# 简介

AngularJS 通过 指令 扩展了 HTML，且通过 表达式 绑定数据到 HTML。

<script src="http://apps.bdimg.com/libs/angular.js/1.4.6/angular.min.js"></script>

⚫AngularJS 通过 ng-directives 扩展了 HTML。

⚫ng-app 指令定义一个 AngularJS 应用程序。从这里启动

⚫ng-model 指令把元素值（比如输入域的值）绑定到应用程序。

⚫ng-bind 指令把应用程序数据绑定到 HTML 视图。



# 表达式

AngularJS 表达式写在双大括号内：{{ expression }}。

AngularJS 表达式把数据绑定到 HTML，这与 ng-bind 指令有异曲同工之妙。

AngularJS 将在表达式书写的位置"输出"数据。

AngularJS 表达式 很像 JavaScript 表达式：它们可以包含文字、运算符和变量。

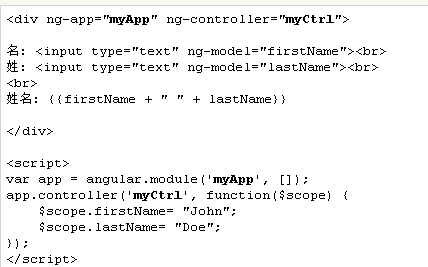
实例 {{ 5 + 5 }} 或 {{ firstName + " " + lastName }}

# 控制器

AngularJS 模块（Module） 定义了 AngularJS 应用。

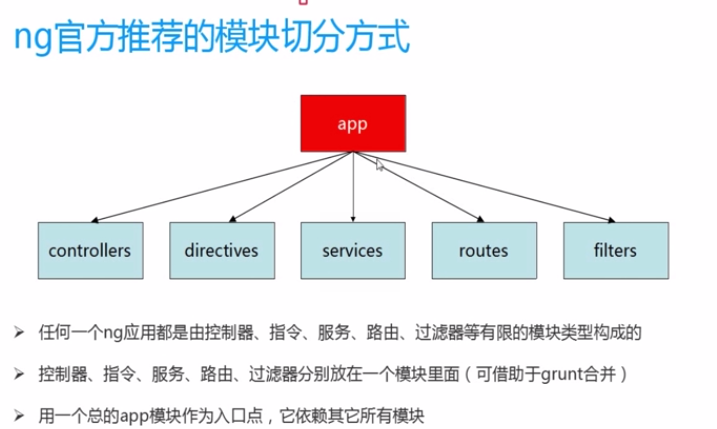
AngularJS 控制器（Controller） 用于控制 AngularJS 应用。

ng-app指令定义了应用, ng-controller 定义了控制器。



注意：不推荐在控制器里面操作dom，应该在指令的link方法中操作

# 官方模块切分方式



# 什么时候用ng-bind，什么时候用｛｛｝｝

当angular加载完后，用｛｛｝｝就不会出现刷新的时候出现｛｛xx.xx｝｝所以这个时候可以用｛｛｝｝

在index.html或者说加载angular的页面中用ng-bind

# 双向数据绑定

view中<input type=’text’ ng-model=”name”>

controller中改变$scope中的name值，view中对应的ng-model的name值也会改变。

相反，改变view中的name，controller中$scope的name也会改变

# 校验

formName.inputName.$error.type 检错误

type：

required 必须

email 邮箱格式

formNmae.inputName.type2

type2:



## 简单使用

<form ng-app="" name="myForm" ng-init="myText = 'test@runoob.com'">

Email:

<input type="email" name="myAddress" ng-model="myText" required>

<p>编辑邮箱地址，查看状态的改变。</p>

<h1>状态</h1>

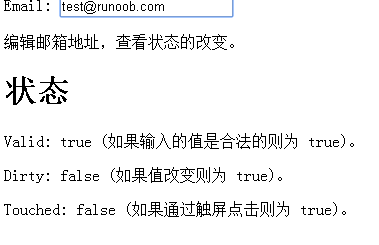
<p>Valid: {{myForm.myAddress.$valid}} (如果输入的值是合法的则为 true)。</p>

<p>Dirty: {{myForm.myAddress.$dirty}} (如果值改变则为 true)。</p>

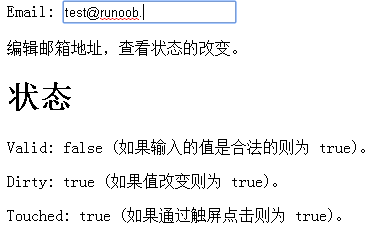
<p>Touched: {{myForm.myAddress.$touched}} (如果通过触屏点击则为 true)。</p>

</form>

**结果1**：



**结果**2：



## 校验css类

<style>

/\*校验失败背景色变lightblue\*/

input.ng-invalid {

background-color: lightblue;

}

/\*校验成功背景色变yellow\*/

input.ng-valid{

background-color: yellow;

}

</style>

<body>

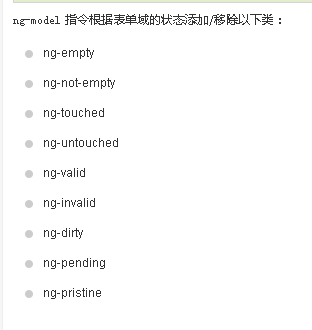
<form ng-app="" name="myForm">

输入你的名字:

<input name="myAddress" ng-model="text" required>

</form>

## 指令根据表单域的状态添加/移除以下类：



# run方法

var module = anguler.module(‘xx’,[])

module.run(function(){

}) run方法是注射器加载完所有模块的时候，会执行一次

# 内置指令

## ng-init

初始化内容作用域。这个指令一般会出现在比较小的应用中，比如给个demo什么的...

<div ng-init="job='fighter'">

I'm a/an {{job}}

</div>

## ng-class

<div ng-class='{error: isError, warning: isWarning}'></div>

isError是true就会引用name为error的class，isWarning是true就会引用name为isWarning的class

## ng-show

<div ng-app="" ng-init="showDiv=true">

<p ng-show="showDiv">我是可见的。</p>

<input type="checkbox" ng-model="showDiv" />显示/隐藏

</div>

可以根据value的值来显示/隐藏

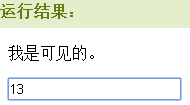
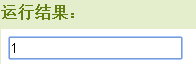
<div ng-app="" ng-init="hour=13">

<p ng-show="hour > 12">我是可见的。</p>

<input type="text" ng-model="hour">

</div>

效果：



## ng-repeat

$index 下标

$odd 是否偶数

$even 是否基数

$first 是否第一个

$last 是否最后一个

$middle 是否中间那个

## ng-disable

<div ng-app="" ng-init="mySwitch=true">

<p>

<button ng-disabled="mySwitch">点我!</button>

</p>

<p>

<input type="checkbox" ng-model="mySwitch"/>按钮

</p>

## ng-include

在 HTML 中包含 HTML 文件

<div ng-include="'.html'"></div>

里面是一个不包含html，body的页面

## 下拉框ng-options

### 使用

<div ng-app="myApp" ng-controller="myCtrl">

<select ng-model="selectedName" ng-options="x for x in names">

</select>

</div>

<script>

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

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

$scope.names = ["Google", "Runoob", "Taobao"];

});

</script>

也可以使用ng-repeat 指令来创建下拉列表

<select>  
<option ng-repeat="x in names">{{x}}</option>  
</select>

ng-option处理的数据是对象的时候：  
如：$scope.sites = [

{site : "Google", url : "http://www.google.com"},

{site : "Runoob", url : "http://www.runoob.com"},

{site : "Taobao", url : "http://www.taobao.com"}

];

<select ng-model="selectedName" ng-options=” x.site for x in sites”></select>

或者根据键值key-value方法

如：

$scope.sites = {

site01 : "Google",

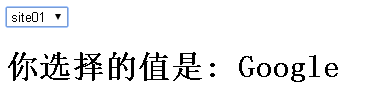
site02 : "Runoob",

site03 : "Taobao"

}

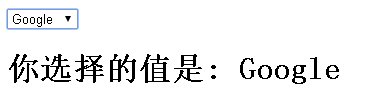
<select ng-model="selectedName" ng-options=” **x for (x, y) in sites**”></select>

效果：



x for (x,y)中 x代表key值，y代表value

如果写成y for (x,y) 效果如下：



### option中value值

使用as绑定显示值对应的value

$scope.colors = [

{

code : 1,

name : "红色"

},{

code : 2,

name : "蓝色"

},{

code : 3,

name : "绿色"

},{

code : 4,

name : "黄色"

}

]

<select ng-model="myColor" ng-options="x.code as x.name for x in colors">

<option value="">-- 请选择 --</option>

</select>

## ng-click

<div ng-app="myApp" ng-controller="myCtrl">

<button ng-click="count = count + 1">点我！</button>

<p>{{ count }}</p>

</div>

<script>

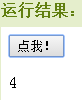
var app = angular.module('myApp', []);

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

$scope.count = 0;

});

</script>



<div ng-app="myApp" ng-controller="personCtrl">

<button ng-click="toggle()">隐藏/显示</button>

<p ng-hide="myVar">

名: <input type=text ng-model="firstName"><br>

姓: <input type=text ng-model="lastName"><br><br>

姓名: {{firstName + " " + lastName}}

</p>

</div>

<script>

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

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

$scope.firstName = "John",

$scope.lastName = "Doe"

$scope.myVar = false;

$scope.toggle = function() {

$scope.myVar = !$scope.myVar;

}

});

</script>

## ****ng-readonly****

通过表达式返回值true/false将表单输入字段设为只读。

## ****ng-selected****

给<select>里面的<option>用的

用于默认选中的

## ****ng-change****

结合ng-model使用，以ng-change为例:

<input type="text" ng-model="calc.arg" ng-change="calc.result = calc.arg\*2" />

<code>{{ calc.result }}</code>



## ng-if

如果ng-if中的表达式为false，则对应的元素整个会从DOM中移除而非隐藏，但审查元素时你可以看到表达式变成注释了。

<div>

<input ng-model="show" />

<div ng-if="show%2 == 0">什么情况</div>

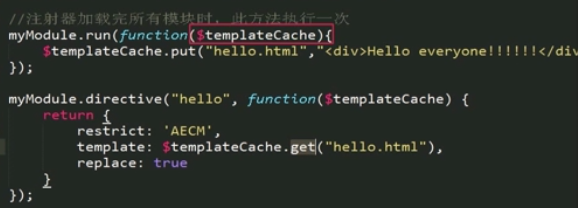
</div>

# 指令

## restrict匹配模式



把模版缓存起来



## require



依赖accordion指令

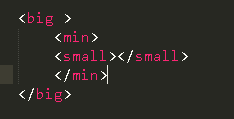
具体如何获取controller呢？require选项的值可以分别用前缀？、^ 和？^进行修饰，也可以不修饰。

如果不进行修饰，比如require:'thisDirective'，那么require只会在当前指令中查找控制器

如果想要指向上游的指令，那么就是用^进行修饰，比如require:'^parentDirective'，如果没有找到，那就会抛出一个错误。

如果使用？前缀，就意味着如果在当前指令没有找到控制器，就将null作为link的第四个参数；

那么，如果将？和^结合起来，我们就可以既指定上游指令，又可以在找不到时，不抛出严重的错误。





## ng-transclude：true

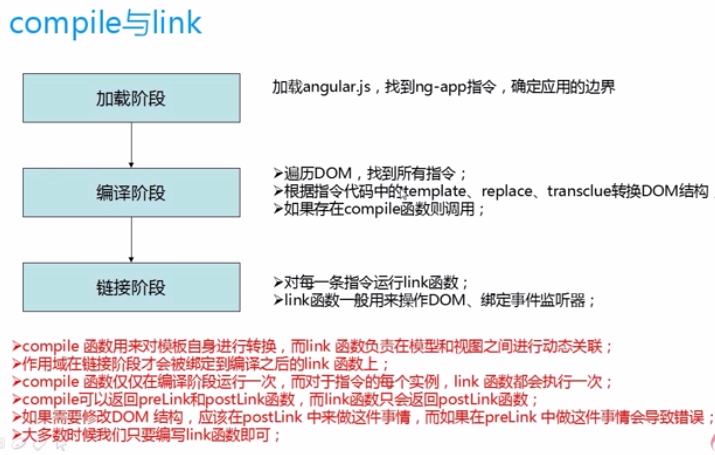
保留原来的内容





在span里面加了之后，hello里面的内容就会出现在span里面

## compile和link方法



如果有自定义compile函数，自定义link函数会无效，因为compile函数应该返回一个link函数供后续处理

compile: function(ele,attrs,transclude){

console.log("编译阶段",transclude);

return function(scope,elem,attrs,ctrl){

console.log("连接阶段",elem)

}

},

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

console.log(“连接阶段”)

}

link中的ctrl就是值指令中的控制器

## 内置指令



## 模板

templateUrl: 模板文件

template:模板字符串

## scope

### @

指令中的取值为html中的字面量/直接量；建立一个local scope property到DOM属性的绑定。因为属性值总是String类型，所以这个值总是返回一个字符串。如果没有通过@attr指定属性名称，那么本地名称将与DOM属性的名称一致。

例如<widget my-attr=”hello {{name}}”>，

widget的scope定义为：{localName:’@myAttr’}。那么，widget scope property的localName会映射出”hello {{name}}"转换后的真实值。name属性值改变后，widget scope的localName属性也会相应地改变（仅仅单向）。name属性是在父scope读取的（不是组件scope）

### =

指令中的属性取值为controller中对应$scope上属性的取值，可用于双向数据的绑定

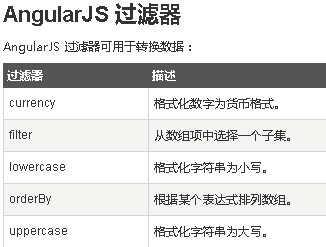
### &

指令中的取值为Contoller中对应$scope上的属性，但是这属性必须为一个函数回调

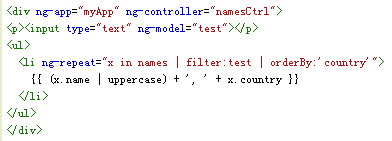
# 过滤器

使用 : s{{ value | filter}}

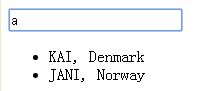
## 官方的过滤器



filter：（感觉类似模糊搜索）



结果：



<p ng-repeat="num in array | small:1:2:3:4">{{num.name}}</p>

## 自定义过滤器

myModule.filter( filterName ,function(){

return function(input,p1,p2,p3,…..){

return 处理后的数据;

}

});

return function() 第一个参数是需要过滤的数据，如array数组。之后的是参数，如:p1=1, p2=2,p3=3….

例子

自定义指令<input ng-model="num" /><br/>

过滤后的值:{{num | myfilter:1}}

app.filter("myfilter",function(){

return function(input,param){

console.log("filter",param) //输出：1

if(!!input){

return input\*input;

}else {

return 0;

}

}

})

# 服务

## $location服务



## $http服务

最常用的服务，用于向服务器发送请求，应用相应服务器传送过来的数据。

### 链式调用

$http({

url:'data.json',

method:'GET',

params:{

'name':'value'

},

data:字符串或对象(如发送一个blob对象)

}).success(function(data,header,config,status){

//响应成功

}).error(function(data,header,config,status){

//处理响应失败

});

### 返回一个promise对象

var promise=$http({

method:'GET',

url: 'url',

params:{

'name':'value'

}

});

promise.success(function(data,status,config,headers){

//处理成功的响应

});

promise.error(function(data,status,hedaers,config){

//处理失败后的响应

});

### 响应对象

◇data

这个数据代表转换过后的响应体（如果定义了转换的话）

◇status

响应的HTTP状态码

◇headers

这个函数是头信息的getter函数，可以接受一个参数，用来获取对应名字值

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

app.controller('customersCtrl', function($scope, $http) {

//快捷get方式

$http.get("url")

.success(function (response) {

//请求成功

response是后台返回的数据

})

.error(function(){

//请求失败

});

});

## $timeout 服务

和window的setTimeout用法差不多。

<script>

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

app.controller('myCtrl', function($scope, $timeout) {

$scope.myHeader = "Hello World!";

$timeout(function () {

$scope.myHeader = "How are you today?";

}, 2000);

});

</script>

## $interval 服务

和window的setInterval用法差不多。

<script>

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

app.controller('myCtrl', function($scope, $interval) {

$scope.theTime = new Date().toLocaleTimeString();

$interval(function () {

$scope.theTime = new Date().toLocaleTimeString();

}, 1000);

});

</script>

## $scope.$watch

### 使用

$watch(watchExpression, listener, objectEquality);

每个参数的说明如下：

1. watchExpression：监听的对象，它可以是一个angular表达式如'name',或函数如function(){return $scope.name}。
2. listener:当watchExpression变化时会被调用的函数或者表达式,它接收3个参数：newValue(新值), oldValue(旧值), scope(作用域的引用)
3. objectEquality：是否深度监听，如果设置为true,它告诉Angular检查所监控的对象中每一个属性的变化. 如果你希望监控数组的个别元素或者对象的属性而不是一个普通的值, 那么你应该使用它

例子：（监控input的变化）

<div ng-app="myApp" ng-controller="myCtrl">

<input ng-model="ng-model-name" />

</div>

<script>

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

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

$scope.$watch(‘ng-model-name’ , function(newValue,oldValue,scope){

scope.say();

})

$scope.say = function(){

alert($scope.name);

}

});

</script>

### 性能问题

太多的$watch将会导致性能问题，$watch如果不再使用，我们最好将其释放掉。

$watch函数返回一个注销监听的函数

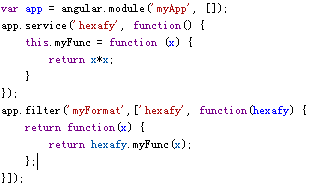
释放可以使用一下的方式：

var watch = $scope.$watch(watchExpression, listener);

watch(); //调用方法释放

## 创建自定义的服务





# Angular API

常用的api



# 动画

# 路由

引入 <script type="text/javascript" src="./js/angular-ui-router.min.js"></script>

## 加依赖：

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

## 配置：

app.config(function($stateProvider, $urlRouterProvider) {

$urlRouterProvider.when("/","/main");

$stateProvider.state("main",option对象)

});

### url

页面地址

### template

页面模板（字符串）

### templateUrl

页面模板（路径）

### controller

控制器，名字或直接函数

### resolve

初始化数据

resolve:{

person:function(){

return {

id:123,

name:’ybb’

}

}

}

## when(what,handler)

what：需要重定向的传入路径。  
handler：你想要重定向的路径/处理程序。

## **$state**

$state服务负责代表状态及提供状态之间的转换。它还提供你当前的状态及上一个状态。

### get(stateOrName,context); stateOrName：如果提供此参数，返回的是指定状态的配置对象；如果不提供此参数，则返回全部状态的配置对象。 context：当context是一个相对的参考状态，状态会在相关上下文中检索。

### ****go(to,params,options);****

to：string，即将跳转的状态。  
params:object，跳转所带的参数。  
options：object，可选配置对象,有以下参数：

location(是否更新地址栏的url，或以什么字符串替换url)，

inherit(是否继承当前url的参数)，

relative(当变化相对路径：如"^,定义的状态是相对的)，

notify(是否广播$stateChangeStart和$stateChangeSuccess事件)，

reload(是否重新载入)。

### reload(state)****;****

重新载入当前状态的方法。

state：一个状态名称或者状态对象。

### transitionTo(to,toParams,options);

过渡到一个新状态的方法。

to：状态名称。  
toParams：将会发送到下一个状态的参数。  
options：可选参数。有location，inherit，relative，notify，reload。

## 跳转

### ui-sref

在点击的元素上添加，连接到特定状态

### $state.go

$state.go(to,params,option)