How to create (singleton) AngularJS services in 4 different ways

Posted on March 16, 2013 by Emil van Galen Tweet

Next to creating controllers and directives, AngularJS also supports “singleton” services. Services, like on the server-side, offer a great way for separating logic from your controllers.

In AngularJS anything that’s either a primitive type, function or object can be a service.

Although the concept of service is quite straight forward, the declaration of them in AngularJS isn’t:

- There are 4 different ways to declare a service.
  - Registering a existing value as a service
  - Registering a factory function to create the singleton service instance
  - Registering a constructor function to create the singleton service instance
  - Registering a service factory which can be configured
- Only 1 of them is extensively documented

The other 3 are only barely documentedThis post describes each option and when to use it in more detail.

Registering a existing value as a service

You can register an existing value as service, using the following methods of the `Module` type:

- “`constant(name, value)`“. intended for registering configuration data and should therefore only be used with primitives or object containing just data.
- “`value(name, value)`“: registers a primitive, existing object instance or function.
  The big difference between the “constant” and “value” methods is that:
  - “constant” should only be used for… constants.
  - services registered with “value” as well as the 3 other ways can even be proxied.

Consider the following example using both styles:

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

app.constant('magicNumber', 42);
app.constant('bookTitle', "Hitchhiker's Guide");

function UsingConstantServiceCtrl(magicNumber, bookTitle) {
  $scope.magicNumber = magicNumber;
  $scope.bookTitle = bookTitle;
}

(function() {
  var existingServiceInstance = {
    getMagicNumber: function() {
      return 42; // Note that we are using an "hard-coded" magic number
    }
  };

  app.value('magicNumberService', existingServiceInstance);
}());

function UsingValueServiceCtrl(magicNumberService) {
  $scope.magicNumberFromService = magicNumberService.getMagicNumber();
}
```

Registering a factory function to create the singleton service instance

Instead of supplying an existing value, you could also register a factory function for the service.
However since services in AngularJS are “singletons” the factory function is invoked **once**.
A factory function can optionally take parameters which, just like controllers and directives, which will be injected by AngularJS.

Using the earlier registered magicNumber ‘service’, you can now declare a service using the “`factory(name, providerFunction)`” (extensively documented) method:

```
(function() {
  // registers a service factory with "magicNumber" injected
  app.factory('magicNumberService', function(magicNumber) {
    return {
      getMagicNumber: function() {
        return magicNumber;
      }
    };
  });
}());
```

Registering a constructor function to create the singleton service instance

Instead of registering a factory function that returns an instance of a service,
you could also register a constructor function for your service object using the
“`service(name, constructor)`” method:

```
(function() {
  var MyService = function(magicNumber) { // "magicNumber" is injected
    this.getMagicNumber = function() {
      return magicNumber;
    };
  };

```

```
  app.service('magicNumberService', MyService);
}());
```

Registering a service factory which can be configured

Last but not least… there is way more advanced way to register a service factory using "`provider(name, providerType)`" which can be configured used the "`Module#config(configFn)`" .

Using the "`provider(name, providerType)`" you can register a so-called "providerType".
This "providerType" can be either an existing object or a constructor function containing:

- any number of configuration methods (i.e. "setMagicNumber")
- a "$get" factory function just like the one used with "`factory(name, providerFunction)`"

Using "`provider(name, providerType)`" we can make our service configurable:

```
app.provider('magicNumberService', {
  // internal configuration data; configured through setter function
  magicNumber: null,

  // configuration method for setting the magic number
  setMagicNumber: function(magicNumber) {
    this.magicNumber = magicNumber;
  },

  $get: function(magicNumber) {
    // use the magic number explicitly provided through "setMagicNumber"
    // otherwise default to the injected "magicNumber" constant
    var toBeReturnedMagicNumber = this.magicNumber || magicNumber;

    // return the service instance
    return {
      getMagicNumber: function() {
        return toBeReturnedMagicNumber;
      }
    };
  }
});
```

To allow configuration each service registered with "`provider(name, providerType)`" automatically gets a "special" additional service with the "Provider" postfix.
This special "…Provider" service (i.e. "magicNumberServiceProvider") can be used solely in combination with "`Module#config(configFn)`" to configure the service prior to its construction:

```
app.config('magicNumberServiceProvider', function() {
  magicNumberServiceProvider.setMagicNumber(99);
});
```

When should you use which way?

There is no right or wrong when it comes the various way in which you can register a service.
Some might argue that "`factory(name, providerFunction)`" would be preferable since it's extensively documented.
Others would prefer registering existing service instances using "`value(name, value)`".

To ease choosing between the 4 different ways I will shortly recap all of them:

- Use either "`value(name, value)`" or "`constant(name, value)`" to register an existing value:
  - you typically would use "value" to register a service object or a function
  - whereas "constant" should only be used for configuration data
- "`factory(name, providerFunction)`": registers a factory function responsible for creating the "singleton" service instance.
  Just like "value(name, value)" a factory function could return anything from primitive type, function or object instance.
- "`service(name, constructor)`": registers the constructor function which will be constructed using a "new" keyword.
- "`provider(name, providerType)`": the most advanced way to register a service.
  Most often there is no need for your service to be configurable through a "…Provider" service, unless you are writing a reusable JavaScript library containing AngularJS services.

References

- Google Groups discussion titled "What is the difference between module.factory and module.service and how might both be applied?" containing a very useful explanation from Miško Hevery (search for "Lets look at the simplest scenario" to find it).
- Gist from Mithrandir0x title "Difference between Service, Factory and Provider in AngularJS"
- The "Registering Services" paragraph from the "Creating Services" section from the AngularJS "Developer Guide" describes how use the "factory(name, providerFunction)" method from the Module type to register services.

This entry was posted in AngularJS, Coding, Javascript, REST by Emil van Galen. Bookmark the permalink [http://blog.jdriven.com/2013/03/how-to-create-singleton-angularjs-services-in-4-different-ways/] .

About Emil van Galen

My name is Emil van Galen, I work for JDriven. I'm passionate about software development and continuously seeking for ways to improve code, learn new technologies, improving software design and keeping things fresh.
View all posts by Emil van Galen →

---

20 THOUGHTS ON "HOW TO CREATE (SINGLETON) ANGULARJS SERVICES IN 4 DIFFERENT WAYS"

Pingback: AngularJS学习资源列表 | 天天三国杀

Pingback: AngularJS - Beginners guide / Yoosuf Muhammad

Pingback: Currently Open Tabs: Resources For Using AngularJS - Anti-Code.com

Pingback: angularjs 学习大礼包大赠送【权威英文版】！ _ Angular _ 小G的大Q

Samuel Smith
on **July 11, 2013 at 00:08** said:

Your examples are confusing since you use the same code to define a constructor function and then when you register it for the factory you construct it.

Emil van Galen
on **July 12, 2013 at 09:25** said:

Thank you for your useful comment… I just changed the code example to make it (hopefully) less confusing.

Aravind MS
on **August 5, 2013 at 10:21** said:

In the section ‘Registering a service factory which can be configured’,

app.provider(name, providerType): helps to register a service.

How do we inject the dependencies such as $q, $rootScope, etc?

Emil van Galen
on **August 5, 2013 at 20:56** said:

Aravind,

As it turned out my usage sample of “Module#provider” did actually contain an error (which I just corrected) which might have caused some confusion.

The “$get” (factory) function can be injected just like other AngularJS

code.
Since at first I wasn't completely sure of this myself I just created a Plunk based of the (corrected) "Module#provider" usage sample from the blog post that proofs that is works (check the console log to see that $q is injected):
http://plnkr.co/edit/AnwHRE3khBUMPX4AuCLp?p=preview

Kind regards,

Emil

Rob K
on **September 13, 2013 at 03:59** said:

This is a very good article, certainly help me understand the differences. The only suggestion I would make is to add a concrete/real world example for when you would use each type.

Emil van Galen
on **September 13, 2013 at 14:06** said:

Thank you for your suggestion… reading back my blog post I have to agree to some parts would me more descriptive with some real world examples.
Additionally I'm thinking of explaining "$provide" and the fact that Module#constant was also be wired into the Module#config function.

Pingback: AngularJS Service, Factory, Provider diff | gelistirme.org

Pingback: Having trouble creating a Singleton AngularJS Service | Technology & Programming

Manu
on **November 27, 2013 at 13:57** said:

This helped a lot. Great extend to the angular documentation.

Pingback: AngularJS-Learning | Nisar Khan

Pingback: Learning AngularJS – best resources to learn angularjs for beginners. | Bhavin Patel

Pingback: AngularJS – Beginners guide | Bhavin Patel

Pingback: AngularJS – Service VS Factory VS Provider | Eureka!

Pingback: How could you become zero to hero in AngularJS? | Milap Bhojak

Pingback: AngularJS Provider Multiple Instance | Alonso

Pingback: web标准学习-小辉博客 | 学习AngularJS【转】