'}"

/><br />

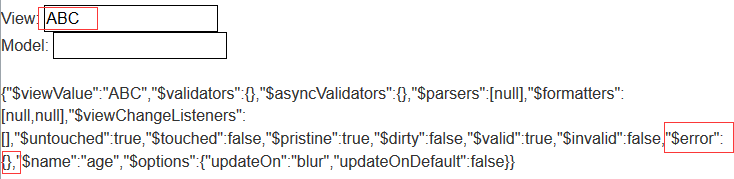
Model: <input ng-model="user.age" /><br />

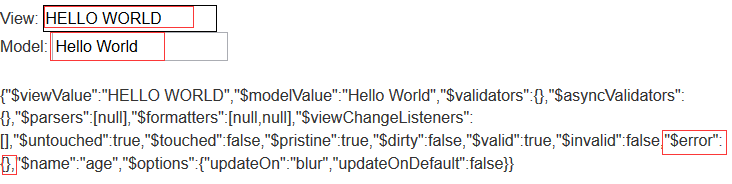
<br />

{{yourform.age}}<br /><br />

<button ng-click="show()">SHOW</button><br />

</form>





**分析**：

1) 从 $modelValue 向 $viewValue 赋值不会，触发死循环递归： 即不会发生：$viewValue 改变了值再触发向 $modelValue改写

5) $parsers 可以做一些更改格式的事情：

app.directive("evennum", function () {

return {

restrict: "AE",

**require**: 'ngModel',

link: function (sc, el, attrs, ctrl) {

ctrl.$parsers.push(function (viewValue) {

var veiwValue1 = viewValue;

if (viewValue % 2 == 0) {

ctrl.$setValidity("lwh", true);

} else {

ctrl.$setValidity("lwh", false);

}

if (viewValue.length >= 1 && viewValue.length <= 3) {

viewValue1 = "(" + viewValue + ")";

}

if (viewValue.length > 3 && viewValue.length <= 6) {

viewValue1 = "(" + viewValue.slice(0, 3) + ") " + viewValue.slice(3);

}

if (viewValue.length > 6 && viewValue.length <= 10) {

viewValue1 = "(" + viewValue.slice(0, 3) + ") " + viewValue.slice(3, 6) + "-" + viewValue.slice(6, 10);

} else {

viewValue1 = viewValue;

}

return viewValue1;

})

}

}

});

<form name="yourform" novalidate>

Name: <input ng-model="user.name" /><br /><br />

View: <input ng-model="user.age" name="age" evennum /><br />

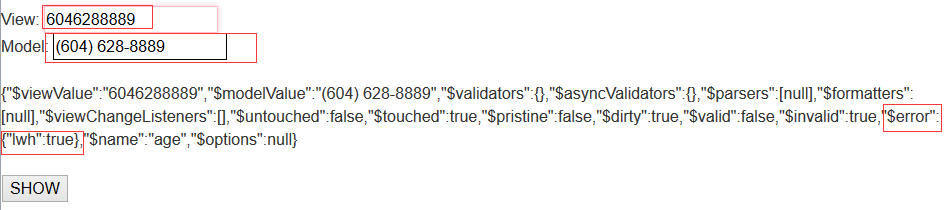
Model: <input ng-model="user.age" /><br />

<br />

{{yourform.age}}<br /><br />

<button ng-click="show()">SHOW</button><br />

</form>

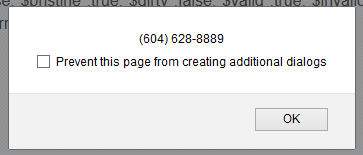


点击 “show” 按钮

$scope.show = function () {

alert($scope.user.age);

}



* **ng-message**

前提是：

<script type="text/javascript" src="angular-messages.min.js"></script>

var app = angular.module("myApp", ["ngRoute", "ngMessages"]);

对于显示信息可以使用 ng-message

<div ng-messages="myform.email.$error" role="alert" ng-messages-multiple>

<div ng-message="required">You did not enter your email address</div>

<div ng-message="minlength">Your email less than 5 haracters long</div>

<div ng-message="maxlength">Your email more than 10 haracters long</div>

<div ng-message="email">Your email is invalid</div>

</div>

<input type="checkbox" ng-model="mm.male" />Male<br />

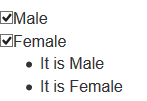
<input type="checkbox" ng-model="mm.female" />Male<br />

<ul ng-messages="mm" role="alertdialog" ng-messages-multiple>

<li ng-message="male">It is Male</li>

<li ng-message="female">It is FeMale</li>

</ul>



* **预防输入抖动**

通过 debounce 延迟模型更新

<input ng-model="country" ng-model-options="{debounce: 1000}" /> : {{ country }}

失去焦点马上更新，

<input ng-model="country" ng-model-options="

{ updateOn:'default blur',

debounce:{default:2000, blur:0}

}" /> : {{ country }}

ng-model-options="{updateOn:'default blur', debounce:{default:3000, blur:1000}}"

* **$timeout 预防抖动**

link: function (sc, el, atrs, pCtr) {

var ts;

sc.$watch("fname", function (nval, oval) {

// 不管队列里有没有都先取消，避免重复函数调用的队列

var ok = $timeout.cancel(ts);

console.log(ok);

// 然后再重新定义新的

ts = $timeout(function () {

console.log("link watch: " + nval + ":" + oval);

}, 2000);

});

}

* **$q 实现异步**

function okToGreet(name) {

return name === 'Robin Hood';

}

function asyncGreet(name) {

var deferred = $q.defer();-- 必须注入 $q

setTimeout(function () {

deferred.notify('About to greet ' + name + '.'); -- 这句会跳转到 notify

if (okToGreet(name)) {

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

deferred.resolve('Hello, ' + name + '!'); -- 这句会跳转到 success

deferred.notify('Complete to greet ' + name + '.'); -- 这句是不会被执行的

} else {

deferred.reject('Greeting ' + name + ' is not allowed.'); -- 跳转到 error

}

}, 2000);

return deferred.promise; - 必须返回 promise

}

$scope.count = 0;

var promise = asyncGreet('Robin Hood');

promise.then(

function (greeting) {

alert('Success: ' + greeting);

},

function (reason) {

alert('Failed: ' + reason);

},

function (update) {

$scope.count++;

alert('Got notification[' + $scope.count + ']: ' + update);

}

);

directive:

* **ng-class**

<style>

.fn-color {

background-color: #808080;

color: orange;

}

.bg-color {

background-color: yellow;

}

.td-odd {

font-style: italic;

text-decoration-color: #ff6a00;

text-decoration-line: line-through;

text-decoration-style: wavy;

}

.td-even {

font-style: normal;

text-decoration-color: green;

text-decoration-line: underline;

text-decoration-style: double;

}

</style>

第一种使用方式：直接赋值使用 , 记得一定不要忘记引号

<span ng-class='"fn-color"'>{{ name }}</span> 双引号字符串

<span ng-class="'fn-color'">{{ name }}</span> 单引号字符串

组合样式：'fn-color bg-color' 放在一个字符串里

<span ng-class="'fn-color bg-color'">{{ name }}</span>

组合样式注意错误： 'fn-color' 'bg-color' 将出错

<span ng-class="'fn-color' 'bg-color'">{{ name }}</span>

第二种使用方式： $scope 变量绑定

<span ng-class="selectClass">{{ name }}</span>

<input type="text" ng-model="selectClass" placeholder="Type: cust1 or cust2" value="" />

<input type="radio" ng-model="selectClass" value="bg-color" />

<input type="radio" ng-model="selectClass" value="fn-color" />

<input type="radio" ng-model="selectClass" value="td-odd" />

<input type="radio" ng-model="selectClass" value="td-even" />{{selectClass}}

ng-class 与 ng-model 绑定变量 selectClass

以下的效果同上：

<span class="{{selectClass}}">{{ name }}</span>

<input type="text" ng-model="selectClass" placeholder="Type: cust1 or cust2" value="" />

<input type="radio" ng-model="selectClass" value="bg-color" />

<input type="radio" ng-model="selectClass" value="fn-color" />

<input type="radio" ng-model="selectClass" value="td-odd" />

<input type="radio" ng-model="selectClass" value="td-even" />{{selectClass}}

class 与ng-model 需要使用 bind {{selectClass}}

容易犯的错误：

<span ng-class="{{selectClass}}">{{ name }}</span>

<span ng-class=" '{{selectClass}}' ">{{ name }}</span> 带引号的，只对初始化值有效， 虽然数据模型可以改变其值 'bg-color', 'td-odd' 等.. 但是并不会对view html效果进行更新。这和angular的工作原理有关。

第三种使用方式： 字符串数组

<span ng-class="['bg-color', 'fn-color', 'td-even']">{{ name }}</span>

<span ng-class="[bg, fn, odd]">{{ name }}</span>

<input type="text" ng-model="bg" value="" />

<input type="text" ng-model="fn" value="" />

<input type="text" ng-model="odd" value="" />

同理也可以：

$scope.selectClass = ["bg-color", "fn-color"];

<span ng-class="selectClass">{{ name }}</span>

也可以是数组里选择其中一个元素：

<span ng-class="['bg-color', 'fn-color', 'td-odd'][bg]">{{ name }}</span>

<input type="text" ng-model="bg" placeholder="Type: 0, 1, 2" value="" />

字符串数组可以来自于变量：

$scope.selectClass = ["bg-color", "fn-color", "td-odd"];

<span ng-class="selectClass[bg]">{{ name }}</span>

<input type="text" ng-model="bg" placeholder="Type: 0, 1, 2" value="" />

第四种使用方式： 字符串数组多选一的变异

<span ng-class="{'goo':'bg-color', 'bad':'fn-color', 'up':'td-odd'}[bg]">{{ name }}</span>

<input type="text" ng-model="bg" placeholder="Type: goo , bad, up" value="" />

等价于：

JSON["goo"] = 'bg-color';

JSON["bad"] = 'fn-color';

JSON["up"] = 'td-odd;

也可以是变量：

$scope.selectClass = {"a": "bg-color", "b": "fn-color", "c": "td-even", "d": "td-odd"};

<span ng-class="selectClass[bg]">{{ name }}</span>

<input type="text" ng-model="bg" placeholder="Type: a, b, c, d" value="" />

第五种使用方式：条件选择

<span ng-class="{'bg-color': bg, 'fn-color': fn, 'td-even': ev, 'td-odd': od }">{{ name }}</span>

<input type="text" ng-model="bg" placeholder="Type: a, b, c, d" value="" />

<input type="text" ng-model="fn" placeholder="Type: cust1 or cust2" value="" />

<input type="text" ng-model="od" placeholder="Type: cust1 or cust2" value="" />

<input type="text" ng-model="ev" placeholder="Type: cust1 or cust2" value="" />

null, undefined, "" 都为 false

其他值是 true

注意：

属性型的 css: 'td-odd' 等于 tdOdd , 可以使用不带引号的tdOdd, 但是不能使用不带引号的 td-odd

<span ng-class="{'bg-color': bg, 'fn-color': fn, 'td-even': ev, tdOdd: od }">{{ name }}</span>

HTML 的class 则不可以使用

<span class="tdOdd">{{ name }}</span> 错误的写法

必须是: <span class="td-odd">{{ name }}</span> 正确的HTML写法

可以使用checkbox来多选

<span ng-class='{ "bg-color": bg, "fn-color": fn, "td-even": ev, "td-odd": od }'> {{ name }}</span>

<input type="checkbox" ng-model="bg" value="1" />

<input type="checkbox" ng-model="fn" value="2" />

<input type="checkbox" ng-model="ev" value="3" />

<input type="checkbox" ng-model="od" value="4" />

注意checkbox 绑定到 $scope 变量的值是 true/false 而并非是 value="1,2,3,4"

第六种使用方式：多种条件组合

<span ng-class='["bg-color", cust, fn, {"td-even": ev, "td-odd": od}]'>{{ name }}</span>

<input type="text" ng-model="cust" value="1" />

<input type="text" ng-model="fn" value="2" />

<input type="checkbox" ng-model="ev" value="3" />

<input type="checkbox" ng-model="od" value="4" />

<p ng-class="[style4, {orange: warning}]">Using Array and Map Syntax</p>

<input ng-model="style4" placeholder="Type: bold, strike"><br>

<input type="checkbox" ng-model="warning"> warning (apply "orange" class)

代码不工作？原因未知

也可以使用表达式： 'cust' + style4

<p ng-class="['cust' + style4, {'orange': warning}]">Using Array and Map Syntax</p>

<input ng-model="style4" placeholder="Type: bold, strike"><br>

<input type="checkbox" ng-model="warning"> warning (apply "orange" class)

* **ng-bind-html -** [Expression](https://docs.angularjs.org/guide/expression) to evaluate
* **ng-bind-template -** template of form {{ expression }} to eval.

区别：

1） ng-bind-html 里面的是变量

2） ng-bind-template 里面的内容就是 html 内容， 而需要输出 $scope 变量， 则需要绑定输出

{{ scopeVar }}

Template: <span ng-bind-template="abcd + myColor"></span><br />

HTML : <span ng-bind-html=" 'abcd' + ' : ' + myColor"></span><br />

<br />

<select id="sel"

ng-model="myColor"

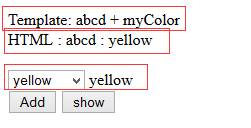
ng-options="color.name as color.name for color in colors">

<option value=""> - select -</option>

</select>

{{myColor}}

<br />



Template: <span ng-bind-template="abcd + {{ myColor|json }}"></span><br />

HTML : <span ng-bind-html=" 'abcd' + ' : ' + ( myColor|json )"></span><br />

<br />

<select id="sel"

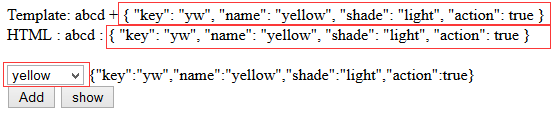
ng-model="myColor"

ng-options="color.name for color in colors">

<option value=""> - select -</option>

</select>

{{myColor}}



* **ng-cloak** 解决未准备好就输出 html的闪烁问题， 对于网速慢的页面可以预防输出的问题

需要在 CSS 加入样式才能起作用

[ng\:cloak],

[ng-cloak],

[data-ng-cloak],

[x-ng-cloak],

.ng-cloak,

.x-ng-cloak {

display: none !important;

}

* **ng-include** 才是真正把其他模板文件添加进来

test.html 的内容：

<html>

<head>

<script language="javascript" type="text/javascript">

function show1() {

alert("hello world this is show");

}

</script>

<link type="text/css" href="style.css" rel="stylesheet" />

<style>

.c1

{

background-color: pink;

font-size: 8px;

}

#span\_id {

font-size: 72px;

}

</style>

</head>

<body>

<span id="span\_id" class="c1 c2 c3" style="color: black;" >Hello World</span><br>

<span style="color: red; font-size: 36px;">Good Morning</span><br>

<input type="button" onclick="show1()" value="show" />

</body>

</html>

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

注意： 1）必须是字符串类型或者是$scope.var 变量

2）很多 ng-xxx 通常要求的是数据模型， 如果是字符串，则不要忘记引号 ‘’

3) 可以通过变量来切换页面

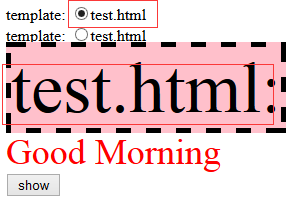
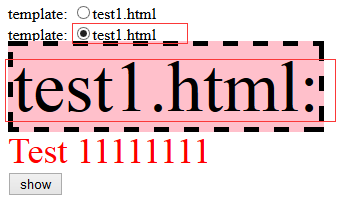
4) 模板里的 javascript 代码虽然被加载进来但将失效，

5) 会将 html ， head去掉

template: <input type="radio" name="rsw" ng-model="sw" ng-value="'test.html'" />{{sw}}<br />

template: <input type="radio" name="rsw" ng-model="sw" ng-value="'test1.html'" />{{sw}}<br />

<div ng-include="sw"></div>

**<div class="ng-scope" ng-include="sw">**

<script class="ng-scope" type="text/javascript" language="javascript">

function show1() {

alert("hello world this is show");

}

</script>

<link class="ng-scope" rel="stylesheet" href="style.css" type="text/css">

<style class="ng-scope">

.c1

{

background-color: pink;

font-size: 8px;

}

#span\_id {

font-size: 72px;

}

</style>

<span id="span\_id" class="c1 c2 c3 ng-scope" style="color: black;">test1.html:</span>

<br class="ng-scope">

<span class="ng-scope" style="color: red; font-size: 36px;">Test 11111111</span>

<br class="ng-scope">

<input class="ng-scope" type="button" value="show" onclick="show1()">

**</div>**

* **ng-show | ng-hide**
* **ng-if –** 区别是完全没有dom存在, 而ng-show|ng-hide是设置display:none属性

默认是 改变 display：none ， 也可以改写如下：

.ng-hide {

/\* this is just another form of hiding an element \*/

display: block!important;

position: absolute;

top: -9999px;

left: -9999px;

}

* **ng-disabled**
* **ng-readonly**

<input type="checkbox" ng-model="dis" />

<input ng-model="state" ng-readonly="dis" />

<input ng-model="city" ng-disabled="dis" /><br />



* **form - html** 标准表单元素. 有如下特性：

1. 如果表单只有一个元素， 按回车，会触发 submit, 触发onsubmit事件

function myform\_submit(e) {

alert("myform submit: " + e);

console.log(e);

return false;

}

<form action="index.html" onsubmit="return myform\_submit(200);" name="myform">

<input name="books" required /><br />

</form>

按回车以后先执行 onsubmit 事件， 页面是否会提交并跳转到action指定的页面，则取决于

onsubmit 事件是 true 则提交跳转， false则不提交并返回不动

常见错误：

1） onsubmit="myform\_submit(200);" 如果onsubmit只是执行函数， 则无论函数返回 false

页面照样提交跳转， 因为函数的返回值并没有被再次放回给onsubmit

正确做法是： onsubmit="return myform\_submit(200);"

或者铁定不提交（通常是做异步提交）: onsubmit="myform\_submit(200); return false;"

方法一：

<form action="index.html" onsubmit="myform\_submit(200); return false;" name="myform">

<input name="books" required /><br />

</form>

方法二：

<form action="javascript:void(0);" onsubmit="myform\_submit(200);" name="myform">

<input name="books" required /><br />

</form>

方法三： 会执行 myform\_submit, 但是前提函数不能有任何返回值，即不能有 return

function myform\_submit(e) {

alert("myform submit: " + e);

console.log(e);

return false;

}

<form action="javascript: myform\_submit(200);" name="myform">

<input name="books" required /><br />

</form>

否则会跳转到 : 

方法四： 会先执行 myform\_submit 再执行 show3

<form action="javascript:show3(300);" onsubmit="myform\_submit(200);" name="myform">

<input name="books" required /><br />

</form>

function show3(e) {

alert("show3:" + e);

return xxx; - action: javascript 不能有返回return，否者如上

}

function myform\_submit(e) {

alert("myform submit: " + e);

console.log(e);

return true;

}

2. 如果表单有两个以上元素，但是没有按钮的情况下， 按回车， 则没有任何动作

function myform\_submit(e) {

alert("myform submit: " + e);

console.log(e);

return true;

}

<form action="index.html" onsubmit="return myform\_submit(200);" name="myform">

<input name="books" required /><br />

<input name="titles" /><br />

</form>

3. 如果表单有两个以上元素，有带按钮 按回车，

情况一：

带有多个按钮<input type="button">：

<form action="index.html" onsubmit="return myform\_submit(200);" name="myform">

<input name="books" required /><br />

<input name="titles" /><br />

<input type="button" onclick="never\_run()" value="Show" />

<input type="button" onclick="show1()" value="Show" />

<input type="button" onclick="show2()" value="Show" />

</form>

无论表单录入元素有几个，都不会执行 <input type="button">

情况二：

<form action="index.html" onsubmit="return myform\_submit(200);" name="myform">

<input name="books" required /><br />

<input name="titles" /><br />

<input type="button" onclick="never\_run()" value="Show" />

<br />

<button onclick="show2()">Show2</button>

<button onclick="show1()">Show1</button>

</form>

1) 先执行第一个 <button onclick="show2()"> 其他button不执行，

2) 然后是onsubmit，

3) 最后是action , 取决于 onsubmit return true；

4）如果 button onclick 返回return false; 则执行完即终止，不再往下执行

function show2() {

alert("show2"); return false;

}

<form action="index.html" onsubmit="return myform\_submit(200);" name="myform">

<input name="books" required /><br />

<input name="titles" /><br />

<input type="button" onclick="never\_run()" value="Show" />

<br />

<button onclick="return show2();">Show2</button>

<button onclick="show1();return false;">Show1</button>

</form>

情况三：

<form action="index.html" onsubmit="return myform\_submit(200);" name="myform">

<input name="books" required /><br />

<input name="titles" /><br />

<input type="button" onclick="never\_run()" value="Show" />

<br />

<button onclick="show2();">Show2</button>

<button onclick="show1();">Show1</button>

<input type="submit" onclick="show3()" value="Submit" />

<input type="submit" onclick="show4()" value="Submit" />

</form>

混合按钮 <button> 与 <input type="submit"> 也是执行第一个按钮，谁先出现执行谁

注意：

1） submit button 上的 onsubmit 事件是不会被触发的， onsubmit事件只能放在form元素上

<input type="submit" onsubmit="show1()" value="Submit" />

button 或者 submit button 只有onclick事件

所以不会存在多个onsubmit

2） 点击按钮button或者submit，会执行本事的点击事件之外，会触发表单提交

3) 而点击 <input type="button"> 这种按钮，则不会提交表单

* **ng-form** - html 不允许 form 嵌套使用， 而angularjs的 ng-form可以嵌套使用

ng-form 的嵌套使用只是为了分组， 它没有提交action和onsubmit 事件功能

<form action="index.html" name="myform">

<ng-form action="f1.html" onsubmit="never\_run()" name="f1">

Books: <input ng-model="books" name="books" required /><br />

Email: <input ng-model="email" type="email" name="email" /><br />

<input type="submit" onsubmit="show1()" value="Submit" />

</ng-form><br />

f1: {{f1.$error}}

<br />

<ng-form action="f2.html" onsubmit="never\_run()" name="f2">

Title: <input ng-model="title" name="title" required /><br />

Quant: <input ng-model="number" type="number" name="number" /><br />

<input type="submit" onclick="show2()" value="Submit" />

</ng-form><br />

f2: {{f2.$error}}

<br />

<br />

<input type="submit" onsubmit="show3()" value="Submit" />

</form>

ng-form 的action="f2.html" onsubmit="never\_run()" 都不会被执行

* **ng-repeat**

有重要的属性可以使用：

Special properties are exposed on the local scope of each template instance, including:

| **Variable** | **Type** | **Details** |
| --- | --- | --- |
| $index | [number](https://docs.angularjs.org/) | iterator offset of the repeated element (0..length-1) |
| $first | [boolean](https://docs.angularjs.org/) | true if the repeated element is first in the iterator. |
| $middle | [boolean](https://docs.angularjs.org/) | true if the repeated element is between the first and last in the iterator. |
| $last | [boolean](https://docs.angularjs.org/) | true if the repeated element is last in the iterator. |
| $even | [boolean](https://docs.angularjs.org/) | true if the iterator position $index is even (otherwise false). |
| $odd | [boolean](https://docs.angularjs.org/) | true if the iterator position $index is odd (otherwise false). |

.first {

background-color: pink;

}

.last {

background-color: orange;

}

.even {

color: blue;

}

.odd {

color: green;

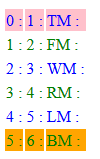
}

$scope.hob = [

{ id: 1, title: "TM" },

{ id: 2, title: "FM" },

{ id: 3, title: "WM" },

 { id: 4, title: "RM" },

{ id: 5, title: "LM" },

{ id: 6, title: "BM" }

];

<table>

<tr ng-repeat="hhh in hob"

ng-class="{'first':$first, 'last':$last, 'even':$even, 'odd':$odd}">

<td>{{$index}} : </td>

<td>{{hhh.id}} : </td>

<td>{{hhh.title}} : </td>

</tr>

</table>

$scope.hob = [

{ id: 1, title: "TM" },

{ id: 2, title: "FM" },

{ id: 3, title: "WM" },

{ id: 4, title: "RM" },

{ id: 5, title: "LM" },

{ id: 5, title: "BM" }

];

<table>

<tr ng-repeat="hhh in hob track by hhh.id" – id 有重复值，所以会出错

ng-class="{'first':$first, 'last':$last, 'even':$even, 'odd':$odd}">

<td>{{$index}} : </td>

<td>{{hhh.id}} : </td>

<td>{{hhh.title}} : </td>

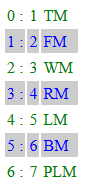
</tr>

</table>

* **ng-class-odd**
* **ng-class-even**

**需要结合 ng-repeat来一起使用**

.even {

 color: blue;

background-color: #cccccc;

}

.odd {

color: green;

background-color: #fffff;

}

<table cellpadding="2">

<tr ng-repeat="hhh in hob"

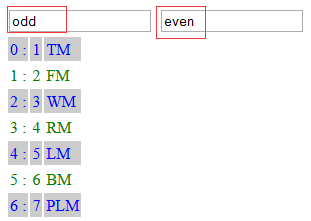
ng-class-even="'even'" ng-class-odd="'odd'">

<td>{{$index}} : </td>

<td>{{hhh.id}}</td>

<td>{{hhh.title}}</td>

</tr>

</table>

奇偶数是从 1 开始算起， 不是从 0 开始的， $index 则是从 0 开始的

也可以这样写： class="ng-class-even: expression;"

<input ng-model="seven" placeholder="input even" />

<input ng-model="sodd" placeholder="input odd" /><br />

<table cellpadding="2">

<tr ng-repeat="hhh in hob"

class="ng-class-even:seven; ng-class-odd:sodd">

<td>{{$index}} : </td>

<td>{{hhh.id}}</td>

<td>{{hhh.title}}</td>

</tr>

</table>

* **input - input[text]**

**<input**

**ng-model="string"**

**[name="string"]**

**[required="string"]**

**[ng-required="boolean"]**

**[ng-minlength="number"]**

**[ng-maxlength="number"]**

**[ng-pattern="string"]**

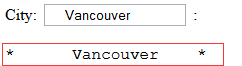
**[ng-change="string"]**

**[ng-trim="boolean"]> -** 默认是 true ，即自动裁剪前后空白字符

**...**

**</input>**

City: <input ng-model="ccc" ng-trim="false" ng-init="ccc=' Vancouver '" /> : <pre>\*{{ccc}}\*</pre>

ng-trim="false" 保留前后空白 ****

注意：1） ng-init 的执行顺序在 controller 之后， 所以 ng-init 的值会修改 $scope 的值

$scope.ccc = " Toronto ";

ng-init="ccc=' Vancouver '" - 输出结果仍然是 Vancouver

* **input[checkbox]**

**<input type="checkbox"**

**ng-model="string"**

**[name="string"]**

**[ng-true-value="expression"]**

**[ng-false-value="expression"]**

**[ng-change="string"]>**

.ck-red {

color: red;

}

$scope.hob = [

{ id: 1, title: "TM" },

{ id: 2, title: "FM" },

{ id: 3, title: "WM" },

{ id: 4, title: "RM" },

{ id: 5, title: "LM" },

{ id: 6, title: "BM" },

{ id: 7, title: "PLM" }

];

$scope.myhob = {1:true, 3:true, 9:true}; - 9不存在，但是会一直保留

<span ng-repeat="hhh in hob">

<label ng-class="{'ck-red':myhob[hhh.id]}">

<input type="checkbox"

ng-click="show()"

ng-model="myhob[hhh.id]" />

{{ ($index + 1) + " : " + hhh.title}}

</label>

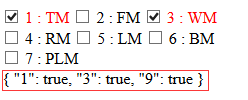
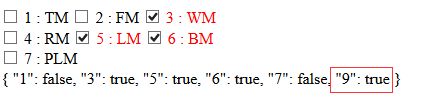
<br ng-if="($index+1)%3==0" />

</span>

<br />

{{myhob|json}}<br />

<br />

* **ng-checked与 input[checkbox]**

$scope.hob = [

{ id: 1, title: "TM" },

{ id: 2, title: "FM" },

{ id: 3, title: "WM" },

{ id: 4, title: "RM" },

{ id: 5, title: "LM" },

{ id: 6, title: "BM" },

{ id: 7, title: "PLM" }

];

//$scope.myhob = { 1: true, 3: true, 9: true };

$scope.myhob = {};

$scope.lists = [1, 5];

<span ng-repeat="hhh in hob">

<label ng-class-odd="'odd'" ng-class-even="'even'" ng-class="{'ck-red':myhob[hhh.id]}">

<input type="checkbox"

ng-click="show()"

ng-model="myhob[hhh.id]"

ng-checked="lists.indexOf(hhh.id)>=0"

ng-init="myhob[hhh.id]=lists.indexOf(hhh.id)>=0?true:false" />-将初始值同步到 $scope

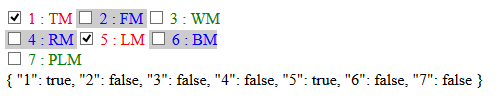
{{ ($index + 1) + " : " + hhh.title}}

</label>

<br ng-if="($index+1)%3==0" /> - 每行三个

</span>

{{myhob|json}}<br />



* **ng-options 与 <select>**

**格式1：** label **for** value **in** array

label : 是现实在列表上的值

value： 是对象数值的枚举出来的单个对象， 默认也是返回这个单个对象

array: 是对象数组

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

app.controller("sales", function ($scope, $location) {

$scope.colors = [

{ key: "bk", name: "black", shade: "dark", action: false },

{ key: "wt", name: "white", shade: "light", action: true },

{ key: "rd", name: "red", shade: "dark" },

{ key: "bl", name: "blue", shade: "dark" },

{ key: "yw", name: "yellow", shade: "light", action: true },

{ key: "or", name: "orange", shade: "dark" },

{ key: "nv", name: "navy", shade: "light", action: true }

];

$scope.myColor = $scope.colors[1];

$scope.add = function () {

$scope.colors.push({ key: "new", name: "new color" });

}

});

function show() {

alert($("#sel").val());

console.log($("#sel").val())

}

<select id="sel" ng-model="myColor" ng-options="color.name for color in colors">

<option value=""> - select -</option>

</select>

{{myColor}}

<br />

<button ng-click="add()">Add</button>

<input type="button" onclick="show()" value="show" />

html ：

<select id="sel" class="ng-valid ng-dirty ng-valid-parse ng-touched" ng-options="color.name for color in colors" ng-model="myColor">

<option class="" value=""> - select -</option>

<option value="0" label="black">black</option>

<option value="1" selected="selected" label="white">white</option>

<option value="2" label="red">red</option>

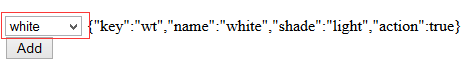
<option value="3" label="blue">blue</option>

<option value="4" label="yellow">yellow</option>

<option value="5" label="orange">orange</option>

<option value="6" label="navy">navy</option>

</select>



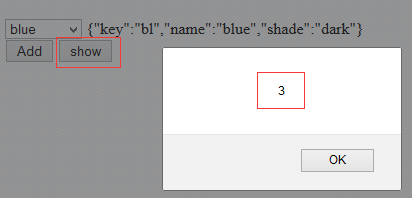
分析：

1）select ng-model ng-options 与传统的html select 不同了

ng-model="myColor" 返回的是对象数组里，被选择的对象

2） 默认的 <option value> 是按序号 "0" 开始递增

3） 如果我们使用 DOM 获取 select 的值 ： 则是 option value="3" 的值。



**格式2：**select **as** label **for** value **in** array

select: 选择以后返回的值

app.controller("sales", function ($scope, $location) {

$scope.colors = [

{ key: "bk", name: "black", shade: "dark", action: false },

{ key: "wt", name: "white", shade: "light", action: true },

{ key: "rd", name: "red", shade: "dark" },

{ key: "bl", name: "blue", shade: "dark" },

{ key: "yw", name: "yellow", shade: "light", action: true },

{ key: "or", name: "orange", shade: "dark" },

{ key: "nv", name: "navy", shade: "light", action: true }

];

$scope.myColor = "yw:yellow";

$scope.add = function () {

$scope.colors.push({ key: "new", name: "new color" });

}

});

<select id="sel"

ng-model="myColor"

ng-options="(color.key + ':' + color.name) as color.name for color in colors">

<option value=""> - select -</option>

</select>

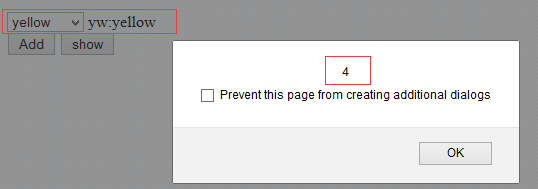
{{myColor}}

<br />

<button ng-click="add()">Add</button>

<input type="button" onclick="show()" value="show" />





<select id="sel" class="ng-valid ng-dirty ng-valid-parse ng-touched"

ng-options="(color.key + ':' + color.name) as color.name for color in colors"

ng-model="myColor">

<option class="" value=""> - select -</option>

<option value="0" label="black">black</option>

<option value="1" label="white">white</option>

<option value="2" label="red">red</option>

<option value="3" label="blue">blue</option>

<option value="4" selected="selected" label="yellow">yellow</option>

<option value="5" label="orange">orange</option>

<option value="6" label="navy">navy</option>

</select>

分析：

1） ng-model="myColor" 返回的值是： color.key + ':' + color.name

**格式3：**label **for** value **in** array **track by** trackExpr

trackExpr: 修改 option value

app.controller("sales", function ($scope, $location) {

$scope.colors = [

{ key: "bk", name: "black", shade: "dark", action: false },

{ key: "wt", name: "white", shade: "light", action: true },

{ key: "rd", name: "red", shade: "dark" },

{ key: "bl", name: "blue", shade: "dark" },

{ key: "yw", name: "yellow", shade: "light", action: true },

{ key: "or", name: "orange", shade: "dark" },

{ key: "nv", name: "navy", shade: "light", action: true }

];

$scope.myColor = $scope.colors[1];

$scope.add = function () {

$scope.colors.push({ key: "new", name: "new color" });

}

});

<select id="sel" ng-model="myColor"

ng-options="color.name for color in colors track by color.key ">

<option value=""> - select -</option>

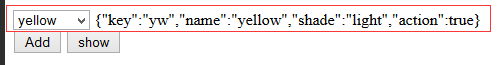
</select>

{{myColor}}

<br />

<button ng-click="add()">Add</button>

<input type="button" onclick="show()" value="show" />



<select id="sel" class="ng-valid ng-dirty ng-valid-parse ng-touched"

ng-options="color.name for color in colors track by color.key "

ng-model="myColor">

<option class="" value=""> - select -</option>

<option value="bk" label="black">black</option>

<option value="wt" selected="selected" label="white">white</option>

<option value="rd" label="red">red</option>

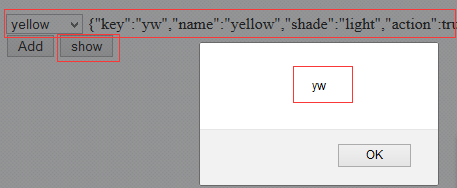
<option value="bl" label="blue">blue</option>

<option value="yw" label="yellow">yellow</option>

<option value="or" label="orange">orange</option>

<option value="nv" label="navy">navy</option>

</select>



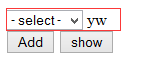
分析：

1） ng-model="myColor" 返回的值是：仍热是对象

{"key":"yw","name":"yellow","shade":"light","action":true}

2）html DOM 的返回值： track by color.key 即 color.key

3) 注意 select **as** 不能与 **track by** trackExpr 同时使用，会造成选择不了同步不了的问题。



**格式4：**select **as** label **group by** groupExpr **for** value **in** array

<select id="sel"

ng-model="myColor"

ng-options="color.key+':'+color.name as color.name group by color.shade[0] for color in colors">

<option value=""> - select -</option>

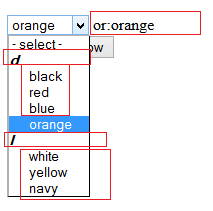
</select>

{{myColor}}

<br />

<button ng-click="add()">Add</button>

<input type="button" onclick="show()" value="show" />



<select id="sel" class="ng-valid ng-dirty ng-valid-parse ng-touched"

ng-options="color.key+':'+color.name as color.name group by color.shade[0] for color in colors"

ng-model="myColor">

<option class="" value=""> - select -</option>

<optgroup label="d">

<option value="0" label="black">black</option>

<option value="2" label="red">red</option>

<option value="3" label="blue">blue</option>

<option value="5" selected="selected" label="orange">orange</option>

</optgroup>

<optgroup label="l">

<option value="1" label="white">white</option>

<option value="4" label="yellow">yellow</option>

<option value="6" label="navy">navy</option>

</optgroup>

</select>

**格式5：**label **group by** groupExpr **for** value **in** array **track by** trackExpr

<select id="sel"

ng-model="myColor"

ng-options="color.name group by color.shade for color in colors track by color.key">

<option value=""> - select -</option>

</select>

{{myColor}}

<br />

<button ng-click="add()">Add</button>

<input type="button" onclick="show()" value="show" />

app.controller("sales", function ($scope, $location) {

$scope.colors = [

{ key: "bk", name: "black", shade: "dark", action: false },

{ key: "wt", name: "white", shade: "light", action: true },

{ key: "rd", name: "red", shade: "dark" },

{ key: "bl", name: "blue", shade: "dark" },

{ key: "yw", name: "yellow", shade: "light", action: true },

{ key: "or", name: "orange", shade: "dark" },

{ key: "nv", name: "navy", shade: "light", action: true }

];

$scope.myColor = $scope.colors[1];

$scope.add = function () {

$scope.colors.push({ key: "new", name: "new color" });

}

});

**HTML输出：**

<select id="sel" class="ng-pristine ng-valid ng-touched"

ng-options="color.name group by color.shade for color in colors track by color.key"

ng-model="myColor">

<option class="" value=""> - select -</option>

<optgroup label="dark">

<option value="bk" label="black">black</option>

<option value="rd" label="red">red</option>

<option value="bl" label="blue">blue</option>

<option value="or" label="orange">orange</option>

</optgroup>

<optgroup label="light">

<option value="wt" selected="selected" label="white">white</option>

<option value="yw" label="yellow">yellow</option>

<option value="nv" label="navy">navy</option>

</optgroup>

</select>

**格式6：可以适当使用 filter: orderBy**

<select id="sel"

ng-model="myColor"

ng-options="color.name group by color.shade

for color in colors|orderBy:['-shade','name'] track by color.key">

<option value=""> - select -</option>

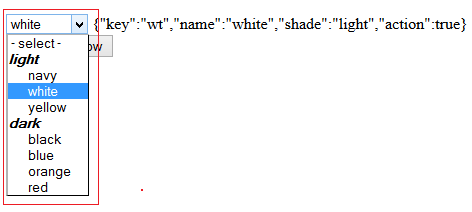
</select>

{{myColor}}

<br />

<button ng-click="add()">Add</button>

<input type="button" onclick="show()" value="show" />



**格式7：**label disable when boolExpr **for** value **in** array

**格式8：传统机制：**

<input type="checkbox"

onclick="show1(this.value)"

ng-model="myhob[hhh.id]"

ng-checked="lists.indexOf(hhh.id)>=0"

value="{{hhh.id}}"/>

* **ng-value 与 <select>**

ng-value 只能用于 <select> - <option> 或者 input[radio]

ng-value 不能用于给其他赋值如：input[text] input[checkbox]

input[text] ng-model="sc\_var" 可以使用 ng-init=" sc\_var='xxxx' "

$scope.colors = [

{ key: "bk", name: "black", shade: "dark", action: false },

{ key: "wt", name: "white", shade: "light", action: true },

{ key: "rd", name: "red", shade: "dark" },

{ key: "bl", name: "blue", shade: "dark" },

{ key: "yw", name: "yellow", shade: "light", action: true },

{ key: "or", name: "orange", shade: "dark" },

{ key: "nv", name: "navy", shade: "light", action: true }

];

<select id="yycolor" ng-model="yourcolor">

<option ng-repeat="color in colors" ng-value="color.key" ng-bind="color.name"></option>

</select>

{{yourcolor}}



<option class="ng-binding ng-scope"

ng-bind="color.name"

ng-value="color.key"

ng-repeat="color in colors"

value="bl">blue</option>

注意：

1) ng-value="" 或者 ng-value=不存在的 ， 则取 ng-bind 的值。

<select id="yycolor" ng-model="yourcolor">

<option ng-repeat="color in colors"

ng-value="color.keje"

ng-bind="color.name">

</option>

</select>

{{yourcolor}}



2）ng-value="color" JSON对象

<select id="yycolor" ng-model="yourcolor">

<option ng-repeat="color in colors" ng-value="color" ng-bind="color.name"></option>

</select>

{{yourcolor}}

注意返回的并不是 JSON 对象， 而是字符串[object Object]

<option class="ng-binding ng-scope"

ng-bind="color.name"

ng-value="color"

ng-repeat="color in colors"

value="[object Object]">black</option> - 取值就都变成了字符串"[object Object]"

* **ng-selected 与 <select> - <option>**

$scope.lists = ["bk", "yw", "or"];

<select id="yycolor" ng-model="yourcolor">

<option ng-repeat="color in colors"

ng-value="color.key"

ng-bind="color.name"

ng-selected="lists.indexOf(color.key)>=0"

></option>

</select>



$scope.myfar = "yellow";

<select id="yycolor" ng-model="yourcolor">

<option ng-repeat="color in colors"

ng-value="color.key"

ng-bind="color.name"

ng-selected="color.name==myfar"

></option>

</select>

{{yourcolor}}

****

* **input[radio]**

在传统的html里

function showSex() {

alert($("input:radio[name='sex']:checked").val());

}

function setSex(val) {

$("input:radio[name='sex']").attr("checked", false);

$("input:radio[name='sex'][value='" + val + "']").attr("checked", true);

}

<input type="radio" name="sex" id="sex1" value="Male" />Male

<input type="radio" name="sex" id="sex2" value="FeMale" />FeMale

<input type="radio" name="sex" id="sex3" value="TransSex" />TransSex

<br />

<input type="button" onclick="showSex()" value="Get Gender" />

<input type="button" onclick="setSex('Male')" value="Set Male" />

分析：

1. 在传统的html 里， radio是以 name属性来分组的
2. 选中某个选项是使用 checked 而不是 selected
3. 如上操作 :checked 来获取选中的值
4. 虽然可以为同一组 radio 赋予多个 checked=true， 实际上只有一个起作用，以页面显示选中的为准。

<input type="radio"

ng-model="string"

value="string"

[name="string"]

[ng-change="string"]

ng-value="string">

$scope.showG = function () {

alert($scope.gender);

console.log($scope.gender);

}

$scope.setG = function (v) {

$scope.gender = v;

}

Cust: <input ng-model="custSex" /><br />

<input type="radio" name="sex1" id="sex1" ng-model="gender" ng-value="custSex" value="Male" />{{custSex}}

<input type="radio" name="sex2" id="sex2" ng-model="gender" ng-value="'MM'" value="FeMale" />FeMale

<input type="radio" name="sex3" id="sex3" ng-model="gender" ng-value="'SS'" value="TransSex" />TransSex

<br />

<button ng-click="showG()">Show</button>

<button ng-click="setG(custSex)">Set</button>

分析

1）如果是用 ng-model来控制 radio, 则使用 ng-model来分组radio , 而不是使用name

<input type="radio" name="sex1" id="sex1" ng-model="gender" ng-value="custSex" value="Male" />{{custSex}}

<input type="radio" name="sex2" id="sex2" ng-model="gender" ng-value="'MM'" value="FeMale" />FeMale

<input type="radio" name="sex3" id="sex3" ng-model="gender1" ng-value="'SS'" value="TransSex" />TransSex

<input type="radio" name="sex3" id="sex3" ng-model="gender2" ng-value="'BB'" value="BlankSex" />BlankSex

此时分两组： gender， gender1



2）ng-value是表达式, 如果是字符串记得引号 ‘MM’, ‘SS’, ‘BB’

3) ng-value 如果是$scope变量 则直接写变量名

4）value则是字符值，不需要引号， 如果是绑定，则记得使用 {{ scope\_var }}

5) value 是字符值， ng-value 的值优先级高，而且会覆盖 value的值

6) value 是绑定， ng-value是字符串， 则 value的值优先级别高，

7）value 是绑定， ng-value也是变量，则ng-value的值有限级别高。

8）当我们获取 ng-value的数据模型的值， 实际上是获取 value的值。

Cust: <input ng-model="custSex" /><br />

Cust1: <input ng-model="custSex1" /><br />

Cust2: <input ng-model="custSex2" /><br />

Cust3: <input ng-model="custSex3" /><br />

<input type="radio" name="sex1" ng-model="gender" ng-value="custSex" value="Male" />{{custSex}}

<input class="ng-pristine ng-untouched ng-valid" type="radio"

ng-value="custSex"

ng-model="gender"

name="sex1"

value="AAA">

<input type="radio" name="sex2" ng-model="gender" ng-value="'MM'" value="FeMale" />FeMale

<input class="ng-pristine ng-untouched ng-valid" type="radio"

value="MM"

ng-value="'MM'"

ng-model="gender" name="sex2">

<input type="radio" name="sex3" ng-model="gender" ng-value="'SS'" value="{{custSex1}}" />TransSex

<input class="ng-pristine ng-untouched ng-valid" type="radio"

value="BBB" ng-value="'SS'" ng-model="gender" name="sex3">

当我们获取 ng-value的数据模型的值， 实际上是获取 value的值。

<input type="radio" name="sex3" ng-model="gender" ng-value="custSex3" value="{{custSex2}}" />BlankSex

<input class="ng-pristine ng-untouched ng-valid" type="radio"

ng-value="custSex3"

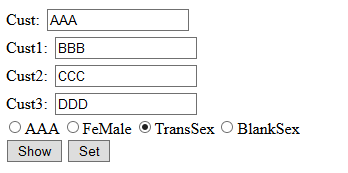
ng-model="gender" name="sex3"

value="DDD">

<br />

<button ng-click="showG()">Show</button>

<button ng-click="setG(custSex)">Set</button>



ng-value 的数据模型不仅仅支持字符串，也支持各种对象

$scope.custSex1 = { id: 100, name: "William" };

$scope.custSex2 = [50, 80, 12];

$scope.showG = function () {

alert($scope.gender);

console.log($scope.gender);

}

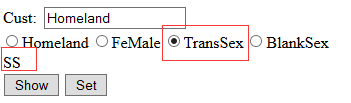
Cust: <input ng-model="custSex" /><br />

<input type="radio" name="sex1" ng-model="gender" ng-value="custSex" value="Male" />{{custSex}}

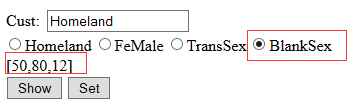
<input type="radio" name="sex2" ng-model="gender" ng-value="'MM'" value="FeMale" />FeMale

<input type="radio" name="sex3" ng-model="gender" ng-value="'SS'" value="{{custSex1}}" />TransSex

--此时 value绑定对象， 则ng-value的优先级高



<input type="radio" name="sex3" ng-model="gender" ng-value="custSex2" />BlankSex



<br />

{{gender}}

<br />

<button ng-click="showG()">Show</button>

<button ng-click="setG(custSex)">Set</button>

<input type="radio" name="sex3" ng-model="gender" value="{{custSex1}}" />TransSex

如果没有ng-model， 则 value也支持对象

则输出

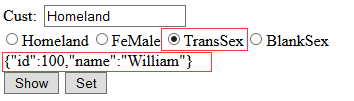
<input class="ng-valid ng-dirty ng-touched ng-valid-parse"

type="radio"

value="{"id":100,"name":"William"}"

ng-model="gender"

name="sex3">



使用代码选择:

$scope.custSex1 = { id: 100, name: "William" };

$scope.custSex2 = [50, 80, 12];

$scope.showG = function () {

alert($scope.gender);

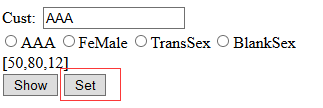
console.log($scope.gender);

}

$scope.setG = function (v) {

$scope.gender = [50, 80, 12];

}



虽然值相同也选择不了, 但是数据模型的值确实改变了, 但是没有选中

$scope.setG = function (v) {

$scope.gender = { id: 100, name: "William" };

$scope.gender = '{ id: 100, name: "William" }';

$scope.gender = $scope.custSex1;

}

<input type="radio" name="sex3" ng-model="gender" value="{{custSex1}}" />TransSex

都选不上

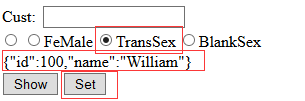
$scope.custSex1 = { id: 100, name: "William" };

$scope.setG = function (v) {

$scope.gender = $scope.custSex1;

}

<input type="radio" name="sex3" ng-model="gender" ng-value="custSex1" />TransSex



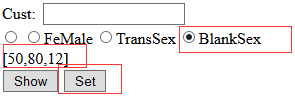
可以选上.

$scope.setG = function (v) {

$scope.gender = $scope.custSex2;

}

<input type="radio" name="sex3" ng-model="gender" ng-value="custSex2" />BlankSex



$scope.setG = function () {

$scope.gender = $scope.custSex2;

}

$scope.setG1 = function () {

$scope.gender = "MM";

}

Cust: <input ng-model="custSex" /><br />

<input type="radio" name="sex1" ng-model="gender" ng-value="custSex" value="Male" />{{custSex}}

<input type="radio" name="sex2" ng-model="gender" ng-value="'MM'" value="FeMale" />FeMale

<input type="radio" name="sex3" ng-model="gender" ng-value="custSex1" />TransSex

<input type="radio" name="sex3" ng-model="gender" ng-value="custSex2" />BlankSex

<br />

{{gender}}

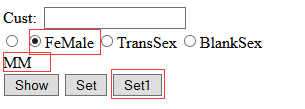
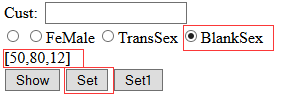
<br />

<button ng-click="showG()">Show</button>

<button ng-click="setG()">Set</button>

<button ng-click="setG1()">Set1</button>

<br />



* **ng-list -** 通过设置隔离符号，分割返回字符串数值，默认隔离符号是 ，

$scope.hob = [

{ id: 1, title: "TM" },

{ id: 2, title: "FM" },

{ id: 3, title: "WM" },

{ id: 4, title: "RM" },

{ id: 5, title: "LM" },

{ id: 6, title: "BM" },

{ id: 7, title: "PLM" }

];

$scope.ccc = ["TM", "WM"];

Horby: <input ng-model="ccc" **ng-list** /> : <pre>\*{{ccc|json}}\*</pre>

<br />

<span ng-repeat="hhh in hob">

<label ng-class-odd="'odd'" ng-class-even="'even'" ng-class="{'ck-red':myhob[hhh.id]}">

<input type="checkbox"

ng-model="myhob[hhh.id]"

ng-checked="ccc.indexOf(hhh.title)>=0"

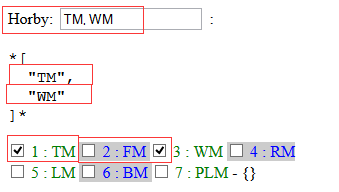
/>

{{ ($index + 1) + " : " + hhh.title}}

</label>

<br ng-if="($index+1)%4==0" />

</span> - {{myhob|json}}<br />



* **ng-switch**

<ANY ng-switch="expression"> 或者 ng-switch on="expression"

<ANY ng-switch-when="matchValue1">...</ANY>

<ANY ng-switch-when="matchValue2">...</ANY>

<ANY ng-switch-default>...</ANY>

</ANY>

$scope.colors = [

{ key: "bk", name: "black", shade: "dark", action: false },

{ key: "wt", name: "white", shade: "light", action: true },

{ key: "rd", name: "red", shade: "dark" },

{ key: "bl", name: "blue", shade: "dark" },

{ key: "yw", name: "yellow", shade: "light", action: true },

{ key: "or", name: "orange", shade: "dark" },

{ key: "nv", name: "navy", shade: "light", action: true }

];

$scope.fullName = "William Liu";

$scope.show = function () {

console.log($scope.fullName);

}

<select ng-model="sel\_temp" ng-options="color.key as color.name for color in colors"></select>

{{sel\_temp | json}}

<div ng-switch on="sel\_temp">

<div ng-switch-when="bk"><span ng-include="'test.html'"></span></div>

<div ng-switch-when="wt"><span ng-include="'test1.html'"></span></div>

<div ng-switch-when="rd">

<input ng-model="fullName" /> - 每次选中都创建新的继承的 scope

<input type="button" ng-click="show()" value="FullName" /> - 方法没有定义，则执行父的方法，

所以输出的也是父的 fullName

</div>

<div ng-switch-default>Default</div>

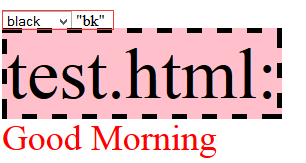
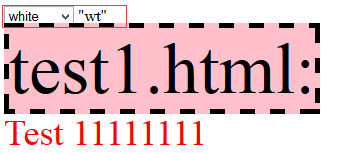
</div>

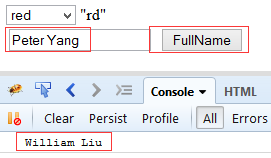
分析：

1） ng-switch-when 不符合条件的内容会从DOM 移除

2） ng-switch-when 里面的内容会自动创建新的scope, 新的scope 会继承自 parent, 而且是每次被选中时

都会创建新的继承自parent的 scope, 这是一个不容易发现的陷阱



方法没有定义，则执行父的方法，输出的也是父的 fullName

switch back again 

每次选回来，创建新的继承的scope

3) 如果不想ng-switch-when 每次创建新的scope， 解决方案是使用 $parent

<div ng-switch on="sel\_temp">

<div ng-switch-when="bk"><span ng-include="'test.html'"></span></div>

<div ng-switch-when="wt"><span ng-include="'test1.html'"></span></div>

<div ng-switch-when="rd">

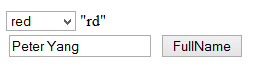
<input ng-model="$parent.fullName" />

<input type="button" ng-click="show()" value="FullName" />

</div>

<div ng-switch-default>Default</div>

</div>

 switch back again 

* **树形迭代：**

app.directive("menus", function ($compile) {

return {

restrict: "E",

replace: true,

transclude: true,

scope: {

data: "="

},

template: "<ul><li ng-repeat='menu in data'>{{menu.title}}<span ng-if='menu.menus.length>0'><menus data='menu.menus'></menus></span></li></ul>",

关键是一下函数， 否则会提示说: 太多的迭代错误

compile: function (el) {

console.log(el);

var contents = el.contents().remove();

return function (scope, el) {

$compile(contents)(scope, function (clone) {

el.append(clone);

});

};

}

}

});

$scope.menus = [

{ title: 'Menu1' },

{ title: "Menu2",

menus: [

{ title: "Menu22",

menus: [

{ title: 'Menu222' },

{ title: 'menu223' }

]

}

]

},

{ title: 'Menu3',

menus: [

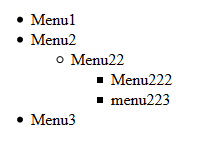
{ title: 'menu33' }

]

}

];

<menus data="menus"></menus>



app.directive("menus", function ($compile) {

return {

restrict: "E",

replace: true,

transclude: true,

scope: {

data: "="

},

template: "<ul></ul>",

compile: function (el) {

var contents = $(el).empty();

return function (scope, el) {

var ell = $compile( "<li ng-repeat='menu in data'>{{menu.title}}

<menus data='menu.menus'></menus>

</li>")(scope);

el.append(ell);

/\*

$compile(contents)(scope, function (clone) {

el.append(clone);

});

\*/

};

}

}

});

也可以这样写， 关键是理解 $compile 函数的作用：对 html 元素进行 scope关联编译.