master

/ Developer Guide (guide) / Controllers (guide/controller)

Show / Hide Table of Contents

://github.com/angular/angular.js/edit/master/docs/content/guide/controller.ngdoc?message=docs(guide%2FControllers)%3A%20describe%20your%20change...)

Understanding Controllers

In AngularJS, a Controller is defined by a JavaScript **constructor function** that is used to augment the AngularJS Scope (guide/scope).

When a Controller is attached to the DOM via the ng-controller (api/ng/directive/ngController) directive, AngularJS will instantiate a new Controller object, using the specified Controller's **constructor function**. A new **child scope** will be created and made available as an injectable parameter to the Controller's constructor function as `$scope`.

If the controller has been attached using the `controller as` syntax then the controller instance will be assigned to a property on the new scope.

Use controllers to:

- Set up the initial state of the `$scope` object.
- Add behavior to the `$scope` object.

Do not use controllers to:

- Manipulate DOM — Controllers should contain only business logic. Putting any presentation logic into Controllers significantly affects its testability. AngularJS has databinding (guide/databinding) for most cases and directives (guide/directive) to encapsulate manual DOM manipulation.
- Format input — Use AngularJS form controls (guide/forms) instead.
- Filter output — Use AngularJS filters (guide/filter) instead.
- Share code or state across controllers — Use AngularJS services (guide/services) instead.
- Manage the life-cycle of other components (for example, to create service instances).

Setting up the initial state of a `$scope` object

Typically, when you create an application you need to set up the initial state for the AngularJS `$scope`. You set up the initial state of a scope by attaching properties to the `$scope` object. The properties contain the **view model** (the model that will be presented by the view). All the `$scope` properties will be available to the template (guide/templates) at the point in the DOM where the Controller is registered.

The following example demonstrates creating a **`GreetingController`**, which attaches a `greeting` property containing the string `'Hola!'` to the `$scope`:

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

myApp.controller('GreetingController', ['$scope', function($scope) {
  $scope.greeting = 'Hola!';
}]);
```

We create an AngularJS Module (guide/module), `myApp`, for our application. Then we add the controller's constructor function to the module using the `.controller()` method. This keeps the controller's constructor function out of the global scope.

We have used an **inline injection annotation** to explicitly specify the dependency of the Controller on the `$scope` service provided by AngularJS. See the guide on Dependency Injection (guide/di) for more information.

We attach our controller to the DOM using the `ng-controller` directive. The `greeting` property can now be data-bound to the template:

```
<div ng-controller="GreetingController">
  {{ greeting }}
</div>
```

Adding Behavior to a Scope Object

In order to react to events or execute computation in the view we must provide behavior to the scope. We add behavior to the scope by attaching methods to the `$scope` object. These methods are then available to be called from the template/view.

master

Show / Hide Table of Contents

✖ Close ()

The following example uses a Controller to add a method, which doubles a number, to the scope:

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

myApp.controller('DoubleController', ['$scope', function($scope) {
  $scope.double = function(value) { return value * 2; };
}]);
```

Once the Controller has been attached to the DOM, the `double` method can be invoked in an AngularJS expression in the template:

```
<div ng-controller="DoubleController">
  Two times <input ng-model="num"> equals {{ double(num) }}
</div>
```

As discussed in the Concepts (guide/concepts) section of this guide, any objects (or primitives) assigned to the scope become model properties. Any methods assigned to the scope are available in the template/view, and can be invoked via AngularJS expressions and `ng` event handler directives (e.g. ngClick (api/ng/directive/ngClick)).

Using Controllers Correctly

In general, a Controller shouldn't try to do too much. It should contain only the business logic needed for a single view.

The most common way to keep Controllers slim is by encapsulating work that doesn't belong to controllers into services and then using these services in Controllers via dependency injection. This is discussed in the Dependency Injection (guide/di) and Services (guide/services) sections of this guide.

Associating Controllers with AngularJS Scope Objects

You can associate Controllers with scope objects implicitly via the ngController directive (api/ng/directive/ngController) or $route service (api/ngRoute/service/$route).

Simple Spicy Controller Example

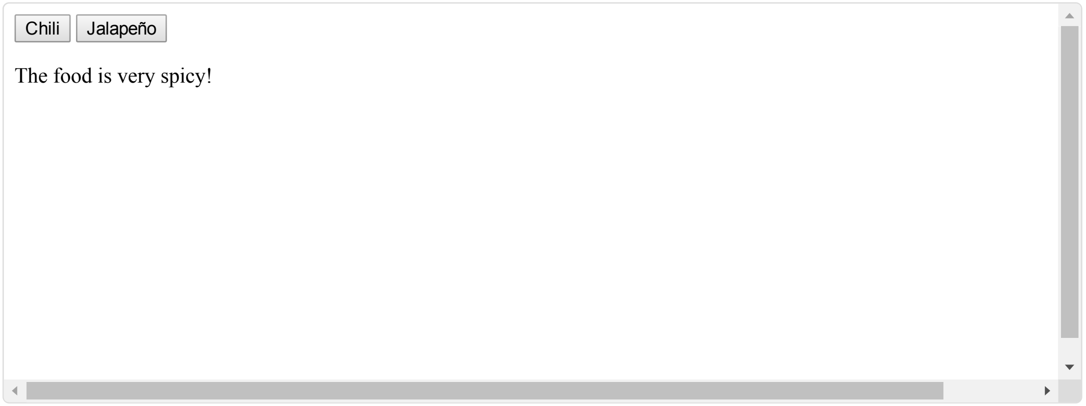
To illustrate further how Controller components work in AngularJS, let's create a little app with the following components:

- A template (guide/templates) with two buttons and a simple message
- A model consisting of a string named `spice`
- A Controller with two functions that set the value of `spice`

The message in our template contains a binding to the `spice` model which, by default, is set to the string "very". Depending on which button is clicked, the `spice` model is set to `chili` or `jalapeño` , and the message is automatically updated by data-binding.

index.html () app.js () Edit in Plunker

```
<div ng-controller="SpicyController">
 <button ng-click="chiliSpicy()">Chili</button>
 <button ng-click="jalapenoSpicy()">Jalapeño</button>
 <p>The food is {{spice}} spicy!</p>
</div>
```



The food is very spicy!

master

Show / Hide Table of Contents

✖ Close ()

Things to notice in the example above:

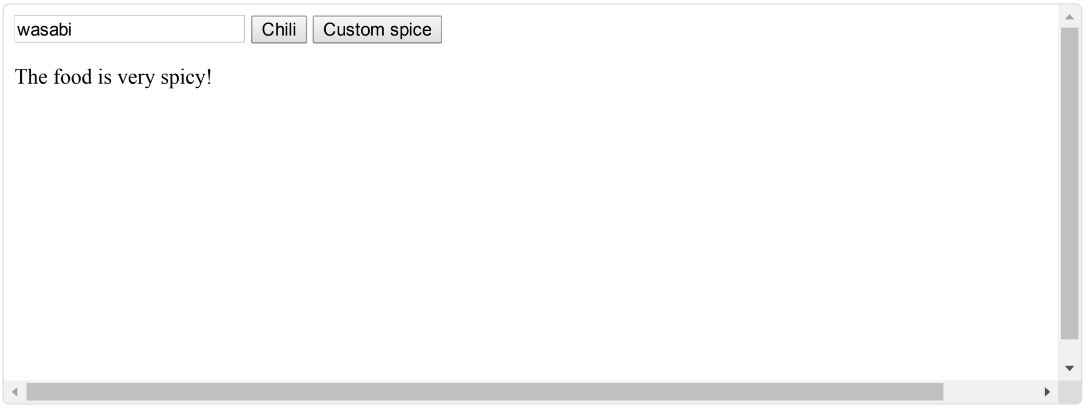
- The `ng-controller` directive is used to (implicitly) create a scope for our template, and the scope is augmented (managed) by the **`SpicyController`** Controller.
- **`SpicyController`** is just a plain JavaScript function. As an (optional) naming convention the name starts with capital letter and ends with "Controller".
- Assigning a property to `$scope` creates or updates the model.
- Controller methods can be created through direct assignment to scope (see the `chiliSpicy` method)
- The Controller methods and properties are available in the template (for both the **`<div>`** element and its children).

Spicy Arguments Example

Controller methods can also take arguments, as demonstrated in the following variation of the previous example.

index.html () app.js () Edit in Plunker

```
<div ng-controller="SpicyController">
 <input ng-model="customSpice">
 <button ng-click="spicy('chili')">Chili</button>
 <button ng-click="spicy(customSpice)">Custom spice</button>
 <p>The food is {{spice}} spicy!</p>
</div>
```



The food is very spicy!

Notice that the **`SpicyController`** Controller now defines just one method called `spicy`, which takes one argument called `spice`. The template then refers to this Controller method and passes in a string constant `'chili'` in the binding for the first button and a model property `customSpice` (bound to an input box) in the second button.

Scope Inheritance Example

It is common to attach Controllers at different levels of the DOM hierarchy. Since the ng-controller (api/ng/directive/ngController) directive creates a new child scope, we get a hierarchy of scopes that inherit from each other. The `$scope` that each Controller receives will have access to properties and methods defined by Controllers higher up the hierarchy. See Understanding Scopes (https://github.com/angular/angular.js/wiki/Understanding-Scopes) for more information about scope inheritance.

index.html () app.css () app.js () Edit in Plunker

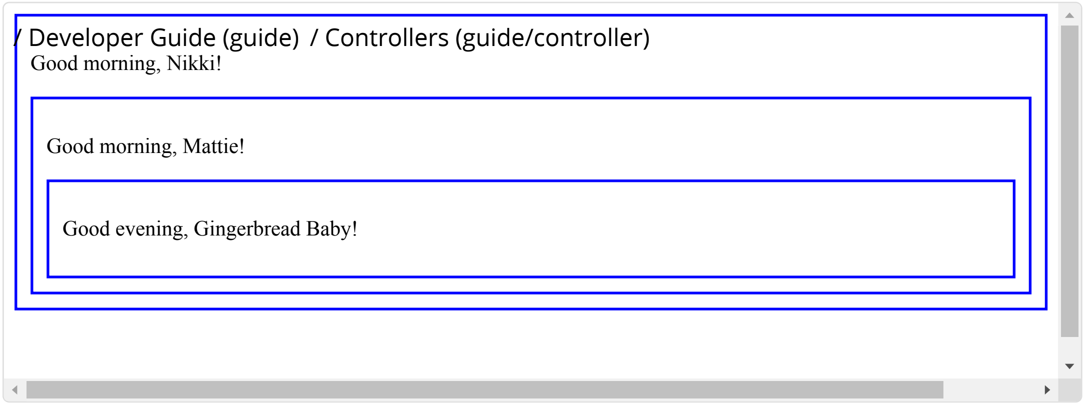
```
<div class="spicy">
  <div ng-controller="MainController">
    <p>Good {{timeOfDay}}, {{name}}!</p>

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

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

master

Show / Hide Table of Contents

✖ Close ()

/ Developer Guide (guide) / Controllers (guide/controller)


Good morning, Nikki!

Good morning, Mattie!

Good evening, Gingerbread Baby!

Notice how we nested three `ng-controller` directives in our template. This will result in four scopes being created for our view:

- The root scope
- The **`MainController`** scope, which contains `timeOfDay` and `name` properties
- The **`ChildController`** scope, which inherits the `timeOfDay` property but overrides (shadows) the `name` property from the previous scope
- The **`GrandChildController`** scope, which overrides (shadows) both the `timeOfDay` property defined in **`MainController`** and the `name` property defined in **`ChildController`**

Inheritance works with methods in the same way as it does with properties. So in our previous examples, all of the properties could be replaced with methods that return string values.

Testing Controllers

Although there are many ways to test a Controller, one of the best conventions, shown below, involves injecting the $rootScope (api/ng/service/$rootScope) and $controller (api/ng/service/$controller):

Controller Definition:

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

myApp.controller('MyController', function($scope) {
  $scope.spices = [{"name":"pasilla", "spiciness":"mild"},
                   {"name":"jalapeno", "spiciness":"hot hot hot!"},
                   {"name":"habanero", "spiciness":"LAVA HOT!!"}];
  $scope.spice = "habanero";
});
```

Controller Test:

```
describe('myController function', function() {

  describe('myController', function() {
    var $scope;

    beforeEach(module('myApp'));

    beforeEach(inject(function($rootScope, $controller) {
      $scope = $rootScope.$new();
      $controller('MyController', {$scope: $scope});
    }));

    it('should create "spices" model with 3 spices', function() {
      expect($scope.spices.length).toBe(3);
    });

    it('should set the default value of spice', function() {
      expect($scope.spice).toBe('habanero');
    });
  });
});
```

If you need to test a nested Controller you must create the same scope hierarchy in your test that exists in the DOM:

master

/ Developer Guide (guide) / Controllers (guide/controller)

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

    beforeEach(module('myApp'));

    beforeEach(inject(function($rootScope, $controller) {
        mainScope = $rootScope.$new();
        $controller('MainController', {$scope: mainScope});
        childScope = mainScope.$new();
        $controller('ChildController', {$scope: childScope});
        grandChildScope = childScope.$new();
        $controller('GrandChildController', {$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('Gingerbread Baby');
    });
```

Show / Hide Table of Contents

✖ Close ()