Dependency Injection

It is a software design in which components are given their dependencies instead of hard coding them within the component. It relieves a component from locating the dependency and makes dependencies configurable. It also helps in making components reusable, maintainable and testable.

AngularJS provides a supreme Dependency Injection mechanism. It provides following core components which can be injected into each other as dependencies.

* Value
* Factory
* Service
* Provider
* Constant

Why the duplication?

1. <script>
2. angular.module('DemoApp', [])
3. .controller('DemoController', ['$scope', '$log', function($scope, $log) {
4. $scope.message = "Hello World";
5. $log.debug('logging hello');
6. }]);
8. </script>

You might wonder why do we need that duplication of the names. Why do we need both the string '$scope' and then the parameter $scope.

Actually we don't need that, but without that our code will break when we minify it.

Introspection

There are many AngularJS examples where the above code looks like this:

angular.module('DemoApp', [])

.controller('DemoController', function($scope, $log) {

$scope.message = "Hello World";

$log.debug('logging hello');

});

The difference is that in this case the second parameter the controller function receives is the function implementing it. There is no array wrapping it. The function has the parameters it is expecting, but they are not listed earlier.

So the duplication of names is gone, but you might be wondering how does Angular know which objects to pass to this function and in which order?

The explanation is in "introspection". While the JavaScript code runs, it can look at its own source code, inspect it and know what variable names a given function is expecting. Then it can call that function passing in the correct objects in the expected order.

The problem with this approach is that if we minify our JavaScript code the parameter names of our script will be also shortened and Angular won't know what objects to pass to the function call. The previous version, the one with the duplication, solves this problem as the strings in the array declaring the values that are expected won't change even during minification. That way Angular will know what objects to pass to the function call.

# [**Use of $inject in controllers**](https://stackoverflow.com/questions/18698963/i-dont-understand-the-use-of-inject-in-controllers)

That is one approach to support Dependency Injection after your code is minified (if you choose to minify).

When you declare a controller, the function takes parameters:

function ($scope, notify)

When you minify the code, your function will look like this:

function (a, b)

Since AngularJS uses the function parameter names to infer DI, your code will break because AngularJS doesn't know about a or b.

To solve this problem, they provided additional ways to declare controllers (or other services/factories/etc) for that matter:

1. For controllers, use the $inject method - here you pass an array of literals that map to the parameters of your controller function. So, if you provide

MyController.$inject = ['$scope', 'notify'];

then the value of the first parameter to your function will be the a scope object associated with this controller and the second parameter will be the notify service.

1. When declaring new controllers, services, etc, you can use the array literal syntax. Here, you do something like this:

angular.module('myModule').controller('MyController', ['$scope', 'notify', function ($scope, notify) {

...

}]);

## Value

Value is a simple JavaScript object, which is required to pass values to the controller during config phase (config phase is when AngularJS bootstraps itself).