**The difference between Service, Provider, and Factory in AngularJS**

If you are searching for this it’s probably because you are trying to figure out which one is the right one for you to use. Or because you’ve come across the three of them and are trying to determine the difference because they seem similar.If you think they are similar – you’re right. They are very similar. In fact, they are all the the same thing.They are all providers. The factory and the service are just special cases of the provider, but you can accomplish everything you want using just provider. I’ll show you.

#### The Provider

We’re going to create a provider that returns a value and simply display that value, you would do this:

<body ng-app="MyModule">

<div ng-controller="MyController"></div>

</body>

var mod = angular.module("MyModule", []);

mod.provider("myProvider", function() {

this.$get = function() {

return "My Value";

};

});

mod.controller("MyController", function(myProvider) {

console.log("MyController - myProvider: " + myProvider);

});

**CONSOLE OUTPUT**

MyController - myProvider: My Value

There, so a “provider” at the core let’s you “provide” a value. That value could be anything. In this case it’s a string with a value of “My Value” but it could easily have been a function or an object.

Note in further code samples I’m going to exclude the <body> tag and the definition of the mod for the purpose of keeping the code excerpts short and to the point.

#### Angular only gets the value once – ever

Note that angular only “gets” the value once, no matter how many times the provider is injected. That means it calls $get() only once ever, stores the value provided by $get(), and gives you that same stored value every time.

To show you what I mean I’ll create another controller and inject the provider again with a console statement so that you can see what’s happening.

<div ng-controller="MyController"></div>

<div ng-controller="MyController2"></div>

mod.provider("myProvider", function() {

this.$get = function() {

console.log("MyProviderFunction.$get() called."); **// ADDED this line**

return "My Value";

};

});

mod.controller("MyController", function(myProvider) {

console.log("MyController - myProvider: " + myProvider);

});

mod.controller("MyController2", function(myProvider) { **// ADDED this controller**

console.log("MyController2 - myProvider: " + myProvider);

});

**CONSOLE OUTPUT**

MyProviderFunction.$get() called.

MyController - myProvider: My Value

MyController2 - myProvider: My Value

As you can see the $get() function was only called once.

Note that we wrote a bunch of code for the provider only for the purpose of creating a method called $get(). Why not instead of giving angular a function that defines another function, why not just give it the function we want to run directly instead? Well you can, that’s what Angular calls afactory.

#### A Factory

With a factory you just provide the function body for the $get method and Angular does the rest. Below is what the new code looks like, as you will see it behaves exactly the same.

mod.**factory**("myProvider", function() { **// CHANGED “provider" to “factory"**

console.log("Factory function called.");

return "My Value";

});

mod.controller("MyController", function(myProvider) {

console.log("MyController - myProvider: " + myProvider);

});

mod.controller("MyController2", function(myProvider) {

console.log("MyController2 - myProvider: " + myProvider);

});

**CONSOLE OUTPUT**

Factory function called.

MyController - myProvider: My Value

MyController2 - myProvider: My Value

Now you might be wondering why would you ever use a provider if you can accomplish the same thing with a factory with less code. There are a couple of reasons and I’ll go into that later, right now I want to stay true to the title of this post and address the difference between these two (provider and factory) and a service.

So far we’ve returned a simple string value, but in practice what we probably want to return most of the time is an object. Well that wouldn’t change our example very much we can very easily swap out the string we’re returning with an object instead.

For example let’s do that by returning an object that contains a function called getValue(). Now there are several ways you can create an object in JavaScript, we’re going to use the “Object Constructor” approach where we create a function that populates an object with properties and functions and uses the new keyword to instantiate it.

function MyObject() { **// ADDED our object constructor**

this.getValue = function() {

return "My Value";

};

}

mod.factory("myProvider", function() {

console.log("Factory function called.");

return new MyObject(); **// CREATE an instance of our object**

});

mod.controller("MyController", function(myProvider) {

console.log("MyController - myProvider: " + myProvider**.getValue()**); **// CHANGED to call getValue()**

});

mod.controller("MyController2", function(myProvider) {

console.log("MyController2 - myProvider: " + myProvider**.getValue()**); **// CHANGED to call getValue()**

});

**CONSOLE OUTPUT**

Factory function called.

MyController - myProvider: My Value

MyController2 - myProvider: My Value

[Open in JS Fiddle](http://jsfiddle.net/luisperezphd/onqngmvk/)

Now I want to make one small tweak to this because it will lead nicely into the next concept. In our example we create the “Object Constructor” function MyObject(), but since we’re only instantiating it in one place we can use an anonymous function instead.

This is a very small tweak. Instead of this:

function MyObject() {

this.getValue = function() {

return "My Value";

};

}

mod.factory("myProvider", function() {

console.log("Factory function called.");

return new MyObject();

});

We do this:

mod.factory("myProvider", function() {

console.log("Factory function called.");

return new function() { **// INLINED our object constructor**

this.getValue = function() {

return "My Value";

};

};

});

So the whole thing now looks like this:

mod.factory("myProvider", function() {

console.log("Factory function called.");

return new function() { **// INLINED our object constructor**

this.getValue = function() {

return "My Value";

};

};

});

mod.controller("MyController", function(myProvider) {

console.log("MyController - myProvider: " + myProvider.getValue());

});

mod.controller("MyController2", function(myProvider) {

console.log("MyController2 - myProvider: " + myProvider.getValue());

});

[Open in JS Fiddle](http://jsfiddle.net/luisperezphd/x70pnaja/)

Now since our whole factory is made up of a single object, wouldn’t it be nice if we could just give Angular the object constructor function instead of having to write that funky looking factory. Well you are in luck, this is exactly what a service is.

#### At your Service

Here is that same code except using a service instead of a factory.

mod.**service**("myProvider", function() { **// CHANGED "factory" to "service"**

**// NOTE that the only function being passed is the object constructor from before**

this.getValue = function() {

return "My Value";

};

});

mod.controller("MyController", function(myProvider) {

console.log("MyController - myProvider: " + myProvider.getValue());

});

mod.controller("MyController2", function(myProvider) {

console.log("MyController2 - myProvider: " + myProvider.getValue());

});

**CONSOLE OUTPUT**

MyController - myProvider: My Value

MyController2 - myProvider: My Value

#### Provider vs Factory vs Service

So in summary, provider, factory, and service are all providers. A factory is a special case of a provider when all you need in your provider is a $get() function. It allows you to write it with less code. A service is a special case of a factory when you want to return an instance of a new object, with the same benefit of writing less code.

#### When to use one versus the other?

The answer is you use the most specialize version that accomplishes your goal. Say for example you are returning an existing object defined somewhere else that takes constructor arguments. You can’t pass arguments to the service, so you would make the call with a factory instead.

mod.factory("myProvider", function() {

console.log("Factory function called.");

return new SomeMessageBoxClass("custom argument");

});

One of the main factors of deciding between a provider and factory is whether you want to be able to configure the object that is generated before it’s generated. You do this by calling module.config() and getting an instance to the provider itself (instead of the object returned by the provider). You do this by appending “Provider” to the end of your provider’s name when you are injecting it.

Here is an example of how you would do that:

mod.provider("myProvider", function() {

this.value = "My Value";

this.setValue = function(newValue) {

this.value = newValue;

};

this.$get = function() {

return this.value;

};

});

mod.controller("MyController", function(myProvider) {

console.log("MyController - myProvider: " + myProvider);

});

mod.config(function(myProvider**Provider**) { **// ADDED config section**

**// Note the extra "Provider" suffix**

myProviderProvider.setValue("New Value");

});

That covers when to use the three providers: provider, factory, and service. There is one additional provider that was not mentioned here that is yet another special case and that’s the value provider.

If you remember when we first introduced the factory provider above we gave the simple example of returning a string value. That looked like this:

mod.factory("myProvider", function() {

return "My Value";

});

Well we could have actually done that using the value provider instead, again the benefit being that you can do it in less code. The code below does the same thing as the code above:

mod.value("myProvider", "My Value");

So when would you use one versus the other? Presumably you would use the factory provider when you want to calculate the value based on some other data, for example data from another value provider or an external source. And/or when you want to calculate the value if and only when it’s first requested. Here are some examples:

// Example where factory depends on a "value" provider

mod.value("multiple", 3);

mod.factory("value", function(multiple) {

return 10 \* multiple;

});

// Example where factory depends on external data

mod.factory("value", function(multiple) {

var multiple = getDateFromExternalPage();

return 10 \* multiple;

});

Did I imply that value was the only other provider? Well I lied, there is another that is very similar to value with two minor differences. That provider is called constant.

The difference between value and constant is that a value specified using constant is available during the configuration phase. You might remember from earlier that I mentioned that provider was accessible from the configuration phase, but service and factory were not.

Well it’s the same for value and constant. constant is available from the configuration phase and value is not. The other difference is as the name implies you can’t change the value of a constant. The first value you assign it is the value it keeps, if you try to assign it a different value later it will be ignored.

Here’s an example:

mod.value("myValue", "First Assignment");

mod.value("myValue", "Second Assignment");

mod.constant("myConstant", "First Assignment");

mod.constant("myConstant", "Second Assignment");

mod.controller("MyController", function(myValue, myConstant) {

console.log("myValue: " + myValue);

console.log("myConstant: " + myConstant);

});

**CONSOLE OUTPUT**

myValue: Second Assignment

myConstant: First Assignment