AngularJS: Factory vs Service vs Provider

- By Tyler
- On May 4, 2014
- With 120 Comments
- In Technical
-

When you first get started with Angular, you' ll naturally find yourself flooding your controllers and scopes with unnecessary logic. It' s important to realize early on that your controller should be very thin; meaning, most of the business logic and persistent data in your application should be taken care of or stored in a service. I see a few questions a day on Stack Overflow regarding someone trying to have persistent data in his or her controller. That' s just not the purpose of a controller. For memory purposes, controllers are instantiated only when they are needed and discarded when they are not. Because of this, every time you switch a route or reload a page, Angular cleans up the current controller. Services however provide a means for keeping data around for the lifetime of an application while they also can be used across different controllers in a consistent manner.

Angular provides us with three ways to create and register our own service.

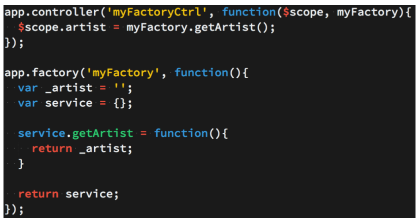
1) Factory

2) Service

3) Provider

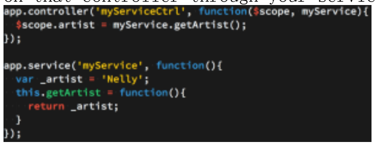
TL;DR

1) When you' re using a Factory you create an object, add properties to it, then return that same object. When you pass this service into your controller, those properties on the object will now be available in that controller through your factory.



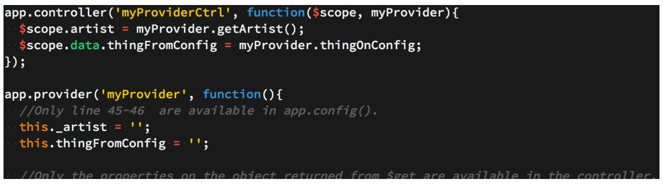
```
app.controller('myFactoryCtrl', function($scope, myFactory){
  $scope.artist = myFactory.getArtist();
});

app.factory('myFactory', function(){
  var _artist = '';
  var service = {};

  service.getArtist = function(){
    return _artist;
  }

  return service;
});
```

2) When you' re using Service, it' s instantiated with the 'new' keyword. Because of that, you' ll add properties to 'this' and the service will return 'this' . When you pass the service into your controller, those properties on 'this' will now be available on that controller through your service.

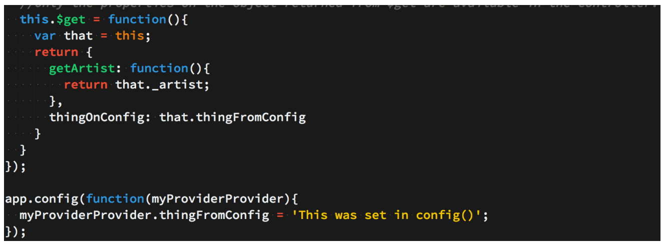


```
app.controller('myServiceCtrl', function($scope, myService){
  $scope.artist = myService.getArtist();
});

app.service('myService', function(){
  var _artist = 'Nelly';
  this.getArtist = function(){
    return _artist;
  }
});
```

3) Providers are the only service you can pass into your .config() function. Use a provider when you want to provide module-wide configuration for your service object before making it available.



```
app.controller('myProviderCtrl', function($scope, myProvider){
  $scope.artist = myProvider.getArtist();
  $scope.data.thingFromConfig = myProvider.thingOnConfig;
});

app.provider('myProvider', function(){
  //Only line 45-46  are available in app.config().
  this._artist = '';
  this.thingFromConfig = '';

  //Only the properties on the object returned from $get are available in the controller.
```



```
  this.$get = function(){
    var that = this;
    return {
      getArtist: function(){
        return that._artist;
      },
      thingOnConfig: that.thingFromConfig
    }
  }
});

app.config(function(myProviderProvider){
  myProviderProvider.thingFromConfig = 'This was set in config()';
});
```

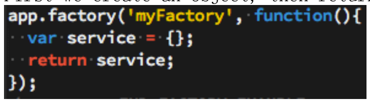
NON TL;DR

In order to extensively show the difference between a Factory, Service, and Provider, we’ re going to build the same service in three separate ways. The services are going to utilize the iTunes API as well as promises with $q.

1) Factory

Factories are the most popular way to create and configure a service. There’ s really not much more than what the TL;DR said. You just create an object, add properties to it, then return that same object. Then when you pass the factory into your controller, those properties on the object will now be available in that controller through your factory. A more extensive example is below.

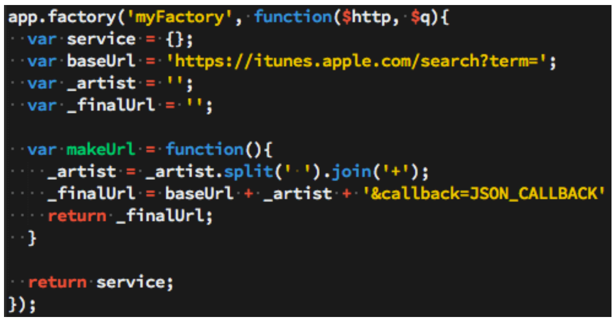
First we create an object, then return that object like so.



```
app.factory('myFactory', function(){
  var service = {};
  return service;
});
```

Now whatever properties we attach to ‘service’ will be available to us when we pass ‘myFactory’ into our controller.

Now let’ s add some ‘private’ variables to our callback function. These won’ t be directly accessible from the controller, but we will eventually set up some getter/setter methods on ‘service’ to be able to alter these ‘private’ variables when needed.



```
app.factory('myFactory', function($http, $q){
  var service = {};
  var baseUrl = 'https://itunes.apple.com/search?term=';
  var _artist = '';
  var _finalUrl = '';

  var makeUrl = function(){
    _artist = _artist.split(' ').join('+');
    _finalUrl = baseUrl + _artist + '&callback=JSON_CALLBACK'
    return _finalUrl;
  }

  return service;
});
```

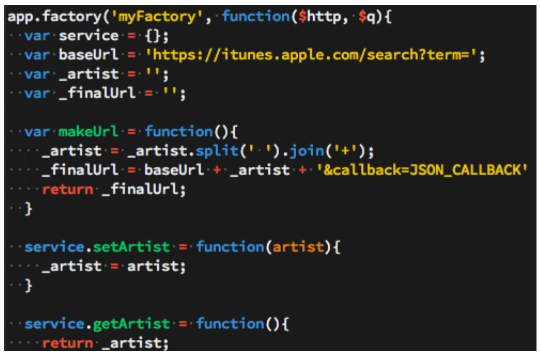
Here you’ ll notice we’ re not attaching those variables/function to ‘service’ . We’ re simply creating them in order to either use or modify them later.

- baseUrl is the base URL that the iTunes API requires
- _artist is the artist we wish to lookup
- _finalUrl is the final and fully built URL to which we’ ll make the call to iTunes
- makeUrl is a function that will create and return our iTunes friendly URL.

Now that our helper/private variables and function are in place, let’ s add some properties to the ‘service’ object. Whatever we put on ‘service’ we’ ll be able to directly use in whichever controller we pass ‘myFactory’ into.

We are going to create setArtist and getArtist methods that simply return or set the artist. We are also going to create a method that will call the iTunes API with our created URL. This method is going to return a promise that will fulfill once the data has come back from the iTunes API. If you haven’ t had much experience using promises in Angular, I highly recommend doing a deep dive on them.

- setArtist accepts an artist and allows you to set the artist
- getArtist returns the artist
- callItunes first calls makeUrl() in order to build the URL we’ ll use with our $http request. Then it sets up a promise object, makes an $http request with our final url, then because $http returns a promise, we are able to call .success or .error after our request. We then resolve our promise with the iTunes data, or we reject it with a message saying ‘There was an error’ .



```
app.factory('myFactory', function($http, $q){
  var service = {};
  var baseUrl = 'https://itunes.apple.com/search?term=';
  var _artist = '';
  var _finalUrl = '';

  var makeUrl = function(){
    _artist = _artist.split(' ').join('+');
    _finalUrl = baseUrl + _artist + '&callback=JSON_CALLBACK'
    return _finalUrl;
  }

  service.setArtist = function(artist){
    _artist = artist;
  }

  service.getArtist = function(){
    return _artist;
  }

  service.callItunes = function(){
    makeUrl();
    var deferred = $q.defer();
    $http({
      method: 'JSONP',
      url: _finalUrl
    }).success(function(data){
      deferred.resolve(data);
    }).error(function(){
      deferred.reject('There was an error')
    })
    return deferred.promise;
  }

  return service;
});
```

Now our factory is complete. We are now able to inject 'myFactory' into any controller and we' ll then be able to call our methods that we attached to our service object (setArtist, getArtist, and callItunes).

```
app.controller('myFactoryCtrl', function($scope, myFactory){
  $scope.data = {};
  $scope.updateArtist = function(){
    myFactory.setArtist($scope.data.artist);
  };

  $scope.submitArtist = function(){
    myFactory.callItunes()
      .then(function(data){
        $scope.data.artistData = data;
      }, function(data){
        alert(data);
      })
  }
});
```

In the controller above we' re injecting in the 'myFactory' service. We then set properties on our $scope object that are coming from data from 'myFactory' . The only tricky code above is if you' ve never dealt with promises before. Because callItunes is returning a promise, we are able to use the .then() method and only set $scope.data.artistData once our promise is fulfilled with the iTunes data. You' ll notice our controller is very 'thin' . All of our logic and persistent data is located in our service, not in our controller.

2) Service

Perhaps the biggest thing to know when dealing with creating a Service is that that it' s instantiated with the 'new' keyword. For you JavaScript gurus this should give you a big hint into the nature of the code. For those of you with a limited background in JavaScript or for those who aren' t too familiar with what the 'new' keyword actually does, let' s review some JavaScript fundamentals that will eventually help us in understanding the nature of a Service.

To really see the changes that occur when you invoke a function with the 'new' keyword, let' s create a function and invoke it with the 'new' keyword, then let' s show what the interpreter does when it sees the 'new' keyword. The end results will both be the same.

First let' s create our Constructor.

```
var Person = function(name, age){
  this.name = name;
  this.age = age;
}
```

This is a typical JavaScript constructor function. Now whenever we invoke the Person function using the 'new' keyword, 'this' will be bound to the newly created object.

Now let' s add a method onto our Person' s prototype so it will be available on every instance of our Person 'class' .

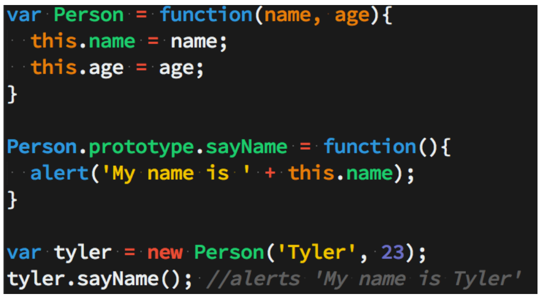
```
Person.prototype.sayName = function(){
  alert('My name is ' + this.name);
}
```

Now, because we put the sayName function on the prototype, every instance of Person will be able to call the sayName function in order alert that instance' s name.

Now that we have our Person constructor function and our sayName function on its prototype, let' s actually create an instance of Person then call the sayName function.

```
var tyler = new Person('Tyler', 23);
tyler.sayName(); //alerts 'My name is Tyler'
```

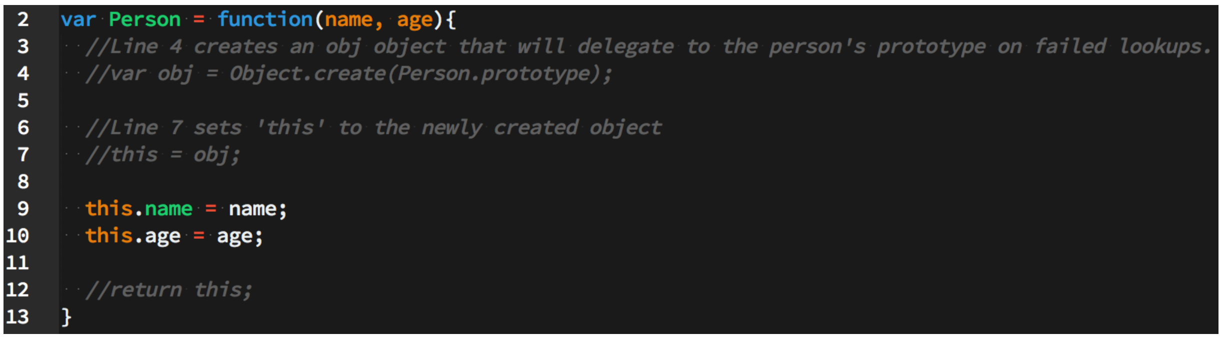
So all together the code for creating a Person constructor, adding a function to it’s prototype, creating a Person instance, and then calling the function on its prototype looks like this.



```
var Person = function(name, age){
  this.name = name;
  this.age = age;
}

Person.prototype.sayName = function(){
  alert('My name is ' + this.name);
}

var tyler = new Person('Tyler', 23);
tyler.sayName(); //alerts 'My name is Tyler'
```

Now let’s look at what actually is happening when you use the ‘new’ keyword in JavaScript. First thing you should notice is that after using ‘new’ in our example, we’re able to call a method (sayName) on ‘tyler’ just as if it were an object – that’s because it is. So first, we know that our Person constructor is returning an object, whether we can see that in the code or not. Second, we know that because our sayName function is located on the prototype and not directly on the Person instance, the object that the Person function is returning must be delegating to its prototype on failed lookups. In more simple terms, when we call tyler.sayName() the interpreter says “OK, I’m going to look on the ‘tyler’ object we just created, locate the sayName function, then call it. Wait a minute, I don’t see it here – all I see is name and age, let me check the prototype. Yup, looks like it’s on the prototype, let me call it.”.

Below is code for how you can think about what the ‘new’ keyword is actually doing in JavaScript. It’s basically a code example of the above paragraph. I’ve put the ‘interpreter view’ or the way the interpreter sees the code inside of notes.



```
2  var Person = function(name, age){
3    //Line 4 creates an obj object that will delegate to the person's prototype on failed lookups.
4    //var obj = Object.create(Person.prototype);
5
6    //Line 7 sets 'this' to the newly created object
7    //this = obj;
8
9    this.name = name;
10   this.age = age;
11
12   //return this;
13 }
```

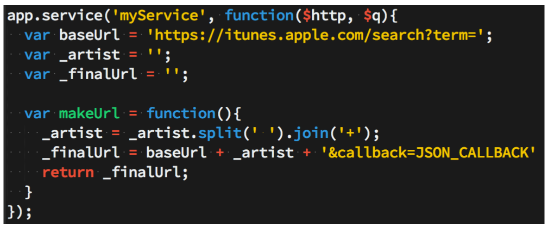
Now having this knowledge of what the ‘new’ keyword really does in JavaScript, creating a Service in Angular should be easier to understand now.

The biggest thing to understand when creating a Service is knowing that Services are instantiated with the ‘new’ keyword. Combining that knowledge with our examples above, you should now recognize that you’ll be attaching your properties and methods directly to ‘this’ which will then be returned from the Service itself. Let’s take a look at this in action.

Unlike what we originally did with the Factory example, we don’t need to create an object then return that object because, like

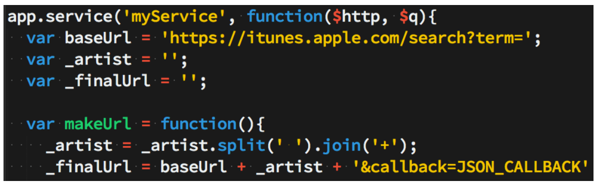
mentioned many times before, we used the ‘new’ keyword so the interpreter will create that object, have it delegate to it’s prototype, then return it for us without us having to do the work.

First things first, let’s create our ‘private’ and helper function. This should look very familiar since we did the exact same thing with our factory. I won’t explain what each line does here because I did that in the factory example, if you’re confused, re-read the factory example.

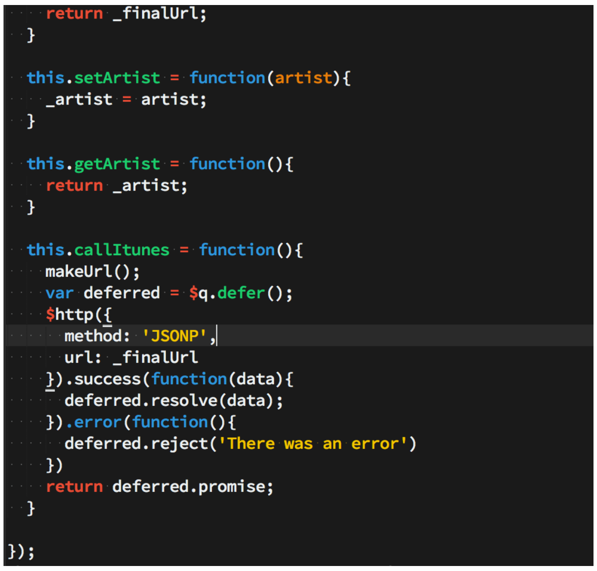


```
app.service('myService', function($http, $q){
  var baseUrl = 'https://itunes.apple.com/search?term=';
  var _artist = '';
  var _finalUrl = '';

  var makeUrl = function(){
    _artist = _artist.split(' ').join('+');
    _finalUrl = baseUrl + _artist + '&callback=JSON_CALLBACK'
    return _finalUrl;
  }
});
```

Now, we’ll attach all of our methods that will be available in our controller to ‘this’.

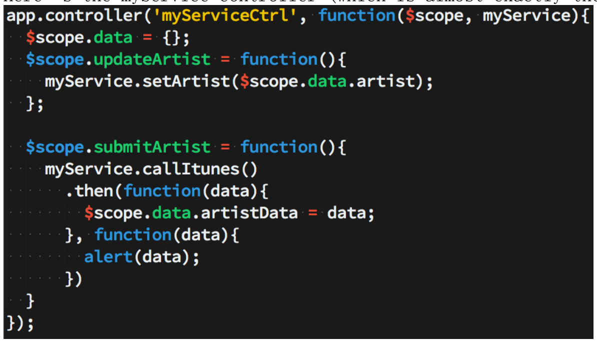


```
app.service('myService', function($http, $q){
  var baseUrl = 'https://itunes.apple.com/search?term=';
  var _artist = '';
  var _finalUrl = '';

  var makeUrl = function(){
    _artist = _artist.split(' ').join('+');
    _finalUrl = baseUrl + _artist + '&callback=JSON_CALLBACK'
```



```
    return _finalUrl;
  }

  this.setArtist = function(artist){
    _artist = artist;
  }

  this.getArtist = function(){
    return _artist;
  }

  this.callItunes = function(){
    makeUrl();
    var deferred = $q.defer();
    $http({
      method: 'JSONP',
      url: _finalUrl
    }).success(function(data){
      deferred.resolve(data);
    }).error(function(){
      deferred.reject('There was an error')
    })
    return deferred.promise;
  }

});
```

Now just like in our factory, setArtist, getArtist, and callItunes will be available in whichever controller we pass myService into. Here’ s the myService controller (which is almost exactly the same as our factory controller).



```
app.controller('myServiceCtrl', function($scope, myService){
  $scope.data = {};
  $scope.updateArtist = function(){
    myService.setArtist($scope.data.artist);
  };

  $scope.submitArtist = function(){
    myService.callItunes()
      .then(function(data){
        $scope.data.artistData = data;
      }, function(data){
        alert(data);
      })
  }
});
```

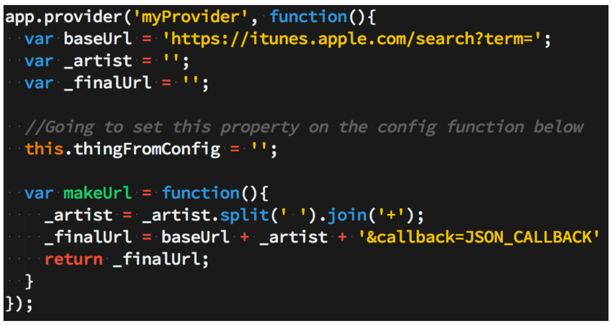
Like I mentioned before, once you really understand what ‘new’ does, Services are almost identical to factories in Angular.

3) Provider

The biggest thing to remember about Providers is that they’ re the only service that you can pass into the app.config portion of

your application. This is of huge importance if you’ re needing to alter some portion of your service object before it’ s available everywhere else in your application. Although very similar to Services/Factories, there are a few differences which we’ ll discuss.

First we set up our Provider in a similar way we did with our Service and Factory. The variables below are our ‘private’ and helper function.

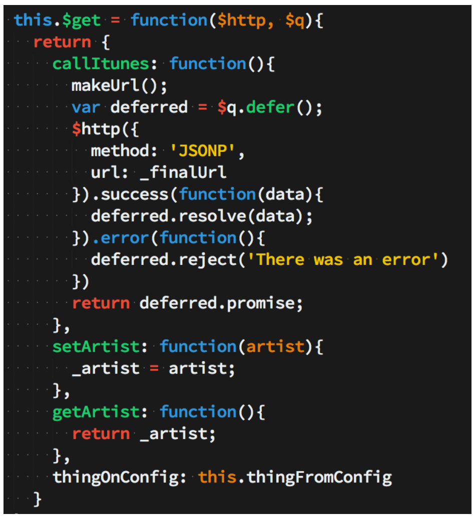


```
app.provider('myProvider', function(){
  var baseUrl = 'https://itunes.apple.com/search?term=';
  var _artist = '';
  var _finalUrl = '';

  //Going to set this property on the config function below
  this.thingFromConfig = '';

  var makeUrl = function(){
    _artist = _artist.split(' ').join('+');
    _finalUrl = baseUrl + _artist + '&callback=JSON_CALLBACK'
    return _finalUrl;
  }
});
```

*Again if any portion of the above code is confusing, check out the Factory section where I explain what it all does it greater details.

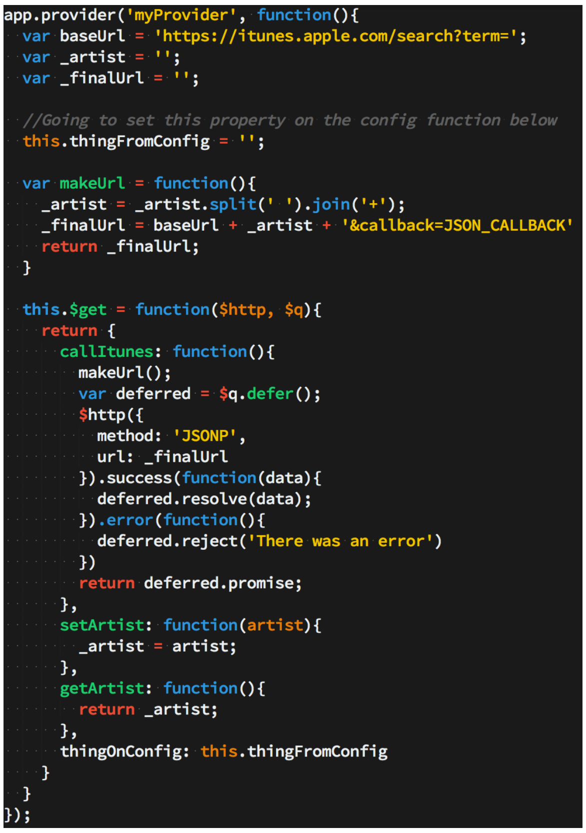
You can think of Providers as having three sections. The first section is the ‘private’ variables/functions that will be modified/set later (shown above). The second section is the variables/functions that will be available in your app.config function and are therefore available to alter before they’ re available anywhere else (also shown above). It’ s important to note that those variables need to be attached to the ‘this’ keyword. In our example, only ‘thingFromConfig’ will be available to alter in the app.config. The third section (shown below) is all the variables/functions that will be available in your controller when you pass in the ‘myProvider’ service into that specific controller.

When creating a service with Provider, the only properties/methods that will be available in your controller are those properties/methods which are returned from the $get() function. The code below puts $get on ‘this’ (which we know will eventually be returned from that function). Now that $get function returns all the methods/properties we want to be available in the be returned from that function). Now, that $get function returns all the methods/properties we want to be available in the controller. Here’s a code example.



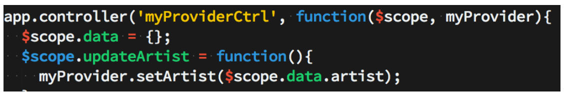
```
this.$get = function($http, $q){
  return {
    callItunes: function(){
      makeUrl();
      var deferred = $q.defer();
      $http({
        method: 'JSONP',
        url: _finalUrl
      }).success(function(data){
        deferred.resolve(data);
      }).error(function(){
        deferred.reject('There was an error')
      })
      return deferred.promise;
    },
    setArtist: function(artist){
      _artist = artist;
    },
    getArtist: function(){
      return _artist;
    },
    thingOnConfig: this.thingFromConfig
  }
```

Now the full Provider code looks like this

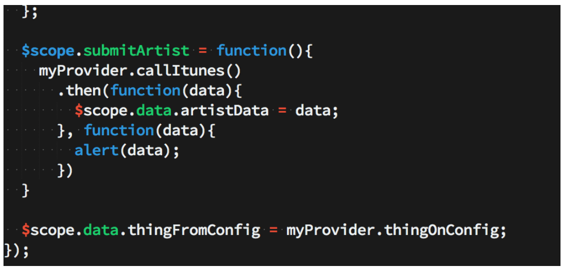


```
app.provider('myProvider', function(){
  var baseUrl = 'https://itunes.apple.com/search?term=';
  var _artist = '';
  var _finalUrl = '';

  //Going to set this property on the config function below
  this.thingFromConfig = '';

  var makeUrl = function(){
    _artist = _artist.split(' ').join('+');
    _finalUrl = baseUrl + _artist + '&callback=JSON_CALLBACK'
    return _finalUrl;
  }

  this.$get = function($http, $q){
    return {
      callItunes: function(){
        makeUrl();
        var deferred = $q.defer();
        $http({
          method: 'JSONP',
          url: _finalUrl
        }).success(function(data){
          deferred.resolve(data);
        }).error(function(){
          deferred.reject('There was an error')
        })
        return deferred.promise;
      },
      setArtist: function(artist){
        _artist = artist;
      },
      getArtist: function(){
        return _artist;
      },
      thingOnConfig: this.thingFromConfig
    }
  }
});
```

Now just like in our factory and Service, setArtist, getArtist, and callItunes will be available in whichever controller we pass myProvider into. Here’s the myProvider controller (which is almost exactly the same as our factory/Service controller).

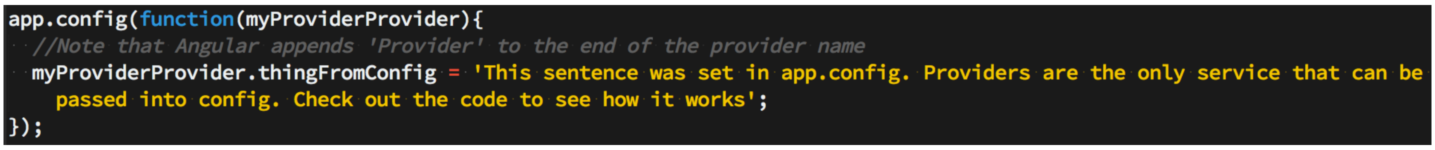


```
app.controller('myProviderCtrl', function($scope, myProvider){
  $scope.data = {};
  $scope.updateArtist = function(){
    myProvider.setArtist($scope.data.artist);
```



```
  };

  $scope.submitArtist = function(){
    myProvider.callItunes()
      .then(function(data){
        $scope.data.artistData = data;
      }, function(data){
        alert(data);
      })
  }

  $scope.data.thingFromConfig = myProvider.thingOnConfig;
});
```

As mentioned before, the whole point of creating a service with Provider is to be able to alter some variables through the app.config function before the final object is passed to the rest of the application. Let’s see an example of that.



```
app.config(function(myProviderProvider){
  //Note that Angular appends 'Provider' to the end of the provider name
  myProviderProvider.thingFromConfig = 'This sentence was set in app.config. Providers are the only service that can be
    passed into config. Check out the code to see how it works';
});
```

Now you can see how ‘thingFromConfig’ is as empty string in our provider, but when that shows up in the DOM, it will be ‘This sentence was set…’.

Thank you for reading and I hoped this helped you to be able to discern the difference between Factory, Service, and Provider in Angular.

*To see the full code example and see the code in action, feel free to fork my repo at https://github.com/tylermcginnis/AngularServices or visit my StackOverflow answer to this question here.

Tyler McGinnis

@tylermcginnis33

Share this post: on Twitter on Facebook on Google+

120 Comments

1. - Blaine Gunn
   - May 5, 2014
   - Reply
   -

   This was very helpful in illustrating the major differences between these 3 services. I prefer the service method it’s very simple. Thank you for dropping this knowledge on us Mr. McGinnis!

   - - Tyler
     - May 8, 2014
     - Reply
     -

     Thanks Blaine!

   - - Samuele
     - January 22, 2015
     - Reply
     -

     Really nice post!! The first I found that clearly explains the differences between the three major ways to inject code into the controller! It also resolve my doubts about this. Thanks!

2. - Austin
   - May 5, 2014
   - Reply
   -

   Really nice explanation and examples (especially for the factory / service relationship).

   PS. Every time you use images for code a kitten dies… 😟

   - - Tyler
     - May 5, 2014
     - Reply
     -

     Thanks for the nice comment. Sorry about the images! Every time I try to do code it jacks it all up and it’s super frustrating. Here’s the code if you want to play around with the examples for yourself.

     - - Richard Clayton
       - July 4, 2014
       - Reply
       -

       If you are blogging in pure HTML or Markdown you can always embed a Github Gist: https://gist.github.com/.

- - Tyler
  - July 6, 2014
  - Reply
  -

  Great idea. I' ll do that next time.

- - rr
  - October 2, 2014
  - Reply
  -

  classic! +1

- - Omar
  - December 7, 2014
  - Reply
  -

  "Every time you use images for code a kitten dies" ? Noooo!

  I guess that, and prefer:

  "Every time you use images for code a …" + skilled-soon-to-be-programmer is born.
  Because there wont be the temptation to copy/paste the code, but to WRITE IT instead.
  😐

  - - Tyler
    - December 7, 2014
    - Reply
    -

    That' s a fantastic way to look at it. Copy and Paste gives you a false sense of security. Makes you feel like you get it until it comes time to actually write it yourself. Great comment.

3. - Britton
   - May 5, 2014
   - Reply
   -

   Great post Tyler, keep the Angular posts coming!

4. - Jason
   - May 5, 2014
   - Reply
   -

   Nice post. I understand the difference between how services and factories are constructed but from the perspective of the consuming controller are there any significant difference. There does not appear to be. Is the choice between using a factory and a service simply a matter of how you prefer to structure and write your code?

5. - Johannes
   - May 5, 2014
   - Reply
   -

   In your example for configuring the provider the configured property 'thingFromConfig' needs to be a object property instead of a local variable. That is you need to access it via

   this.thingFromConfig

   A function scoped local variable would not be accessible from app.config. Otherwise a nice article! Thanks for posting.

   - - Tyler
     - May 5, 2014
     - Reply
     -

     Ah great catch. I changed that last minute. Thanks for the feedback, it' s been fixed.

6. - Brock Whitten
   - May 5, 2014
   - Reply
   -

   Thanks for writing this terrific article.

   - - Tyler
     - May 5, 2014
     - Reply
     -

     No problem. Thanks for reading.

7. - Enrico
   - May 5, 2014
   - Reply
   -

   Thank you very much for this article Tyler!

   - - Tyler
     - May 5, 2014
     - Reply
     -

     You are very welcome. Thanks for reading.

8.
   - fraserxu
   - May 5, 2014
   - Reply
   -

   The best post I’ve ever read on Angularjs Service/Factory!

   -
      - Tyler
      - May 5, 2014
      - Reply
      -

      Thanks so much! Glad you enjoyed it.

9.
   - Johan
   - May 6, 2014
   - Reply
   -

   Nice post however I still feel the reason to use service vs factory is unclear.

   It’s important that a factory can return any of a number of types – primitive, object, function or instance of a custom type. For example you could return a constructor function and use “new” to create separate instances of your factory across your application.

   Since a service is instantiated with “new” it is ideal for custom types.

   The angular services (including value and constant service types) docs are actually very good (the original docs were not very clear):

   https://docs.angularjs.org/guide/providers

   -
      - Tyler
      - May 7, 2014
      - Reply
      -

      Great comment. Thanks for the insights.

10.
   - Sony Arouje
   - May 6, 2014
   - Reply
   -

   Thank you so much Tyler. I am new to javascript and I am learning Angular and this post is very helpful for me to understand Services and Providers. As you said at the beginning, I was writing all the code inside the controller :). Thanks for giving me the right direction.

   Wish you might write a post about require js explaining in simple manner like this.

   Thanks once again Tyler.

   -
      - Tyler
      - May 7, 2014
      - Reply
      -

      Thank you for the kind words Sony! I’m glad you enjoyed it. I’ll try to post more frequently.

11.
   - Raoel
   - May 7, 2014
   - Reply
   -

   Great! Several times I have tried to grasp the difference between service and factory, but your detailed example have pushed me past that barrier: I get it now 😃

   I was still puzzled when I should use a Factory and when a Service and then I found this sentence in the AngularJS-docs Johan posted:
   “Factory and Service are the most commonly used recipes. The only difference between them is that Service recipe works better for objects of custom type, while Factory can produce JavaScript primitives and functions.”

   So there you have it. You can both use them, but the intention behind them is different. And it’s a good advice to always try to use a framework the way it was intented.

   Applying this knowledge on the example given in this article, I would say you should use a service for the class that calls iTunes.

   -
      - Tyler
      - May 7, 2014
      - Reply
      -

      Awesome comment. Thanks for the info!

12.
   - James
   - May 10, 2014
   - Reply
   -

   Great article, this really helped me understand the differences. I have one issue, I converted some of my factories to follow your concept of putting all public methods into another object “service” which gets returned and get all private methods as individual variables. The problem I’m having is testing, because you don’t have anyway to access the private methods outside the service (rightly so). Is there anything process here which will still allow me to test the private methods?

   -
      - Tyler
      - May 13, 2014
      - Reply

Hi James, I may be wrong on this, but here’s my intuition. The whole point of setting up the ‘service’ object was to create those private getter/setter methods as you mentioned. Yes you won’t have direct access to test those private variables, but you do have direct access to test the interface which interacts with those variables (the getter and setter methods). What’s the disadvantage of testing the interface rather than the private variables? Seems like they’re one in the same to me. Thanks for reading!

13.
- Allen Kim
- May 13, 2014
- Reply

Well Done, I wrote the differences about these three in stackflow as a popular answer, but I found out that I was misleading the readers about service and factory. So I took it down and did not have time to rewrite the answer again. Thanks for this, you did not only saved my time but also did gave me the confidence of AngularJS community.

- Tyler
- May 13, 2014
- Reply

Allen, thanks for those kind words. I’m glad you enjoyed the post!

14.
- Praveenkumar Arepalli
- May 13, 2014
- Reply

It was really good article.I struggled to find the difference.Now its clear.
thank you Tyler.

15.
- Savannah
- May 13, 2014
- Reply

YAY! Thanks Tyler!

- Tyler
- May 13, 2014
- Reply

Savannah! You rock. Hope all is well at HR. Tell everyone I say hello.

16.
- Yoorek
- May 13, 2014
- Reply

Nice article.

One comment just to complicate stuff – you can use ‘service’ for returning objects as you do with ‘factory’

For example:

app.service(‘MyService’, function () {

return {
getName: function (message) {
return “Hello ” + message;
}
};
});

Will it work? It does not do what constructor should do with this.getName = … etc.

But it works!!!

Angular constructs service instance the same manner as for factory and does not call ‘new’ on service constructor function directly
($injector.instantiate in Angular source code explains it)

So, in your examples you could just change function name app.factory to app.service and it would work!!

- Tyler
- May 14, 2014
- Reply

Awesome! Thanks for the reply.

- rr
- October 2, 2014
- Reply

so can you summarize and say they are the same functionally ?

17.
- Jamie
- May 14, 2014
- Reply

Awesome! I see the light 😀 Thanks

- Tyler

- May 14, 2014
- Reply
-

Thanks for reading! Glad I could help.

18. - Ctibor Laky
    - May 14, 2014
    - Reply
    -

    Nice reading,
    You could also mentioned here that there are some other providers in angular that this article wont be discussed, but they exist. (for those who want to know, look here: http://blog.xebia.com/2013/09/01/differences-between-providers-in-angularjs/ or official docs: https://docs.angularjs.org/guide/providers)
    You should also mentioned pros and cons of each attitude. Something like Thomas answer here:
    http://stackoverflow.com/questions/18939709/angularjs-when-to-use-service-instead-of-factory.
    From Angular docs: Service works better for objects of custom type, while Factory can produce JavaScript primitives and functions.
    But anyway this is good start for understanding them 😉 Thanks

    - - Tyler
      - May 14, 2014
      - Reply
      -

      Thanks for the comment and the extra info.

19. - Maarten
    - June 3, 2014
    - Reply
    -

    Wow, thanks for your detailed EFFORT to mould complicated stuff into easy chunks…very smooth reading this was.

    - - Tyler
      - June 3, 2014
      - Reply
      -

      You are very welcome. Thanks for reading.

20. - Ravi Teja
    - June 4, 2014
    - Reply
    -

    Great article….. Thanks alot…. now it was very clear to me and much helpful for my future projects. Please keep posting

21. - Dhruv
    - June 9, 2014
    - Reply
    -

    Really good article. Hope to see many more in the future ^^. By the way can you also make an article about promises :D?

22. - Tom
    - June 19, 2014
    - Reply
    -

    best explanation I' ve seen thus far

    - - Tyler
      - June 19, 2014
      - Reply
      -

      Thanks for the kind remarks.

23. - Satadipa
    - July 1, 2014
    - Reply
    -

    Great article with good explaination ! Thanks 😃

24. - Leo
    - July 13, 2014
    - Reply
    -

    As someone who' s new to Angular, this post was extremely helpful! I also appreciated the screenshots (is that the brogrammer theme?). I don' t think this could be put any simpler. Although I' m leaning towards the FGR stack (firebase, go, reactjs), stuff like this makes me realize that the community for angular is so valuable that it' s hard not to use it. But I digress. Love the post, love your passion, and it looks like you know what you' re talking about.

    - - Tyler
      - July 14, 2014
      - Reply
      -

      Leo Lambda,

      You' re a good *man. Thanks for the nice comment. 😃

      Tyler

25.
   - TimF
   - July 15, 2014
   - Reply
   -

   Services (of type Service) are created with the ‘new’ keyword. One would think that means that when injected into one controller it creates a new object, and then when injected into another controller, it creates another new object. But that does not seem to be the case. Services are static. So when one controller manipulates the variables within a service, it affects all other controllers also using that service. So how do you create a service that can be used by multiple controllers where the controllers don’t step on each other?

26.
   - Chetan
   - July 16, 2014
   - Reply
   -

   Awesome explanation.

27.
   - Alex
   - July 23, 2014
   - Reply
   -

   Thanks for the great article… Angular, and JS are both very new things for me, and you really helped me understand the underlying mechanisms… 😃
   I will probably reference you article again once I’ll actually need to decide which to choose, but definitely gave me a good sense… 😃

28.
   - Dave
   - July 28, 2014
   - Reply
   -

   Pretty good but an explanation about why you would choose one over the other would be nice.

29.
   - nitu bansal
   - July 29, 2014
   - Reply
   -

   great explanation.. thanks for this article.. it helped to clear concepts about all three..

30.
   - Sacx
   - August 13, 2014
   - Reply
   -

   Wow, I will be able to implement these in my code without any hesitation now. If anyone asks me in the midnight, awkening from sleep, to explain these three things I can do that.

   Thank you so much.

31.
   - Kwailo
   - August 19, 2014
   - Reply
   -

   This post was astonishingly clear and well-written. I am really thankful to you for finally clarifying these concepts for me. I have been frustrated by the fact that that despite having read mountains of words on this topic, I was never able to get it, but this was exactly the level of detail I needed (Including the explanation on how “new” works).

32.
   - Rafael
   - August 22, 2014
   - Reply
   -

   Perfect explanation! Read a lot of other articles and didn’t understand, this one finally helped me. Couldn’t be better (for the “NON TL;DR” explanation)!

   Only the “TL;DR” could be a bit better.

   a) Factory
   You provide a function F to be used as a factory. AngularJS does the following to get your factory:
   var yourFactory = F(…);

   b) Service
   You provide a function S to be used as a service. AngularJS does the following to get your service:
   var yourService = new F(…);

   c) Provider
   Your provide a function P to be used as a provider. AngularJS does the following to get your provider:
   var yourProviderProvider = new P(…);

   Then, you can write a function be passed to the config. This function will receive yourProviderProvider. After that, AngularJS gets your provider doing:
   var yourProvider = yourProviderProvider.$get(…);

   -
     - burt
     - August 28, 2014
     - Reply
     -

     Thanks, this helped!

     -
       - rr
       - October 2, 2014

- Reply
-

Very well put, so all in all, I can see that F & S make no damn difference at all at the end (just there to confuse the beejeezus out of us) whereas P acts like a singleton.
Thanks Tyler…too bad this appeared all the way at the bottom of a google search ! Deserves to be right on top (and I agree and wholeheartedly laughed at the dead kittens statement – he’s so right, for some of us its just too much color and becomes unreadable. I set all my terminals to m/c. Cheers !

33. - Ali
    - September 3, 2014
    - Reply
    -

    The clearest and most thorough explanation of service, provider and provider.

    Thanks.

34. - Anand
    - September 4, 2014
    - Reply
    -

    Great work Tyler. It helped me to understand the difference between Factory, Service and Provider. Thanks a lot

35. - Nick
    - September 10, 2014
    - Reply
    -

    There are tons of thank you comments, but I think you needed one more. This really made it clear in my head – thank you!

36. - dauber
    - September 10, 2014
    - Reply
    -

    Multiple times in the “Service” explanation you say that a service is instantiated with the “new” keyword, but I don’t see “new” ANYWHERE in your code. Could you clarify?

    - - Tyler
      - September 10, 2014
      - Reply
      -

      You (the user) don’t instantiate your services with ‘new’, but the Angular source code does.

37. - Steven
    - September 11, 2014
    - Reply
    -

    Very nice article. Easy to understand. Do you do other Angular related articles? Thanks!

    - - Tyler
      - September 11, 2014
      - Reply
      -

      Thanks for the kind words. I’ve committed to myself that I’m going to start doing technical blog posts once a week because they’ve had such good feedback in the past. So keep an eye out.

38. - Vitaly
    - September 21, 2014
    - Reply
    -

    Very nice and useful examples, thank you!

    I also like the WebStorm theme that you are using. What is it called and where can I download it?

39. - Anthony
    - September 23, 2014
    - Reply
    -

    Excellent article, very easy to read with clear explanations. I will definitely keep it in my bookmarks to refresh my mind each time I forget the difference between service, factory and provider 😉

    Thanks a lot !

40. - Julie

    - September 26, 2014
    - Reply
    -

    Thank you for this very easy to follow writeup. As someone who’s new to JavaScript, I appreciated the section on what ‘new’ does.

41. - Francisc
    - September 27, 2014
    - Reply
    -

    You can also pass `constant`s in a module’s config block.

42. - Matt

- September 28, 2014
- Reply
-

I’ve been searching Stackoverflow for a while for help on this issue, and stumbling upon this post was a breath of fresh air. Thanks for explaining it in such simple terms and including the section on the ‘new’ keyword for javascript beginners — that helped a lot.

43.
- Deepak
- October 13, 2014
- Reply
-

This is really helpful. Clears up the air.

44.
- Venkat
- October 26, 2014
- Reply
-

Very nice post with clear examples. Thank you.

45.
- Sam
- November 5, 2014
- Reply
-

As a newbee to angular , best explanation I red so far …. Easy to understand and straight to the point… Keep up the good work

Cheers

46.
- Nhut Le
- November 11, 2014
- Reply
-

It’s so helpful. Finally, they are clear to me

47.
- Jeff
- November 13, 2014
- Reply
-

Awesome…first time ever i properly understood the differences in creating services…thanks mate

48.
- Karthik
- December 5, 2014
- Reply
-

Awesome. Perfect. Excellent. I prefer angular service.

49.
- Stefan
- December 16, 2014
- Reply
-

Thank you for taking the time and write these very easy to understand and helpful posts. Mostly appreciated!

50.
- Paul
- December 22, 2014
- Reply
-

Thanks for posting such a lucid description.. really helped this old coder get his head around some of this ng stuff! Cheers…

51.
- Ronny Karam
- December 23, 2014
- Reply
-

This is by far the best explanation I’ve read; thank you for taking the time to write it down.

52.
- giri
- December 26, 2014
- Reply
-

Wow Nice explanation and i am very happy now over service and factory, provider
once again thanks

53.
- Jon Kristian Bernhardsen
- January 11, 2015
- Reply
-

Great blog post!

You explained the differences between them very well, but you did not go much into detail on when to use the different types. I think this is important as well and would probably be useful for a lot of people.

Thanks.

54.
- Dana Edwards
- January 13, 2015
- Reply
-

Thank you so much for this explanation. Really cleared a lot of things up for me.

55. - fin
    - January 23, 2015
    - Reply
    -

    You make skip 3 chapters of angular book I’m currently reading. Thank you so much for this awesome post 🙂

56. - Aparna
    - January 28, 2015
    - Reply
    -

    Thanks Tyler for explaining this horrible topic.

57. - Jodevan
    - February 12, 2015
    - Reply
    -

    Thanks a lot, dude!

    Your explanation is simply perfect! It helped me a lot!

58. - Dimos
    - February 27, 2015
    - Reply
    -

    Extremely nice article!! Thanks!

59. - Aman Kothari
    - March 23, 2015
    - Reply
    -

    Thanks!II Great post!!!!

60. - Mingjay
    - March 23, 2015
    - Reply
    -

    I think your explanation is awesome, thank you.

61. - Thang Phuoc Nguyen (Qtysth)
    - April 3, 2015
    - Reply
    -

    Awesome and really helpful, thank you!

62. - Bhau Sangale
    - April 8, 2015
    - Reply
    -

    Thanks for helpful Article !

63. - Eric Richards
    - April 28, 2015
    - Reply
    -

    Thanks, man. This is pretty sweet. Ever thought about doing instructional videos?

    - - Tyler
      - April 28, 2015
      - Reply
      -

      https://egghead.io/series/react-native-fundamentals 🙂

64. - HR
    - May 23, 2015
    - Reply
    -

    Great blog post.
    I am build an angular app which will call API to get data from server. “My question is should I use a Factory or Service to call API’s.”

65. - Elektrische zahnburste test 2015
    - June 5, 2015
    - Reply
    -

    Appreciation to my father who stated to me concerning this webpage, this weblog is really remarkable.

66. - Ajit
    - June 11, 2015
    - Reply
    -

    It is very nice article. I was totally new to all these terms/concepts, your article helped to understand it very well. Thanks

Tyler

67.
- Luiz
- June 12, 2015
- Reply
-

No need to create a new promise in callitunes. Your can simply return what $http returns which is also a promise.
Great article!

-
  - Tyler
  - June 17, 2015
  - Reply
  -

  Yes. I was young and naive when I wrote this.

68.
- S
- June 15, 2015
- Reply
-

$http returns a promise. can't the calliTunes method simply return $http()?

-
  - Tyler
  - June 17, 2015
  - Reply
  -

  Yes. I was young and naive when I wrote this.

69.
- Shinjan Mitra
- June 16, 2015
- Reply
-

Thank You so much for writing this article. I though I knew what services and factories do before I read this, but my knowledge was far from complete. Thanks again!

70.
- Javier
- June 17, 2015
- Reply
-

So great article Tyler, its helpfull. But I have a doubt, if u could answer it.

I now understand the difference between factory, service and provider. But, i cant get in my mind when to use each of them, except the provider. Could you specify when its better to use factory instead of service? Thanks a lot in advance.

71.
- Jack
- June 17, 2015
- Reply
-

This is so useful.

72.
- joe
- June 19, 2015
- Reply
-

First, great article like everyone else said. I enjoyed it. I'm new to AngularJS but have been a developer for a long time.

I've read all the comments and several people ask when its best to use factory over service over provider, but you never replied.

I think provider would be used when you want to configure your app in a single location (app.JS) but hide the methods elsewhere.

Factory properties are shared between controllers while a Service is unique since its created with 'new'. I may be wrong

I, like the others, would like to know best practice vs comfort level on when to use each. Maybe a couple more simple examples.

Looking forward to reading more of your posts! Thx

73.
- Binh Thanh Nguyen
- June 27, 2015
- Reply
-

Thanks, nice explanation

74.
- Felipe
- July 8, 2015
- Reply
-

Thanks very much! Beautiful explanation.

75.
- miglior tablet samsung 2015
- July 21, 2015
- Reply
-

Thanks designed for sharing such a pleasant thought, post is good, thats why
i have read it entirely

Add a comment

Add a comment

Name

Email

Website

Comment

Submit Comment