Understanding Dependency Injection

Vi edited this page on Aug 27, 2015 · 20 revisions

Dependency injection in AngularJS is supremely useful, and the key to making easily testable components. This article explains how Angular's dependency injection system works.

The Provider (`$provide`)

The `$provide` service is responsible for telling Angular how to create new injectable things; these things are called **services**. Services are defined by things called **providers**, which is what you're creating when you use `$provide` . Defining a provider is done via the `provider` method on the `$provide` service, and you can get hold of the `$provide` service by asking for it to be injected into an application's `config` function. An example might be something like this:

```
myMod.config(function($provide) {
  $provide.provider('greeting', function() {
    this.$get = function() {
      return function(name) {
        alert("Hello, " + name);
      };
    };
  });
});
```

Here we've defined a new provider for a service called `greeting` ; we can inject a variable named `greeting` into any injectable function (like controllers, more on that later) and Angular will call the provider's `$get` function in order to return a new instance of the service. In this case, the thing that will be injected is a function that takes a `name` parameter and `alert` s a message based on the name. We might use it like this:

```
myMod.controller('MainController', function($scope, greeting) {
  $scope.onClick = function() {
    greeting('Ford Prefect');
  };
});
```

Now here's the trick. `factory` , `service` , and `value` are all just shortcuts to define various parts of a provider--that is, they provide a means of defining a provider without having to type all that stuff out. For example, you could write that **exact same provider** just like this:

```
myMod.config(function($provide) {
  $provide.factory('greeting', function() {
    return function(name) {
      alert("Hello, " + name);
    };
  });
});
```

It's important to understand, so I'll rephrase: under the hood, AngularJS is calling the **exact same code** that we wrote above (the `$provide.provider` version) *for* us. There is literally, 100% no difference in the two versions. `value` works just the same way--if whatever we would return from our `$get` function (aka our `factory` function) is always exactly the same, we can write even less code using `value` . For example, since we always return the same function for our `greeting` service, we can use `value` to define it, too:

```
myMod.config(function($provide) {
  $provide.value('greeting', function(name) {
    alert("Hello, " + name);
  });
});
```

Again, this is 100% identical to the other two methods we've used to define this function--it's just a way to save some typing.

Now you probably noticed this annoying `myMod.config(function($provide) { ... })` thing I've been using. Since defining new providers (via *any* of the given methods above) is so common, AngularJS exposes the `$provide` methods directly on the module object, to save even more typing:

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

myMod.provider("greeting", ...);
myMod.factory("greeting", ...);
myMod.service("greeting", ...);
myMod.value("greeting", ...);
```

These all do the same thing as the more verbose `app.config(...)` versions we used previously.

The one injectable I've skipped so far is `constant`. For now, it's easy enough to say that it works just like `value`. We'll see there's one difference later.

To review, *all* these pieces of code are doing the *exact* same thing:

```
myMod.provider('greeting', function() {
  this.$get = function() {
    return function(name) {
      alert("Hello, " + name);
    };
  };
});

myMod.factory('greeting', function() {
  return function(name) {
    alert("Hello, " + name);
  };
});

myMod.service('greeting', function() {
  return function(name) {
    alert("Hello, " + name);
  };
});

myMod.value('greeting', function(name) {
  alert("Hello, " + name);
});
```

The Injector (`$injector`)

The injector is responsible for actually creating instances of our services using the code we provided via `$provide` (no pun intended). Any time you write a function that takes injected arguments, you're seeing the injector at work. Each AngularJS application has a single `$injector` that gets created when the application first starts; you can get a hold of it by injecting `$injector` into any injectable function (yes, `$injector` knows how to inject itself!)

Once you have `$injector`, you can get an instance of a defined service by calling `get` on it with the name of the service. For example,

```
var greeting = $injector.get('greeting');
greeting('Ford Prefect');
```

The injector is also responsible for injecting services into functions; for example, you can magically inject services into any function you have using the injector's `invoke` method;

```
var myFunction = function(greeting) {
  greeting('Ford Prefect');
};
$injector.invoke(myFunction);
```

It's worth noting that the injector will only create an instance of a service *once*. It then caches whatever the provider returns by the service's name; the next time you ask for the service, you'll actually get the exact same object.

So, it stands to reason that you can inject services into **any function that is called with `$injector.invoke`**. This includes

- controller definition functions
- directive definition functions
- filter definition functions
- the `$get` methods of providers (aka the `factory` definition functions)

Since `constant`s and `value`s always return a static value, they are not invoked via the injector, and thus you cannot inject them with anything.

Configuring Providers

You may be wondering why anyone would bother to set up a full-fledged provider with the `provide` method if `factory`, `value`, etc. are so much easier. The answer is that providers allow a lot of configuration. We've already mentioned that when you create a service via the provider (or any of the shortcuts Angular gives you), you create a new provider that defines how that service is constructed. What I *didn't* mention is that these providers can be injected into `config` sections of your application so you can interact with them!

First, Angular runs your application in two phases--the `config` and `run` phases. The `config` phase, 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.

You can add additional code to be run in these phases with the `myMod.config` and `myMod.run` functions--each take a function to run during that specific phase. 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 are any *providers* for services you've made:

```
myMod.config(function(greetingProvider) {
  // a-ok!
});
```

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

Whenever you defined a provider for a service, that provider gets named `serviceProvider`, 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.

Since this is a more complex example, here's a working demonstration: http://jsfiddle.net/BinaryMuse/9GjYg/

Controllers (`$controller`)

You can inject things into controllers, but you can't inject controllers into things. That's because controllers aren't created via the provider. Instead, there is a built-in Angular service called `$controller` that is responsible for setting up your controllers. When you call `myMod.controller(...)`, you're actually accessing this service's provider, just like in the last section.

For example, when you define a controller like this:

```
myMod.controller('MainController', function($scope) {
  // ...
});
```

What you're actually doing is this:

```
myMod.config(function($controllerProvider) {
  $controllerProvider.register('MainController', function($scope) {
    // ...
  });
});
```

Later, when Angular needs to create an instance of your controller, it uses the `$controller` service (which in turn uses the `$injector` to invoke your controller function so it gets its dependencies injected too).

Filters and Directives

`filter` and `directive` work exactly the same way as `controller`; `filter` uses a service called `$filter` and its provider `$filterProvider`, while `directive` uses a service called `$compile` and its provider `$compileProvider`. Some links:

- $filter: http://docs.angularjs.org/api/ng.$filter
- $filterProvider: http://docs.angularjs.org/api/ng.$filterProvider
- $compile: http://docs.angularjs.org/api/ng.$compile
- $compileProvider: http://docs.angularjs.org/api/ng.$compileProvider

As per the other examples, `myMod.filter` and `myMod.directive` are shortcuts to configuring these services.

Summary

So, to summarize, any function that gets called with `$injector.invoke` **can be injected into**. This includes, but is not limited to:

- controller
- directive
- factory
- filter
- provider `$get` (when defining provider as an object)
- provider function (when defining provider as a constructor function)
- service

The provider creates new services **that can be injected into things**. This includes:

- constant
- factory
- provider
- service
- value

That said, built-in services like `$controller` and `$filter` *can* be injected, and you can *use* those services to get hold of the new filters and controllers you defined with those methods (even though the things you defined aren't, by themselves, able to be injected into things).

Other than that, any injector-invoked function can be injected with any provider-provided service--there is no restriction (other than the `config` and `run` differences listed herein).

Adapted from this Stack Overflow answer