## AngularJs official document

<https://docs.angularjs.org/api/ng/directive/ngController>

## Quick start

. Build html template

. Include AngularJs script

. Add ng bind including ng-module, ng-controller, etc.

. Inject data into html by directives {{}} or ng-bind

. Define event handler in $scope like ng-click, ng-doubleClick etc.

## Basic Concepts

. View and directive

The prefix of directives are ng-data-\*, ng\_data\_\*, ng\_\*, ng-\*, {{}}

. $scope and hierarchy

. $watch, $digest, $apply

. Ajax

3 ways to send ajax requests. $http, jsonp, REST

* $http.get(url, config)
* $http.post(url, data, config)
* $http.put(url, data, config)
* $http.delete(url, config)
* $http.head(url, config)

Notice that the $http.post() and $http.put() functions take a data parameter which contains data to be sent to the server. The rest of the $http functions cannot take a data parameter.

The data parameter will be converted to a JSON string. This string will be be included in the request body when the HTTP request is sent to the server. AngularJS will consider all properties starting with a $ as private, and thus exclude these from the string. If you need to include properties starting with a$ in the data string, convert the data object to a string yourself using JSON.stringify(data).

To understand angularjs basic concepts is most important before we dig into the deep things.

. Injection

In angularjs framework, if we like to make something injectable (or like to share some common functions cross multiple controllers), we have to declare them as service by $provider service. (service, factory are special providers).

The injection function is implemented by angularjs $injector. When we inject a service, in fact the injection will automatically call $injector.get(serviceName). When we call the function of the service, the $injector.invoke(functionName) will be triggered.

.Two phases in AngularJs

The reason of having $provider is to initialize the configuration before angularJs starts running. Angular runs your application in two phases--the config and run phases. The configphase, as we've seen, is where you can set up any providers as necessary. This is also where directives, controllers, filters, and the like get set up. The run phase, as you might guess, is where Angular actually compiles your DOM and starts up your app.

As we saw in the first section, these functions are injectable--we injected the built-in $provide service in our very first code sample. However, what's worth noting is that **during the config phase, only providers can be injected** (with the exception of the services in the AUTO module--$provide and $injector).

For example, the following is **not allowed**:

myMod.config(function(greeting) {

// WON'T WORK -- greeting is an \*instance\* of a service.

// Only providers for services can be injected in config blocks.

});

What you *do* have access to any *providers* for services you've made:

myMod.config(function(greetingProvider) {

// a-ok!

});

There is one important exception: constants, since they cannot be changed, are allowed to be injected inside config blocks (this is how they differ from values). They are accessed by their name alone (no Provider suffix necessary).

Be aware of the Angularjs naming convention, whenever you defined a provider for a service, that provider gets named servicenameProvider(serviceName+suffix of Provider when we use it in config phase), where service is the name of the service. Now we can use the power of providers to do some more complicated stuff!

myMod.provider('greeting', function() {

var text = 'Hello, ';

this.setText = function(value) {

text = value;

};

this.$get = function() {

return function(name) {

alert(text + name);

};

};

});

myMod.config(function(greetingProvider) {

greetingProvider.setText("Howdy there, ");

});

myMod.run(function(greeting) {

greeting('Ford Prefect');

});

Now we have a function on our provider called setText that we can use to customize our alert; we can get access to this provider in a config block to call this method and customize the service. When we finally run our app, we can grab the greeting service, and try it out to see that our customization took effect.

. understand $scope and controller

The ngController directive attaches a controller class to the view. This is the key aspect of how angular supports the principles behind the MVC design pattern.

MVC components in angular:

* Model — Models are the properties of a scope; scopes are attached to the DOM where scope properties are accessed through bindings.
* View — The template (HTML with data bindings) that is rendered into the View.
* Controller — The ngController directive specifies a Controller class; the class contains business logic behind the application to decorate the scope with functions and values

Note that you can also attach controllers to the DOM by declaring it in a route definition via the [$route](https://docs.angularjs.org/api/ngRoute/service/$route) service. A common mistake is to declare the controller again using ng-controller in the template itself. This will cause the controller to be attached and executed twice.

Two ways to build the controller with scope (details: <https://docs.angularjs.org/api/ng/directive/ngController> )

1. The controller instance is pushed into a scope property by specifying ng-controller="as propertyName"
2. <div id="ctrl-as-exmpl" ng-controller="SettingsController1 as settings">
3. <label>Name: <input type="text" ng-model="settings.name"/></label>
4. <button ng-click="settings.greet()">greet</button><br/>
5. Contact:
6. <ul>
7. <li ng-repeat="contact in settings.contacts">
8. <select ng-model="contact.type" aria-label="Contact method" id="select\_{{$index}}">
9. <option>phone</option>
10. <option>email</option>
11. </select>
12. <input type="text" ng-model="contact.value" aria-labelledby="select\_{{$index}}" />
13. <button ng-click="settings.clearContact(contact)">clear</button>
14. <button ng-click="settings.removeContact(contact)" aria-label="Remove">X</button>
15. </li>
16. <li><button ng-click="settings.addContact()">add</button></li>
17. </ul>
18. </div>
19. angular.module('controllerAsExample', [])
20. .controller('SettingsController1', SettingsController1);
21. function SettingsController1() {
22. this.name = "John Smith";
23. this.contacts = [
24. {type: 'phone', value: '408 555 1212'},
25. {type: 'email', value: 'john.smith@example.org'} ];
26. }
27. SettingsController1.prototype.greet = function() {
28. alert(this.name);
29. };
30. SettingsController1.prototype.addContact = function() {
31. this.contacts.push({type: 'email', value: 'yourname@example.org'});
32. };
33. SettingsController1.prototype.removeContact = function(contactToRemove) {
34. var index = this.contacts.indexOf(contactToRemove);
35. this.contacts.splice(index, 1);
36. };
37. SettingsController1.prototype.clearContact = function(contact) {
38. contact.type = 'phone';
39. contact.value = '';
40. };
41. one injects $scope into the controller: ng-controller="SettingsController2"

. understand $q, defer, promise

. boostrap