参考：<http://baike.baidu.com/link?url=NLznqlfUO6Ddy7KHNRTEI9TTboWQ-6WhIE8IT7eoIYWMCCxK4hf4S3cdjI0O5ZGyuxIS6x7Ob-I7cRSGv5Wv0ou13wvJQZaMY_ak5LQ1Vi_>

AngularJS

AngularJS[1]  诞生于2009年，由Misko Hevery 等人创建，后为Google所收购。是一款优秀的前端JS框架，已经被用于Google的多款产品当中。AngularJS有着诸多特性，最为核心的是：MVC、模块化、自动化双向数据绑定、语义化标签、依赖注入等等。

AngularJS是为了克服[HTML](http://baike.baidu.com/view/692.htm)在构建应用上的不足而设计的。HTML是一门很好的伪静态文本展示设计的声明式语言，但要构建WEB应用的话它就显得乏力了。所以我做了一些工作（你也可以觉得是小花招）来让浏览器做我想要的事。

通常，我们是通过以下技术来解决静态网页技术在构建动态应用上的不足：

类库 - 类库是一些函数的集合，它能帮助你写WEB应用。起主导作用的是你的代码，由你来决定何时使用类库。类库有：jQuery等

框架 - 框架是一种特殊的、已经实现了的WEB应用，你只需要对它填充具体的业务逻辑。这里框架是起主导作用的，由它来根据具体的应用逻辑来调用你的代码。框架有：knockout、sproutcore等。

AngularJS使用了不同的方法，它尝试去补足HTML本身在构建应用方面的缺陷。AngularJS通过使用我们称为指令(directives)的结构，让浏览器能够识别新的语法。例如：

使用双大括号{{}}语法进行数据绑定；

使用DOM控制结构来实现迭代或者隐藏DOM片段；

支持表单和表单的验证；

能将逻辑代码关联到相关的DOM元素上；

能将HTML分组成可重用的组件。

解决方案

[编辑](http://baike.baidu.com/javascript:;)

AngularJS试图成为WEB应用中的一种客户端的解决方案。这意味着它不只是你的WEB应用中的一个小部分，还是一个完整的客户端的解决方案。这会让AngularJS在构建一个CRUD（增加Create、查询Retrieve、更新Update、删除Delete）的应用时显得很“固执”（原文为 opinionated,意指没有太多的其他方式）。但是，尽管它很“固执”，它仍然能确保它的“固执”只是在你构建应用的起点，并且你仍能灵活变动。AngularJS的一些出众之处如下：

构建一个CRUD应用可能用到的全部内容包括：数据绑定、基本模板标识符、表单验证、路由、深度链接、组件重用、依赖注入。

测试方面包括：单元测试、端对端测试、模拟和自动化测试框架。

具有目录布局和测试脚本的种子应用作为起点。

可爱之处

[编辑](http://baike.baidu.com/javascript:;)

AngularJS通过为开发者呈现一个更高层次的抽象来简化应用的开发。如同其他的抽象技术一样，这也会损失一部分灵活性。换句话说，并不是所有的应用都适合用AngularJS来做。AngularJS主要考虑的是构建CRUD应用。幸运的是，至少90%的WEB应用都是CRUD应用。但是要了解什么适合用AngularJS构建，就得了解什么不适合用AngularJS构建。

如游戏，图形界面编辑器，这种DOM操作很频繁也很复杂的应用，和CRUD应用就有很大的不同，它们不适合用AngularJS来构建。像这种情况用一些更轻量、简单的技术如jQuery可能会更好。

**实例**

下面是一个包含了一个表单的典型CRUD应用。表单值先经过验证，然后用来计算总值，这个总值会被格式化成本地的样式。下面有一些开发者常见的概念，你需要先了解一下：

1.将数据模型(data-model)关联到视图(UI)上；

2.写、读、验证用户的输入；

3.根据模型计算新的值；

4.将输出格式本地化，

index.html:

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9  10  11  12  13  14  15  16  17  18  19  20  21  22  23 | <!doctype html>  <html ng-app>  <head>  <script src="angular-1.1.0.min.js"></script>  <script src="script.js"></script>  </head>  <body>  <div ng-controller="InvoiceCntl">  <b>Invoice:</b>  <br>  <br>  <table>  <tr><td>Quantity</td><td>Cost</td></tr>  <tr>  <td><input type="integer" min="0" ng-model="qty" required ></td>  <td><input type="number" ng-model="cost" required ></td>  </tr>  </table>  <hr>  <b>Total:</b> {{qty \* cost | currency}}  </div>  </body>  </html> |

script.js

|  |  |
| --- | --- |
| 1  2  3  4 | function InvoiceCntl($scope) {  $scope.qty = 1;  $scope.cost = 19.95;  } |

试一下上面这个例子。然后我们一起来看下这个例子的工作原理。

在<html>标签里，我们用一个ng-app标识符标明这是一个AngularJS应用。这个ng-app标识符会使AngularJS自动初始化(auto initialize)你的应用。

我们用<script>标签来加载AngularJS脚本：

<script src="angular-1.1.0.min.js"></script>通过设置<input>标签里的ng-model属性， AngularJS会自动对数据进行双向绑定。我们还同时进行了一些简单的数据验证：

Quantity:<input type="integer"min="0"ng-model="qty"required >Cost:<input type="number"ng-model="cost"required >这个输入框的widget看起来很普通，但如果认识到以下几点那它就不普通了：

当页面加载完后，AngularJS会依照widget里的声明的模型名字（qty、cost）生成同名变量。你可以把这些变量认为是MVC设计模式中的M(Model)；

注意上面widget里的input有着特殊的能力。如果你们没有输入数据或者输入的数据无效，这个input输入框会自动变红。输入框的这种新特性，能让开发者更容易实现CRUD应用里常见的字段验证功能。

终于，我们可以来看一下神秘的双大括号{{}}了:

Total:{{qty \*cost |currency}}这个{{表达式}}标记是AngularJS的数据绑定。其中的表达式可以是表达式和过滤器({{ expression | filter }})的组合。AngularJS提供了过滤器来对输入输出数据格式化。

上面的这个例子里，{{}}里的表达式让AngularJS把从输入框中获得的数据相乘，然后把相乘结果格式化成本地货币样式，然后输出到页面上。

值得一提的是，我们既没有调用任何AngularJS的方法，也没有像用框架一样去编写某个具体逻辑，就是完成了上述功能。这个实现的背后是因为浏览器做了比以往生成 静态页面更多的工作，让它能满足动态WEB应用的需要。AngularJS使得动态WEB应用的开发门槛降到不需要类库或者框架的程度。

理念

[编辑](http://baike.baidu.com/javascript:;)

Angular信奉的是，当组建视图(UI)同时又要写软件逻辑时，声明式的代码会比命令式的代码好得多，尽管命令式的代码非常适合用来表述业务逻辑。

将DOM操作和应用逻辑解耦是一种非常好的思路，它能大大改善代码的可调性；

将测试和开发同等看待是一种非常非常好的思路，测试的难度在很大程度上取决于代码的结构；

将客户端和服务器端解耦是一种特别好的做法，它能使两边并行开发，并且使两边代码都能实现重用；

如果框架能够在整个开发流程里都引导着开发者：从设计UI，到编写业务逻辑，再到测试，那对开发者将是极大的帮助；

“化繁为简，化简为零”总是好的。

AngularJS能将你从以下的噩梦中解脱出来：

使用回调：回调的使用会打乱你的代码的可读性，让你的代码变得支离破碎，很难看清本来的业务逻辑。移除一些常见的代码，例如回调，是件好事。大幅度地减少你因为JavaScript这门语言的设计而不得不写的代码，能让你把自己应用的逻辑看得更清楚。

手动编写操作DOM元素的代码：操作DOM是AJAX应用很基础的一部分，但它也总是很“笨重”并且容易出错。用声明的方式描述的UI界面可随着应用状态的改变而变化，能让你从编写低级的DOM操作代码中解脱出来。绝大部分用AngularJS写的应用里，开发者都不用再自己去写操作DOM的代码，不过如果你想的话还是可以去写。

对UI界面读写数据：AJAX应用的很大一部是CRUD操作。一个经典的流程是把服务端的数据组建成内部对象，再把对象编成HTML表单，用户修改表单后再验证表单，如果有错再显示错误，然后将数据重新组建成内部对象，再返回给服务器。这个流程里有太多太多要重复写的代码，使得代码看起来总是在描述应用的全部执行流程，而不是具体的业务逻辑和业务细节。

开始前得写大量的基础性的代码：通常你需要写很多的基础性的代码才能实现一个“Hello World”的应用。用AngularJS的话，它会提供一些服务让你很容易地正式开始写你的应用，而这些服务都是以一种Guice-like dependency-injection式的依赖注入自动加入到你的应用中去的，这让你能很快的进入你应用的具体开发。特别的是，你还能全盘掌握自动化测试的初始化过程。

AngularJS 表达式

[编辑](http://baike.baidu.com/javascript:;)

AngularJS 表达式写在双大括号内：{{ expression }}。[2]

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 }}

参考：<http://www.runoob.com/angularjs/angularjs-tutorial.html>

# AngularJS 教程



AngularJS 通过新的属性和表达式扩展了 HTML。

AngularJS 可以构建一个单一页面应用程序（SPAs：Single Page Applications）。

AngularJS 学习起来非常简单。

小试牛刀：

<!DOCTYPE html>  
<html>  
<head>  
<meta charset="utf-8">  
<script src="http://cdn.static.runoob.com/libs/angular.js/1.4.6/angular.min.js"></script>  
</head>  
<body>  
  
<div ng-app="">  
  <p>名字 : <input type="text" ng-model="name"></p>  
  <h1>Hello {{name}}</h1>  
</div>  
  
</body>  
</html>

## AngularJS 历史

AngularJS 是比较新的技术，版本 1.0 是在 2012 年发布的。

AngularJS 是由 Google 的员工 Miško Hevery 从 2009 年开始着手开发。

这是一个非常好的构想，该项目目前已由 Google 正式支持，有一个全职的开发团队继续开发和维护这个库。

# AngularJS 简介

ng-model

AngularJS 是一个 **JavaScript 框架**。它可通过 <script> 标签添加到 HTML 页面。

AngularJS 通过 [**指令**](http://www.runoob.com/angularjs/angularjs-directives.html) 扩展了 HTML，且通过 [**表达式**](http://www.runoob.com/angularjs/angularjs-expressions.html) 绑定数据到 HTML。

{{表达式}}

## AngularJS 是一个 JavaScript 框架

AngularJS 是一个 JavaScript 框架。它是一个以 JavaScript 编写的库。

AngularJS 是以一个 JavaScript 文件形式发布的，可通过 script 标签添加到网页中：

<script src="http://cdn.static.runoob.com/libs/angular.js/1.4.6/angular.min.js"></script>

|  |  |
| --- | --- |
| **Note** | 我们建议把脚本放在 <body> 元素的底部。 这会提高网页加载速度，因为 HTML 加载不受制于脚本加载。 |

各个 angular.js 版本下载： <https://github.com/angular/angular.js/releases>

## AngularJS 扩展了 HTML

指令

AngularJS 通过 **ng-directives** 扩展了 HTML。

**ng-app** 指令定义一个 AngularJS 应用程序。

**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。

## AngularJS 实例

<!DOCTYPE html>

<html>

<head>

<meta charset="utf-8">

<script src="http://cdn.static.runoob.com/libs/angular.js/1.4.6/angular.min.js"></script>

</head>

<body>

<div ng-app="">

<p>名字 : <input type="text" ng-model="name"></p>

<h1>Hello {{name}}</h1>

</div>

</body>

</html>

[**尝试一下 »**](http://www.runoob.com/try/try.php?filename=try_ng_intro)

实例讲解：

当网页加载完毕，AngularJS 自动开启。注：是在网页加载完毕，angularjs才开启，因此某些变量可能在网页加载过程直接显示，如{{name}}，网页加载完后就又会消失

**ng-app** 指令告诉 AngularJS，<div> 元素是 AngularJS **应用程序** 的"所有者"。

**ng-model** 指令把输入域的值绑定到应用程序变量 **name ——>ng-model=”name”**。

**ng-bind** 指令把应用程序变量 name 绑定到某个段落的 innerHTML。

|  |  |
| --- | --- |
| **Note** | 如果您移除了 **ng-app** 指令，HTML 将直接把表达式显示出来，不会去计算表达式的结果。 |

## 什么是 AngularJS？

AngularJS 使得开发现代的单一页面应用程序（SPAs：Single Page Applications）变得更加容易。

* AngularJS 把应用程序数据绑定到 HTML 元素。
* AngularJS 可以克隆和重复 HTML 元素。
* AngularJS 可以隐藏和显示 HTML 元素。
* AngularJS 可以在 HTML 元素"背后"添加代码。
* AngularJS 支持输入验证。

## AngularJS 指令

正如您所看到的，AngularJS 指令是以 **ng** 作为前缀的 HTML 属性(是angluarjs的指令，同时是html的属性)。

**ng-init** 指令初始化 AngularJS 应用程序变量。

## AngularJS 实例

<div ng-app="" ng-init="firstName='John'"> <p>姓名为 <span ng-bind="firstName"></span></p> </div>

[**尝试一下 »**](http://www.runoob.com/try/try.php?filename=try_ng_intro_directives)

|  |  |
| --- | --- |
| **Note** | HTML5 允许扩展的（自制的）属性，以 **data-** 开头。 AngularJS 属性以 **ng-** 开头，但是您可以使用 **data-ng-** 来让网页对 HTML5 有效。 |

带有有效的 HTML5：

## AngularJS 实例

<div data-ng-app="" data-ng-init="firstName='John'"> <p>姓名为 <span data-ng-bind="firstName"></span></p> </div>

[**尝试一下 »**](http://www.runoob.com/try/try.php?filename=try_ng_intro_directives_html)

既可以使用表达式，也可以使用bind，使用bind的话就会替换拥有此属性的标签的所有innerhtml

## AngularJS 表达式

AngularJS 表达式写在双大括号内：**{{ expression }}（变量是最简单的表达式）**。

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 }}

## AngularJS 实例

<!DOCTYPE html>

<html>

<head>

<meta charset="utf-8">

<script src="http://cdn.static.runoob.com/libs/angular.js/1.4.6/angular.min.js">

</script>

</head>

<body>

<div ng-app=""><p>我的第一个表达式： {{ 5 + 5 }}</p>

</div>

</body>

</html>

[**尝试一下 »**](http://www.runoob.com/try/try.php?filename=try_ng_intro_expression)

## AngularJS 应用

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。

## AngularJS 实例

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

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

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

姓名: {{firstName + " " + lastName}}

</div>

<script>

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

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

$scope.firstName= "John";

$scope.lastName= "Doe";

});

</script>

[**尝试一下 »**](http://www.runoob.com/try/tryit.php?filename=try_ng_intro_controller)

AngularJS 模块定义应用:

## AngularJS 模块

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

AngularJS 控制器控制应用:

## AngularJS 控制器

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

$scope.firstName= "John";

$scope.lastName= "Doe";

});

在接下来的教程中你将学习到更多的应用和模块的知识。

# AngularJS 表达式

AngularJS 使用 **表达式** 把数据绑定到 HTML。

但是ng-bind和{{}}又不同

## AngularJS 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 }}

## AngularJS 实例

<!DOCTYPE html>

<html>

<head>

<meta charset="utf-8">

<script src="http://cdn.static.runoob.com/libs/angular.js/1.4.6/angular.min.js"></script>

</head>

<body>

<div ng-app="">

<p>我的第一个表达式: {{ 5 + 5 }}</p>

</div>

</body>

</html>

[**尝试一下 »**](http://www.runoob.com/try/try.php?filename=try_ng_expression)

## AngularJS 数字

AngularJS 数字就像 JavaScript 数字：

## AngularJS 实例

<div ng-app="" ng-init="quantity=1;cost=5">

<p>总价： {{ quantity \* cost }}</p>

</div>

[**尝试一下 »**](http://www.runoob.com/try/try.php?filename=try_ng_expressions)

使用 ng-bind 的相同实例：

## AngularJS 实例

<div ng-app="" ng-init="quantity=1;cost=5">

<p>总价： <span ng-bind="quantity \* cost"></span></p>

</div>

[**尝试一下 »**](http://www.runoob.com/try/try.php?filename=try_ng_bind_numbers)

|  |  |
| --- | --- |
| **IMG_256** | 使用 **ng-init** 不是很常见。您将在控制器一章中学习到一个更好的初始化数据的方式。 |

## AngularJS 字符串

AngularJS 字符串就像 JavaScript 字符串：

## AngularJS 实例

<div ng-app="" ng-init="firstName='John';lastName='Doe'"> <p>姓名： {{ firstName + " " + lastName }}</p>

</div>

[**尝试一下 »**](http://www.runoob.com/try/try.php?filename=try_ng_expressions_strings)

使用 ng-bind 的相同实例：

## AngularJS 实例

<div ng-app="" ng-init="firstName='John';lastName='Doe'"> <p>姓名： <span ng-bind="firstName + ' ' + lastName"></span></p>

</div>

[**尝试一下 »**](http://www.runoob.com/try/try.php?filename=try_ng_bind_strings)

## AngularJS 对象

AngularJS 对象就像 JavaScript 对象：

## AngularJS 实例

<div ng-app="" ng-init="person={firstName:'John',lastName:'Doe'}"> <p>姓为 {{ person.lastName }}</p> </div>

[**尝试一下 »**](http://www.runoob.com/try/try.php?filename=try_ng_expressions_objects)

使用 ng-bind 的相同实例：

## AngularJS 实例

<div ng-app="" ng-init="person={firstName:'John',lastName:'Doe'}"> <p>姓为 <span ng-bind="person.lastName"></span></p> </div>

[**尝试一下 »**](http://www.runoob.com/try/try.php?filename=try_ng_bind_objects)

## AngularJS 数组

AngularJS 数组就像 JavaScript 数组：

## AngularJS 实例

<div ng-app="" ng-init="points=[1,15,19,2,40]"> <p>第三个值为 {{ points[2] }}</p> </div>

[**尝试一下 »**](http://www.runoob.com/try/try.php?filename=try_ng_expressions_arrays)

使用 ng-bind 的相同实例：

## AngularJS 实例

<div ng-app="" ng-init="points=[1,15,19,2,40]"> <p>第三个值为 <span ng-bind="points[2]"></span></p> </div>

[**尝试一下 »**](http://www.runoob.com/try/try.php?filename=try_ng_bind_arrays)

## AngularJS 表达式 与 JavaScript 表达式

类似于 JavaScript 表达式，AngularJS 表达式可以包含字母，操作符，变量。

与 JavaScript 表达式不同，AngularJS 表达式可以写在 HTML 中。

与 JavaScript 表达式不同，AngularJS 表达式不支持条件判断，循环及异常。

与 JavaScript 表达式不同，AngularJS 表达式支持过滤器。

# AngularJS 指令

AngularJS 通过被称为 **指令** 的新属性来扩展 HTML。

AngularJS 通过内置的指令来为应用添加功能。

AngularJS 允许你自定义指令。

## AngularJS 指令

AngularJS 指令是扩展的 HTML 属性，带有前缀 **ng-**。

**ng-app** 指令初始化一个 AngularJS 应用程序。

输入值可以在此应用程序中通过绑定的变量访问

**ng-init** 指令初始化应用程序数据。

**ng-model** 指令把元素值（比如输入域的值）绑定到应用程序。

完整的指令内容可以参阅 [AngularJS 参考手册](http://www.runoob.com/angularjs/angularjs-reference.html)。

## AngularJS 实例

<div ng-app="" ng-init="firstName='John'">

<p>在输入框中尝试输入：</p>

<p>姓名：<input type="text" ng-model="firstName"></p>

<p>你输入的为： {{ firstName }}</p>

</div>

[**尝试一下 »**](http://www.runoob.com/try/try.php?filename=try_ng_directives)

**ng-app** 指令告诉 AngularJS，<div> 元素是 AngularJS **应用程序** 的"所有者"。

|  |  |
| --- | --- |
| **IMG_256** | 一个网页可以包含多个运行在不同元素中的 AngularJS 应用程序。 |

## 数据绑定

上面实例中的 **{{ firstName }}** 表达式是一个 AngularJS 数据绑定表达式。

AngularJS 中的数据绑定，同步了 AngularJS 表达式与 AngularJS 数据。

**{{ firstName }}** 是通过 **ng-model="firstName"** 进行同步。

在下一个实例中，两个文本域是通过两个 ng-model 指令同步的：

## AngularJS 实例

<div ng-app="" ng-init="quantity=1;price=5">

<h2>价格计算器</h2>

数量： <input type="number" ng-model="quantity">

价格： <input type="number" ng-model="price">

<p><b>总价：</b> {{ quantity \* price }}</p>

</div>

[**尝试一下 »**](http://www.runoob.com/try/try.php?filename=try_ng_binding)

|  |  |
| --- | --- |
| **IMG_257** | 使用 **ng-init** 不是很常见。您将在控制器一章中学习到一个更好的初始化数据的方式。 |

## 重复 HTML 元素

**ng-repeat** 指令会重复一个 HTML 元素：

## AngularJS 实例

<div ng-app="" ng-init="names=['Jani','Hege','Kai']">

<p>使用 ng-repeat 来循环数组</p>

<ul>

<li ng-repeat="x in names"> {{ x }}

</li>

</ul>

</div>

[**尝试一下 »**](http://www.runoob.com/try/try.php?filename=try_ng_repeat_array)

**ng-repeat** 指令用在一个对象数组上：

## AngularJS 实例

<div ng-app="" ng-init="names=[ {name:'Jani',country:'Norway'}, {name:'Hege',country:'Sweden'}, {name:'Kai',country:'Denmark'}]">

<p>循环对象：</p>

<ul> <li ng-repeat="x in names"> {{ x.name + ', ' + x.country }} </li>

</ul>

</div>

[**尝试一下 »**](http://www.runoob.com/try/try.php?filename=try_ng_repeat_object)

|  |  |
| --- | --- |
| **IMG_258** | AngularJS 完美支持数据库的 CRUD（增加Create、读取Read、更新Update、删除Delete）应用程序。 把实例中的对象想象成数据库中的记录。 |

## ng-app 指令

加载未完成时不会起作用

**ng-app** 指令定义了 AngularJS 应用程序的 **根元素**。

**ng-app** 指令在网页加载完毕时会**自动引导**（自动初始化）应用程序。

稍后您将学习到 **ng-app** 如何通过一个值（比如 ng-app="myModule"）连接到代码模块。

## ng-init 指令

**ng-init** 指令为 AngularJS 应用程序定义了 **初始值**。

通常情况下，不使用 ng-init。您将使用一个控制器或模块来代替它。

稍后您将学习更多有关控制器和模块的知识。

## ng-model 指令

同步了html元素数据到变量的值

**ng-model** 指令 **绑定 HTML 元素** 到应用程序数据。

**ng-model** 指令也可以：

* 为应用程序数据提供类型验证（number、email、required）。
* 为应用程序数据提供状态（invalid、dirty、touched、error）。
* 为 HTML 元素提供 CSS 类。
* 绑定 HTML 元素到 HTML 表单。

## ng-repeat 指令

**ng-repeat** 指令对于集合中（数组中）的每个项会 **克隆一次 HTML 元素**。

## 创建自定义的指令

除了 AngularJS 内置的指令外，我们还可以创建自定义指令。

你可以使用 **.directive** 函数来添加自定义的指令。

要调用自定义指令，HTML 元素上需要添加自定义指令名。

使用驼峰法来命名一个指令， **runoobDirective**, 但在使用它时需要以 **-** 分割, **runoob-directive**:

## AngularJS 实例

<body ng-app="myApp">  
  
<runoob-directive></runoob-directive>  
  
<script>  
var app = angular.module("myApp", []);  
app.directive("runoobDirective", function() {  
    return {  
        template : "<h1>自定义指令!</h1>"  
    };  
});  
</script>  
  
</body>

[**尝试一下 »**](http://www.runoob.com/try/try.php?filename=try_ng_directive_tagname)

你可以通过以下方式来调用指令：

* 元素名
* 属性
* 类名 需要设置restrict为C
* 注释 需要设置restrict为M，且replace为true

以下实例方式也能输出同样结果:

元素名

<runoob-directive></runoob-directive>

[**尝试一下 »**](http://www.runoob.com/try/try.php?filename=try_ng_directive_element)

属性

<div runoob-directive></div>

[**尝试一下 »**](http://www.runoob.com/try/try.php?filename=try_ng_directive_attribute)

类名

<div class="runoob-directive"></div>

[**尝试一下 »**](http://www.runoob.com/try/try.php?filename=try_ng_directive_class)

注释

<!-- directive: runoob-directive -->

[**尝试一下 »**](http://www.runoob.com/try/try.php?filename=try_ng_directive_comment)

## 限制使用

你可以限制你的指令只能通过特定的方式来调用。

### 实例

通过添加 **restrict** 属性,并设置只值为 "A", 来设置指令只能通过属性的方式来调用:

var app = angular.module("myApp", []);  
app.directive("runoobDirective", function() {  
    return {  
        restrict : "A",  
        template : "<h1>自定义指令!</h1>"  
    };  
});

[**尝试一下 »**](http://www.runoob.com/try/try.php?filename=try_ng_directive_restrictions)

**restrict** 值可以是以下几种:

* E 作为元素名使用
* A 作为属性使用
* C 作为类名使用
* M 作为注释使用

**restrict** 默认值为 EA, 即可以通过元素名和属性名来调用指令。

# AngularJS ng-model 指令

ng-model 指令用于绑定应用程序数据到 HTML 控制器(input, select, textarea)的值。

## ng-model 指令

ng-model 指令可以将输入域的值与 AngularJS 创建的变量绑定。

## AngularJS 实例

<div ng-app="myApp" ng-controller="myCtrl">  
    名字: <input ng-model="name">  
</div>  
  
<script>  
var app = angular.module('myApp', []);  
app.controller('myCtrl', function($scope) {  
    $scope.name = "John Doe";  
});  
</script>

[**尝试一下 »**](http://www.runoob.com/try/try.php?filename=try_ng_model)

## 双向绑定

双向绑定，在修改输入域的值时， AngularJS 属性的值也将修改：

## AngularJS 实例

<div ng-app="myApp" ng-controller="myCtrl">  
    名字: <input ng-model="name">  
    <h1>你输入了: {{name}}</h1>  
</div>

[**尝试一下 »**](http://www.runoob.com/try/try.php?filename=try_ng_model_two-way)

## 验证用户输入

## AngularJS 实例

<form ng-app="" name="myForm">  
    Email:  
    <input type="email" name="myAddress" ng-model="text">  
    <span ng-show="myForm.myAddress.$error.email">不是一个合法的邮箱地址</span>  
</form>

[**尝试一下 »**](http://www.runoob.com/try/try.php?filename=try_ng_model_validate)

以上实例中，提示信息会在 ng-show 属性返回 true 的情况下显示。

## 应用状态

ng-model 指令可以为应用数据提供状态值(invalid, dirty, touched, error):

## AngularJS 实例

<form ng-app="" name="myForm" ng-init="myText = 'test@runoob.com'">  
    Email:  
    <input type="email" name="myAddress" ng-model="myText" required></p>  
    <h1>状态</h1>  
    {{myForm.myAddress.$valid}}  
    {{myForm.myAddress.$dirty}}  
    {{myForm.myAddress.$touched}}  
</form>

[**尝试一下 »**](http://www.runoob.com/try/try.php?filename=try_ng_model_status)

## CSS 类

ng-model 指令基于它们的状态为 HTML 元素提供了 CSS 类：

## AngularJS 实例

<style>

input.ng-invalid {  
    background-color: lightblue;  
}

</style>  
<body>  
  
<form ng-app="" name="myForm">  
    输入你的名字:  
    <input name="myAddress" ng-model="text" required>  
</form>

[**尝试一下 »**](http://www.runoob.com/try/try.php?filename=try_ng_model_css)

ng-model 指令根据表单域的状态添加/移除以下类：

* ng-empty
* ng-not-empty
* ng-touched
* ng-untouched
* ng-valid
* ng-invalid
* ng-dirty
* ng-pending
* ng-pristine

# AngularJS Scope(作用域)

Scope(作用域) 是应用在 HTML (视图) 和 JavaScript (控制器)之间的纽带。

Scope 是一个对象，有可用的方法和属性。

Scope 可应用在视图和控制器上。

## 如何使用 Scope

当你在 AngularJS 创建控制器时，你可以将 **$scope** 对象当作一个参数传递:

## AngularJS 实例

控制器中的属性对应了视图上的属性：

<div ng-app="myApp" ng-controller="myCtrl">  
  
<h1>{{carname}}</h1>  
  
</div>  
  
<script>

var app = angular.module('myApp', []);  
  
app.controller('myCtrl', function($scope) {  
    $scope.carname = "Volvo";  
});

</script>

[**尝试一下 »**](http://www.runoob.com/try/try.php?filename=try_ng_scope)

当在控制器中添加 **$scope** 对象时，视图 (HTML) 可以获取了这些属性。

视图中，你不需要添加 **$scope** 前缀, 只需要添加属性名即可，如： **{{carname}}**。

## Scope 概述

AngularJS 应用组成如下：

* View(视图), 即 HTML。
* Model(模型), 当前视图中可用的数据。
* Controller(控制器), 即 JavaScript 函数，可以添加或修改属性。

scope 是模型。

scope 是一个 JavaScript 对象，带有属性和方法，这些属性和方法可以在视图和控制器中使用。

## AngularJS 实例

如果你修改了视图，模型和控制器也会相应更新：

<div ng-app="myApp" ng-controller="myCtrl">  
  
<input ng-model="name">  
  
<h1>我的名字是 {{name}}</h1>  
  
</div>  
  
<script>

var app = angular.module('myApp', []);  
  
app.controller('myCtrl', function($scope) {  
    $scope.name = "John Dow";  
});

</script>

[**尝试一下 »**](http://www.runoob.com/try/try.php?filename=try_ng_scope_sync)

## Scope 作用范围

了解你当前使用的 scope 是非常重要的。

在以上两个实例中，只有一个作用域 scope，所以处理起来比较简单，但在大型项目中， HTML DOM 中有多个作用域，这时你就需要知道你使用的 scope 对应的作用域是哪一个。

## AngularJS 实例

当我们使用 **ng-repeat** 指令时，每个重复项都访问了当前的重复对象:

<div ng-app="myApp" ng-controller="myCtrl">  
  
<ul>  
    <li ng-repeat="x in names">{{x}}</li>  
</ul>  
  
</div>  
  
<script>

var app = angular.module('myApp', []);  
  
app.controller('myCtrl', function($scope) {  
    $scope.names = ["Emil", "Tobias", "Linus"];  
});

</script>

[**尝试一下 »**](http://www.runoob.com/try/try.php?filename=try_ng_scope_repeat)

每个 **<li>** 元素可以访问当前的重复对象，这里对应的是一个字符串， 并使用变量 **x** 表示。

## 根作用域

所有的应用都有一个 **$rootScope**，它可以作用在 **ng-app** 指令包含的所有 HTML 元素中。

**$rootScope** 可作用于整个应用中。是各个 controller 中 scope 的桥梁。用 rootscope 定义的值，可以在各个 controller 中使用。

## AngularJS 实例

创建控制器时，将 $rootScope 作为参数传递，可在应用中使用：

<div ng-app="myApp" ng-controller="myCtrl">  
  
<h1>{{lastname}} 家族成员:</h1>  
  
<ul>  
    <li ng-repeat="x in names">{{x}} {{lastname}}</li>  
</ul>  
  
</div>  
  
<script>

var app = angular.module('myApp', []);  
  
app.controller('myCtrl', function($scope, $rootScope) {  
    $scope.names = ["Emil", "Tobias", "Linus"];  
    $rootScope.lastname = "Refsnes";  
});

</script>

[**尝试一下 »**](http://www.runoob.com/try/try.php?filename=try_ng_scope_rootscope)

# AngularJS 控制器

 AngularJS 控制器 **控制** AngularJS 应用程序的数据。

 AngularJS 控制器是常规的 **JavaScript 对象**。

## AngularJS 控制器

AngularJS 应用程序被控制器控制。

**ng-controller** 指令定义了应用程序控制器。

控制器是 **JavaScript 对象**，由标准的 JavaScript **对象的构造函数** 创建。

## AngularJS 实例

<div ng-app="myApp" ng-controller="myCtrl">  
  
名: <input type="text" ng-model="firstName"><br>  
姓: <input type="text" ng-model="lastName"><br>  
<br>  
姓名: {{firstName + " " + lastName}}  
  
</div>  
  
<script>  
var app = angular.module('myApp', []);  
app.controller('myCtrl', function($scope) {  
    $scope.firstName = "John";  
    $scope.lastName = "Doe";  
});  
</script>

[**尝试一下 »**](http://www.runoob.com/try/tryit.php?filename=try_ng_controller)

应用解析：

AngularJS 应用程序由 **ng-app** 定义。应用程序在 <div> 内运行。

**ng-controller="myCtrl"** 属性是一个 AngularJS 指令。用于定义一个控制器。

**myCtrl** 函数是一个 JavaScript 函数。

AngularJS 使用**$scope** 对象来调用控制器。

在 AngularJS 中， $scope 是一个应用对象(属于应用变量和函数)。

控制器的 **$scope** （相当于作用域、控制范围）用来保存AngularJS Model(模型)的对象。

控制器在作用域中创建了两个属性 (**firstName** 和 **lastName**)。

**ng-model** 指令绑定输入域到控制器的属性（firstName 和 lastName）。

## 控制器方法

上面的实例演示了一个带有 lastName 和 firstName 这两个属性的控制器对象。

控制器也可以有方法（变量和函数）：

## AngularJS 实例

<div ng-app="myApp" ng-controller="personCtrl">  
  
名: <input type="text" ng-model="firstName"><br>  
姓: <input type="text" ng-model="lastName"><br>  
<br>  
姓名: {{fullName()}}  
  
</div>  
  
<script>  
var app = angular.module('myApp', []);  
app.controller('personCtrl', function($scope) {  
    $scope.firstName = "John";  
    $scope.lastName = "Doe";  
    $scope.fullName = function() {  
        return $scope.firstName + " " + $scope.lastName;  
    }  
});  
</script>

[**尝试一下 »**](http://www.runoob.com/try/tryit.php?filename=try_ng_controller_property)

## 外部文件中的控制器

在大型的应用程序中，通常是把控制器存储在外部文件中。

只需要把 <script> 标签中的代码复制到名为 [personController.js](http://www.runoob.com/try/demo_source/personController.js) 的外部文件中即可：

## AngularJS 实例

<div ng-app="myApp" ng-controller="personCtrl">  
  
First Name: <input type="text" ng-model="firstName"><br>  
Last Name: <input type="text" ng-model="lastName"><br>  
<br>  
Full Name: {{firstName + " " + lastName}}  
  
</div>  
  
<script src="personController.js"></script>

[**尝试一下 »**](http://www.runoob.com/try/tryit.php?filename=try_ng_controller_js)

## 其他实例

以下实例创建一个新的控制器文件:

angular.module('myApp', []).controller('namesCtrl', function($scope) { $scope.names = [ {name:'Jani',country:'Norway'}, {name:'Hege',country:'Sweden'}, {name:'Kai',country:'Denmark'} ]; });

保存文件为  [namesController.js](http://www.runoob.com/try/demo_source/namesController.js):

然后，在应用中使用控制器文件:

## AngularJS 实例

<div ng-app="myApp" ng-controller="namesCtrl">  
  
<ul>  
  <li ng-repeat="x in names">  
    {{ x.name + ', ' + x.country }}  
  </li>  
</ul>  
  
</div>  
  
<script src="namesController.js"></script>

[**尝试一下 »**](http://www.runoob.com/try/tryit.php?filename=try_ng_controller_names)

# AngularJS 过滤器

过滤器可以使用一个管道字符（|）添加到表达式和指令中。

## AngularJS 过滤器

AngularJS 过滤器可用于转换数据：

|  |  |
| --- | --- |
| **过滤器** | **描述** |
| currency | 格式化数字为货币格式。 |
| filter | 从数组项中选择一个子集。 |
| lowercase | 格式化字符串为小写。 |
| orderBy | 根据某个表达式排列数组。 |
| uppercase | 格式化字符串为大写。 |

## 表达式中添加过滤器

过滤器可以通过一个管道字符（|）和一个过滤器添加到表达式中。.

(（下面的两个实例，我们将使用前面章节中提到的 person 控制器）)

**uppercase** 过滤器将字符串格式化为大写：

## AngularJS 实例

<div ng-app="myApp" ng-controller="personCtrl">  
  
<p>姓名为 {{ lastName | uppercase }}</p>  
  
</div>

[**尝试一下 »**](http://www.runoob.com/try/tryit.php?filename=try_ng_filters_uppercase)

**lowercase** 过滤器将字符串格式化为小写：

## AngularJS 实例

<div ng-app="myApp" ng-controller="personCtrl">  
  
<p>姓名为 {{ lastName | lowercase }}</p>  
  
</div>

[**尝试一下 »**](http://www.runoob.com/try/tryit.php?filename=try_ng_filters_lowercase)

## currency 过滤器

**currency** 过滤器将数字格式化为货币格式：

## AngularJS 实例

<div ng-app="myApp" ng-controller="costCtrl">  
  
<input type="number" ng-model="quantity">  
<input type="number" ng-model="price">  
  
<p>总价 = {{ (quantity \* price) | currency }}</p>  
  
</div>

[**尝试一下 »**](http://www.runoob.com/try/tryit.php?filename=try_ng_filters_currency)

## 向指令添加过滤器

过滤器可以通过一个管道字符（|）和一个过滤器添加到指令中。

**orderBy** 过滤器根据表达式排列数组：

## AngularJS 实例

<div ng-app="myApp" ng-controller="namesCtrl">  
  
<ul>  
  <li ng-repeat="x in names | orderBy:'country'">  
    {{ x.name + ', ' + x.country }}  
  </li>  
</ul>  
  
</div>

[**尝试一下 »**](http://www.runoob.com/try/tryit.php?filename=try_ng_filters_orderby)

## 过滤输入

输入过滤器可以通过一个管道字符（|）和一个过滤器添加到指令中，该过滤器后跟一个冒号和一个模型名称。

**filter** 过滤器从数组中选择一个子集：

## AngularJS 实例

<div ng-app="myApp" ng-controller="namesCtrl">  
  
<p><input type="text" ng-model="test"></p>  
  
<ul>  
  <li ng-repeat="x in names | filter:test | orderBy:'country'">  
    {{ (x.name | uppercase) + ', ' + x.country }}  
  </li>  
</ul>  
  
</div>

[**尝试一下 »**](http://www.runoob.com/try/tryit.php?filename=try_ng_filters_input)

# AngularJS 服务(Service)

AngularJS 中你可以创建自己的服务，或使用内建服务。

## 什么是服务？

在 AngularJS 中，服务是一个函数或对象，可在你的 AngularJS 应用中使用。

AngularJS 内建了30 多个服务。

有个 **$location** 服务，它可以返回当前页面的 URL 地址。

### 实例

var app = angular.module('myApp', []);  
app.controller('customersCtrl', function($scope, $location) {  
    $scope.myUrl = $location.absUrl();  
});

[**尝试一下 »**](http://www.runoob.com/try/tryit.php?filename=try_ng_services)

注意 **$location** 服务是作为一个参数传递到 controller 中。如果要使用它，需要在 controller 中定义。

## 为什么使用服务?

在很多服务中，比如 $location 服务，它可以使用 DOM 中存在的对象，类似 window.location 对象，但 window.location 对象在 AngularJS 应用中有一定的局限性。

AngularJS 会一直监控应用，处理事件变化， AngularJS 使用 **$location** 服务比使用**window.location** 对象更好。

### $location vs window.location

|  |  |  |
| --- | --- | --- |
|  | **window.location** | **$location.service** |
| 目的 | 允许对当前浏览器位置进行读写操作 | 允许对当前浏览器位置进行读写操作 |
| API | 暴露一个"裸聊"的能被读写的对象 | 暴露jquery风格的读写器 |
| 是否在AngularJS应用生命周期中和应用整合 | 否 | 可获取到应用生命周期内的每一个阶段，并且和$watch整合 |
| 是否和HTML5 API的无缝整合 | 否 | 是（对低级浏览器优雅降级） |
| 和应用的上下文是否相关 | 否，window.location.path返回"/docroot/actual/path" | 是，$location.path()返回"/actual/path" |

## $http 服务

**$http** 是 AngularJS 应用中最常用的服务。 服务向服务器发送请求，应用响应服务器传送过来的数据。

### 实例

使用 **$http** 服务向服务器请求数据:

var app = angular.module('myApp', []);  
app.controller('myCtrl', function($scope, $http) {  
    $http.get("welcome.htm").then(function (response) {  
        $scope.myWelcome = response.data;  
    });  
});

[**尝试一下 »**](http://www.runoob.com/try/tryit.php?filename=try_ng_services_http)

以上是一个非常简单的 **$http** 服务实例，更多 **$http** 服务应用请查看 [AngularJS Http 教程](http://www.runoob.com/angularjs/angularjs-http.html)。

## $timeout 服务

AngularJS **$timeout** 服务对应了 JS **window.setTimeout** 函数。

### 实例

两秒后显示信息:

var app = angular.module('myApp', []);  
app.controller('myCtrl', function($scope, $timeout) {  
    $scope.myHeader = "Hello World!";  
    $timeout(function () {  
        $scope.myHeader = "How are you today?";  
    }, 2000);  
});

[**尝试一下 »**](http://www.runoob.com/try/tryit.php?filename=try_ng_services_timeout)

## $interval 服务

AngularJS **$interval** 服务对应了 JS **window.setInterval** 函数。

### 实例

每两秒显示信息:

var app = angular.module('myApp', []);  
app.controller('myCtrl', function($scope, $interval) {  
    $scope.theTime = new Date().toLocaleTimeString();  
    $interval(function () {  
        $scope.theTime = new Date().toLocaleTimeString();  
    }, 1000);  
});

[**尝试一下 »**](http://www.runoob.com/try/tryit.php?filename=try_ng_services_interval)

## 创建自定义服务

你可以创建访问自定义服务，链接到你的模块中：

创建名为**hexafy** 的访问:

app.service('hexafy', function() {  
    this.myFunc = function (x) {  
        return x.toString(16);  
    }  
});

要使用访问自定义服务，需要在定义过滤器的时候独立添加:

### 实例

使用自定义的的服务 **hexafy** 将一个数字转换为16进制数:

app.controller('myCtrl', function($scope, **hexafy**) {  
    $scope.hex = **hexafy**.myFunc(255);  
});

[**尝试一下 »**](http://www.runoob.com/try/tryit.php?filename=try_ng_services_custom)

## 过滤器中，使用自定义服务

当你创建了自定义服务，并连接到你的应用上后，你可以在控制器，指令，过滤器或其他服务中使用它。

在过滤器 **myFormat** 中使用服务 **hexafy**:

app.filter('myFormat',[**'hexafy'**, function(**hexafy**) {  
    return function(x) {  
        return **hexafy**.myFunc(x);  
    };  
}]);

[**尝试一下 »**](http://www.runoob.com/try/tryit.php?filename=try_ng_services_filter)

在对象数组中获取值时你可以使用过滤器：

创建服务 **hexafy**:

<ul>  
<li ng-repeat="x in counts">{{x | myFormat}}</li>  
</ul>

[**尝试一下 »**](http://www.runoob.com/try/tryit.php?filename=try_ng_services_filter2)

# AngularJS XMLHttpRequest

**$http** 是 AngularJS 中的一个核心服务，用于读取远程服务器的数据。

## 读取 JSON 文件

以下是存储在web服务器上的 JSON 文件：

## http://www.runoob.com/try/angularjs/data/sites.php

{ "sites": [ { "Name": "菜鸟教程", "Url": "www.runoob.com", "Country": "CN" }, { "Name": "Google", "Url": "www.google.com", "Country": "USA" }, { "Name": "Facebook", "Url": "www.facebook.com", "Country": "USA" }, { "Name": "微博", "Url": "www.weibo.com", "Country": "CN" } ] }

## AngularJS $http

AngularJS $http 是一个用于读取web服务器上数据的服务。

$http.get(url) 是用于读取服务器数据的函数。

## AngularJS 实例

<div ng-app="myApp" ng-controller="siteCtrl"> <ul> <li ng-repeat="x in names"> {{ x.Name + ', ' + x.Country }} </li> </ul> </div> <script> var app = angular.module('myApp', []); app.controller('siteCtrl', function($scope, $http) { $http.get("http://www.runoob.com/try/angularjs/data/sites.php") .success(function (response) {$scope.names = response.sites;}); }); </script>

[**尝试一下 »**](http://www.runoob.com/try/tryit.php?filename=try_ng_customers_json)

**应用解析:**

注意：以上代码的 get 请求是本站的服务器，你不能直接拷贝到你本地运行，会存在跨域问题，解决办法就是将 Customers\_JSON.php 的数据拷贝到你自己的服务器上，附：[PHP Ajax 跨域问题最佳解决方案](http://www.runoob.com/w3cnote/php-ajax-cross-border.html)。

AngularJS 应用通过 **ng-app** 定义。应用在 <div> 中执行。

**ng-controller** 指令设置了 **controller 对象** 名。

函数 **customersController** 是一个标准的 JavaScript **对象构造器**。

控制器对象有一个属性: **$scope.names**。

**$http.get()** 从web服务器上读取静态 **JSON 数据**。

服务器数据文件为：  <http://www.runoob.com/try/angularjs/data/sites.php>。

当从服务端载入 JSON 数据时，**$scope.names** 变为一个数组。

|  |  |
| --- | --- |
| **IMG_256** | 以上代码也可以用于读取数据库数据。 |

# AngularJS Select(选择框)

AngularJS 可以使用数组或对象创建一个下拉列表选项。

## 使用 ng-options 创建选择框

在 AngularJS 中我们可以使用 **ng-option** 指令来创建一个下拉列表，列表项通过对象和数组循环输出，如下实例:

### 实例

<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>

[**尝试一下 »**](http://www.runoob.com/try/tryit.php?filename=try_ng_select)

## ng-options 与 ng-repeat

我们也可以使用**ng-repeat** 指令来创建下拉列表：

### 实例

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

[**尝试一下 »**](http://www.runoob.com/try/tryit.php?filename=try_ng_select_repeat)

**ng-repeat** 指令是通过数组来循环 HTML 代码来创建下拉列表，但 **ng-options** 指令更适合创建下拉列表，它有以下优势：

使用 **ng-options** 的选项的一个对象， **ng-repeat** 是一个字符串。

## 应该用哪个更好?

假设我们使用以下对象:

$scope.sites = [

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

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

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

**ng-repeat** 有局限性，选择的值是一个字符串:

### 实例

使用 **ng-repeat**:

<select ng-model="selectedSite">  
<option ng-repeat="x in sites" value="{{x.url}}">{{x.site}}</option>  
</select>  
  
<h1>你选择的是: {{selectedSite}}</h1>

[**尝试一下 »**](http://www.runoob.com/try/tryit.php?filename=try_ng_select_repeat_selected)

使用 **ng-options** 指令，选择的值是一个对象：

### 实例

使用 **ng-options**:

<select ng-model="selectedSite" ng-options="x.site for x in sites">  
</select>  
  
<h1>你选择的是: {{selectedSite.site}}</h1>  
<p>网址为: {{selectedSite.url}}</p>

[**尝试一下 »**](http://www.runoob.com/try/tryit.php?filename=try_ng_select_selected)

当选择值是一个对象时，我们就可以获取更多信息，应用也更灵活。

## 数据源为对象

前面实例我们使用了数组作为数据源，以下我们将数据对象作为数据源。

$scope.sites = {

site01 : "Google",

site02 : "Runoob",

site03 : "Taobao"};

**ng-options** 使用对象有很大的不同，如下所示：

### 实例

使用对象作为数据源, **x** 为键(key), **y** 为值(value):

<select ng-model="selectedSite" ng-options="**x for (x, y) in sites**">  
</select>  
  
<h1>你选择的值是: {{selectedSite}}</h1>

[**尝试一下 »**](http://www.runoob.com/try/tryit.php?filename=try_ng_select_object)

你选择的值为在 key-**value** 对中的 **value**。

**value** 在 key-**value** 对中也可以是个对象：

### 实例

选择的值在 key-**value** 对的 **value** 中, 这里它是一个对象:

$scope.cars = {  
car01 : {brand : "Ford", model : "Mustang", color : "red"},  
car02 : {brand : "Fiat", model : "500", color : "white"},  
car03 : {brand : "Volvo", model : "XC90", color : "black"}  
};

[**尝试一下 »**](http://www.runoob.com/try/tryit.php?filename=try_ng_select_object_object)

在下拉菜单也可以不使用 **key**-value 对中的 **key** , 直接使用对象的属性：

### 实例

<select ng-model="selectedCar" ng-options="**y.brand** for (x, y) in cars">  
</select>

[**尝试一下 »**](http://www.runoob.com/try/tryit.php?filename=try_ng_select_object_y)

# AngularJS 表格

ng-repeat 指令可以完美的显示表格。

## 在表格中显示数据

使用 angular 显示表格是非常简单的：

## AngularJS 实例

<div ng-app="myApp" ng-controller="customersCtrl">   
  
<table>  
  <tr ng-repeat="x in names">  
    <td>{{ x.Name }}</td>  
    <td>{{ x.Country }}</td>  
  </tr>  
</table>  
  
</div>  
  
<script>  
var app = angular.module('myApp', []);  
app.controller('customersCtrl', function($scope, $http) {  
    $http.get("http://www.runoob.com/try/angularjs/data/Customers\_JSON.php")  
    .success(function (response) {$scope.names = response.records;});  
});  
</script>

[**尝试一下 »**](http://www.runoob.com/try/tryit.php?filename=try_ng_tables_simple)

## 使用 CSS 样式

为了让页面更加美观，我们可以在页面中使用CSS：

## CSS 样式

<style>  
table, th , td {  
  border: 1px solid grey;  
  border-collapse: collapse;  
  padding: 5px;  
}  
table tr:nth-child(odd) {  
  background-color: #f1f1f1;  
}  
table tr:nth-child(even) {  
  background-color: #ffffff;  
}  
</style>

[**尝试一下 »**](http://www.runoob.com/try/tryit.php?filename=try_ng_tables_css)

## 使用 orderBy 过滤器

排序显示，可以使用 **orderBy** 过滤器:

## AngularJS 实例

<table>  
  <tr ng-repeat="x in names | orderBy : 'Country'">  
    <td>{{ x.Name }}</td>  
    <td>{{ x.Country }}</td>  
  </tr>  
</table>

[**尝试一下 »**](http://www.runoob.com/try/tryit.php?filename=try_ng_tables_orderby)

## 使用 uppercase 过滤器

使用 **uppercase** 过滤器转换为大写:

## AngularJS 实例

<table>  
  <tr ng-repeat="x in names">  
    <td>{{ x.Name }}</td>  
    <td>{{ x.Country | uppercase }}</td>  
  </tr>  
</table>

[**尝试一下 »**](http://www.runoob.com/try/tryit.php?filename=try_ng_tables_ucase)

## 显示序号 ($index)

表格显示序号可以在 <td> 中添加 **$index**:

## AngularJS 实例

<table>  
  <tr ng-repeat="x in names">  
    <td>{{ $index + 1 }}</td>  
    <td>{{ x.Name }}</td>  
    <td>{{ x.Country }}</td>  
  </tr>  
</table>

[**尝试一下 »**](http://www.runoob.com/try/tryit.php?filename=try_ng_tables_index)

## 使用 $even 和 $odd

## AngularJS 实例

<table>  
<tr ng-repeat="x in names">  
<td ng-if="$odd" style="background-color:#f1f1f1">{{ x.Name }}</td>  
<td ng-if="$even">{{ x.Name }}</td>  
<td ng-if="$odd" style="background-color:#f1f1f1">{{ x.Country }}</td>  
<td ng-if="$even">{{ x.Country }}</td>  
</tr>  
</table>

[**尝试一下 »**](http://www.runoob.com/try/tryit.php?filename=try_ng_tables_even)

# AngularJS SQL

在前面章节中的代码也可以用于读取数据库中的数据。

## 使用 PHP 从 MySQL 中获取数据

## AngularJS 实例

<div ng-app="myApp" ng-controller="customersCtrl">   
  
<table>  
  <tr ng-repeat="x in names">  
    <td>{{ x.Name }}</td>  
    <td>{{ x.Country }}</td>  
  </tr>  
</table>  
  
</div>  
  
<script>  
var app = angular.module('myApp', []);  
app.controller('customersCtrl', function($scope, $http) {  
    $http.get("http://www.runoob.com/try/angularjs/data/Customers\_MySQL.php")  
    .success(function (response) {$scope.names = response.records;});  
});  
</script>

[**尝试一下 »**](http://www.runoob.com/try/tryit.php?filename=try_ng_customers_mysql)

## ASP.NET 中执行 SQL 获取数据

## AngularJS 实例

<div ng-app="myApp" ng-controller="customersCtrl">   
  
<table>  
  <tr ng-repeat="x in names">  
    <td>{{ x.Name }}</td>  
    <td>{{ x.Country }}</td>  
  </tr>  
</table>  
  
</div>  
  
<script>  
var app = angular.module('myApp', []);  
app.controller('customersCtrl', function($scope, $http) {  
    $http.get("http://www.runoob.com/try/angularjs/data/Customers\_SQL.aspx")  
    .success(function (response) {$scope.names = response.records;});  
});  
</script>

[**尝试一下 »**](http://www.runoob.com/try/tryit.php?filename=try_ng_customers_sql)

## 服务端代码

以下列出了几种服务端代码类型：

1. 使用 PHP 和 MySQL。返回 JSON。
2. 使用 PHP 和 MS Access。返回 JSON。
3. 使用 ASP.NET, VB, 及 MS Access。 返回 JSON。
4. 使用 ASP.NET, Razor, 及 SQL Lite。 返回 JSON。

## 跨域 HTTP 请求

如果你需要从不同的服务器（不同域名）上获取数据就需要使用跨域 HTTP 请求。

跨域请求在网页上非常常见。很多网页从不同服务器上载入 CSS, 图片，Js脚本等。

在现代浏览器中，为了数据的安全，所有请求被严格限制在同一域名下，如果需要调用不同站点的数据，需要通过跨域来解决。

以下的 PHP 代码运行使用的网站进行跨域访问。

header("Access-Control-Allow-Origin: \*"); 任何浏览器都可以访问这个文件，因为它这里的http请求的是ajax

更多跨域访问解决方案可参阅：[PHP Ajax 跨域问题最佳解决方案](http://www.runoob.com/w3cnote/php-ajax-cross-border.html)。

## 1. PHP 和 MySql 代码实例

<?php  
header("Access-Control-Allow-Origin: \*");  
header("Content-Type: application/json; charset=UTF-8");  
  
$conn = new mysqli("myServer", "myUser", "myPassword", "Northwind");  
  
$result = $conn->query("SELECT CompanyName, City, Country FROM Customers");  
  
$outp = "";  
while($rs = $result->fetch\_array(MYSQLI\_ASSOC)) {  
    if ($outp != "") {$outp .= ",";}  
    $outp .= '{"Name":"'  . $rs["CompanyName"] . '",';  
    $outp .= '"City":"'   . $rs["City"]        . '",';  
    $outp .= '"Country":"'. $rs["Country"]     . '"}';   
}  
$outp ='{"records":['.$outp.']}';  
$conn->close();  
  
echo($outp);  
?>

## 2. PHP 和 MS Access 代码实例

<?php  
header("Access-Control-Allow-Origin: \*");  
header("Content-Type: application/json; charset=ISO-8859-1");  
  
$conn = new COM("ADODB.Connection");  
$conn->open("PROVIDER=Microsoft.Jet.OLEDB.4.0;Data Source=Northwind.mdb");  
  
$rs = $conn->execute("SELECT CompanyName, City, Country FROM Customers");  
  
$outp = "";  
while (!$rs->EOF) {  
    if ($outp != "") {$outp .= ",";}  
    $outp .= '{"Name":"'  . $rs["CompanyName"] . '",';  
    $outp .= '"City":"'   . $rs["City"]        . '",';  
    $outp .= '"Country":"'. $rs["Country"]     . '"}';   
    $rs->MoveNext();  
}  
$outp ='{"records":['.$outp.']}';  
  
$conn->close();  
  
echo ($outp);  
?>

## 3. ASP.NET, VB 和 MS Access 代码实例

<%@ Import Namespace="System.IO"%>  
<%@ Import Namespace="System.Data"%>  
<%@ Import Namespace="System.Data.OleDb"%>  
<%  
Response.AppendHeader("Access-Control-Allow-Origin", "\*")  
Response.AppendHeader("Content-type", "application/json")  
Dim conn As OleDbConnection  
Dim objAdapter As OleDbDataAdapter  
Dim objTable As DataTable  
Dim objRow As DataRow  
Dim objDataSet As New DataSet()  
Dim outp  
Dim c  
conn = New OledbConnection("Provider=Microsoft.Jet.OLEDB.4.0;data source=Northwind.mdb")  
objAdapter = New OledbDataAdapter("SELECT CompanyName, City, Country FROM Customers", conn)  
objAdapter.Fill(objDataSet, "myTable")  
objTable=objDataSet.Tables("myTable")  
  
outp = ""  
c = chr(34)  
for each x in objTable.Rows  
if outp <> "" then outp = outp & ","  
outp = outp & "{" & c & "Name"    & c & ":" & c & x("CompanyName") & c & ","  
outp = outp &       c & "City"    & c & ":" & c & x("City")        & c & ","   
outp = outp &       c & "Country" & c & ":" & c & x("Country")     & c & "}"  
next  
  
outp ="{" & c & "records" & c & ":[" & outp & "]}"  
response.write(outp)  
conn.close  
%>

## 4. ASP.NET, VB Razor 和 SQL Lite 代码实例

@{  
Response.AppendHeader("Access-Control-Allow-Origin", "\*")  
Response.AppendHeader("Content-type", "application/json")  
var db = Database.Open("Northwind");  
var query = db.Query("SELECT CompanyName, City, Country FROM Customers");  
var outp =""  
var c = chr(34)  
}  
@foreach(var row in query)  
{  
if outp <> "" then outp = outp + ","  
outp = outp + "{" + c + "Name"    + c + ":" + c + @row.CompanyName + c + ","  
outp = outp +       c + "City"    + c + ":" + c + @row.City        + c + ","  
outp = outp +       c + "Country" + c + ":" + c + @row.Country     + c + "}"  
}  
outp ="{" + c + "records" + c + ":[" + outp + "]}"  
@outp

# AngularJS HTML DOM

AngularJS 为 HTML DOM 元素的属性提供了绑定应用数据的指令。

## ng-disabled 指令

**ng-disabled** 指令直接绑定应用程序数据到 HTML 的 disabled 属性。

## AngularJS 实例

<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>  
  
<p>  
{{ mySwitch }}  
</p>  
  
</div>

[**尝试一下 »**](http://www.runoob.com/try/tryit.php?filename=try_ng_html_disabled)

实例讲解：

**ng-disabled** 指令绑定应用程序数据 "mySwitch" 到 HTML 的 disabled 属性。

**ng-model** 指令绑定 "mySwitch" 到 HTML input checkbox 元素的内容（value）。

如果 **mySwitch** 为**true**, 按钮将不可用:

<p>  
<button disabled>点我！</button>  
</p>

如果 **mySwitch** 为**false**, 按钮则可用:

<p>  
<button>点我!</button>  
</p>

## ng-show 指令

**ng-show** 指令隐藏或显示一个 HTML 元素。

## AngularJS 实例

<div ng-app="">  
  
<p ng-show="true">我是可见的。</p>  
  
<p ng-show="false">我是不可见的。</p>  
  
</div>

[**尝试一下 »**](http://www.runoob.com/try/tryit.php?filename=try_ng_html_show)

ng-show 指令根据 **value** 的值来显示（隐藏）HTML 元素。

你可以使用表达式来计算布尔值（ true 或 false）:

## AngularJS 实例

<div ng-app="" ng-init="hour=13"> <p ng-show="hour > 12">我是可见的。</p> </div>

[**尝试一下 »**](http://www.runoob.com/try/tryit.php?filename=try_ng_html_show_if)

|  |  |
| --- | --- |
| **IMG_256** | 在下一个章节中，我们将为大家更多通过点击按钮来隐藏 HTML 元素的实例。 |

## ng-hide 指令

**ng-hide** 指令用于隐藏或显示 HTML 元素。

## AngularJS 实例

<div ng-app="">  
  
<p ng-hide="true">我是不可见的。</p>  
  
<p ng-hide="false">我是可见的。</p>  
  
</div>

[**尝试一下 »**](http://www.runoob.com/try/tryit.php?filename=try_ng_html_hide)

# AngularJS 事件

AngularJS 有自己的 HTML 事件指令。

## ng-click 指令

**ng-click** 指令定义了 AngularJS 点击事件。

## AngularJS 实例

<div ng-app="" ng-controller="myCtrl">  
  
<button ng-click="count = count + 1">点我！</button>  
  
<p>{{ count }}</p>  
  
</div>

[**尝试一下 »**](http://www.runoob.com/try/tryit.php?filename=try_ng_events_click)

## 隐藏 HTML 元素

**ng-hide** 指令用于设置应用部分是否可见。

**ng-hide="true"** 设置 HTML 元素不可见。

**ng-hide="false"** 设置 HTML 元素可见。

## AngularJS 实例

<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>  
姓名: <input type="text" ng-model="lastName"><br>  
<br>  
Full Name: {{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>

[**尝试一下 »**](http://www.runoob.com/try/tryit.php?filename=try_ng_events_hide)

应用解析:

第一部分 **personController**与控制器章节类似。

应用有一个默认属性: **$scope.myVar = false;**

**ng-hide** 指令设置 <p>元素及两个输入域是否可见， 根据 **myVar** 的值 (true 或 false) 来设置是否可见。

**toggle()** 函数用于切换 **myVar** 变量的值（true 和 false）。

**ng-hide="true"** 让元素 **不可见**。

## 显示 HTML 元素

**ng-show** 指令可用于设置应用中的一部分是否**可见** 。

**ng-show="false"** 可以设置 HTML 元素 **不可见**。

**ng-show="true"** 可以以设置 HTML 元素可见。

以下实例使用了 ng-show 指令:

## AngularJS 实例

<div ng-app="myApp" ng-controller="personCtrl">  
  
<button ng-click="toggle()">隐藏/显示</button>  
  
<p ng-show="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 = true;  
    $scope.toggle = function() {  
        $scope.myVar = !$scope.myVar;  
    }  
});  
</script>

[**尝试一下 »**](http://www.runoob.com/try/tryit.php?filename=try_ng_events)

# AngularJS 模块

模块定义了一个应用程序。——module

模块是应用程序中不同部分的容器。

模块是应用控制器的容器。

控制器通常属于一个模块。

## 创建模块

你可以通过 AngularJS 的 **angular.module** 函数来创建模块：

<div ng-app="myApp">...</div>  
  
<script>

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

</script>

"myApp" 参数对应执行应用的 HTML 元素。

现在你可以在 AngularJS 应用中添加控制器，指令，过滤器等。

## 添加控制器

你可以使用 **ng-controller** 指令来添加应用的控制器:

## AngularJS 实例

<div ng-app="**myApp**" ng-controller=**"myCtrl"**>  
{{ firstName + " " + lastName }}  
</div>  
  
<script>

var app = angular.module(**"myApp"**, []);  
  
app.controller(**"myCtrl"**, function($scope) {  
    $scope.firstName = "John";  
    $scope.lastName = "Doe";  
});

</script>

[**尝试一下 »**](http://www.runoob.com/try/try.php?filename=try_ng_module)

你可以在 [AngularJS 控制器](http://www.runoob.com/angularjs/angularjs-controllers.html)章节学到更多关于控制器的知识。

## 添加指令

AngularJS 提供了很多内置的指令，你可以使用它们来为你的应用添加功能。

完整的指令内容可以参阅 [AngularJS 参考手册](http://www.runoob.com/angularjs/angularjs-reference.html)。

此外，你可以使用模块来为你应用添加自己的指令：

## AngularJS 实例

<div ng-app="myApp" runoob-directive></div>  
  
<script>

var app = angular.module("myApp", []);  
  
app.directive("runoobDirective", function() {  
    return {  
        template : "我在指令构造器中创建!"  
    };  
});

</script>

[**尝试一下 »**](http://www.runoob.com/try/try.php?filename=try_ng_directive)

你可以在 [AngularJS 指令](http://www.runoob.com/angularjs/angularjs-directives.html)章节学到更多关于指令的知识。

## 模块和控制器包含在 JS 文件中

通常 AngularJS 应用程序将模块和控制器包含在 JavaScript 文件中。

在以下实例中， "myApp.js" 包含了应用模块的定义程序， "myCtrl.js" 文件包含了控制器：

## AngularJS 实例

<!DOCTYPE html>  
<html>  
<script src="http://apps.bdimg.com/libs/angular.js/1.4.6/angular.min.js"></script>  
<body>  
  
<div ng-app="**myApp**" ng-controller="**myCtrl**">  
{{ firstName + " " + lastName }}  
</div>  
  
<script src="**myApp.js**"></script>  
<script src="**myCtrl.js**"></script>  
  
</body>  
</html>

[**尝试一下 »**](http://www.runoob.com/try/tryit.php?filename=try_ng_module_files)

## myApp.js

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

|  |  |
| --- | --- |
| **IMG_256** | 在模块定义中 [] 参数用于定义模块的依赖关系。 中括号[]表示该模块没有依赖，如果有依赖的话会在中括号写上依赖的模块名字。 |

## myCtrl.js

app.controller(**"myCtrl"**, function($scope) {  
    $scope.firstName = "John";  
    $scope.lastName= "Doe";  
});

## 函数会影响到全局命名空间

JavaScript 中应避免使用全局函数。因为他们很容易被其他脚本文件覆盖。

AngularJS 模块让所有函数的作用域在该模块下，避免了该问题。

## 什么时候载入库?

|  |  |
| --- | --- |
| **IMG_257** | 在我们的实例中，所有 AngularJS 库都在 HTML 文档的头部载入。 |

对于 HTML 应用程序，通常建议把所有的脚本都放置在 <body> 元素的最底部。

这会提高网页加载速度，因为 HTML 加载不受制于脚本加载。

在我们的多个 AngularJS 实例中，您将看到 AngularJS 库是在文档的 <head> 区域被加载。

在我们的实例中，AngularJS 在 <head> 元素中被加载，因为对 angular.module 的调用只能在库加载完成后才能进行。

另一个解决方案是在 <body> 元素中加载 AngularJS 库，但是必须放置在您的 AngularJS 脚本前面：

## AngularJS 实例

<!DOCTYPE html>  
<html>  
<head>  
<meta charset="utf-8">  
<script src="http://apps.bdimg.com/libs/angular.js/1.4.6/angular.min.js"></script>  
</head>  
<body>  
  
<div ng-app="myApp" ng-controller="myCtrl">  
{{ firstName + " " + lastName }}  
</div>  
<script>  
var app = angular.module("myApp", []);  
app.controller("myCtrl", function($scope) {  
    $scope.firstName = "John";  
    $scope.lastName = "Doe";  
});  
</script>  
  
</body>  
</html>

[**尝试一下 »**](http://www.runoob.com/try/tryit.php?filename=try_ng_module_body)

## PHP Ajax 跨域问题最佳解决方案

### 分类 [编程技术](http://www.runoob.com/w3cnote/w3cnote_genre/code)

本文通过设置**Access-Control-Allow-Origin**来实现跨域。

例如：客户端的域名是client.runoob.com，而请求的域名是server.runoob.com。

如果直接使用ajax访问，会有以下错误：

XMLHttpRequest cannot load http://server.runoob.com/server.php. No 'Access-Control-Allow-Origin' header is present on the requested resource.Origin 'http://client.runoob.com' is therefore not allowed access.

## 1、允许单个域名访问

指定某域名（http://client.runoob.com）跨域访问，则只需在http://server.runoob.com/server.php文件头部添加如下代码：

header('Access-Control-Allow-Origin:http://client.runoob.com');

## 2、允许多个域名访问

指定多个域名（http://client1.runoob.com、http://client2.runoob.com等）跨域访问，则只需在http://server.runoob.com/server.php文件头部添加如下代码：

$origin = isset($\_SERVER['HTTP\_ORIGIN'])? $\_SERVER['HTTP\_ORIGIN'] : '';

$allow\_origin = array(

'http://client1.runoob.com',

'http://client2.runoob.com' );

if(in\_array($origin, $allow\_origin)){

header('Access-Control-Allow-Origin:'.$origin); }

## 3、允许所有域名访问

允许所有域名访问则只需在http://server.runoob.com/server.php文件头部添加如下代码：

header('Access-Control-Allow-Origin:\*');

Angularjs学习随笔：

腾讯课堂-智能社angularjs教学

Angularjs学习要点：

1. angularjs和javascript不能互通--需要通过某些手段--即controller

函数不互通（不能在{{}}中使用js函数），变量不互通（angular不能使用js的变量），事件也不互通

1. Angularjs开发模式和传统的开发模式完全不同，关注和围绕的核心是数据，一切以数据为核心

angularjs接管了UI--html，即html的变化需要通过angularjs来操作

Controller:控制器--逻辑层