AngularJS Services

provider()

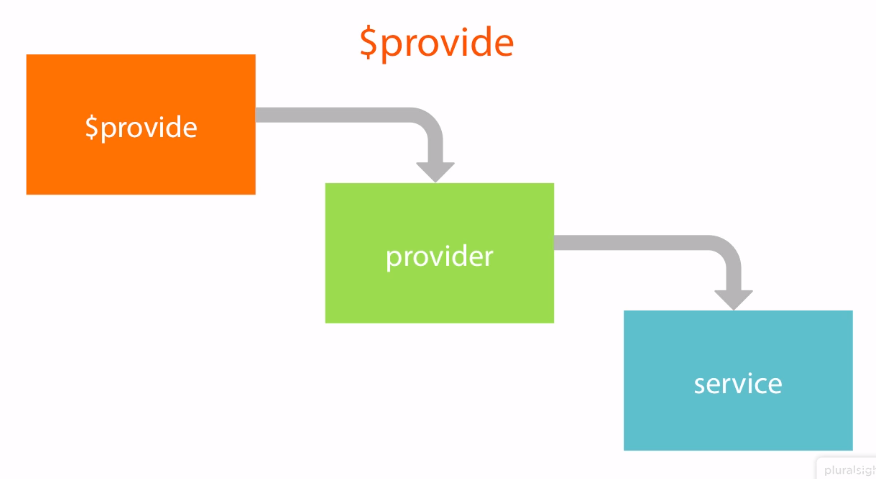
factory()

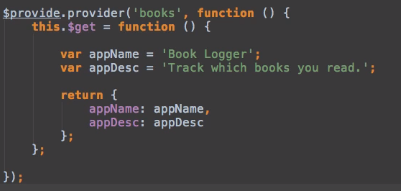
service()

value()

constant()

# Providers



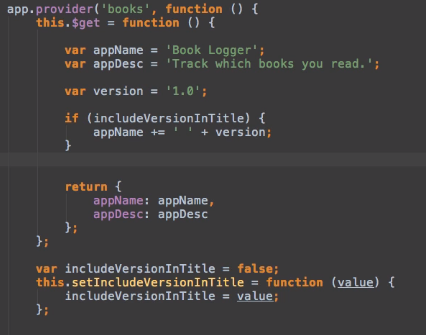


Or angular.module.(‘app’, []).provider(‘books’, function….)

Every service has a provider!  
‘squareService’ => ‘squareServiceProvider’

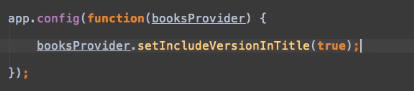
Providers must have $get property, Angular uses its function to create your service, and the return value of that function represents your service.

You can customize behavior of providers:



If you inject ‘books’, only books.appName & books.appDesc is available.

However, if you inject ‘booksProvider’ in the app config, you can call the ‘setIncludeVersionInTitle’

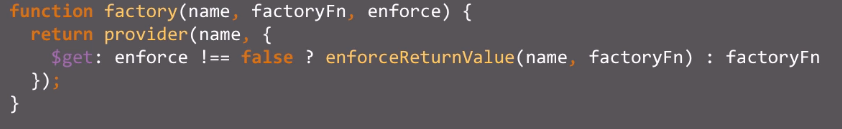


Examples where we use this (src => app => core => config):

$logProvider.debugEnabled => controls whether angular logs debug messages (you don’t want this in production!)  
$mdAriaProvider.disableWarnings => controls whether angular material shows aria warnings (aka no aria-label on buttons or icons that have no text)  
$compileProvider.debugInfoEnabled => show less info in the HTML in production mode, makes AngularJS quite a bit faster if disabled  
exceptionHandlerProvider.configure(conf.appErrorPrefix);  
routerHelperProvider.configure({ docTitle: conf.appTitle + ': ' });  
ngWigToolbarProvider.addStandardButton(‘underline’….)  
….

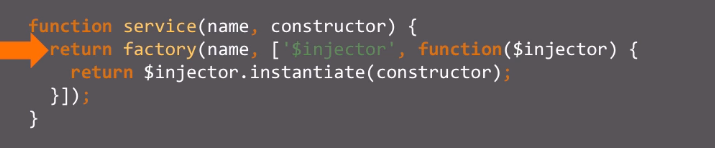
<https://docs.angularjs.org/error/$injector/unpr>

# Factory

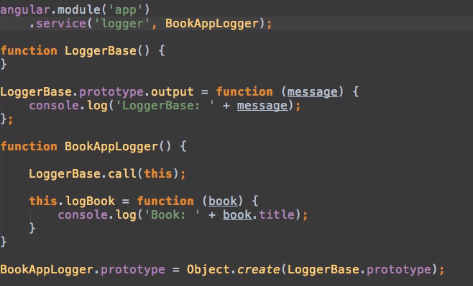


Calls the provider fn and assigns the value you pass to the factory function as the value of the get property on the provider. Enforce is to make sure it passes an object

# Service



The function you call to the service function is created as a constructor, thus “new” is called by the $injector.instantiate. Why do you want this? Inheritance hierarchy. In our case we don’t really need services but use it quite a bit, but it’s not big deal



# Value

= short syntax for factory without parameters  
you can use value instead of factory if you don’t need to inject anything into it (like $http, $q, …)

Value can’t be injected in angular.module().config but constant can

# Constant

Registers service with injector, no factory/provider calls. Can be an object literal, global object (like “\_”, “moment”, … or a string)

# Annotations

Inform injector what to inject  
Used to support minimization

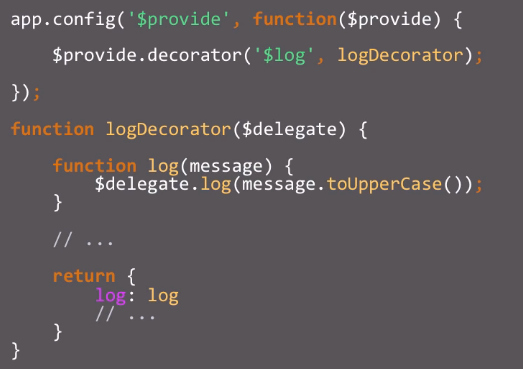
1. Use same names as the registered services to inject -> works but breaks when minified:  
   function BooksController(logger) => works  
   function a(b) => minified, breaks!
2. Use $inject (like we do)  
   BooksController.$inject = [‘logger’];  
   function BooksController(logger)  
   minified =>>  
   a.$inject = [‘logger’];  
   function a(b)
3. Use inline array notation (we use(d)? this in some cases)  
   (route file)  
    resolve: {  
    userExists: ['authService', function (authService) {  
    return authservice.updateUser();  
    }],  
    },

# Decorators

Tweak behavior of existing service or add function to a library

Decorator pattern: Design pattern that allows behavior to be added to an individual object, either statically or dynamically

Decorate angular service:



$delegate will always be injected. It’s the service being decorated, in this case “$log”.

You override $log.log() to always log the message in uppercase before logging.

Exception-handler.provider