

[推酷](#)

- [文章](#)
- [站点](#)
- [主题](#)
- [公开课](#)
- [活动](#)
- [客户端](#) [荐](#)
- [周刊](#)
  - [编程狂人](#)
  - [设计匠艺](#)
  - [创业周刊](#)
  - [科技周刊](#)
  - [Guru Weekly](#)
  - [一周拾遗](#)

# AngularJS之Factory vs Service vs Provider

[登录](#)

时间 2014-05-09 08:20:52 [LinuxEden](#)

原文 <http://www.linuxeden.com/html/news/20140509/151538.html>

主题 [AngularJS Prototype](#)

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。我每天都会看到几个同类的问题，关于如何在 controller 里保存持久化数据。这可不是 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

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 放进 controller 之后，在 controller 里这个对象里的属性就可以通过 factory 使用了。

```

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) 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;
});
```

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

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 和一个 getter 方法，以便必要时修改这些“private”变量。

```
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 iTunes URL 的函数

既然我们的帮手/私有变量和函数放在合适的位置，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。

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)。

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

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。

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'
```

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。

```
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 }
```

现在，既然我们了解了在JavaScript中新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，如有疑问，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。

```
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：

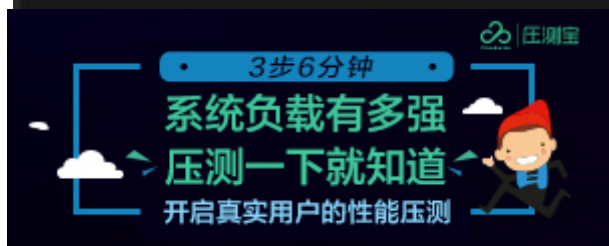
```

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;
  }

```



```

    getArtist: function(){
      return _artist;
    },
    thingOnConfig: thingFromConfig
  }
}
});

```

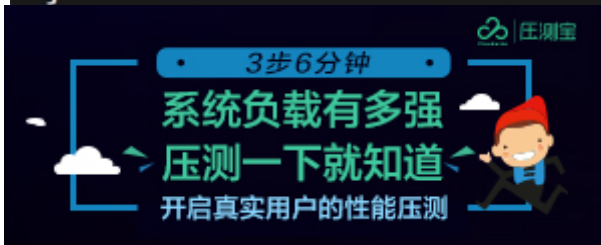
现在，与我们的Factory和Service类似，setArtist，getArtist，和callItunes可以在任何一个传入了 myProvider 的控制器中访问。下面是myProvider的控制器（几乎和我们Factory/Service中的控制器一样）。

```

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);
      })
  }
});

```



3步6分钟  
系统负载有多强  
压测一下就知道  
开启真实用户的性能压测

```
Provider.thingOnConfig;
```

一个service的独特之处是，你可以在Provider对象传递到对其进行修改。让我们来看一个对应的例子。



100offer  
高端招聘平台  
告别盲目投简历  
让海量好机会主动来找你  
更多详情

```

// ser name
app.config, Providers are the only service that can be

```

么样地在我们的provider中是空字符串，而当它出现在...’。

在Angular中Factory, Service, 和 Provider之间的差

码，可以自由地fork我的repo:  
[angularService](#)



分享

收藏

纠错



## 专访：听道哥聊互联网江湖

“根据判断，下一个被黑产盯上的行业很有可能是直播行业。现在来看，这件事情已经发生了。”

推荐文章

- 1. [Sparrow - 轻量移动前端框架](#)
- 2. [JavaScript进阶学习（三）—— 基于html5 File API的文件操作](#)
- 3. [OpenResty Web 开发框架 lor](#)
- 4. [深入理解BOM系列第三篇——定时器应用\(时钟、倒计时、秒表和闹钟\)](#)
- 5. [vue-slider 一个轻量的 vue 滑动组件](#)
- 6. [H5打造属于自己的视频播放器（JS篇1）](#)

相关推刊

```
= function(name, age){
  e = name;
  = age;

  prototype.sayName = function
  y name is ' + this.name);

= new Person('Tyler', 23)
ame(); //alerts 'My name
```

- [bv VoLi](#) [《AngularJS》](#) 2



[《Javascript Lib》](#) 16

- [bv flvinhighwi](#) [《angularis》](#) 4

我来评几句

请输入评论内容...

登录后评论

已发表评论数(1)



[qingbingyan](#)08-03 22:12  
写得好，解决了我的困惑

相关站点



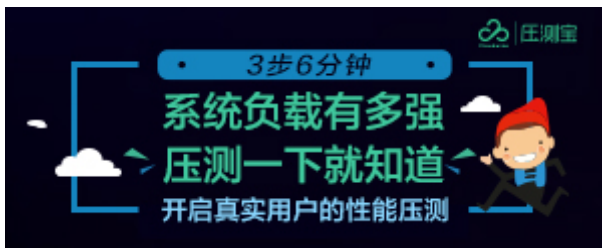


[LinuxEden](#)

+ 订阅

热门文章

- 1. [Sparrow - 轻量移动前端框架](#)
- 2. [JavaScript进阶学习（三）—— 基于html5 File API的文件操作](#)
- 3. [OpenResty Web 开发框架 lor](#)
- 4. [vue-slider 一个轻量的 vue 滑动组件](#)
- 5. [H5打造属于自己的视频播放器（JS篇1）](#)





收藏到推刊

[创建推刊](#)

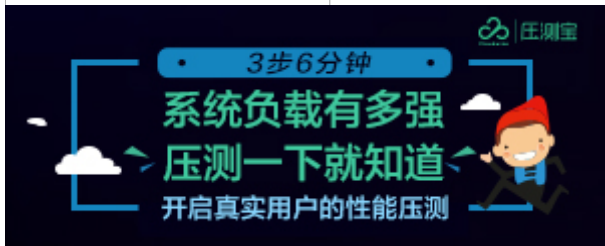
收藏

取消

推刊名(必填)

请填写推刊名

推刊描述



个字符!



提交



tuicool2012

友情链接

[人人都是产品经理](#) [PM256](#) [移动信息化](#) [行晓网](#) [智城外包网](#) [虎嗅](#) [IT耳朵](#) [创媒工场](#) [经理人分享](#) [市场部网](#) [砍柴网](#) [CocoaChina](#) [北风网](#) [云智慧](#) [我赢职场](#) [大数据时代](#) [奇笛网](#) [咕噜网](#) [红联linux](#) [Win10之家](#) [鸟哥笔记](#) [爱游戏](#) [投资潮](#) [31会议网](#) [极光推送](#) [Teambition](#) [硅谷网](#) [leangoo](#) [ZEALER中国](#) [OpenSNS](#) [小牛学堂](#) [handone](#) [Scrum中文网](#) [比戈大生](#) [又拍云](#) [更多链接>>](#)