



## 第七讲 指令

# 教学目标

---

1. 自定义指令
2. 创建复杂指令
3. 高级指令

1

自定义指令

# 序

## 引导案例-理解指令

```
app.directive( 'say' ,function(){  
    var json= {  
        restrict: 'E' , //做为元素名使用  
        template: '<div>hello!</div>' , //模板  
    };  
    return json;  
});
```

<say> **<div>hello!</div>** </say>

指令(directive)的作用： 自定义标签

# 序

## 指令是什么？

- (1) 指令的目的是**用来自定义HTML标签**，指令是一种标记，用来告诉HTML Parser “这里需要编译”
- (2) **指令本质上就是AngularJS扩展具有自定义功能的HTML元素的途径。**
- (3) 通过AngularJS模块API中的**.directive()**方法，我们可以通过传入一个字符串和一个函数来注册一个新指令。

# ①

## 创建一个自定义指令

调用`module.directive()`方法，自定义指令，传入指令和工厂函数。

```
var app=angular.module("exampleApp", [])  
  
app.directive("unorderedList", function () {  
    .....  
})
```

↓  
新指令名称

↓  
创建指令的函数  
称之为链接函数

Demo:Listing 02.html

## ②

## 应用自定义指令

调用module.**directive**方法，传入指令和工厂函数

```
var app=angular.module("exampleApp", [])
app.directive("unorderedList", function () {
    .....
})
```

↓  
第二个单词首字母大写

↙  
<div unordered-list="products"></div>

应用时会使用 “ - ” 连接，字母则为小写

Demo:Listing 02.html

# 3

## 从作用域获取数据

与angularJS控制器不同，指令并不声明对\$scope服务的依赖。传入的是指令被应用到视图的控制器所创建的作用域。

```
.directive("unorderedList", function () {  
  return function (scope, element, attrs) {  
    .....  
  }  
})
```

作用域对象

html对象

属性集合

按名称索引

视图的控制器所创建的作用域

Demo:Listing 03.html



# 3

## 从作用域获取数据

```
var app=angular.module("exampleApp", [])
app.directive("unorderedList", function () {
  return function (scope, element, attrs) {
    var data = scope[attrs["unorderedList"]];
    if (angular.isArray(data)) {
      for (var i = 0; i < data.length; i++) {
        console.log("Item: " + data[i].name);
      }
    }
  }
})
```

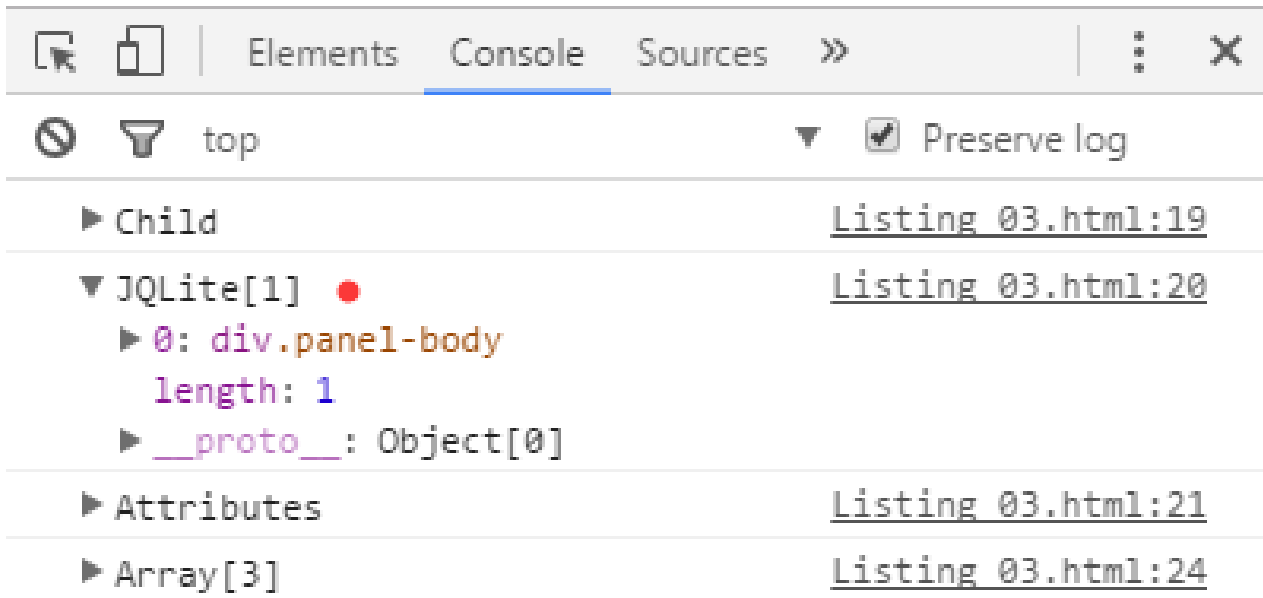
Demo:Listing 04.html

## 4

# 生成HTML

AngularJS包含了一个裁剪过的jQuery，称为“jqLite”

```
console.log(element);
```



Demo:Listing 05.html

# 4

## 生成HTML

```
.directive("unorderedList", function () {  
    return function (scope, element, attrs) {  
        .....  
    }  
})
```

AngularJS创建新元素：

```
var listElem = angular . element ("<ul>");
```

AngularJS插入新元素：

```
element . append ( listElem );
```

↓  
**html对象**

Demo:Listing 05.html

# 4

## 生成HTML

```
var listElem = angular.element("<ul>");  
element.append(listElem);  
for (var i = 0; i < data.length; i++) {  
  var li_list=angular.element('<li>');  
  listElem.append(li_list.text(data[i].name));  
}
```

Demo:Listing 05.html

## 5

## 添加属性

Demo:Listing 06.html

```
.directive("unorderedList", function () {  
    return function (scope, element, attrs) {  
        var data = scope[attrs["unorderedList"]];  
        var propertyName = attrs["listProperty"];  
        if (angular.isArray(data)) {  
            var listElem = angular.element("<ul>");  
            element.append(listElem);  
            for (var i = 0; i < data.length; i++) {  
                var li_list=angular.element('<li>');  
                listElem.append(li_list  
                    .text(data[i][propertyName]));  
            }  
        }  
    }  
})
```

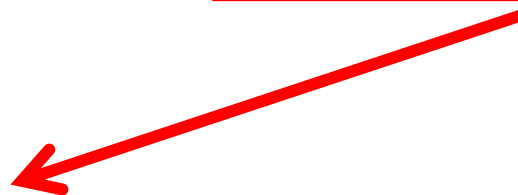
```
<div unordered-list="products" list-property="name">
```

5

## 添加属性-添加过滤器

```
<div unordered-list="products"  
      list-property="price | currency">
```

开发者工具:



```
▼ <div unordered-list="products" list-property="price | currency">  
  <ul></ul>  
</div>
```

对属性值的修改破坏了自定义指令，没有生成li

Demo:Listing 07.html

5

## 添加属性-计算表达式(\$eval)

```
listElem.append(angular.element('<li>')  
  .text(scope.$eval (propertyEx, data[i] ))));
```

将参数作为表达式运算

attrs["listProperty"]

计算的表达式: price | currency

需要用于执行该计算的本地数据

```
▼ <div unordered-list="products" list-property="price | currency">  
  ▼ <ul>  
    <li>$1.20</li>  
    <li>$2.42</li>  
    <li>$2.02</li>  
  </ul>
```

Demo:Listing 08.html

# 6

## 数据的变化

单击按钮时作用域中的数据发生变化，而视图中的元素内容没有自动更新。

```
$scope.incrementPrices = function () {  
    for (var i = 0; i < $scope.products.length; i++) {  
        $scope.products[i].price++;  
    }  
}
```

```
<button class="btn btn-primary" ng-click="incrementPrices()">
```

Demo:Listing 09.html



## 6

# 数据的变化-添加监听器

可以使用**\$watch()**方法来监控作用域的变化，这一过程对于自定义指令来说要更复杂一些，因为是从一个属性值中获取待计算的表达式。

```
var itemElement = angular.element('<li>');
var watcherFn = function (watchScope) {
    return watchScope.$eval(propertyExpression, data[i]);
};
scope.$watch(watcherFn, function(newValue, oldValue){
    itemElement.text(newValue);
});
```



字符串表达式

处理函数

# 6

## 数据的变化-添加监听器

执行顺序：

- 1、AngularJS调用链接函数来建立指令。
- 2、for循环开始遍历products数组中的各个元素。
- 3、i的值为0，对应于数组中的第一个元素。
- 4、执行for循环i的值增加为1，对应于数组中的第二个元素。
- 5、执行for循环i的值增加为2，对应于数组中的第三个元素。
- 6、执行for循环i的值增加为3，大于数组长度，结束循环。
- 7、AngularJS计算这三个分别涉及data[i]的监听器函数

Demo:Listing 10.html

# 6

## 数据的变化-修复作用域

```
for (var i = 0; i < data.length; i++) {  
  (function(){  
    ... ..  
    var itemElement = angular.element('<li>');  
    var index = i;  
    var watcherFn = function (watchScope) {  
      return watchScope.$eval(property, data[index]);  
    };  
  })()  
}
```

将循环变量 i 修改为变量 index

( function(){} )() 自执行函数

## 关于DOM的jqLite方法

名称	描述
children( )	返回一组子元素，
eq(index)	从元素集合中返回指定索引下的元素
find(tag)	按tag名定位所有的后代元素
next( )	获得下一个兄弟元素
parent( )	返回父元素

注：不支持jQuery选择器

## 7

## 范例—定位子元素

```
var items = element.children(); 返回一个jqLite对象
for (var i = 0; i < items.length; i++) {
    if (items.eq(i).text() == "Oranges") {
        items.eq(i).css("font-weight", "bold");
    }
}
```

按索引值获取元素

Demo:Listing 12.html

## 7

## 范例—find()

```
var items = element.find('li');
for (var i = 0; i < items.length; i++) {
    if (items.eq(i).text() == "Oranges") {
        items.eq(i).css("font-weight", "bold");
    }
}
```

查找所有li元素

按索引值获取li元素

Demo:Listing 14.html

## 用于修改元素的jqLite方法

名称	描述
<code>addClass(name)</code>	将jqLite对象中的所有元素添加指定的class
<code>attr(name )</code>	获得jqLite对象中的第一个元素的指定属性的值
<code>attr(name,value)</code>	设置jqLite对象中的第一个元素的指定属性的值
<code>css(name)</code>	获得jqLite对象中第一个元素的CSS属性值
<code>css(name,value)</code>	设置jqLite对象中第一个元素的CSS属性值
<code>hasClass(name)</code>	jqLite对象中使用了指定的class时返回true
<code>prop(name)</code>	获得jqLite对象中的第一个元素的指定属性的值
<code>prop(name,value)</code>	设置jqLite对象中的第一个元素的指定属性的值

## 7

# 使用jqLite

## 用于修改元素的jqLite方法

名称	描述
<code>removeAttr(name)</code>	从jqLite对象的所有元素中移除某个属性
<code>removeClass(name)</code>	从jqLite对象中移除具有指定class属性的元素
<code>text( )</code>	获取jqLite对象中所有元素的拼接后的文本内容
<code>text(value)</code>	设置jqLite对象中所有元素的文本内容
<code>toggleClass(name)</code>	为jqLite对象中的所有元素切换Class
<code>val()</code>	获取jqLite对象中第一个元素的value属性值
<code>val(value)</code>	设置jqLite对象中第一个元素的value属性值



## 7

# 范例—修改元素

```
var items = element.find('li');  
  
for (var i = 0; i < items.length; i++) {  
    if (items.eq(i).text() == "Oranges") {  
        items.eq(i).css( "color", "red");  
    }  
}
```

修改元素样式

Demo:Listing 16.html

## 用于创建和删除元素的jqLite方法

名称	描述
<code>angular.element(html)</code>	创建一个元素的jqLite对象
<code>after(elements)</code>	在元素后插入特定的内容
<code>append(elements)</code>	在每个元素上将特定元素作为最后一个子元素插入
<code>clone()</code>	返回复制的新元素
<code>prepend(elements)</code>	在每个元素上将特定元素作为第一个子元素插入
<code>remove()</code>	删除元素
<code>replaceWith(elements)</code>	用指定元素替换当前元素
<code>wrap(elements)</code>	使用特定元素包装每个元素

## 7

## 范例—删除元素

```
var listElem = angular.element("<ol>");  
element.append(listElem);  
for (var i = 0; i < scope.names.length; i++) {  
    listElem.append(angular.element("<li>")  
        .append(angular.element("<span>").text(  
scope.names[i]))));  
}
```

Demo:Listing 18.html

## 7

## jqLite方法—事件处理器

```
var buttons = element.find("button");  
buttons.on("click", function (e) {  
    element.find("li").toggleClass("bold");  
});
```

Demo:Listing 19.html

2

创建复杂指令

# 序

## 自定义指令

```
.directive("listName", function () {  
    return function (scope, element, attrs) {  
        console.log(scope);  
        console.log(element);  
        console.log(attrs);  
    }  
})
```

```
< div list-name ></div>
```

当返回一个链接函数时，所创建的指令被当作属性来使用

可以修改吗？ 可以！

# 序

## 自定义指令的属性

名称	描述
restrict	指定指令如何被使用
template	指定一个将被插入到html文档的模板
templateUrl	指定一个将被插入到html文档的外部模板
replace	指定模板内容是否替换指令所应用到的元素
scope	为指令创建一个新的作用域或一个隔离的作用域

# 1

## 范例-restrict

```
.directive('sayHello',function(){  
    return{  
        restrict: 'M',  
        replace: true,  
        scope: {cont: '=speak'}  
    }  
});
```

注意：需要在该实例添加 **replace** 属性， 否则评论是不可见的。

注意：必须设置 **restrict** 的值为 "M" 才能通过注释来调用指令。

注意：注释两端必须有空格

Demo:Listing 02.html



# 1

## 定义指令如何被使用-restrict

### 用于配置restrict定义选项的字母

名称	描述	示例
E	允许指令被用作一个元素，默认	<a href="#">Listing 03.html</a>
A	允许指令被用作一个属性，默认	<a href="#">Listing 04.html</a>
C	允许指令被用作一个类	<a href="#">Listing 05.html</a>
M	允许指令被用作一个注释	<a href="#">Listing 06.html</a>

## ②

# 使用指令模板-template

自定义指令可以使用jqLite来生成元素，一个可以替代的方法是从一个html模板生成内容，用于替换掉指令所应用到的元素的内容。

```
.directive("unorderedList", function () {  
    return {  
        restrict: "A",  
        template: "<ul><li>{{item.price }}</li></ul>"  
    }  
})
```

↓  
定义html模板

↓  
生成的html元素内容

Demo:Listing 07.html

## ②

# 范例-使用函数做为模板

```
<script type="text/template" id="list">
<ul>
<li ng-repeat="item in data">{{item.price | currency}}
</li>
</ul>
</script>
```

```
.directive("unorderedList", function () {
  return {
    restrict: "A",
    template: function () {
      return angular.element(document.
        querySelector("#list")).html();
    }
  };
});
```

Demo:Listing 08.html

## ②

# 范例-使用函数做为模板

jQuery不支持冠军id属性选择元素，可使用DOM API定位脚本元素并将其包装在一个jQuery对象中。

例：

```
var obj = document.querySelector("#list");  
  
return angular . element( obj ) .html();
```

Demo:Listing 08.html

# 3

## 使用外部模板-templateUrl

可以在一个文档中定义模板的内容，使用templateurl定义对象属性来指定文件名。

### Listing 09.html

```
<p>This is the list ... ..</p>  
<ul><li ng-repeat="item in data">{{item.price|currency}}</li>  
</ul>
```

```
templateUrl: "Listing 09.html"
```


Demo:Listing 10.html

3

## 使用外部模板-templateUrl

通过函数选择一个外部模板：

```
templateUrl: function (elem, attrs) {  
    return attrs["template"] == "table" ?  
        "tableTemplate.html" : "itemTemplate.html";  
}
```



**tableTemplate.html**

<table>... ..</table>

**itemTemplate.html**

<p>... ..</p>

Demo:Listing 12.html

# 4

## 替换元素-replace

replace : 指定模板内容是否替换指令所应用到的元素.

`templateUrl: "table.html",`    不设置replace属性



```
<div unordered-list="products" class="table table-striped">  
  <table>... ..</table>  
</div>
```

`templateUrl: "table.html",`  
`replace: true`    设置replace属性为true



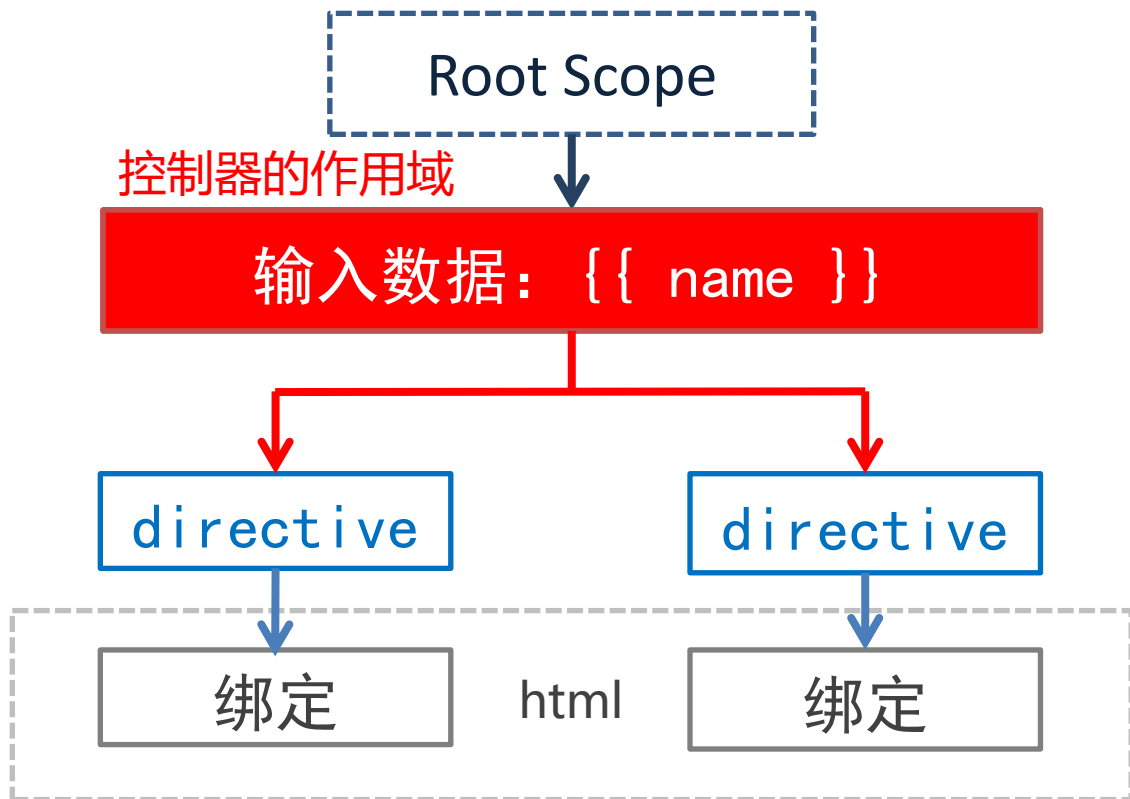
```
<table unordered-list="products" class="table table-striped">  
  ... ..  
</table>
```

Demo:Listing 14.html

## 5

# 自定义指令的作用域

**template:** "<div class='panel-body'>Name: <input **ng-model=name** /></div>"

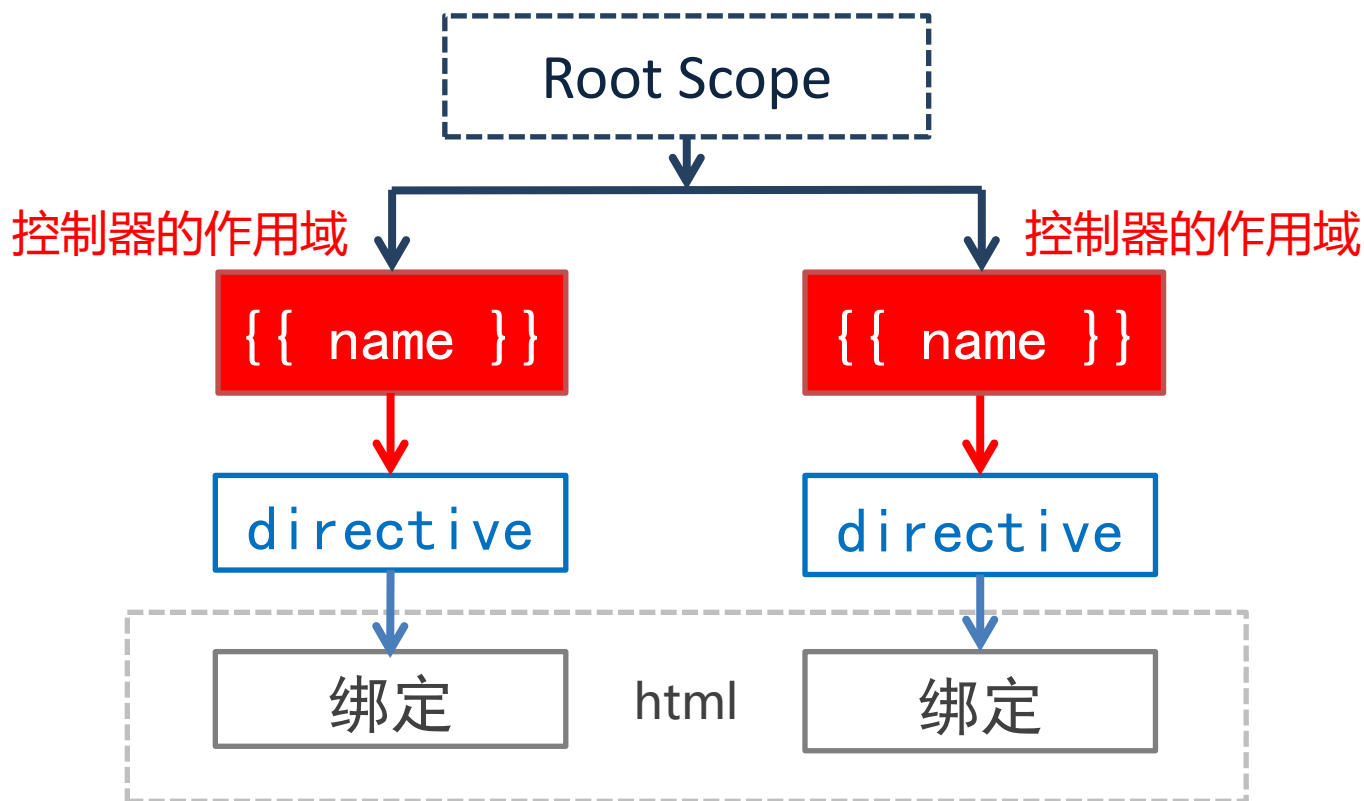


value值的同步是因为绑定了同一个name属性



## 5

# 创建多个作用域



为指令的每个实例创建一个控制器，数据将不再同步

# 5

## 自定义指令的作用域

为每个指令实例创建自己的作用域：

`scope : true` //创建独立的作用域，数据不同步

Demo:Listing 18.html

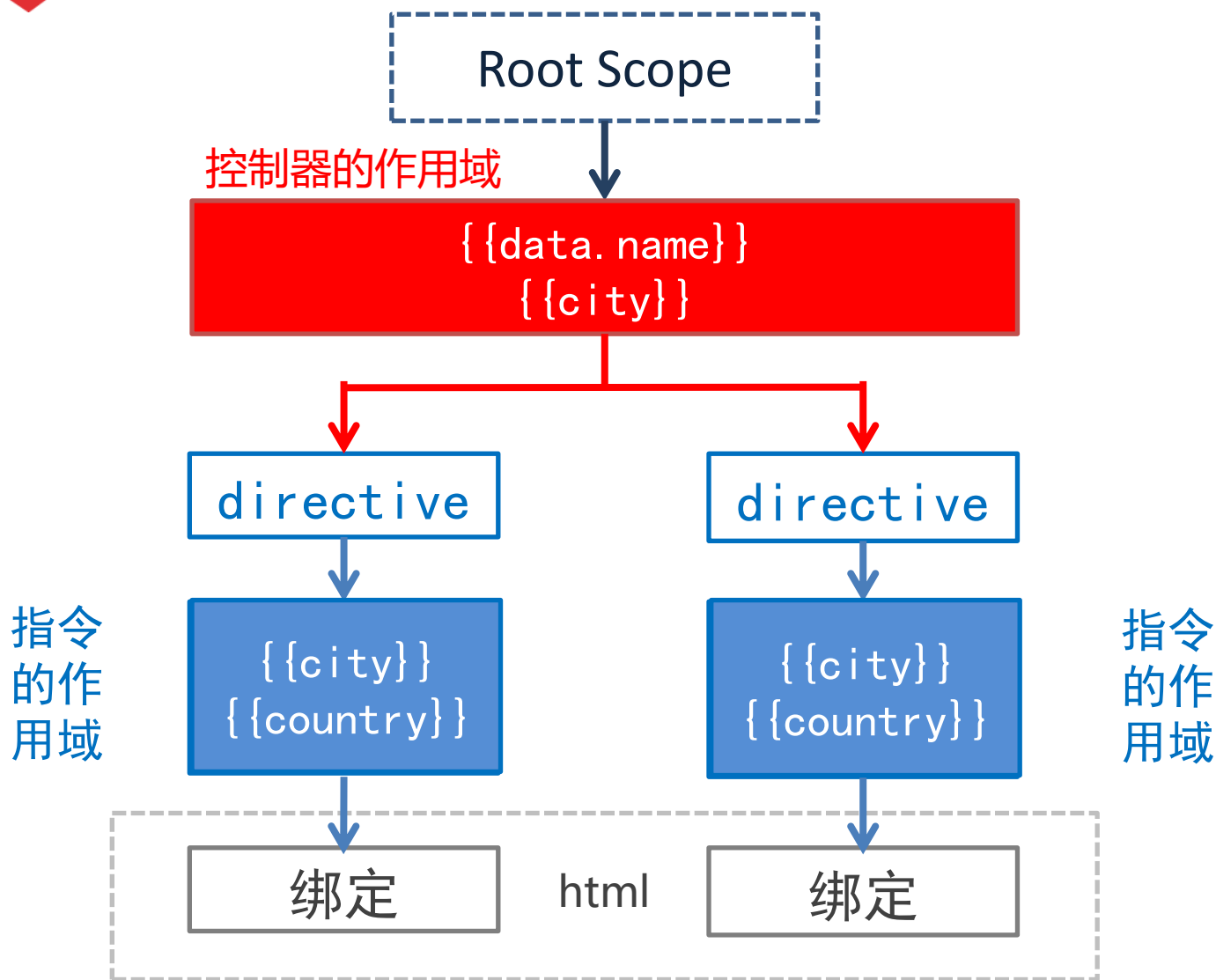
`scope : false` //不创建独立的作用域，数据同步

Demo:Listing 19.html

如果指令实例是一个对象会怎样呢？

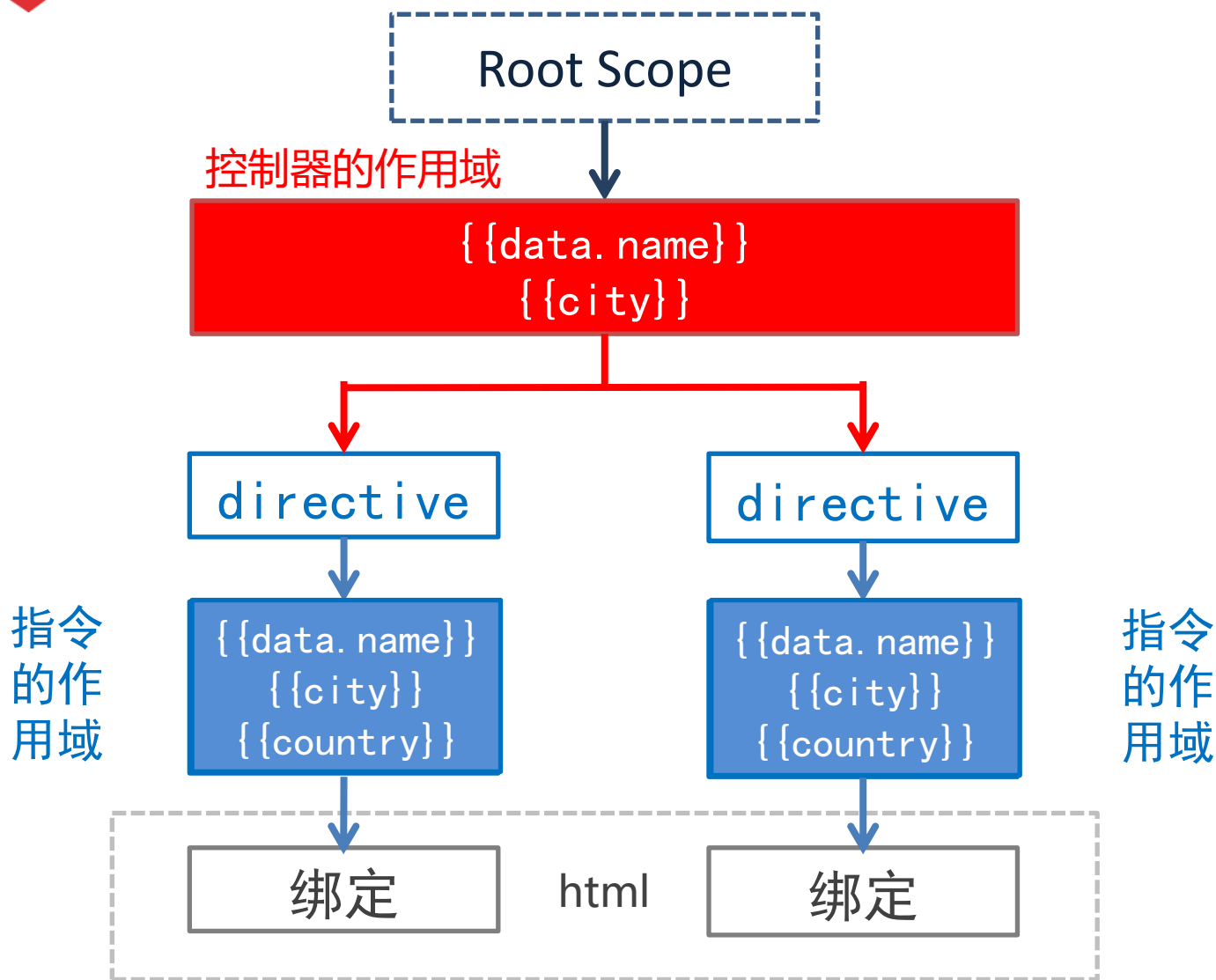
5

# 解析 : Listing 19.html



5

# 创建隔离作用域



## 5

# 创建隔离作用域

如何实现在指令作用域修改对象的属性值？

```
.directive("scopeDemo", function () {  
    return {  
        template: function() { ... ..; },  
        scope: {}  
    }  
})
```

Demo:Listing 20.html

# 5

## 自定义指令的作用域

### scope属性值符号

名称	描述	示例
@	单向数据绑定重用一個指令	<a href="#">Listing 22.html</a>
=	双向数据绑定	<a href="#">Listing 23.html</a>
&	在控制器的作用域里计算表达式	<a href="#">Listing 24.html</a>

# 3

## 高级指令

# 指令的高级属性

名称	描述
<code>transclude</code>	指定指令是否被用于包含任意内容
<code>compile</code>	指定一个编译函数
<code>link</code>	为指令定义链接函数
<code>controller</code>	为指令创建一个控制器
<code>require</code>	声明对某个控制器的依赖



# ①

## 使用嵌入包含

**transclude:**指定指令是否被用于包含任意内容

```
template: function () {... ..
```

```
    <div class="panel-body" ng-transclude>
```

```
    </div>
```

原始内容嵌入位置

```
},
```

**transclude:true**

保留原始内容

# 1

## 使用嵌入包含

```
template: function () {  
  <div class="panel panel-default">  
    <div class="panel-heading">  
      <h4>This is the panel</h4>  
    </div>  
    <div class="panel-body" ng-transclude>  
      </div>  
    </div>  
  },
```

**transclude:true**

Demo:Listing 01.html

## ②

# 使用编译函数

**compile** : 指定一个编译函数

```
compile: function (element, attrs,  
                    transcludeFn) {  
    // somecode ...  
}
```

编译函数负责对模板DOM进行转换，可以返回一个对象或函数。

## 2

# 使用编译函数

```
compile: function (element, attrs, transcludeFn) {  
  return function ( $scope, $element, $attr) {  
    $scope.$watch("data.length", function () {  
      var parent = $element.parent();  
      parent.children().remove();  
      for (var i = 0; i < $scope.data.length; i++) {  
        var childScope = $scope.$new();  
        childScope[$scope.propName] = $scope.data[i];  
        transcludeFn(childScope, function (clone) {  
          parent.append(clone);  
        });  
      }  
    });  
  }  
}
```

Demo:Listing 02.html

# 3

## 使用选择列表

**link** : 为指令定义链接函数

```
link: function (scope, element, attrs, ctrl) {  
    scope.$watch("item.quantity", function () {  
        ctrl.updateTotal();  
    });  
}
```

Demo:Listing 04.html

## ④

## 应用-显示更多

```
.directive('showmore', function (){  
  return {  
    restrict: 'E',  
    template: '<div class="{{show?\'more2\' : \'more\'}}">' + '<a href="javascript:;"  
      ng-click="show=!show">显示更多</a>' +  
      '<span ng-transclude></span></div>',  
    transclude: true  
  };  
});
```

Demo: 显示更多.html

## 5

## 应用-下拉选项

```
directive('dropdownlist', function () {  
  return {  
    restrict: 'E',  
    template: '<input type="text" ng-model="str"/>  
              <ul><li ng-repeat="v in arr "  
                ng-show="v.indexOf(str) != -1">{{v}}  
              </li></ul>  
    };  
  });  
});
```

Demo: 下拉提示.html

# 本课小结

---

1. 自定义指令
2. 创建复杂指令
3. 高级指令



# TNAKS

主讲：王智娟

QQ: 24132228

Email: wangzhijuan@onest.net