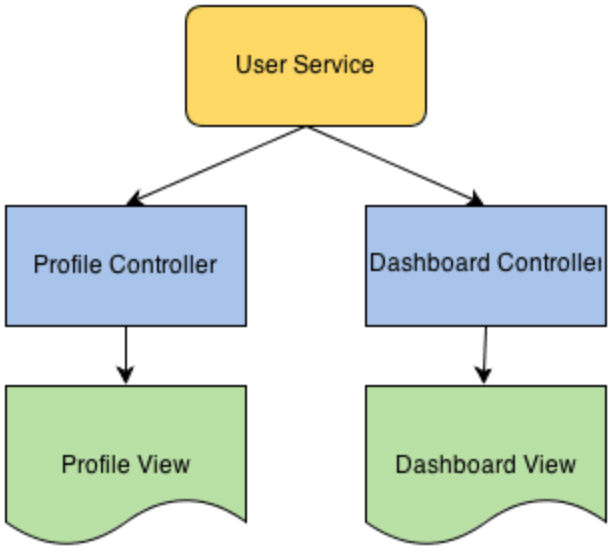
Introduction to AngularJS Services

In AngularJS world, the services are singleton objects or functions that carry out specific tasks. It holds some business logic. Separation of concern is at the heart while designing an AngularJS application. Your controller must be responsible for binding model data to views using $scope. It does not contain logic to fetch the data or manipulating it.

For that we must create singleton objects called services. AngularJS can manage these service objects. Wherever we want to use the service, we just have to specify its name and AngularJS auto-magically inject these objects (more on this later).

Thus service is a stateless object that contains some useful functions. These functions can be called from anywhere; Controllers, Directive, Filters etc. Thus we can divide our application in logical units. The business logic or logic to call HTTP url to fetch data from server can be put within a service object.

Putting business and other logic within services has many advantages. First it fulfills the principle of separation of concern or segregation of duties. Each component is responsible for its own work making application more manageable. Second this way each component can be more testable. AngularJS provides first class support for unit testing. Thus we can quickly write tests for our services making them robust and less error prone.



Consider above diagram. Here we divide our application in two controllers: 1. Profile and 2. Dashboard. Each of these controllers require certain user data from server. Thus instead of

repeating the logic to fetch data from server in each controller, we create a User service which hides the complexity. AngularJS automatically inject User service in both Profile and Dashboard controller. Thus our application becomes for modular and testable.

2. AngularJS internal services

AngularJS internally provides many services that we can use in our application. `$http` is one example (Note: All angularjs internal services starts with **$** sign). There are other useful services such as `$route`,`$window`, `$location` etc.
These services can be used within any Controller by just declaring them as dependencies. For example:

```
module.controller('FooController', function($http){
    //...
});

module.controller('BarController', function($window){
    //...
});
```

3. AngularJS custom services

We can define our own custom services in angular js app and use them wherever required.

There are several ways to declare angularjs service within application. Following are two simple ways:

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

module.service('userService', function(){
    this.users = ['John', 'James', 'Jake'];
});
```

or we can use factory method

```
module.factory('userService', function(){

    var fac = {};

    fac.users = ['John', 'James', 'Jake'];

    return fac;

});
```

Both of the ways of defining a service function/object are valid. We will shortly see the difference between `factory()` and `service()` method. For now just keep in mind that both these APIS defines a singleton service object that can be used within any controller, filter, directive etc.

4. AngularJS Service vs Factory

AngularJS services as already seen earlier are singleton objects. These objects are application wide. Thus a service object once created can be used within any other services or controllers etc.

We saw there are two ways (actually four, but for sake of simplicity lets focus on 2 ways that are widely used) of defining an angularjs service. Using `module.factory` and `module.service`.

```
module.service( 'serviceName', function );

module.factory( 'factoryName', function );
```

When declaring `serviceName` as an injectable argument you will be provided with an instance of the function. In other words `new FunctionYouPassedToService()`. This object instance becomes the service object that AngularJS registers and injects later to other services / controllers if required.
When declaring `factoryName` as an injectable argument you will be provided with the value that is returned by invoking the function reference passed to module.factory.
In below example we define `MyService` in two different ways. Note how in `.service` we create service methods using `this.methodname`. In `.factory` we created a factory object and assigned the methods to it.

AngularJS .service

```
module.service('MyService', function() {
    this.method1 = function() {
            //..
        }

    this.method2 = function() {
            //..
        }
});
```

AngularJS .factory

```
module.factory('MyService', function() {

    var factory = {};

    factory.method1 = function() {
            //..
        }

    factory.method2 = function() {
            //..
        }
```

```
    return factory;
});
```

5. Injecting dependencies in services

Angularjs provides out of the box support for dependency management.

In general the wikipedia definition of dependency injection is:

Dependency injection is a software design pattern that allows the removal of hard-coded dependencies and makes it possible to change them, whether at run-time or compile-time. …

We already saw how to use angularjs dependency management and inject dependencies in controllers. We injected `$scope` object in our controller class.
Dependency injection mainly reduces the tight coupling of code and create modular code that is more maintainable and testable. AngularJS services are the objects that can be injected in any other Angular construct (like controller, filter, directive etc). You can define a service which does certain tasks and inject it wherever you want. In that way you are sure your tested service code works without any glitch.

Like it is possible to inject service object into other angular constructs, you can also inject other objects into service object. One service might be dependence on another.

Let us consider an example where we use dependency injection between different services and controller. For this demo let us create a small calculator app that does two things: squares and cubes. We will create following entities in AngularJS:

1. **MathService** – A simple custom angular service that has 4 methods: add, subtract, multiply and divide. We will only use multiply in our example.
2. **CalculatorService** – A simple custom angular service that has 2 methods: square and cube. This service has dependency on MathService and it uses MathService.multiply method to do its work.
3. **CalculatorController** – This is a simple controller that handler user interactions. For UI we have one textbox to take a number from user and two buttons; one to square another to multiply.

Below is the code:

5.1 The HTML

```
<div ng-app="app">
    <div ng-controller="CalculatorController">
        Enter a number:
        <input type="number" ng-model="number" />
        <button ng-click="doSquare()">X<sup>2</sup></button>
```

```
        <button ng-click="doCube()">X<sup>3</sup></button>

        <div>Answer: {{answer}}</div>
    </div>
</div>
```

5.2 The JavaScript

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

app.service('MathService', function() {
    this.add = function(a, b) { return a + b };

    this.subtract = function(a, b) { return a - b };

    this.multiply = function(a, b) { return a * b };

    this.divide = function(a, b) { return a / b };
});

app.service('CalculatorService', function(MathService){

    this.square = function(a) { return MathService.multiply(a,a); };
    this.cube = function(a) { return MathService.multiply(a, MathService.multiply(a,a)); };

});

app.controller('CalculatorController', function($scope, CalculatorService) {

    $scope.doSquare = function() {
        $scope.answer = CalculatorService.square($scope.number);
    }

    $scope.doCube = function() {
        $scope.answer = CalculatorService.cube($scope.number);
    }
});
```

Thus in the above angularjs injected service object to another service and in turn injected final service object to the controller object. You can inject same service object in multiple controllers. As angularjs service object is inheritedly singleton. Thus only one service object will be created per application