[**我看到的最好的解释AngularJs中Factory和Service和Provide不同**](http://blog.csdn.net/lglgsy456/article/details/36902179)

2014-07-04 17:45 3839人阅读 [评论](http://blog.csdn.net/lglgsy456/article/details/36902179#comments)(3) [收藏](javascript:void(0);) [举报](http://blog.csdn.net/lglgsy456/article/details/36902179#report)

**AngularJS: Factory vs Service vs Provider**

* By [Tyler](http://tylermcginnis.com/author/Tyler/)
* On May 4, 2014
* With [44 Comments](http://tylermcginnis.com/angularjs-factory-vs-service-vs-provider/#comments)
* In [Technical](http://tylermcginnis.com/category/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.

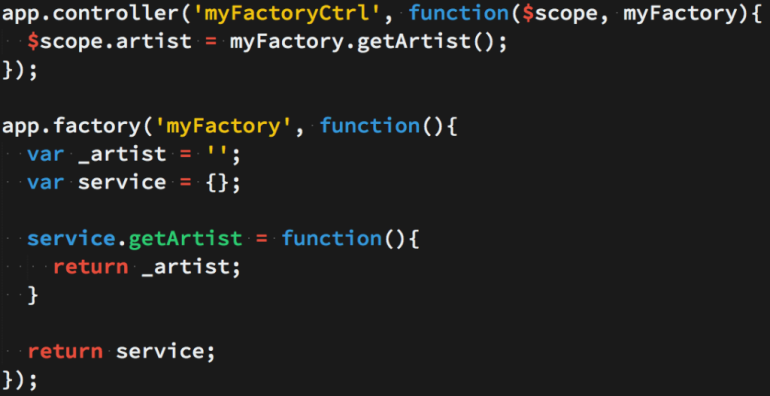
1) Factory

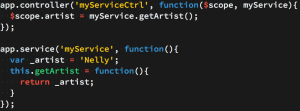
2) Service

3) Provider

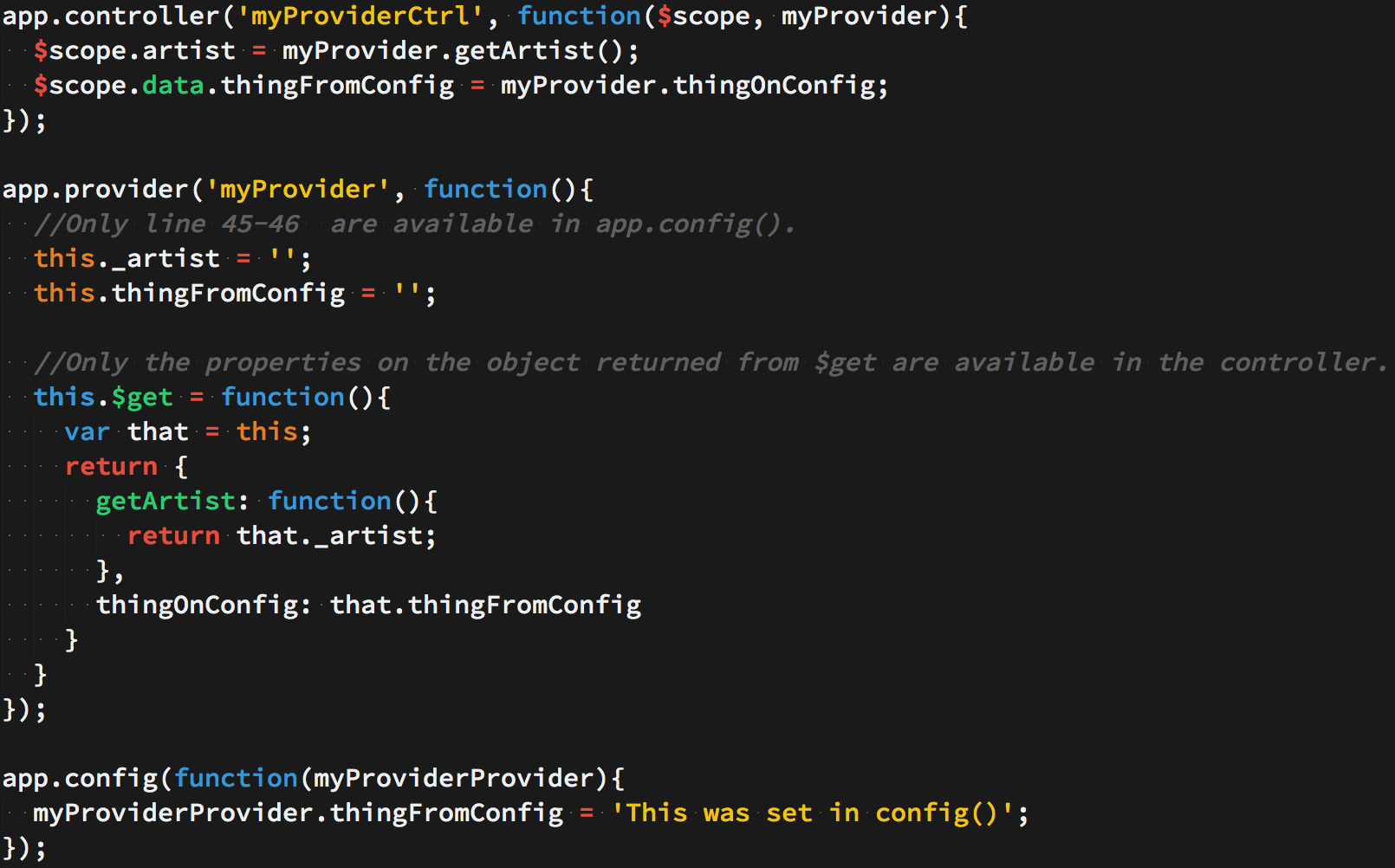
TL;DR

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.

[](http://tylermcginnis.com/wp-content/uploads/2014/05/FactoryExample1.png)

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.  


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.

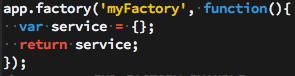
[](http://tylermcginnis.com/wp-content/uploads/2014/05/ProviderExample3.png)

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.  


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.

[](http://tylermcginnis.com/wp-content/uploads/2014/05/factory4-5.png)

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.

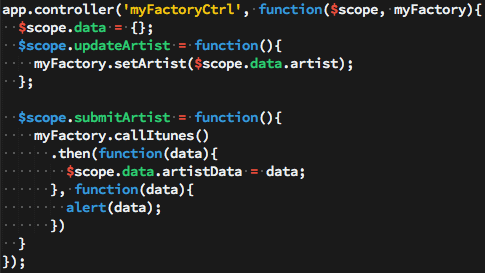
Now that our helper/private variables and function are in place, let’s add some properties to the ‘service’ object. Whatever we put on ‘service’ we’ll be able to directly use in whichever controller we pass ‘myFactory’ into.

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’.

[](http://tylermcginnis.com/wp-content/uploads/2014/05/factory6.png)

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).

[](http://tylermcginnis.com/wp-content/uploads/2014/05/factory2-7.png)

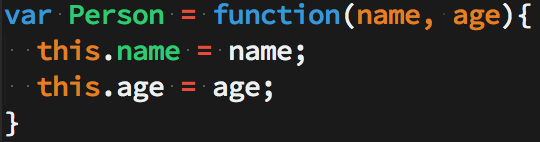
In the controller above we’re injecting in the ‘myFactory’ service. We then set properties on our $scope object that are coming from data from ‘myFactory’. The only tricky code above is if you’ve never dealt with promises before. Because callItunes is returning a promise, we are able to use the .then() 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.

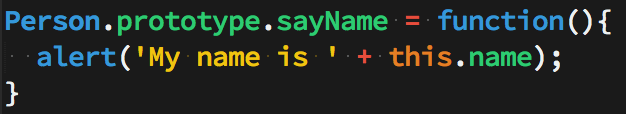
To really see the changes that occur when you invoke a function with the ‘new’ keyword, let’s create a function and invoke it with the ‘new’ keyword, then let’s show what the interpreter does when it sees the ‘new’ keyword. The end results will both be the same.

First let’s create our Constructor.

[](http://tylermcginnis.com/wp-content/uploads/2014/05/personConstructor-8.png)

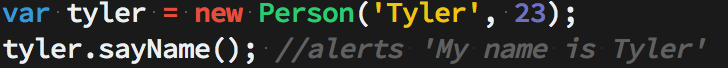
This is a typical JavaScript constructor function. Now whenever we invoke the Person function using the ‘new’ keyword, ‘this’ will be bound to the newly created object.

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

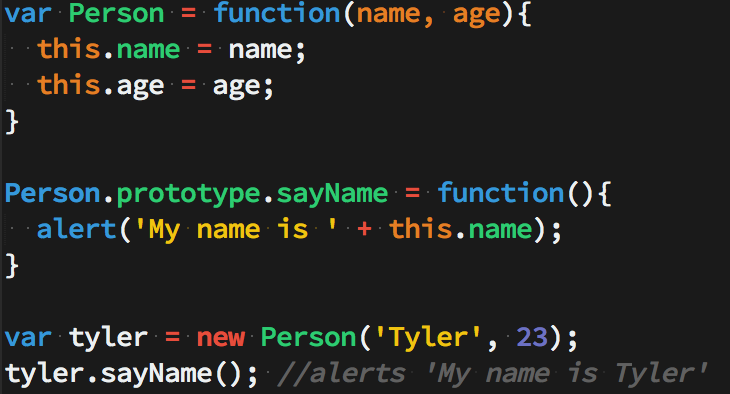
[](http://tylermcginnis.com/wp-content/uploads/2014/05/person.prototype-9.png)

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.

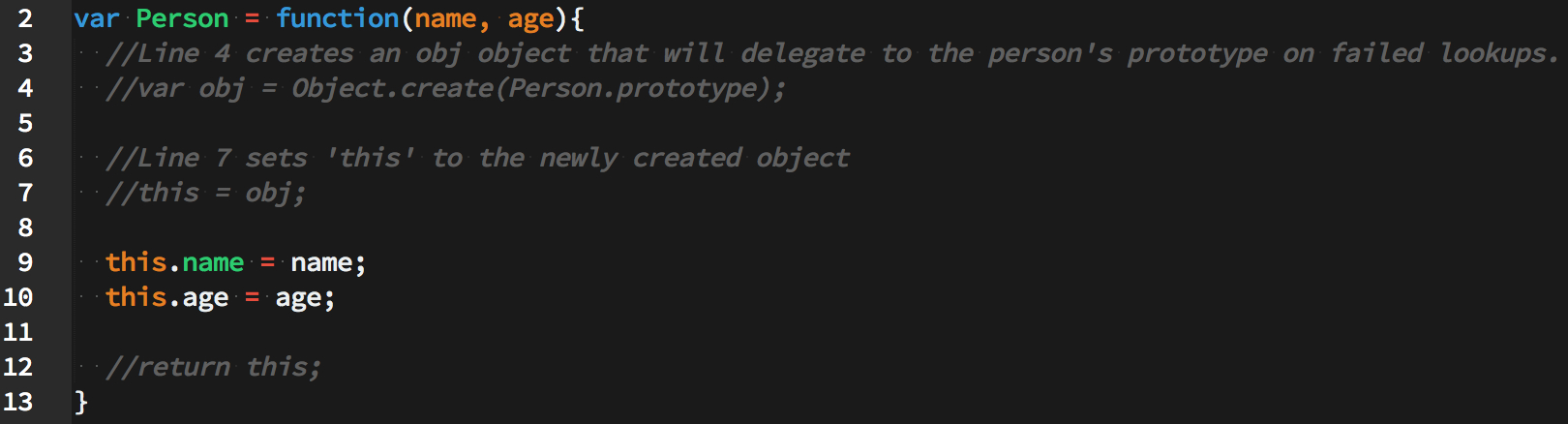
[](http://tylermcginnis.com/wp-content/uploads/2014/05/personInstance-10.png)

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.

[](http://tylermcginnis.com/wp-content/uploads/2014/05/PersonCode-11.png)

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.

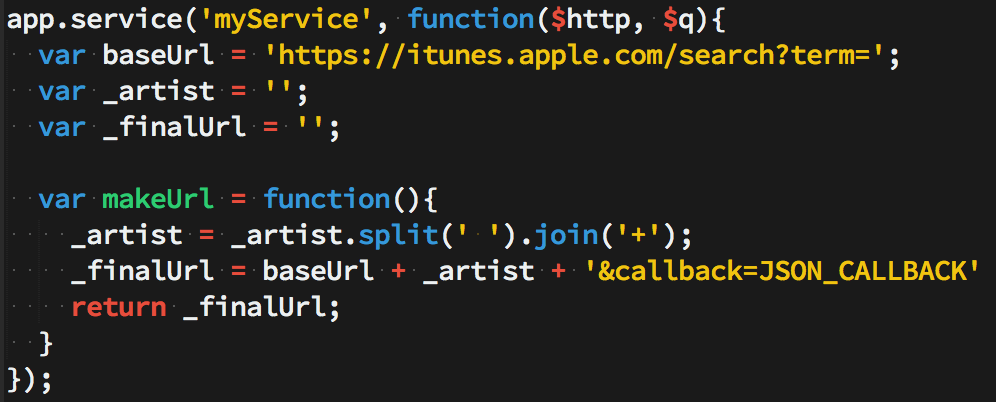
[](http://tylermcginnis.com/wp-content/uploads/2014/05/PersonCommented-12.png)

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

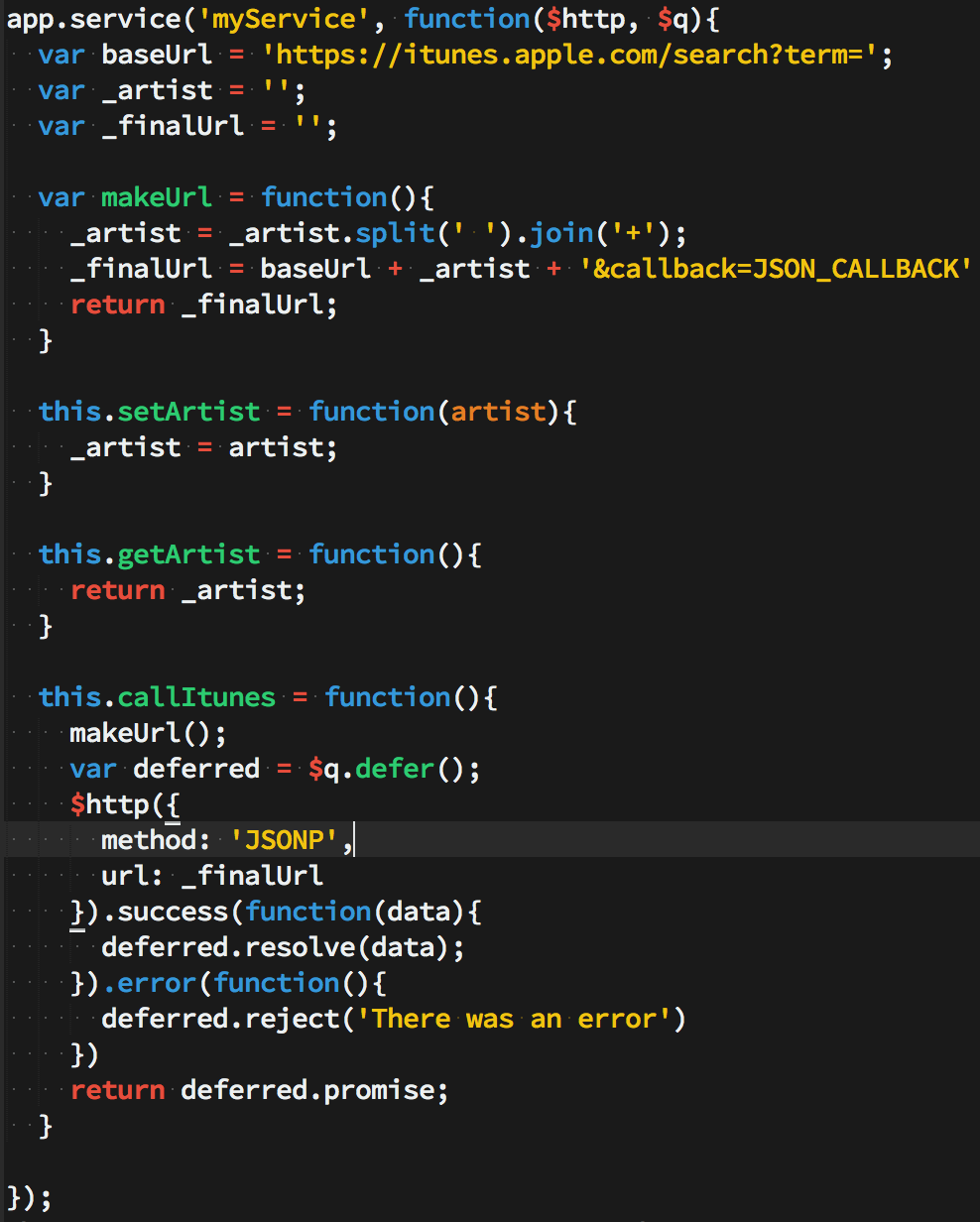
The biggest thing to understand when creating a Service is knowing that Services are instantiated with the ‘new’ keyword. Combining that knowledge with our examples above, you should now recognize that you’ll be attaching your properties and methods directly to ‘this’ which will then be returned from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

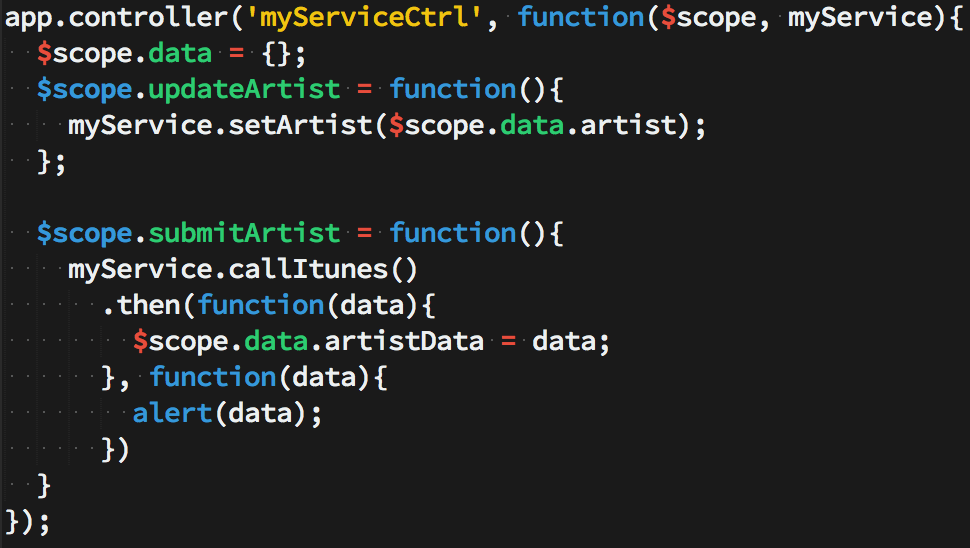
Unlike what we originally did with the Factory example, we don’t need to create an object then return that object because, like mentioned many times before, we used the ‘new’ keyword so the interpreter will create that object, have it delegate to it’s prototype, then return it for us without us having 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.

[](http://tylermcginnis.com/wp-content/uploads/2014/05/ServiceBase-13.png)

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

[](http://tylermcginnis.com/wp-content/uploads/2014/05/serviceservice-14.png)

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).  
[](http://tylermcginnis.com/wp-content/uploads/2014/05/ServiceController-15.png)

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.

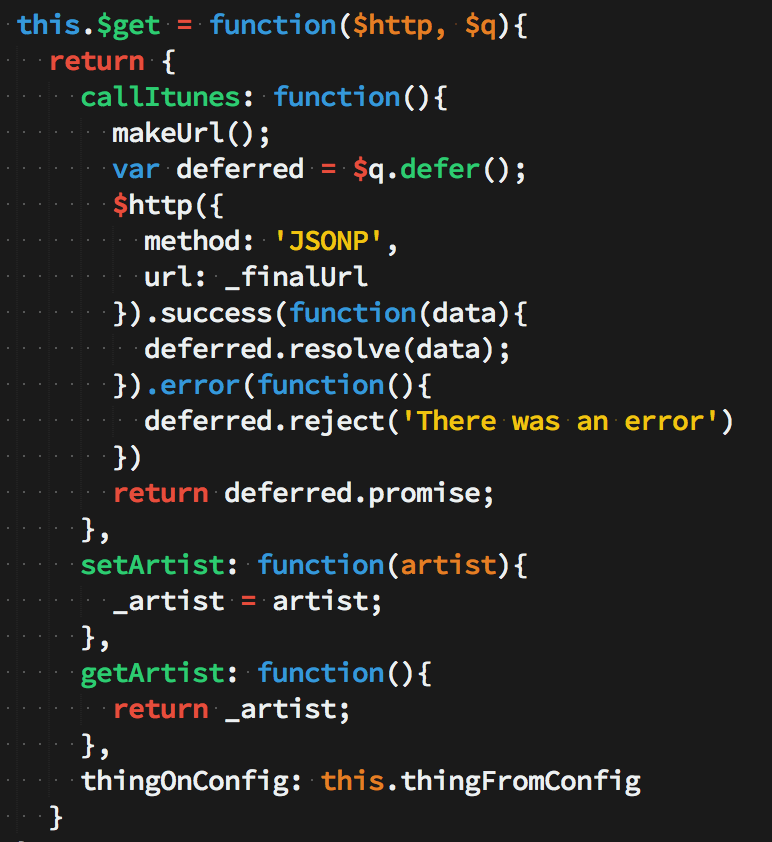
First we set up our Provider in a similar way we did with our Service and Factory. The variables below are our ‘private’ and helper function.

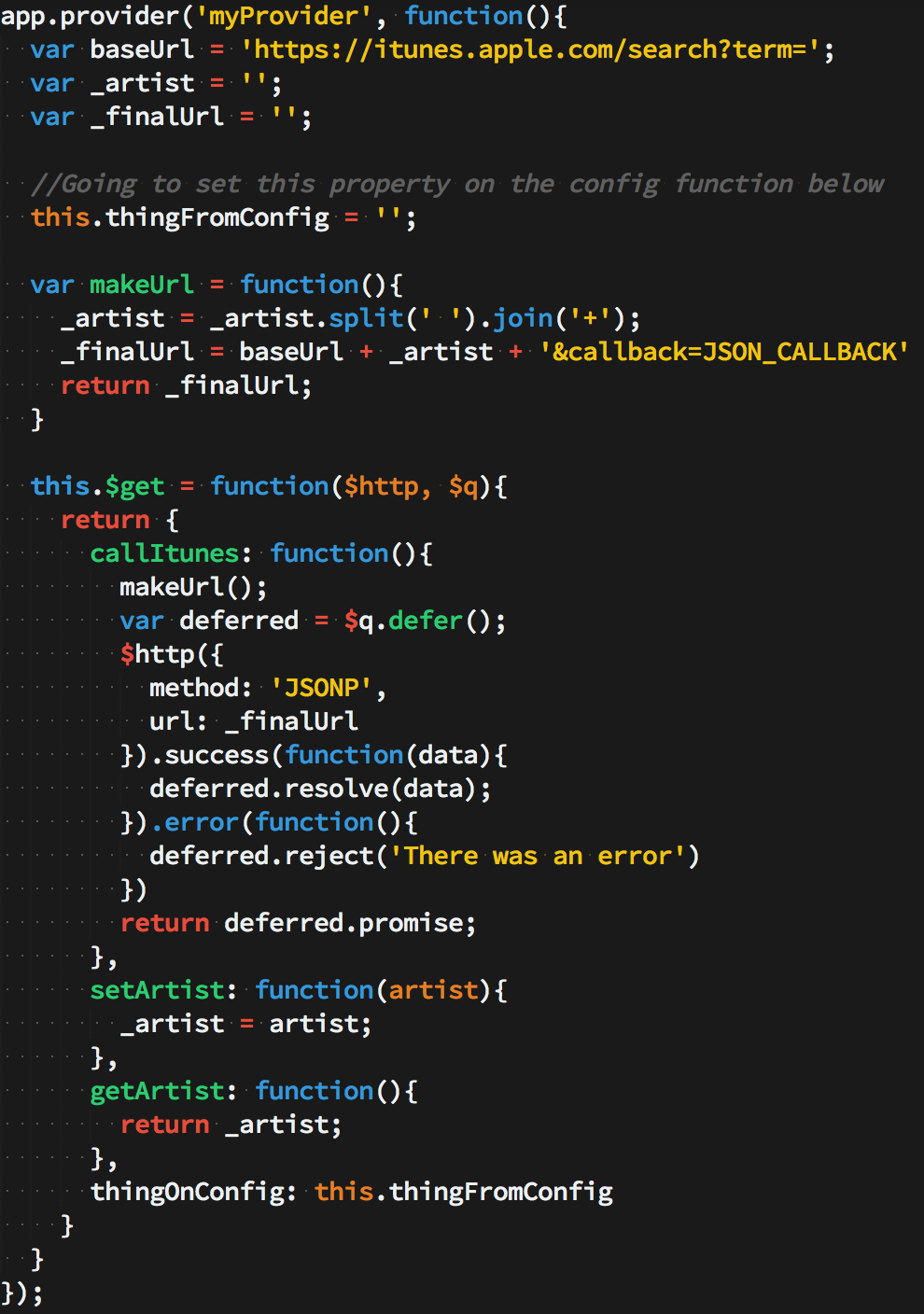
[](http://tylermcginnis.com/wp-content/uploads/2014/05/p16.png)

\*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.

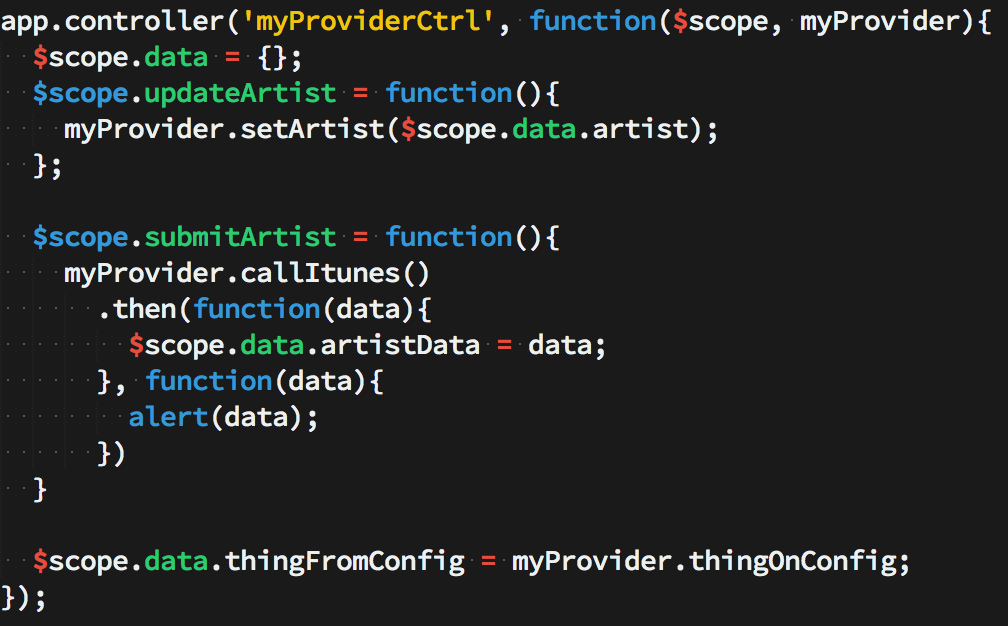
You can think of Providers as having three sections. The first section is the ‘private’ variables/functions that will be modified/set later (shown above). The second section is the variables/functions that will be available in your app.config function and are therefore available to alter before they’re 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 controller. Here’s a code example.

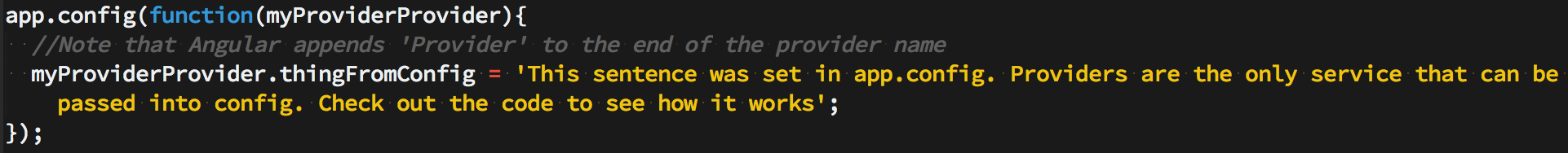
[](http://tylermcginnis.com/wp-content/uploads/2014/05/p17.png)

Now the full Provider code looks like this  
[](http://tylermcginnis.com/wp-content/uploads/2014/05/p18.png)

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).

[](http://tylermcginnis.com/wp-content/uploads/2014/05/ProviderCtrl-19.png)

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.

[](http://tylermcginnis.com/wp-content/uploads/2014/05/appconfig-20.png)

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/tylermcginnis33/AngularServices>

# AngularJS中serivce，factory，provider的区别

 [一、service引导](http://my.oschina.net/tanweijie/blog/295067#OSC_h1_1)

 [二、service](http://my.oschina.net/tanweijie/blog/295067#OSC_h1_2)

 [1、factory()](http://my.oschina.net/tanweijie/blog/295067#OSC_h2_3)

 [‍2、service()‍](http://my.oschina.net/tanweijie/blog/295067#OSC_h2_4)

 [‍3、provider()‍‍](http://my.oschina.net/tanweijie/blog/295067#OSC_h2_5)

# ****一、service引导****

刚开始学习Angular的时候，经常被误解和被初学者问到的组件是 service(), factory(), 和 provide()这几个方法之间的差别。This is where we'll start the twenty-five days of Angular calendar.

# ****二、service****

在Angular里面，services作为单例对象在需要到的时候被创建，只有在应用生命周期结束的时候（关闭浏览器）才会被清除。而controllers在不需要的时候就会被销毁了。

这 就是为什么使用controllers在应用里面传递数据不可靠的原因，特别是使用routing的时候。Services are designed to be the glue between controllers, the minions of data, the slaves of functionality, the worker-bees of our application（就是说services在应用的controllers、 方法、数据之前起到了很关键的作用）

现在我们开始看怎么**创建service**。每个方法我们都会看到下面两个一样的参数：

* name-我们要定义的service的名字
* function-service方法

他们都创建了相同的底层对象类型。实例化后，他们都创建了一个service，这些对象没有什么功能上的差别。

## ****1、factory()****

Angular里面创建service最简单的方式是使用**factory()**方法。

factory()让我们通过返回一个包含service方法和数据的对象来定义一个service。在service方法里面我们可以注入services，比如 $http 和 $q等。

[?](http://my.oschina.net/tanweijie/blog/295067)

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9  10  11  12  13  14  15  16  17 | angular.module('myApp.services')  .factory('User', function($http) { // injectables go here    var backendUrl = "http://localhost:3000";  var service = {    // our factory definition      user: {},      setName: function(newName) {        service.user['name'] = newName;      },      setEmail: function(newEmail) {        service.user['email'] = newEmail;      },      save: function() {        return $http.post(backendUrl + '/users', {          user: service.user        });      }    };  return service;  }); |

* **在应用里面使用factory()方法**

在应用里面可以很容易地使用factory ，需要到的时候简单地注入就可以了

[?](http://my.oschina.net/tanweijie/blog/295067)

|  |  |
| --- | --- |
| 1  2  3  4 | angular.module('myApp')  .controller('MainCtrl', function($scope, User) {    $scope.saveUser = User.save;  }); |

* **什么时候使用factory()方法**

在service里面当我们仅仅需要的是一个方法和数据的集合且不需要处理复杂的逻辑的时候，factory()是一个非常不错的选择。

**注意：需要使用.config()来配置service的时候不能使用factory()方法**

## ****2、service()****

service()通过构造函数的方式让我们创建service，我们可以使用原型模式替代javaScript原始的对象来定义service。

和factory()方法一样我们也可以在函数的定义里面看到服务的注入

[?](http://my.oschina.net/tanweijie/blog/295067)

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9  10  11  12  13  14  15  16  17 | angular.module('myApp.services')  .service('User', function($http) { // injectables go here    var self = this; // Save reference    this.user = {};    this.backendUrl = "http://localhost:3000";    this.setName = function(newName) {      self.user['name'] = newName;    }    this.setEmail = function(newEmail) {      self.user['email'] = newEmail;    }    this.save = function() {      return $http.post(self.backendUrl + '/users', {        user: self.user      })    }  }); |

这里的功能和使用factory()方法的方式一样，service()方法会持有构造函数创建的对象。

* **在应用里面使用service()方法**

[?](http://my.oschina.net/tanweijie/blog/295067)

|  |  |
| --- | --- |
| 1  2  3  4 | angular.module('myApp')  .controller('MainCtrl', function($scope, User) {    $scope.saveUser = User.save;  }); |

* **什么时候适合使用service()方法**

service()方法很适合使用在功能控制比较多的service里面

**注意：需要使用.config()来配置service的时候不能使用service()方法**

## ****3、provider()****

provider()是创建service最底层的方式，这也是唯一一个可以使用.config()方法配置创建service的方法

不像上面提到的方法那样，我们在定义的this.$get()方法里面进行依赖注入

[?](http://my.oschina.net/tanweijie/blog/295067)

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9  10  11  12  13  14  15  16  17  18  19  20  21  22  23  24  25 | angular.module('myApp.services')  .provider('User', function() {    this.backendUrl = "http://localhost:3000";    this.setBackendUrl = function(newUrl) {      if (url) this.backendUrl = newUrl;    }    this.$get = function($http) { // injectables go here      var self = this;      var service = {        user: {},        setName: function(newName) {          service.user['name'] = newName;        },        setEmail: function(newEmail) {          service.user['email'] = newEmail;        },        save: function() {          return $http.post(self.backendUrl + '/users', {            user: service.user          })        }      };      return service;    }  }); |

* **在应用里面使用provider()方法**

为了给service进行配置，我们可以将provider注入到.config()方法里面

[?](http://my.oschina.net/tanweijie/blog/295067)

|  |  |
| --- | --- |
| 1  2  3  4 | angular.module('myApp')  .config(function(UserProvider) {    UserProvider.setBackendUrl("http://myApiBackend.com/api");  }) |

这样我们就可以和其他方式一样在应用里面使用这个service了

[?](http://my.oschina.net/tanweijie/blog/295067)

|  |  |
| --- | --- |
| 1  2  3  4 | angular.module('myApp')  .controller('MainCtrl', function($scope, User) {    $scope.saveUser = User.save;  }); |

* **什么时候使用provider()方法**

1. 当我们希望在应用开始前对service进行配置的时候就需要使用到provider()。比如，我们需要配置services在不同的部署环境里面（开发，演示，生产）使用不同的后端处理的时候就可以使用到了
2. 当我们打算发布开源provider()也是首选创建service的方法，这样就可以使用配置的方式来配置services而不是将配置数据硬编码写到代码里面。

**点击查看完整的代码：**[**https://gist.github.com/auser/7743235**](https://gist.github.com/auser/7743235)