

[AngularJS 教程](#)[AngularJS 简介](#)[AngularJS 表达式](#)[AngularJS 指令](#)[AngularJS 模型](#)[AngularJS Scope\(作用域\)](#)[AngularJS 控制器](#)[AngularJS 过滤器](#)[AngularJS Service](#)[AngularJS Http](#)[AngularJS Select](#)[AngularJS 表格](#)[AngularJS SQL](#)[AngularJS HTML DOM](#)[AngularJS 事件](#)[AngularJS 模块](#)[AngularJS 表单](#)[AngularJS 输入验证](#)[AngularJS API](#)[AngularJS Bootstrap](#)[← AngularJS 动画](#)[AngularJS 路由 →](#)

AngularJS 依赖注入

什么是依赖注入

wiki 上的解释是：依赖注入（Dependency Injection，简称DI）是一种软件设计模式，在这种模式下，一个或更多的依赖（或服务）被注入（或者通过引用传递）到一个独立的对象（或客户端）中，然后成为了该客户端状态的一部分。

该模式分离了客户端依赖本身行为的创建，这使得程序设计变得松耦合，并遵循了依赖反转和单一职责原则。与服务定位器模式形成直接对比的是，它允许客户端了解客户端如何使用该系统找到依赖

一句话 --- 没事你不要来找我，有事我会去找你。

AngularJS 提供很好的依赖注入机制。以下5个核心组件用来作为依赖注入：

- value
- factory
- service
- provider
- constant

value

Value 是一个简单的 javascript 对象，用于向控制器传递值（配置阶段）：

[AngularJS 包含](#)[AngularJS 动画](#)[AngularJS 依赖注入](#)[AngularJS 路由](#)[AngularJS 应用](#)

AngularJS 实例

[AngularJS 实例](#)

AngularJS 参考手册

[AngularJS 参考手册](#)

```
// 定义一个模块
var mainApp = angular.module("mainApp", []);

// 创建 value 对象 "defaultInput" 并传递数据
mainApp.value("defaultInput", 5);
...

// 将 "defaultInput" 注入到控制器
mainApp.controller('CalcController', function($scope, CalcService, defaultInput) {
    $scope.number = defaultInput;
    $scope.result = CalcService.square($scope.number);

    $scope.square = function() {
        $scope.result = CalcService.square($scope.number);
    }
});
```

factory

factory 是一个函数用于返回值。在 service 和 controller 需要时创建。

通常我们使用 factory 函数来计算或返回值。

```
// 定义一个模块
var mainApp = angular.module("mainApp", []);

// 创建 factory "MathService" 用于两数的乘积 provides a method multiply to return multiplication of two numbers
mainApp.factory('MathService', function() {
    var factory = {};

    factory.multiply = function(a, b) {
        return a * b
    }

    return factory;
});
```

```
// 在 service 中注入 factory "MathService"
mainApp.service('CalcService', function(MathService){
    this.square = function(a) {
        return MathService.multiply(a,a);
    }
});
...
```

provider

AngularJS 中通过 provider 创建一个 service、factory等(配置阶段)。

Provider 中提供了一个 factory 方法 get(), 它用于返回 value/service/factory。

```
// 定义一个模块
var mainApp = angular.module("mainApp", []);
...

// 使用 provider 创建 service 定义一个方法用于计算两数乘积
mainApp.config(function($provide) {
    $provide.provider('MathService', function() {
        this.$get = function() {
            var factory = {};

            factory.multiply = function(a, b) {
                return a * b;
            }

            return factory;
        };
    });
});
```

constant

constant(常量)用来在配置阶段传递数值，注意这个常量在配置阶段是不可用的。

```
mainApp.constant("configParam", "constant value");
```

实例

以下实例提供了以上几个依赖注入机制的演示。

AngularJS 实例 - factory

```
var mainApp = angular.module("mainApp", []);
mainApp.value("defaultInput", 5);

mainApp.factory('MathService', function() {
    var factory = {};

    factory.multiply = function(a, b) {
        return a * b;
    }
    return factory;
});

mainApp.service('CalcService', function(MathService){
    this.square = function(a) {
        return MathService.multiply(a,a);
    }
});

mainApp.controller('CalcController', function($scope, CalcService, defaultInput) {
    $scope.number = defaultInput;
    $scope.result = CalcService.square($scope.number);

    $scope.square = function() {
        $scope.result = CalcService.square($scope.number);
    }
});
```

尝试一下 »

AngularJS 实例 - provider

```
var mainApp = angular.module("mainApp", []);

mainApp.config(function($provide) {
    $provide.provider('MathService', function() {
        this.$get = function() {
            var factory = {};

            factory.multiply = function(a, b) {
                return a * b;
            }
            return factory;
        };
    });
});

mainApp.value("defaultInput", 5);

mainApp.service('CalcService', function(MathService){
    this.square = function(a) {
        return MathService.multiply(a,a);
    }
});

mainApp.controller('CalcController', function($scope, CalcService, defaultInput) {
    $scope.number = defaultInput;
    $scope.result = CalcService.square($scope.number);

    $scope.square = function() {
        $scope.result = CalcService.square($scope.number);
    }
});
```

尝试一下 »



笔记列表



1. 一个对别人有依赖的东西，它想要单独测试，就需要在依赖项齐备的情况下进行。如果我们在运行时注入，就可以减少这种依赖

2. 参数由定义方决定

3. 与import还不完全一样

Joey 10个月前 (03-06)



factory的实例

```
<div ng-app='myApp' ng-controller='MathCtrl'>
  <input type="text" ng-model='leftNum'> x
  <input type="text" ng-model='rightNum'>
  =<span >{{multRes}}</span>
</div>
```

尝试一下 »

热爱前端 3个月前 (09-11)

点我分享笔记

笔记需要是本篇文章的内容扩展！

注册邀请码获取方式

