***ANGULARJS-II***

1) Directives have access to the parent scope by default in AngularJS applications. For example, the following directive relies on the parent scope to write out a customer object’s name and street properties:

angular.module('directivesModule').directive('mySharedScope', function () {

return {

template: 'Name: {{customer.name}} Street: {{customer.street}}'

};

});

Although this code gets the job done, you have to know a lot about the parent scope in order to use the directive and could just as easily use **ng-include** and an HTML template to accomplish the same thing (this was discussed in the [first post](http://weblogs.asp.net/dwahlin/archive/2014/04/29/creating-custom-angularjs-directives-part-i-the-fundamentals.aspx)). If the parent scope changes at all the directive is no longer useful.

If you want to make a reuseable directive you can’t rely on the parent scope and must use something called**Isolate Scope**instead.

Creating Isolate Scope in a Directive

Isolating the scope in a directive is a is a simple process. Start by adding a **scope** property into the directive as shown next. This automatically isolates the directive’s scope from any parent scope(s).

angular.module('directivesModule').directive('myIsolatedScope', function () {

return {

scope: {},

template: 'Name: {{customer.name}} Street: {{customer.street}}'

};

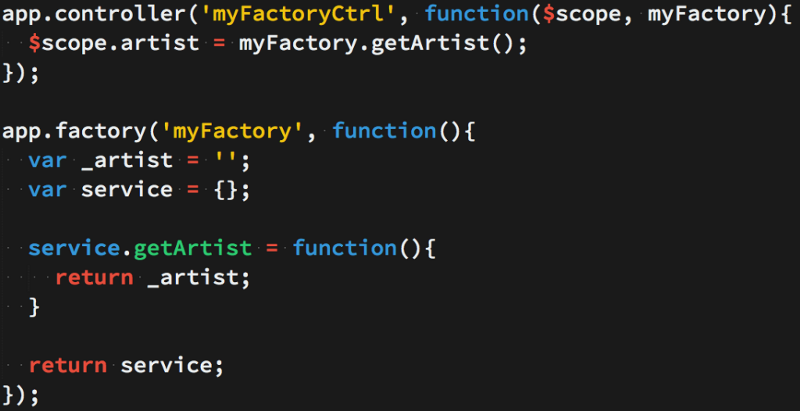
});

Now that the scope is isolated, the customer object from the parent scope will no longer be accessible. When the directive is used in a view it’ll result in the following output (notice that the customer name and street values aren’t rendered)

2). Angular provides us with three ways to create and register our own service.

1) Factory 2) Service 3) Provider

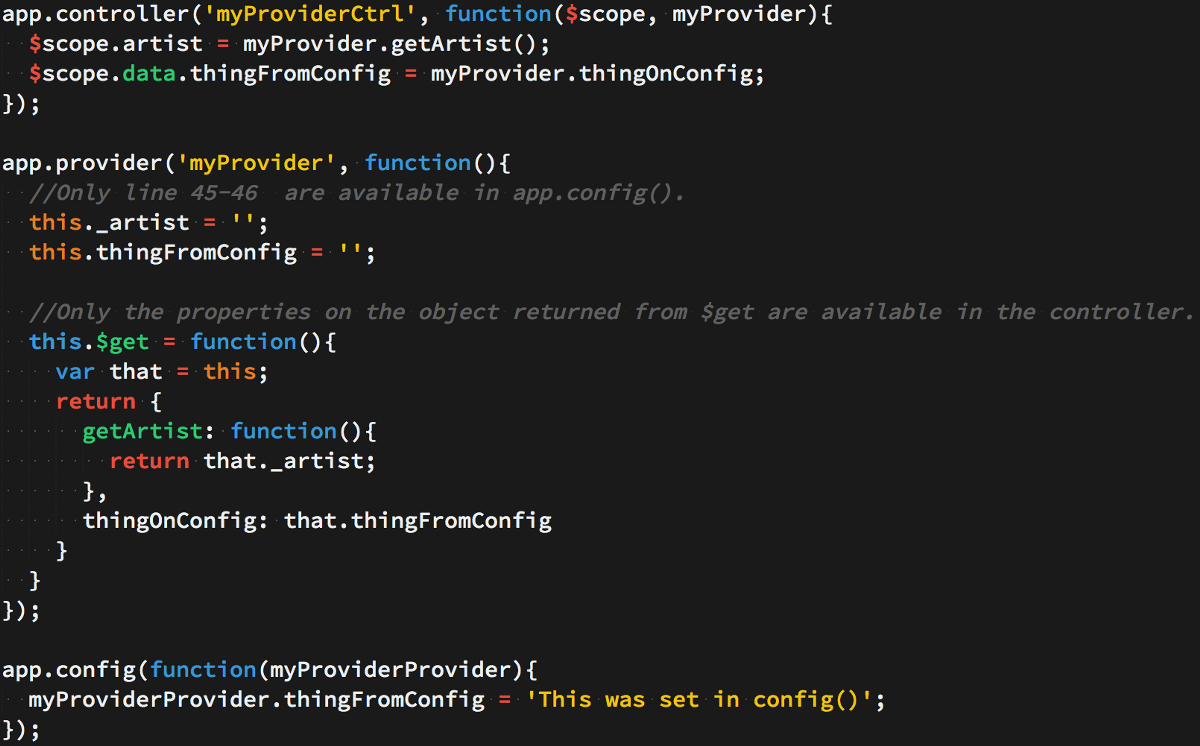
When you’re using a Factory you create an object, add properties to it, then return that same object. When you pass this service into your controller, those properties on the object will now be available in that controller through your factory.



2) When you’re using Service, it’s instantiated with the ‘new’ keyword. Because of that, you’ll add properties to ‘this’ and the service will return ‘this’. When you pass the service into your controller, those properties on ‘this’ will now be available on that controller through your service.



3) Providers are the only service you can pass into your .config() function. Use a provider when you want to provide module-wide configuration for your service object before making it available.



3)Constants and values are pretty much the same with a couple of differences. constants can be injected anywhere including configuration calls. So when you call module.config and set up states or other runtime settings you can inject constant values, but not values.

Also, as mentioned earlier, there is one caveat with constants that can cause them to be editable and that is using anything other than a primitive. Values other than primitives are treated as objects. When using objects with a constant they can be modified, which for a constant makes no semantic sense whatsoever.

This is not to say you can NOT use an object inside of a constant, as long as you or other developers do not modify any of the values contained therein. It is best to stick with values below if you want to modify values.

// Storing a single constant value

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

app.constant(‘appName’, ‘My App’);

// Now we inject our constant value into a test controller

app.controller(‘TestCtrl’, [‘appName’, function TestCtrl(appName) {

console.log(appName);

}]);

// Storing multiple constant values inside of an object

// Keep in mind the values in the object mean they can be modified

// Which makes no sense for a constant, use wisely if you do this

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

app.constant(‘config’, {

appName: ‘My App’,

appVersion: 2.0,

apiUrl: ‘http://www.google.com?api’

});

// Now we inject our constant value into a test controller

app.controller(‘TestCtrl’, [‘config’, function TestCtrl(config) {

console.log(config);

console.log(‘App Name’, config.appName);

console.log(‘App Name’, config.appVersion);

}]);

**Values**

Constants and values are pretty much the same, but remember they are for storing pieces of data that are temporary, they can change anywhere, any time. Even though constants can work the same as values, only values should be modifiable. You should not use an object inside of a constant in the place of what should be a value instead.

You will notice the examples are basically the same as constants, the syntax is the same with exception of the fact that we are setting values as well as reading them.

// Storing a single value

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

app.value(‘usersOnline’, 0);

// Now we inject our constant value into a test controller

app.controller(‘TestCtrl’, [‘usersOnline’, function TestCtrl(usersOnline) {

console.log(usersOnline);

usersOnline = 15;

console.log(usersOnline);

}]);

// Storing multiple values inside of an object

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

app.value(‘user’, {

firstName: ’‘,

lastName: ’‘,

email: ’’

});

// Now we inject our constant value into a test controller

// Values will be empty

app.controller(‘TestCtrl’, [‘user’, function TestCtrl(user) {

console.log(user);

console.log(‘First name: ’, user.firstName);

console.log(‘Last name: ’, user.lastName);

console.log(‘Email: ’, user.email);

}]);

// Storing multiple values inside of an object

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

app.value(‘user’, {

firstName: ’‘,

lastName: ’‘,

email: ’’

});

// Now we inject our constant value into a test controller

// Values will be populated inside of controller

app.controller(‘TestCtrl’, [‘$scope’, ‘user’, function TestCtrl($scope, user) {

user.firstName = ‘Dwayne’;

user.lastName = ‘Charrington’;

user.email = ‘dwayne@ilikekillnerds.com’;

console.log(user);

console.log('First name: ', user.firstName);

console.log('Last name: ', user.lastName);

console.log('Email: ', user.email);

// Pass the user values through to the view

$scope.user = user;

}]);

# 4)$routeProvider

1. [- $route](https://docs.angularjs.org/api/ngRoute/service/$route)
2. - provider in module [ngRoute](https://docs.angularjs.org/api/ngRoute)

Used for configuring routes.

# If you want to navigate to different pages in your application, but you also want the application to be a SPA (Single Page Application), with no page reloading, you can use the ngRoute module.

# The ngRoute module routes your application to different pages without reloading the entire application.

# With the $routeProvider you can define what page to display when a user clicks a link.

# Define the $routeProvider using the config method of your application. Work registered in the config method will be performed when the application is loading. With the $routeProvider you can also define a controller for each "view".

# 5) ngTransclude

# - directive in module [ng](https://docs.angularjs.org/api/ng)

# Directive that marks the insertion point for the transcluded DOM of the nearest parent directive that uses transclusion.

# You can specify that you want to insert a named transclusion slot, instead of the default slot, by providing the slot name as the value of the ng-transclude or ng-transclude-slot attribute.

# If the transcluded content is not empty (i.e. contains one or more DOM nodes, including whitespace text nodes), any existing content of this element will be removed before the transcluded content is inserted. If the transcluded content is empty (or only whitespace), the existing content is left intact. This lets you provide fallback content in the case that no transcluded content is provided.

# Usage:

* ng-transclude
* ng-transclude-slot="string">
* ...
* </ng-transclude>
* as attribute:
* <ANY
* ng-transclude="string">
* ...

</ANY>

* as CSS class:

<ANY class="ng-transclude: string;"> ... </ANY>

# 6) $q

1. [- $qProvider](https://docs.angularjs.org/api/ng/provider/$qProvider)
2. - service in module [ng](https://docs.angularjs.org/api/ng)

A service that helps you run functions asynchronously, and use their return values (or exceptions) when they are done processing.

This is a [Promises/A+](https://promisesaplus.com/)-compliant implementation of promises/deferred objects inspired by [Kris Kowal's Q](https://github.com/kriskowal/q).

$q can be used in two fashions --- one which is more similar to Kris Kowal's Q or jQuery's Deferred implementations, and the other which resembles ES6 (ES2015) promises to some degree.

There are two main differences:

* $q is integrated with the [$rootScope.Scope](https://docs.angularjs.org/api/ng/type/$rootScope.Scope) Scope model observation mechanism in angular, which means faster propagation of resolution or rejection into your models and avoiding unnecessary browser repaints, which would result in flickering UI.
* Q has many more features than $q, but that comes at a cost of bytes. $q is tiny, but contains all the important functionality needed for common async tasks.

$http

* [- $httpProvider](https://docs.angularjs.org/api/ng/provider/$httpProvider)
* - service in module [ng](https://docs.angularjs.org/api/ng)
* The $http service is a core Angular service that facilitates communication with the remote HTTP servers via the browser's [XMLHttpRequest](https://developer.mozilla.org/en/xmlhttprequest) object or via [JSONP](http://en.wikipedia.org/wiki/JSONP).
* For unit testing applications that use $http service, see [$httpBackend mock](https://docs.angularjs.org/api/ngMock/service/$httpBackend).
* For a higher level of abstraction, please check out the [$resource](https://docs.angularjs.org/api/ngResource/service/$resource) service.
* The $http API is based on the [deferred/promise APIs](https://docs.angularjs.org/api/ng/service/$q) exposed by the $q service. While for simple usage patterns this doesn't matter much, for advanced usage it is important to familiarize yourself with these APIs and the guarantees they provide.

The $http service is a function which takes a single argument — a [configuration object](https://docs.angularjs.org/api/ng/service/$http#usage) — that is used to generate an HTTP request and returns a [promise](https://docs.angularjs.org/api/ng/service/$q).

// Simple GET request example:

$http({

method: 'GET',

url: '/someUrl'

}).then(function successCallback(response) {

// this callback will be called asynchronously

// when the response is available

}, function errorCallback(response) {

// called asynchronously if an error occurs

// or server returns response with an error status.

});

The response object has these properties:

* **data** – {string|Object} – The response body transformed with the transform functions.
* **status** – {number} – HTTP status code of the response.
* **headers** – {function([headerName])} – Header getter function.
* **config** – {Object} – The configuration object that was used to generate the request.
* **statusText** – {string} – HTTP status text of the response.

http.get('/someUrl', config).then(successCallback, errorCallback);

$http.post('/someUrl', data, config).then(successCallback, errorCallback);

list of shortcut methods:

* [$http.get](https://docs.angularjs.org/api/ng/service/$http#get)
* [$http.head](https://docs.angularjs.org/api/ng/service/$http#head)
* [$http.post](https://docs.angularjs.org/api/ng/service/$http#post)
* [$http.put](https://docs.angularjs.org/api/ng/service/$http#put)
* [$http.delete](https://docs.angularjs.org/api/ng/service/$http#delete)
* [$http.jsonp](https://docs.angularjs.org/api/ng/service/$http#jsonp)
* [$http.patch](https://docs.angularjs.org/api/ng/service/$http#patch)

7)A promise represents the eventual result of an operation. You can use a promise to specify what to do when an operation eventually succeeds or fails.

the success and error functions actually just return the promise, so we can simplify the code:

$http.get("/api/my/name")

.success(function(name) {

console.log("Your name is: " + name);

})

.error(function(response, status) {

console.log("The request failed with response " + response + " and status code " + status);

});

In fact, success and error are special functions added to a promise by $http - normally with promises we just use then, which takes the success function as the first parameter and the error function as the second:

$http.get("/api/my/name")

.then(

/\* success \*/

function(response) {

console.log("Your name is: " + response.data);

},

/\* failure \*/

function(error) {

console.log("The request failed: " + error);

});

8)In AngularJS, a service is a function, or object, that is available for, and limited to, your AngularJS application.

AngularJS has about 30 built-in services. One of them is the $location service.

The $location service has methods which return information about the location of the current web page:

The $http service is one of the most common used services in AngularJS applications. The service makes a request to the server, and lets your application handle the response.

o create your own service, connect your service to the module:

Create a service named hexafy:

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

To use your custom made service, add it as a dependency when defining the filter:

### **Example**

Use the custom made service named hexafy to convert a number into a hexadecimal number:

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

9) AngularJS application mainly relies on controllers to control the flow of data in the application. A controller is defined using ng-controller directive. A controller is a JavaScript object containing attributes/properties and functions. Each controller accepts $scope as a parameter which refers to the application/module that controller is to control.

<div ng-app = "" ng-controller = "studentController">

...

</div>

10) AngularJS offers client-side form validation.AngularJS monitors the state of the form and input fields (input, textarea, select), and lets you notify the user about the current state.AngularJS also holds information about whether they have been touched, or modified, or not.You can use standard HTML5 attributes to validate input, or you can make your own validation functions.To create your own validation ,You have to add a new directive to your application, and deal with the validation inside a function with certain specified arguments.