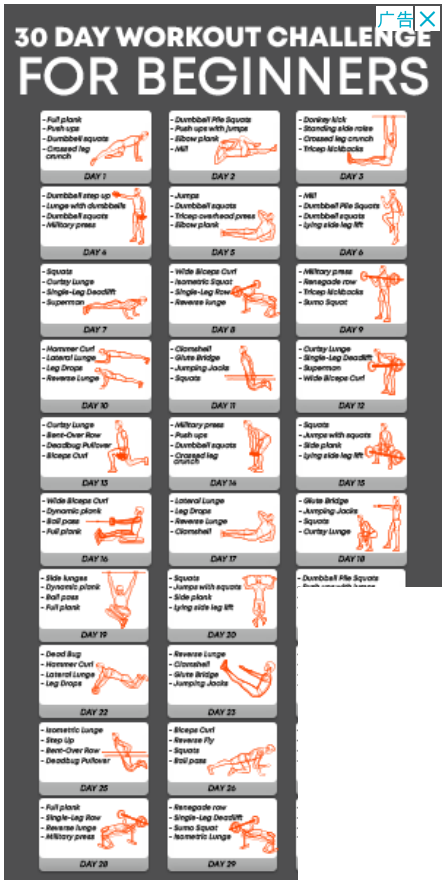

AngularJS 控制器(Controllers)

在Angular中，控制器就像 JavaScript 中的**构造函数**一般，是用来增强 Angular作用域(scope) (../tutorial/12.html) 的。

当一个控制器通过 `ng-controller` () 指令被添加到DOM中时，ng 会调用该控制器的**构造函数**来生成一个控制器对象，这样，就创建了一个新的**子级 作用域(scope)**。在这个构造函数中，作用域(scope)会作为 `$scope` 参数注入其中，并允许用户代码访问它。

一般情况下，我们使用控制器做两件事：

- 初始化 `$scope` 对象
- 为 `$scope` 对象添加行为（方法）

初始化 $scope 对象

当我们创建应用程序时，我们通常要为Angular的 `$scope` 对象设置初始状态，这是通过在 `$scope` 对象上添加属性实现的。这些属性就是供在视图中展示用的**视图模型**（**view model**），它们在与此控制器相关的模板中均可以访问到。

下面的例子中定义了一个非常简单的控制器构造函数： `GreetingCtrl` ，我们在该控制器所创建的 scope 中添加一个 `greeting` 属性：


```
1.      function GreetingCtrl($scope) {
2.          $scope.greeting = 'Hola!';
3.      }
```

如上所示，我们有了一个控制器，它初始化了一个 `$scope` 对象，并且有一个 `greeting` 属性。当我们把该控制器关联到DOM节点上，模板就可以通过数据绑定来读取它：

```
1.      <div ng-controller="GreetingCtrl">
2.
3.      </div>
```

注意：虽然Angular允许我们在全局作用域下(window)定义控制器函数，但**建议不要**用这种方式。在一个实际的应用程序中，推荐在Angular模块 (../tutorial/6.html) 下通过 `.controller` 为你的应用创建控制器，如下所示：

```
1.      var myApp = angular.module('myApp',[]);
2.
3.      myApp.controller('GreetingCtrl', ['$scope', function($scope) {
4.          $scope.greeting = 'Hola!';
5.      }]);
```

在上面例子中，我们使用**内联注入**的方式声明 `GreetingCtrl` 依赖于Angular提供的 `$scope` 服务。更多详情，参阅 依赖注入 (../tutorial/17.html) 。

为 $scope 对象添加行为

为了对事件作出响应，或是在视图中执行计算，我们需要为 scope 提供相关的行为操作的逻辑。上面一节中，我们为 scope 添加属性来让模板可以访问数据模型，现在，我们为 `$scope` 添加方法来让它提供相关的交互逻辑。添加完之后，这些方法就可以在**模板/视图**中被调用了。

下面的例子将演示为控制器的 scope 添加方法，它用来使一个数字翻倍：

```
1.      var myApp = angular.module('myApp',[]);
2.
3.      myApp.controller('DoubleCtrl', ['$scope', function($scope) {
4.          $scope.double = function(value) { return value * 2; };
5.      }]);
```

当上述控制器被添加到DOM之后， `double` 方法即可被调用，如在模板中的一个Angular表达式中：


```
1.      <div ng-controller="DoubleCtrl">
2.        <input ng-model="num"> 翻倍后等于
3.      </div>
```

如 概述 (guide/concepts) 部分所指出的一样，任何对象（或者原生类型的变量）被添加到 scope 后都将成为 scope 的属性，作为数据模型供模板/视图调用。任何方法被添加到 scope 后，也能在模板/视图中通过Angular表达式或是Angular的事件处理器（如： `ngClick` (../api/11.html)）调用。

正确使用控制器

通常情况下，控制器不应被赋予太多的责任和义务，它只需要负责一个单一视图所需的业务逻辑。

最常见的保持控制器“纯度”的方法是将那些不属于控制器的逻辑都封装到服务（services）中，然后在控制器中通过依赖注入调用相关服务。详见指南中的 依赖注入 (guide/di) 服务 (../tutorial/19.html) 这两部分。

注意，下面的场合**千万不要用控制器**：

- 任何形式的DOM操作：控制器只应该包含业务逻辑。DOM操作则属于应用程序的表现层逻辑操作，向来以测试难度之高闻名于业界。把任何表现层的逻辑放到控制器中将会大大增加业务逻辑的测试难度。ng 提供数据绑定 （数据绑定 (guide/databinding)）来实现自动化的DOM操作。如果需要手动进行DOM操作，那么最好将表现层的逻辑封装在 指令 (../tutorial/5.html) 中
- 格式化输入：使用 angular表单控件 (../tutorial/4.html) 代替
- 过滤输出：使用 angular过滤器 (../tutorial/8.html) 代替
- 在控制器间复用有状态或无状态的代码：使用angular服务 (../tutorial/19.html) 代替
- 管理其它部件的生命周期（如手动创建 service 实例）

将控制器与 scope 对象关联

通过两种方法可以实现控制器和 scope 对象的关联：

- `ngController` 指令 (../api/13.html) 这个指令就会创建一个新的 scope
- $route路由服务 (../api/81.html)

简单的控制器范例

为了更深入地阐释Angular的控制器是如何工作的，我们用以下几个部件来构建一个小型应用：

- 一个由两个按钮和一条简单反馈构成的模板 (../tutorial/13.html)
- 一个名为 `spice` 的数据模型对象，是一个字符串
- 一个拥有两个方法的控制器，可以设置 `spice` 的值

模板中的消息包含了一个到数据模型 `spice` 的绑定，默认值为 `very` 。之后，取决于哪个按钮被点击， `spice` 的值会被置为 `chili` 或是 `jalapeño` ，受益于数据绑定，模板中的这个消息会在 `spice` 变化时自动更新。

源码

index.html script.js

```
 1.  <!doctype html>
 2.  <html ng-app="spicyApp1">
 3.    <head>
 4.      <script src="http://code.angularjs.org/1.2.25/angular.min.js"></script>
 5.      <script src="script.js"></script>
 6.    </head>
 7.    <body>
 8.      <div ng-app="spicyApp1" ng-controller="SpicyCtrl">
 9.       <button ng-click="chiliSpicy()">Chili</button>
10.       <button ng-click="jalapenoSpicy()">Jalapeño</button>
11.       <p>The food is  spicy!</p>
12.      </div>
13.    </body>
14.  </html>
```

效果

Chili Jalapeño

The food is very spicy!

上面的例子中有几个值得注意的地方：

- `ng-controller` 指令用来为我们的模板创建一个 scope ，而且它受到 `SpicyCtrl` 控制器的管理
- `SpicyCtrl` 就是一个普通的 JavaScript 函数，只是命名上以首字母大写，以 "Ctrl" 或 "Controller" 结尾
- 把一个属性指定给 `$scope` 这样会创建或更新一个数据模型
- 控制器的方法可以通过在 scope 中添加函数来创建，如 `chiliSpicy` 方法
- 控制器的方法和属性在模板/视图中都是可以获得的，在上例中的 `<div>` 元素及其子节点

控制器范例扩展--带参数

控制器方法可以带参数，我们看一下如下范例（是上面例子的变种）：

源码

index.html script.js

```
<!doctype html>
<html ng-app="spicyApp2">
  <head>
    <script src="http://code.angularjs.org/1.2.25/angular.min.js"></script>
    <script src="script.js"></script>
  </head>
  <body>
    <div ng-app="spicyApp2" ng-controller="SpicyCtrl">
     <input ng-model="customSpice">
     <button ng-click="spicy('chili')">Chili</button>
     <button ng-click="spicy(customSpice)">Custom spice</button>
     <p>The food is  spicy!</p>
    </div>
  </body>
</html>
```

效果

wasabi Chili Custom spice

The food is very spicy!

注意上面的 `SpicyCtrl` 控制器现在只定义了一个 `spicy` 方法，带一个 `spice` 参数。然后在模板中，第一个按钮调用 `spicy` 方法的时候传进一个字符串常量 `'chili'` ；第二个按钮则传进一个与 `<input>` 进行了双向绑定的数据模型 `customSpice` （初始值在 scope 中设置为了 `'wasabi'` ）。（译者注：这样在 `<input>` 输入框输入什么，点击第二个按钮时， `<p>` 标签就会显示 `<input>` 中的当前值。）

Scope 继承范例

我们常常会在不同层级的DOM结构中添加控制器。由于 `ng-controller` (api/ng.directive:ngController) 指令会创建新的子级 scope ，这样我们就会获得一个与DOM层级结构相对应的的基于继承关系的 scope 层级结构。（译者注：由于 Js 是基于原型的继承，所以）底层（内层）控制器的 `$scope` 能够访问在高层控制器的 scope 中定义的属性和方法。详情参见 理解“作用域” (https://github.com/angular/angular.js/wiki/Understanding-Scopes) 。

译者注：下面是一个拥有三层div结构，也就对应有三层 scope 继承关系的层级结构（不包括 rootScope 的话），demo中的蓝色边框很清晰的展现了 scope 的层级和DOM层级的对应关系。它还展示了“scope 是由 `ng-controller` 指令创建并由其对应的控制器所管理”这个概念。

源码

index.html style.css script.js

```
<!doctype html>
<html ng-app="scopeInheritance">
  <head>
    <script src="http://code.angularjs.org/1.2.25/angular.min.js"></script>
    <script src="script.js"></script>
  </head>
  <body>
    <div ng-app="scopeInheritance" class="spicy">
      <div ng-controller="MainCtrl">
        <p>Good {{timeOfDay}}, {{name}}!</p>

        <div ng-controller="ChildCtrl">
          <p>Good {{timeOfDay}}, {{name}} !</p>

          <div ng-controller="GrandChildCtrl">
            <p>Good {{timeOfDay}}, {{name}}!</p>
          </div>
        </div>
      </div>
    </div>
  </body>
</html>
```

效果

Good morning, Nikki!

Good morning, Mattie !

Good evening, Gingerbreak Baby!

注意，上面例子中我们在HTML模板中嵌套了三个 `ng-controller` 指令，这导致我们的视图中有4个 scope：

- root scope，所有作用域的“根”
- `MainCtrl` 控制器管理的 scope （简称 `MainCtrl` scope），拥有 `timeOfDay` 和 `name` 两个属性
- `ChildCtrl` 控制器管理的 scope （简称 `ChildCtrl` scope），继承了 `MainCtrl` scope 中的 `timeOfDay` 属性，但重写了它的 `name` 属性
- `GrandChildCtrl` 控制器管理的 scope （简称 `GrandChildCtrl` scope），重写了 `MainCtrl` scope 中的 `timeOfDay` 属性和 `ChildCtrl` scope 中的 `name` 属性

控制器中，方法继承和属性继承的工作方式是一样的，所以，上面例子中的所有属性，我们也可以改写成能够返回字符串值的方法，同样有效。

控制器的单元测试

虽然我们有很多方法可以对控制器进行测试，但在这里，我们仅展示最常见的一种，包括注入 `$rootScope` (api/ng.$rootScope) 以及 `$controller` (../api/101.html)：

控制器定义：

```
1.       var myApp = angular.module('myApp',[]);
2.
3.       myApp.controller('MyController', function($scope) {
4.         $scope.spices = [{"name":"pasilla", "spiciness":"mild"},
5.                          {"name":"jalapeno", "spiceiness":"hot hot hot!"},
6.                          {"name":"habanero", "spiceness":"LAVA HOT!!"}];
7.         $scope.spice = "habanero";
8.       });
```

控制器测试：

```
1.   describe('myController function', function() {
2.
3.     describe('myController', function() {
4.       var $scope;
5.
6.       beforeEach(module('myApp'));
7.
8.       beforeEach(inject(function($rootScope, $controller) {
9.         $scope = $rootScope.$new();
10.        $controller('MyController', {$scope: $scope});
11.      }));
12.
13.      it('should create "spices" model with 3 spices', function() {
14.        expect($scope.spices.length).toBe(3);
15.      });
16.
17.      it('should set the default value of spice', function() {
18.        expect($scope.spice).toBe('habanero');
19.      });
20.    });
21.  });
```

如果有需要测试嵌套关系的控制器，那么在你的测试代码中，你也得创建对应于 scope 层级结构的测试代码：

```
describe('state', function() {
    var mainScope, childScope, grandChildScope;

    beforeEach(module('myApp'));

    beforeEach(inject(function($rootScope, $controller) {
        mainScope = $rootScope.$new();
        $controller('MainCtrl', {$scope: mainScope});
        childScope = mainScope.$new();
        $controller('ChildCtrl', {$scope: childScope});
        grandChildScope = childScope.$new();
        $controller('GrandChildCtrl', {$scope: grandChildScope});
    }));

    it('should have over and selected', function() {
        expect(mainScope.timeOfDay).toBe('morning');
        expect(mainScope.name).toBe('Nikki');
        expect(childScope.timeOfDay).toBe('morning');
        expect(childScope.name).toBe('Mattie');
        expect(grandChildScope.timeOfDay).toBe('evening');
        expect(grandChildScope.name).toBe('Gingerbreak Baby');
    });
});
```