## AngularJS 之 Factory vs Service vs Provider

11 顶

英文原文：AngularJS: Factory vs Service vs Provider
标签： AngularJS
oschina 推荐于 1年前 (共 15 段, 翻译完成于 05-06) (26评)

143人收藏此文章, 我要收藏

参与翻译(4人)： 中奖啦, 戴仓薯, captain-cao, 赵亮-碧海情天

仅中文 | 中英文对照 | 仅英文 | 打印此文章

当你初试 Angular 时，很自然地就会往 controller 和 scope 里堆满不必要的逻辑。一定要早点意识到，controller 这一层应该很薄；也就是说，应用里大部分的业务逻辑和持久化数据都应该放在 service 里。我每天都会在 Stack Overflow 上看到几个同类的问题，关于如何在 controller 里保存持久化数据。这就不是 controller 该干的事。出于内存性能的考虑，controller 只在需要的时候才会初始化，一旦不需要就会被抛弃。因此，每次当你切换或刷新页面的时候，Angular 会清空当前的 controller。与此同时，service 可以用来永久保存应用的数据，并且这些数据可以在不同的 controller 之间使用。

Angular 提供了3种方法来创建并注册我们自己的 service。

1. Factory
2. Service
3. Provider

翻译的不错哦!

戴仓薯 1年前
顶 翻译的不错哦!

1 人顶

如果你是“太长的不看”

1) 用 Factory 就是创建一个对象，为它添加属性，然后把这个对象返回出来。你把 service 传进 controller 之后，在 controller 里这个对象里的属性就可以通过 factory 使用了。

翻译的不错哦!

```
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;
});
```

戴仓薯
顶 于 1年前

0 人顶

2) Service 是用"new"关键字实例化的。因此，你应该给"this"添加属性，然后 service 返回"this"。你把 service 传进 controller 之后，在controller里 "this" 上的属性就可以通过 service 来使用了。

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

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

3) Providers 是唯一一种你可以传进 .config() 函数的 service。当你想要在 service 对象启用之前，先进行模块范围的配置，那就应该用 provider。

```
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()';
});
```

详细解释（对于不是“太长不看”的读者）

为了准确表现出 Factory、Service 和 Provider 之间的差别，下面我们用 3 种不同的方式来构建同一个服务。这个服务会用到 iTunes API 以及使用 $q 的 promise。

1) Factory

Factory 是创建和配置服务最常见的方式。除了“快速浏览”之外，其实没有什么要补充的。只需创建一个对象，为它添加属性，然后返回这个对象就可以了。当你把 factory 传进 controller 中，对象的这些属性就可以通过 factory 访问。更详细的例子如下：

首先创建一个对象，然后返回这个对象，如下。

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

现在如果我们把"myFactory"传进 controller 里，附加在 "service" 上的任何属性都可以访问到了。

翻译的不

戴仓薯
顶 于 1年前
错哦！

0 人顶

现在让我们向回调函数中添加一些“private” 变量。当然 controller中是无法直接访问这些变量的，不过我们最终还是会在“service”中设置setter和个getter 方法，以便必要时修改这些“private”变量。

翻译的不

captain-cao
顶 于 1年前
错哦！

0 人顶

```
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;
});
```

你可能注意到了，我们没有将变量/函数加到“service”中。我们只是简单的创建他们以便之后的使用和修改。

- baseUrl 是iTunes API要求的根URL
- _artist 是我们想要查找的艺术家
- _finalUrl 是最终的权限定URL，即我们调用iTunes的入口
- makeUrl 是一个创建并返回友好的iTunesURL的函数

既然我们的帮手/私有变量和函数放在的合适的位置，那么让我们向“service”对象中添加一些属性。无论我们向”service“中添加什么， 我们都能在任意一个我们传递进‘myFactory’ 的controller中使用。

翻译的不

captain-cao 丢 1年前
顶 错哦!

0 人顶

我们来创建setArtist和getArtist方法来简单的返回或设置artist。同样创建一个方法使用我们创建的URL来调用iTunes API。这个方法将返回一个从iTunes ＡＰＩ获取数据后便会满足的ｐｒｏｍｉｓｅ。如果你对Ａｎｇｕｌａｒ的ｐｒｏｍｉｓｅ接触不多，我强烈推荐你深入的学习一下它。

- setArtist 接受一个ａｒｔｉｓｔ并且允许你设置ａｒｔｉｓｔ
- getArtist 返回ａｒｔｉｓｔ
- callItunes 首先调用makeUrl()方法以便构建$http请求使用的URL。然后它会设置promise对象，让$http请求我们最终的URL， 再然后呢，因为$http返回一个promise，所以我们可以在请求后调用.success或.error。最后我们可以通过iTunes的数据来解析我们的promise，或者直接‘There was an error’来拒绝它。

```
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;
});
```

现在我们的factory完成了。我们可以将"myFactory"注入到任意controller中了，然后就可以调用我们添加到service对象中的方法了（setArtist，getArtist，和callItunes）。

翻译的不错哦!

```
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);
      })
  }
});
```

captain-cao 于 1年前
顶
1 人顶

在上面的controller中，我们注入了‘myFactory’ service对象。然后我们设置$scope 对象的属性。上面唯一棘手的代码是处理promise。因为callItunes返回一个promise对象，一旦我们的promise满足了，我们可以调用.then()方法以及设置$scope.data.artistData。你会注意到我们的controller是非常的“瘦”。因为我们所有的逻辑和持久化数据都存放在了service中而不是controller中。

2) Service

当我们创建一个Service时，我们所知道的最重要事可能就是Service通过new关键字实例化对象。这应该可以使熟悉JavaScript的人了解到了这段代码的作用。但对于那些JS背景有限，或者不太熟悉new关键字的作用的人来说可能有点困难。那就让我们来重温一下JavaScript的基本功能，以便帮助我们了解Service究竟做了什么。

让我们先定义一个函数，然后通过new关键字来调用它，看看当解释器遇到了new关键字的时候做了些什么工作，以便帮助我们了解使用new关键字来实例化一个函数时究竟有什么变化。这个的最终结果应该和Service是一样的。

翻译的不

中奖啦
顶 吾峨! 1年前
1 人顶

首先，让我们定义一个构造器。

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

这个一个典型的JavaScript式的构造方法。现在，只要我们使用new关键字来调用Person函数，就会将’this’关键字绑定到新创建的对象上。

接下来，让我们给Person的prototype对象添加一个方法，这个方法对所有Person ‘类’的实例都是可用的。

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

现在，由于我们往prototype上添加了一个sayName方法，所以所有的Person实例都可以调用这个方法，并且输出对应实例的name值。

既然我们已经有了一个Person的构造器，并在在其prototype上定义了一个sayName方法，那就让我们去创建一个Person的实例，并调用这个sayName方法。

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

接下来，我们把创建Person构造器、往其prototype上添加方法、创建一个Person实例，并调用sayName方法的代码写在一块，如下所示：

```
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'
```

翻译的不

中奖啦
顶 吾峨! 1年前
0 人顶

现在，让我们看一下当我们在JavaScript中使用new关键字的时候究竟发生了什么。首先你应该已经注意到的是，当我们在例子中使用了new关键字之后，我们可以通过’tyler’来调用方法（sayName），看上去好像tyler是一个对象——那是因为

翻译的不

它确实成了一个对象。所以，我们知道的第一件事就是我们的Person构造器返回了一个对象(object)。其次，我们知道，由于我们的sayName方法是定义在Person的prototype上，而不是直接定义在Person的实例上的，所以Person函数返回的对象（tyler）一定是由于未找到sayName方法，进而去prototype寻找sayName方法的。用更通俗的话来说，当我们调用tyler.sayName()时，JS解释器说，“好吧，我先去我们刚创建的'tyler'对象上查找sayName方法，然后调用它。等一下，我没有在它上面找到sayName方法——我只看到了name和age，那让我去prototype找一下吧。没错，它在prototype上，那就让我调用它吧”。

下面的代码演示了在JavaScript中使用new关键之后所做的事。它是上面这一段文字的一个基本的代码示例。我已经把从JS解释器的角度来看整个过程的代码写在了注释里。

```
var Person = function(name, age){
  //Line 4 creates an obj object that will
  //var obj = Object.create(Person.prototyp

  //Line 7 sets 'this' to the newly created
  //this = obj;

  this.name = name;
  this.age = age;

  //return this;
}
```

现在，既然我们了解了在JavaScript中new关键字是如何工作的，那么在Angular中创建一个Service也应该变得容易理解了。

在创建一个Service时，需要理解的最重要的一件事就是我们使用new关键字去实例化Service对象。结合我们从上面的例子所了解到的知识，你应该已经意识到你可以将一些属性和方法直接添加到this上，之后，在创建Service对象时，this会被作为返回值返回。让我们来看一下这种工作方式。

我们不用像之前Factory中的例子那样创建一个对象，然后返回这个对象。因为我们使用了new关键字来调用，解释器会创建一个对象，并关联它的prototype对象，然后将该对象返回，而不用我们去做这些工作。

首先，让我们创建我们的私有辅助函数。它应该看起来和我们在factory中所作的工作很类似。由于我已经在factory的例子中解释过每一行代码的含义了，所以我不会在这里多作解释，如有疑惑，请再次回味一下factory的例子。

```
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;
  }
});
```

接下来，我们将要把可以从控制器中访问的方法添加到‘this’上。

```
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;
  }

});
```

现在，和使用factory一样，所有将myService作为参数传入的控制器都可以访问到setArtist, getArtist, 和callItunes方法。下面是传入了myService的控制器（基本上和factory的控制器一样）。

```
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);
      })
  }
});
```

正如我之前提到的那样，一旦你了解了new关键字的作用，你就会知道在Angular中，Services和Factories几乎一样。

3) Provider

要记住的关于Provider的最重要的事情是，它们是你可以传递到应用程序的app.config部分唯一的服务。如果你需要在你的服务对象可以在你的应用程序之外任何地方都可用之前改变它的某些部分，这是非常重要的。虽然Services/Factories很相似，但也有一些差异，我们将会讨论它们。

首先，我们用与我们建立Service 和 Factory类似的方式来建立我们的Provider。下面的变量是我们的'私人'和辅助功能。

翻译的不

赵亮-碧海情天 1年前
顶 碧哦!

0 人顶

```
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
  var thingFromConfig = '';

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

*同样地，如果上面的代码的任何部分令你纠结，请看下 Factory 部分，在那里我更详细地解释了这些代码的作用。

必须要注意的一点是只有这些变量和函数是可以在我们的app.config函数中访问的。这曾一度使我感到困惑，所以你最好也要知道这点不同之处。你可以把Provider想象成由两部分组成。第一部分的变量和函数是可以在app.config函数中访问的，因此你可以在它们被其他地方访问到之前来修改它们（如上所示）。第二部分（如下所示） 的变量和函数是可以在任何传入了’myProvider‘的控制器中进行访问的。

当你使用Provider创建一个service时，唯一的可以在你的控制器中访问的属性和方法是通过$get()函数返回内容。下面的代码将$get方法写在了’this‘（最终会被函数返回）上。现在，$get函数会返回所有我们希望在控制器中进行访问的方法和属性。下面是代码示例：

```
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: thingFromConfig
  }
}
```

现在，Provider的完整代码如下所示：

```
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
  var 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: thingFromConfig
    }
  }
});
```

现在，与我们的Factory和Service类似，setArtist, getArtist, 和callItunes可以在任何一个传入了 myProvider 的控制器中访问。下面是myProvider的控制器（几乎和我们Factory/Service中的控制器一样）。

```
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;
});
```

正如前面提到的那样，使用Provider创建一个service的独特之处是，你可以在Provider对象传递到应用程序的其他部分之前在app.config函数对其进行修改。让我们来看一个对应的例子。

```
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. P
    passed into config. Check out the code to see how it works';
});
```

现在你可以明白‘thingFromConfig’是怎么样地在我们的provider中是空字符串，而当它出现在DOM中时，它将是’This sentence was set…’。

谢谢您的阅读，我希望这有助于你能辨别在Angular中Factory, Service, 和 Provider之间的差异。

翻译的不错哦!

*要查看完整的代码示例，看看运行中的代码，可以自由地fork我的repo：https://github.com/tylermcginnis33/AngularService

赵亮-碧海情天 于 1年前

顶 0 人顶





网友评论 共26条 发表评论 回页面顶部

景愿 发表于 2014-05-09 08:36
Only for single page application!

开源中国匿名会员 发表于 2014-05-09 09:22
js是有够烦的。

小鲅鱼 发表于 2014-05-09 10:16
> 引用来自“开源中国匿名会员”的评论
>
> js是有够烦的。

ng给人感觉比较好用～～特别是在web应用中～～～

布洛克斯 发表于 2014-05-09 10:19
为什么是图片啊

opal 发表于 2014-05-09 10:23
本来很简单的东西，弄成这么复杂

傅小黑 发表于 2014-05-09 10:28
代码配色不错

小鲅鱼 发表于 2014-05-09 10:49
> 引用来自“opal”的评论
>
> 本来很简单的东西，弄成这么复杂

设计模式不就是这样的吗

justplaymore 发表于 2014-05-09 11:27
看看英文原文的评论，再看看这里的评论，呵呵。

elsonwu 发表于 2014-05-09 12:06

> 引用来自“opal”的评论
>
> 本来很简单的东西，弄成这么复杂

如果应付简单的需求确实复杂了，但如果原生js应付一个复杂的需求，很容易代码就乱得不得了

danceshow 发表于 2014-05-09 12:07

mark

sevendlong 发表于 2014-05-09 13:09

> 引用来自“opal”的评论
>
> 本来很简单的东西，弄成这么复杂

当你写过复杂的webapp，就会知道ng是多么好的东西

开源中国匿名会员 发表于 2014-05-09 15:14

> 引用来自“开源中国匿名会员”的评论
>
> js是有够烦的。
>
> 引用来自“银星”的评论
> ng给人感觉比较好用～～特别是在web应用中～～～

ng相比较起来是给散沙一盘的js聚拢了一下。 可惜还是乱七八糟的。模块化相当相当蛋头疼。 搞小的还行，搞大了，嗯，你真厉害。

Tu_Minglei 发表于 2014-05-09 15:32

有一个关键的细节被改变了：
Provider那段的贴图里，原本应该是

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

在这里变成了

```
• • •
// Going to set this property on the config function below
var thingFromConfig = '';
• • •
```

这里 this.thingFromConfig = '' 变成 var thingFromConfig = ''，差别是挺大的。

刘-冬-冬 发表于 2014-05-09 15:43

> 引用来自“opal”的评论
>
> 本来很简单的东西，弄成这么复杂
>
> 引用来自“寇德林”的评论
> 当你写过复杂的webapp，就会知道ng是多么好的东西

我赞同你的意见，如何合理的构建大型web应用，ng是很好的。当然了，Backbone和seajs也不错。

limichange 发表于 2014-05-09 17:20

== 你看看js里有多少代码是取值和赋值的

limichange 发表于 2014-05-09 17:21

> 引用来自“opal”的评论
>
> 本来很简单的东西，弄成这么复杂
>
> 引用来自“寇德林”的评论
> 当你写过复杂的webapp，就会知道ng是多么好的东西
>
> 引用来自“刘-冬-冬”的评论
> 我赞同你的意见，如何合理的构建大型web应用，ng是很好的。当然了，Backbone和seajs也不错。

angular商业化的飘过

jluflyingz 发表于 2014-05-09 17:42

> 引用来自“景愿”的评论
>
> Only for single page application!

Nope, SPA is optional in Angular.js if you don't use Angular Router,

jluflyingz 发表于 2014-05-09 17:44

> 引用来自“opal”的评论
>
> 本来很简单的东西，弄成这么复杂
>
> 引用来自“寇德林”的评论
> 当你写过复杂的webapp，就会知道ng是多么好的东西

赞， Angular.js的出现就是为了保证应用的简洁和可测试性，让你从繁重的DOM操作中抽身出来。其实我觉得这篇文章写得挺好的。 Service和Factory我也一度分不清楚，干脆直接用factory了. 现在对为什么要有Provider这个玩意也有了一些理解。

杯面柒 发表于 2014-05-09 18:11

mark， 看完好理解好多了

jQer 发表于 2014-05-09 21:28

现在你不只得到了一个拿着香蕉的猩猩，而且是一大堆拿着香蕉的猩猩，还有几只海龟和小岛。

发表评论

回评论顶部 | 回页面顶部