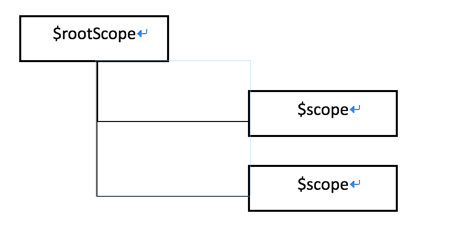
### 1. ng-show/ng-hide 与 ng-if的区别？

我们都知道ng-show/ng-hide实际上是通过display来进行隐藏和显示的。而ng-if实际上控制dom节点的增删除来实现的。因此如果我们是根据不同的条件来进行dom节点的加载的话，那么ng-if的性能好过ng-show.

### 2.解释下什么是$rootScrope以及和$scope的区别？

通俗的说$rootScrope 页面所有$scope的父亲。



我们来看下如何产生$rootScope和$scope吧。

step1:Angular解析ng-app然后在内存中创建$rootScope。

step2:angular回继续解析，找到{{}}表达式，并解析成变量。

step3:接着会解析带有ng-controller的div然后指向到某个controller函数。这个时候在这个controller函数变成一个$scope对象实例。

### 3. 表达式 {{yourModel}}是如何工作的？

它依赖于 $interpolation服务，在初始化页面html后，它会找到这些表达式，并且进行标记，于是每遇见一个{{}}，则会设置一个$watch。而$interpolation会返回一个带有上下文参数的函数，最后该函数执行，则算是表达式$parse到那个作用域上。

### 4. Angular中的digest周期是什么？

每个digest周期中，angular总会对比scope上model的值，一般digest周期都是自动触发的，我们也可以使用$apply进行手动触发。更深层次的研究，可以移步[The Digest Loop and apply](https://www.ng-book.com/p/The-Digest-Loop-and-apply/" \t "http://blog.csdn.net/donggx/article/details/_blank)。

### 5. 如何取消 $timeout, 以及停止一个$watch()?

停止 $timeout我们可以用cancel：

**var** customTimeout = $timeout(**function** () {

// your code

}, 1000);

$timeout.cancel(customTimeout);

停掉一个$watch：

// .$watch() 会返回一个停止注册的函数**function** **that** **we** **store** **to** **a** **variable** **var** **deregisterWatchFn** = **$rootScope**.**$watch**(‘someGloballyAvailableProperty’, function (newVal) {

**if** (newVal) {

// we invoke that deregistration function, to disable the watch

deregisterWatchFn();

...

}

});

### 6. Angular Directive中restrict 中分别可以怎样设置？scope中@,=,&有什么区别？

restrict中可以分别设置:

* A匹配属性
* E匹配标签
* C匹配class
* M 匹配注释

当然你可以设置多个值比如AEC,进行多个匹配。

在scope中，@,=,&在进行值绑定时分别表示

* @获取一个设置的字符串，它可以自己设置的也可以使用{{yourModel}}进行绑定的;
* = 双向绑定，绑定scope上的一些属性；
* & 用于执行父级scope上的一些表达式，常见我们设置一些需要执行的函数

angular.module('docsIsolationExample', [])

.controller('Controller', ['$scope', **function**($scope) {

$scope.alertName = **function**() {

alert('directive scope &');

}

}])

.directive('myCustomer', **function**() {

**return** {

restrict: 'E',

scope: {

clickHandle: '&'

},

template: '<button ng-click="testClick()">Click Me</button>',

controller: **function**($scope) {

$scope.testClick = **function**() {

$scope.clickHandle();

}

}

};

});

<**div** ng-app="docsIsolationExample"> <**div** ng-controller="Controller">

<**my-customer** click-handle="alertName()"></**my-customer**></**div**>

</**div**>

[Codepen Demo](http://codepen.io/Jack_Pu/pen/NrpRBK" \t "http://blog.csdn.net/donggx/article/details/_blank)

* < 进行单向绑定。

### 7. 列出至少三种实现不同模块之间通信方式？

Service

events,指定绑定的事件

使用 $rootScope

controller之间直接使用$parent, $$childHead等

directive 指定属性进行数据绑定

### 8. 有哪些措施可以改善Angular 性能

* 官方提倡的，关闭debug,$compileProvider

myApp.config(**function** ($compileProvider) {

$compileProvider.debugInfoEnabled(false);

});

使用一次绑定表达式即{{::yourModel}}

减少watcher数量

在无限滚动加载中避免使用ng-repeat,关于解决方法可以参考这篇[文章](http://www.williambrownstreet.net/blog/2013/07/angularjs-my-solution-to-the-ng-repeat-performance-problem/" \t "http://blog.csdn.net/donggx/article/details/_blank)

使用性能**[测试](http://lib.csdn.net/base/softwaretest" \o "软件测试知识库" \t "http://blog.csdn.net/donggx/article/details/_blank)**的小工具去挖掘你的angular性能问题，我们可以使用简单的console.time()也可以借助开发者工具以及[Batarang](https://chrome.google.com/webstore/detail/angularjs-batarang/ighdmehidhipcmcojjgiloacoafjmpfk?hl=en" \t "http://blog.csdn.net/donggx/article/details/_blank)

console.time("TimerName"); //your codeconsole.timeEnd("TimerName");

### 9. 你认为在Angular中使用jQuery好么？

这是一个开放性的问题，尽管网上会有很多这样的争论，但是普遍还是认为这并不是一个特别好的尝试。其实当我们学习Angular的时候，我们应该做到从0去接受angular的思想，数据绑定，使用angular自带的一些api，合理的路由组织和，写相关指令和服务等等。angular自带了很多api可以完全替代掉**[jQuery](http://lib.csdn.net/base/jquery" \o "jQuery知识库" \t "http://blog.csdn.net/donggx/article/details/_blank)**中常用的api，我们可以使用angular.element，$http,$timeout,ng-init等。

我们不妨再换个角度，如果业务需求，而对于一个新人（比较熟悉**[jQuery](http://lib.csdn.net/base/jquery" \o "jQuery知识库" \t "http://blog.csdn.net/donggx/article/details/_blank)**）的话，或许你引入jQuery可以让它在解决问题，比如使用插件上有更多的选择，当然这是通过影响代码组织来提高工作效率，随着对于angular理解的深入，在重构时会逐渐摒弃掉当初引入jquery时的一些代码。

所以我觉得两种框架说完全不能一起用肯定是错的，但是我们还是应该尽力去**遵循angular的设计**。

### 10. 如何进行angular的单元测试

我们可以使用karam＋jasmine 进行单元**[测试](http://lib.csdn.net/base/softwaretest" \o "软件测试知识库" \t "http://blog.csdn.net/donggx/article/details/_blank)**,我们通过ngMock引入angular app然后自行添加我们的测试用例。 一段简单的测试代码：

describe('calculator', **function** () {

beforeEach(module('calculatorApp'));

**var** $controller;

beforeEach(inject(**function**(\_$controller\_){

$controller = \_$controller\_;

}));

describe('sum', **function** () {

it('1 + 1 should equal 2', **function** () {

**var** $scope = {};

**var** controller = $controller('CalculatorController', { $scope: $scope });

$scope.x = 1;

$scope.y = 2;

$scope.sum();

expect($scope.z).toBe(3);

});

});

});

### ng-if 跟 ng-show/hide 的区别有哪些？

第一点区别是，ng-if 在后面表达式为 true 的时候才创建这个 dom 节点，ng-show 是初始时就创建了，用 display:block 和 display:none 来控制显示和不显示。

第二点区别是，ng-if 会（隐式地）产生新作用域，ng-switch 、 ng-include 等会动态创建一块界面的也是如此。

这样会导致，在 ng-if 中用基本变量绑定 ng-model，并在外层 div 中把此 model 绑定给另一个显示区域，内层改变时，外层不会同步改变，因为此时已经是两个变量了。

<p>{{name}}</p><div ng-if="true">

<input type="text" ng-model="name"></div>

ng-show 不存在此问题，因为它不自带一级作用域。

避免这类问题出现的办法是，始终将页面中的元素绑定到对象的属性（data.x）而不是直接绑定到基本变量（x）上。

详见 [AngularJS 中的作用域](http://huangtengfei.com/2015/09/scope-in-angularjs/" \t "https://segmentfault.com/a/_blank)

### ng-repeat迭代数组的时候，如果数组中有相同值，会有什么问题，如何解决？

会提示 Duplicates in a repeater are not allowed. 加 track by $index 可解决。当然，也可以 trace by 任何一个普通的值，只要能唯一性标识数组中的每一项即可（建立 dom 和数据之间的关联）。

### ng-click 中写的表达式，能使用 JS 原生对象上的方法吗？

不止是 ng-click 中的表达式，只要是在页面中，都不能直接调用原生的 JS 方法，因为这些并不存在于与页面对应的 Controller 的 $scope 中。

举个栗子：

<p>{{parseInt(55.66)}}<p>

会发现，什么也没有显示。

但如果在 $scope 中添加了这个函数：

$scope.parseInt = function(x){

return parseInt(x);

}

这样自然是没什么问题了。

对于这种需求，使用一个 filter 或许是不错的选择：

<p>{{13.14 | parseIntFilter}}</p>

app.filter('parseIntFilter', function(){

return function(item){

return parseInt(item);

}

})

### {{now | 'yyyy-MM-dd'}} 这种表达式里面，竖线和后面的参数通过什么方式可以自定义？

filter，格式化数据，接收一个输入，按某规则处理，返回处理结果。

##### 内置 filter

ng 内置的 filter 有九种：

date（日期）

currency（货币）

limitTo（限制数组或字符串长度）

orderBy（排序）

lowercase（小写）

uppercase（大写）

number（格式化数字，加上千位分隔符，并接收参数限定小数点位数）

filter（处理一个数组，过滤出含有某个子串的元素）

json（格式化 json 对象）

filter 有两种使用方法，一种是直接在页面里：

<p>{{now | date : 'yyyy-MM-dd'}}</p>

另一种是在 js 里面用：

// $filter('过滤器名称')(需要过滤的对象, 参数1, 参数2,...)

$filter('date')(now, 'yyyy-MM-dd hh:mm:ss');

##### 自定义 filter

// 形式

app.filter('过滤器名称',function(){

return function(需要过滤的对象,过滤器参数1,过滤器参数2,...){

//...做一些事情

return 处理后的对象;

}

});

// 栗子

app.filter('timesFilter', function(){

return function(item, times){

var result = '';

for(var i = 0; i < times; i++){

result += item;

}

return result;

}

})

### factory、service 和 provider 是什么关系？

##### factory

把 service 的方法和数据放在一个对象里，并返回这个对象

app.factory('FooService', function(){

return {

target: 'factory',

sayHello: function(){

return 'hello ' + this.target;

}

}

});

##### service

通过构造函数方式创建 service，返回一个实例化对象

app.service('FooService', function(){

var self = this;

this.target = 'service';

this.sayHello = function(){

return 'hello ' + self.target;

}

});

##### provider

创建一个可通过 config 配置的 service，$get 中返回的，就是用 factory 创建 service 的内容

app.provider('FooService', function(){

this.configData = 'init data';

this.setConfigData = function(data){

if(data){

this.configData = data;

}

}

this.$get = function(){

var self = this;

return {

target: 'provider',

sayHello: function(){

return self.configData + ' hello ' + this.target;

}

}

}

});

// 此处注入的是 FooService 的 provider

app.config(function(FooServiceProvider){

FooServiceProvider.setConfigData('config data');

});

从底层实现上来看，service 调用了 factory，返回其实例；factory 调用了 provider，返回其 $get 中定义的内容。factory 和 service 功能类似，只不过 factory 是普通 function，可以返回任何东西（return 的都可以被访问，所以那些私有变量怎么写，你懂的）；service 是构造器，可以不返回（绑定到 this 的都可以被访问）；provider 是加强版 factory，返回一个可配置的 factory。

详见 [AngularJS 之 Factory vs Service vs Provider](http://www.oschina.net/translate/angularjs-factory-vs-service-vs-provider" \t "https://segmentfault.com/a/_blank)

### angular 的数据绑定采用什么机制？详述原理

脏检查机制。

双向数据绑定是 AngularJS 的核心机制之一。当 view 中有任何数据变化时，会更新到 model ，当 model 中数据有变化时，view 也会同步更新，显然，这需要一个监控。

原理就是，Angular 在 scope 模型上设置了一个 监听队列，用来监听数据变化并更新 view 。每次绑定一个东西到 view 上时 AngularJS 就会往 $watch 队列里插入一条 $watch，用来检测它监视的 model 里是否有变化的东西。当浏览器接收到可以被 angular context 处理的事件时，$digest 循环就会触发，遍历所有的 $watch，最后更新 dom。

举个栗子

<button ng-click="val=val+1">increase 1</button>

click 时会产生一次更新的操作（至少触发两次 $digest 循环）

按下按钮

浏览器接收到一个事件，进入到 angular context

$digest 循环开始执行，查询每个 $watch 是否变化

由于监视 $scope.val 的 $watch 报告了变化，因此强制再执行一次 $digest 循环

新的 $digest 循环未检测到变化

浏览器拿回控制器，更新 $scope.val 新值对应的 dom

$digest 循环的上限是 10 次（超过 10次后抛出一个异常，防止无限循环）。

详见 [关于 AngularJS 的数据绑定](http://huangtengfei.com/2015/09/data-bind-of-angularjs/" \t "https://segmentfault.com/a/_blank)

### 两个平级界面块 a 和 b，如果 a 中触发一个事件，有哪些方式能让 b 知道？详述原理

这个问题换一种说法就是，如何在平级界面模块间进行通信。有两种方法，一种是共用服务，一种是基于事件。

##### 共用服务

在 Angular 中，通过 factory 可以生成一个单例对象，在需要通信的模块 a 和 b 中注入这个对象即可。

##### 基于事件

这个又分两种方式

第一种是借助父 controller。在子 controller 中向父 controller 触发（$emit）一个事件，然后在父 controller 中监听（$on）事件，再广播（$broadcast）给子 controller ，这样通过事件携带的参数，实现了数据经过父 controller，在同级 controller 之间传播。

第二种是借助 $rootScope。每个 Angular 应用默认有一个根作用域 $rootScope， 根作用域位于最顶层，从它往下挂着各级作用域。所以，如果子控制器直接使用 $rootScope 广播和接收事件，那么就可实现同级之间的通信。

详见 [AngularJS 中 Controller 之间的通信](http://huangtengfei.com/2015/09/communication-in-controller/" \t "https://segmentfault.com/a/_blank)

### 一个 angular 应用应当如何良好地分层？

##### 目录结构的划分

对于小型项目，可以按照文件类型组织，比如：

css

js

controllers

models

services

filters

templates

但是对于规模较大的项目，最好按业务模块划分，比如：

css

modules

account

controllers

models

services

filters

templates

disk

controllers

models

services

filters

templates

modules 下最好再有一个 common 目录来存放公共的东西。

##### 逻辑代码的拆分

作为一个 MVVM 框架，Angular 应用本身就应该按照 模型，视图模型（控制器），视图来划分。

这里逻辑代码的拆分，主要是指尽量让 controller 这一层很薄。提取共用的逻辑到 service 中 （比如后台数据的请求，数据的共享和缓存，基于事件的模块间通信等），提取共用的界面操作到 directive 中（比如将日期选择、分页等封装成组件等），提取共用的格式化操作到 filter 中等等。

在复杂的应用中，也可以为实体建立对应的构造函数，比如硬盘（Disk）模块，可能有列表、新建、详情这样几个视图，并分别对应的有 controller，那么可以建一个 Disk 构造函数，里面完成数据的增删改查和验证操作，有跟 Disk 相关的 controller，就注入 Disk 构造器并生成一个实例，这个实例就具备了增删改查和验证方法。这样既层次分明，又实现了复用（让 controller 层更薄了）。

参考 [AngularJS在苏宁云中心的深入实践](http://www.csdn.net/article/2015-08-24/2825527" \t "https://segmentfault.com/a/_blank)

### angular 应用常用哪些路由库，各自的区别是什么？

Angular1.x 中常用 ngRoute 和 ui.router，还有一种为 Angular2 设计的 [new router](https://angular.github.io/router/" \t "https://segmentfault.com/a/_blank)（面向组件）。后面那个没在实际项目中用过，就不讲了。

无论是 ngRoute 还是 ui.router，作为框架额外的附加功能，都必须以 模块依赖 的形式被引入。

##### 区别

ngRoute 模块是 Angular 自带的路由模块，而 ui.router 模块是基于 ngRoute模块开发的第三方模块。

ui.router 是基于 state （状态）的， ngRoute 是基于 url 的，ui.router模块具有更强大的功能，主要体现在视图的嵌套方面。

使用 ui.router 能够定义有明确父子关系的路由，并通过 ui-view 指令将子路由模版插入到父路由模板的 <div ui-view></div> 中去，从而实现视图嵌套。而在 ngRoute 中不能这样定义，如果同时在父子视图中 使用了 <div ng-view></div> 会陷入死循环。

##### 示例

ngRoute

var app = angular.module('ngRouteApp', ['ngRoute']);

app.config(function($routeProvider){

$routeProvider

.when('/main', {

templateUrl: "main.html",

controller: 'MainCtrl'

})

.otherwise({ redirectTo: '/tabs' });

ui.router

var app = angular.module("uiRouteApp", ["ui.router"]);

app.config(function($urlRouterProvider, $stateProvider){

$urlRouterProvider.otherwise("/index");

$stateProvider

.state("Main", {

url: "/main",

templateUrl: "main.html",

controller: 'MainCtrl'

})

### 如果通过angular的directive规划一套全组件化体系，可能遇到哪些挑战？

没有自己用 directive 做过一全套组件，讲不出。

能想到的一点是，组件如何与外界进行数据的交互，以及如何通过简单的配置就能使用吧。

### 分属不同团队进行开发的 angular 应用，如果要做整合，可能会遇到哪些问题，如何解决？

可能会遇到不同模块之间的冲突。

比如一个团队所有的开发在 moduleA 下进行，另一团队开发的代码在 moduleB 下

angular.module('myApp.moduleA', [])

.factory('serviceA', function(){

...

})

angular.module('myApp.moduleB', [])

.factory('serviceA', function(){

...

})

angular.module('myApp', ['myApp.moduleA', 'myApp.moduleB'])

会导致两个 module 下面的 serviceA 发生了覆盖。

貌似在 Angular1.x 中并没有很好的解决办法，所以最好在前期进行统一规划，做好约定，严格按照约定开发，每个开发人员只写特定区块代码。

### angular 的缺点有哪些？

##### 强约束

导致学习成本较高，对前端不友好。

但遵守 AngularJS 的约定时，生产力会很高，对 Java 程序员友好。

##### 不利于 SEO

因为所有内容都是动态获取并渲染生成的，搜索引擎没法爬取。

一种解决办法是，对于正常用户的访问，服务器响应 AngularJS 应用的内容；对于搜索引擎的访问，则响应专门针对 SEO 的HTML页面。

##### 性能问题

作为 MVVM 框架，因为实现了数据的双向绑定，对于大数组、复杂对象会存在性能问题。

可以用来 [优化 Angular 应用的性能](https://github.com/xufei/blog/issues/23" \t "https://segmentfault.com/a/_blank) 的办法：

减少监控项（比如对不会变化的数据采用单向绑定）

主动设置索引（指定 track by，简单类型默认用自身当索引，对象默认使用 $$hashKey，比如改为 track by item.id）

降低渲染数据量（比如分页，或者每次取一小部分数据，根据需要再取）

数据扁平化（比如对于树状结构，使用扁平化结构，构建一个 map 和树状数据，对树操作时，由于跟扁平数据同一引用，树状数据变更会同步到原始的扁平数据）

另外，对于Angular1.x ，存在 脏检查 和 模块机制 的问题。

##### 移动端

可尝试 Ionic，但并不完善。

参考 [如何看2015年1月Peter-Paul Koch对Angular的看法？](https://www.zhihu.com/question/27601964/answer/37309005" \t "https://segmentfault.com/a/_blank)

### 如何看待 angular 1.2 中引入的 controller as 语法？

##### 最根本的好处

在 angular 1.2 以前，在 view 上的任何绑定都是直接绑定在 $scope 上的

function myCtrl($scope){

$scope.a = 'aaa';

$scope.foo = function(){

...

}

}

使用 controllerAs，不需要再注入 $scope，controller 变成了一个很简单的 javascript 对象（POJO），一个更纯粹的 ViewModel。

function myCtrl(){

// 使用 vm 捕获 this 可避免内部的函数在使用 this 时导致上下文改变

var vm = this;

vm.a = 'aaa';

}

##### 原理

从源码实现上来看，controllerAs 语法只是把 controller 这个对象的实例用 as 别名在 $scope 上创建了一个属性。

if (directive.controllerAs) {

locals.$scope[directive.controllerAs] = controllerInstance;

}

但是这样做，除了上面提到的使 controller 更加 POJO 外，还可以避免遇到 AngularJS 作用域相关的一个坑（就是上文中 ng-if 产生一级作用域的坑，其实也是 javascript 原型链继承中值类型继承的坑。因为使用 controllerAs 的话 view 上所有字段都绑定在一个引用的属性上，比如 vm.xx，所以坑不再存在）。

<div ng-controller="TestCtrl as vm">

<p>{{name}}</p>

<div ng-if="vm.name">

<input type="text" ng-model="vm.name">

</div></div>

##### 问题

使用 controllerAs 会遇到的一个问题是，因为没有注入 $scope，导致 $emit、 $broadcast、 $on、 $watch 等 $scope 下的方法无法使用。这些跟事件相关的操作可以封装起来统一处理，或者在单个 controller 中引入 $scope，特殊对待。

参考 [angular controller as syntax vs scope](http://www.cnblogs.com/whitewolf/p/3493362.html" \t "https://segmentfault.com/a/_blank)

### 详述 angular 的 “依赖注入”

##### 栗子

依赖注入是一种软件设计模式，目的是处理代码之间的依赖关系，减少组件间的耦合。

举个栗子，如果没有使用 AngularJS，想从后台查询数据并在前端显示，可能需要这样做：

var animalBox = document.querySelector('.animal-box');

var httpRequest = {

get: function(url, callback){

console.log(url + ' requested');

var animals = ['cat', 'dog', 'rabbit'];

callback(animals);

}

}

var render = function(el, http){

http.get('/api/animals', function(animals){

el.innerHTML = animals;

})

}

render(httpRequest, animalBox);

但是，如果在调用 render 的时候不传参数，像下面这样，会报错，因为找不到 el 和 http（定义的时候依赖了，运行的时候不会自动查找依赖项）

render();// TypeError: Cannot read property 'get' of undefined

而使用 AngularJS，可以直接这样

function myCtrl = ($scope, $http){

$http.get('/api/animals').success(function(data){

$scope.animals = data;

})

}

也就是说，在 Angular App 运行的时候，调用 myCtrl，自动做了 $scope 和 $http 两个依赖性的注入。

##### 原理

AngularJS 是通过构造函数的参数名字来推断依赖服务名称的，通过 toString() 来找到这个定义的 function 对应的字符串，然后用正则解析出其中的参数（依赖项），再去依赖映射中取到对应的依赖，实例化之后传入。

简化一下，大概是这样：

var inject = {

// 存储依赖映射关系

storage: {},

// 注册依赖

register: function(name, resource){

this.storage[name] = resource;

},

// 解析出依赖并调用

resolve: function(target){

var self = this;

var FN\_ARGS = /^function\s\*[^\(]\*\(\s\*([^\)]\*)\)/m;

var STRIP\_COMMENTS = /((\/\/.\*$)|(\/\\*[\s\S]\*?\\*\/))/mg;

fnText = target.toString().replace(STRIP\_COMMENTS, '');

argDecl = fnText.match(FN\_ARGS)[1].split(/, ?/g);

var args = [];

argDecl.forEach(function(arg){

if(self.storage[arg]){

args.push(self.storage[arg]);

}

})

return function(){

target.apply({}, args);

}

}

}

使用这个 injector，前面那个不用 AngularJS 的栗子这样改造一下就可以调用了

inject.register('el', animalBox);

inject.register('ajax', httpRequest);

render = inject.resolve(render);

render();

##### 问题

因为 AngularJS 的 injector 是假设函数的参数名就是依赖的名字，然后去查找依赖项，那如果按前面栗子中那样注入依赖，代码压缩后（参数被重命名了），就无法查找到依赖项了。

// 压缩前function myCtrl = ($scope, $http){

...

}

// 压缩后function myCtrl = (a, b){

...

}

所以，通常会使用下面两种方式注入依赖（对依赖添加的顺序有要求）。

数组注释法

myApp.controller('myCtrl', ['$scope', '$http', function($scope, $http){

...

}])

显式 $inject

myApp.controller('myCtrl', myCtrl);function myCtrl = ($scope, $http){

...

}

myCtrl.$inject = ['$scope', '$http'];

##### 补充

对于一个 DI 容器，必须具备三个要素：依赖项的注册，依赖关系的声明和对象的获取。

在 AngularJS 中，module 和 $provide 都可以提供依赖项的注册；内置的 injector 可以获取对象（自动完成依赖注入）；依赖关系的声明，就是前面问题中提到的那样。

下面是个栗子

// 对于 module，传递参数不止一个，代表新建模块，空数组代表不依赖其他模块// 只有一个参数（模块名），代表获取模块

// 定义 myApp，添加 myApp.services 为其依赖项

angular.module('myApp', ['myApp.services']);// 定义一个 services module，将 services 都注册在这个 module 下面

angular.module('myApp.services', [])

// $provider 有 factory, service, provider, value, constant

// 定义一个 HttpService

angular.module('myApp.services').service('HttpService', ['$http', function($http){

...

}])

参考

[[AngularJS] 自己实现一个简单的依赖注入](https://segmentfault.com/a/1190000003048407cope-in-angularjs/)

[理解angular中的module和injector，即依赖注入](http://www.mamicode.com/info-detail-247448.html" \t "https://segmentfault.com/a/_blank)

[AngularJS中的依赖注入实际应用场景](https://www.zhihu.com/question/28097646" \t "https://segmentfault.com/a/_blank)

### 如何看待angular2

相比 Angular1.x，Angular2的改动很大，几乎算是一个全新的框架。

基于 TypeScript（可以使用 TypeScript 进行开发），在大型项目团队协作时，强语言类型更有利。

组件化，提升开发和维护的效率。

还有 module 支持动态加载，new router，promise的原生支持等等。

迎合未来标准，吸纳其他框架的优点，值得期待，不过同时要学习的东西也更多了（ES next、TS、Rx等）

# Angular 经典面试问题及扩展阅读

## 1.  请解释Angular 2应用程序的生命周期hooks是什么？

Angular 2组件/指令具有生命周期事件，是由@angular/core管理的。@angular/core会创建组件，渲染它，创建并呈现它的后代。当@angular/core的数据绑定属性更改时，处理就会更改，在从DOM中删除其模板之前，就会销毁掉它。Angular提供了一组生命周期hooks（特殊事件），可以被分接到生命周期中，并在需要时执行操作。构造函数会在所有生命周期事件之前执行。每个接口都有一个前缀为ng的hook方法。例如，ngOnint界面的OnInit方法，这个方法必须在组件中实现。

一部分事件适用于组件/指令，而少数事件只适用于组件。

* ngOnChanges：当Angular设置其接收当前和上一个对象值的数据绑定属性时响应。
* ngOnInit：在第一个ngOnChange触发器之后，初始化组件/指令。这是最常用的方法，用于从后端服务检索模板的数据。
* ngDoCheck：检测并在Angular上下文发生变化时执行。每次更改检测运行时，会被调用。
* ngOnDestroy：在Angular销毁指令/组件之前清除。取消订阅可观察的对象并脱离事件处理程序，以避免内存泄漏。

组件特定hooks：

* ngAfterContentInit：组件内容已初始化完成
* ngAfterContentChecked：在Angular检查投影到其视图中的绑定的外部内容之后。
* ngAfterViewInit：Angular创建组件的视图后。
* ngAfterViewChecked：在Angular检查组件视图的绑定之后。

### 

### 2.  使用Angular 2，和使用Angular 1相比，有什么优势？

1. Angular 2是一个平台，不仅是一种语言
2. 更好的速度和性能
3. 更简单的依赖注入
4. 模块化，跨平台
5. 具备ES6和Typescript的好处。
6. 灵活的路由，具备延迟加载功能
7. 更容易学习

### 

### 3.  Angular 2中的路由工作原理是什么？

路由是能够让用户在视图/组件之间导航的机制。Angular 2简化了路由，并提供了在模块级（延迟加载）下配置和定义的灵活性。

Angular应用程序具有路由器服务的单个实例，并且每当URL改变时，相应的路由就与路由配置数组进行匹配。在成功匹配时，它会应用重定向，此时路由器会构建ActivatedRoute对象的树，同时包含路由器的当前状态。在重定向之前，路由器将通过运行保护（[CanActivate](https://blog.thoughtram.io/angular/2016/07/18/guards-in-angular-2.html)）来检查是否允许新的状态。Route Guard只是路由器运行来检查路由授权的接口方法。保护运行后，它将解析路由数据并通过将所需的组件实例化到<router-outlet> </ router-outlet>中来激活路由器状态。

**扩展阅读：**

1. [https://www.codeproject.com/Articles/1164813/Angular-Routing](https://www.codeproject.com/Articles/1164813/Angular-Routing%C2%A0)
2. [https://vsavkin.com/angular-2-router-d9e30599f9ea#.kt4z1v957](https://vsavkin.com/angular-2-router-d9e30599f9ea" \l ".kt4z1v957)

### 

### 4.  什么是事件发射器？它是如何在Angular 2中工作的？

Angular 2不具有双向digest cycle，这是与Angular 1不同的。在Angular2中，组件中发生的任何改变总是从当前组件传播到其所有子组件中。如果一个子组件的更改需要反映到其父组件的层次结构中，我们可以通过使用事件发射器api来发出事件。

简而言之，EventEmitter是在@ angular/core模块中定义的类，由组件和指令使用，用来发出自定义事件。

@output() somethingChanged = new EventEmitter();

我们使用somethingChanged.emit（value）方法来发出事件。这通常用在setter中，当类中的值被更改完成时。

可以通过模块的任何一个组件，使用订阅方法来实现事件发射的订阅。

myObj.somethingChanged.subscribe(val) => this.myLocalMethod(val));

**扩展阅读：**

1. <http://stackoverflow.com/questions/36076700/what-is-the-proper-use-of-an-eventemitter>
2. [https://angular.io/docs/ts/latest/api/core/index/EventEmitter-class.HTML](https://angular.io/docs/ts/latest/api/core/index/EventEmitter-class.html)

### 

### 5.  如何在Angular 2应用程序中使用codelyzer？

所有企业应用程序都会遵循一组编码惯例和准则，以更好的方式维护代码。Codelyzer是一个开源工具，用于运行和检查是否遵循了预定义的编码准则。Codelyzer仅对Angular和TypeScript项目进行静态代码分析。

Codelyzer运行在tslint的顶部，其编码约定通常在tslint.json文件中定义。Codelyzer可以直接通过Angularcli或npm运行。像Visual Studio Code和Atom这样的编辑器也支持codelyzer，只需要通过做一个基本的设置就能实现。

要在Visual Studio代码中设置codelyzer，我们可以在文件 - >选项 - >用户设置中添加tslint规则的路径。

{

"tslint.rulesDirectory": "./node\_modules/codelyzer",

"typescript.tsdk": "node\_modules/typescript/lib"

}

从cli中运行的代码：ng lint。

从npm中运行的代码： npm run lint

**扩展阅读：**

1. <https://github.com/mgechev/codelyzer>
2. <https://www.youtube.com/watch?v=bci-Z6nURgE>

### 

### 6.  什么是延迟加载？如何在Angular 2中启用延迟加载？

大多数企业应用程序包含用各式各样的用于特定业务案例的模块。捆绑整个应用程序代码并完成加载，会在初始调用时，产生巨大的性能开销。延迟加载使我们只加载用户正在交互的模块，而其余的模块会在运行时按需加载。

延迟加载通过将代码拆分成多个包并以按需加载的方式，来加速应用程序初始加载过程。

每个Angular应用程序必须有一个叫AppModule的主模块。代码应该根据应用程序业务案例分为不同的子模块（NgModule）。

启用延迟加载的Plunkr示例：

1. 我们不需要在根模块中导入或声明延迟加载模块。
2. 将路由添加到顶层路由（app.routing.ts）并设置loadChildren。loadChildren会从根文件夹中获取绝对路径。RouterModule.forRoot（）会获取routes数组并配置路由器。
3. 在子模块中导入模块特定路由。
4. 在子模块路由中，将路径指定为空字符串“”，也就是空路径。RouterModule.forChild会再次采用路由数组为子模块组件加载并配置路由器。
5. 然后，导出const路由：ModuleWithProviders = RouterModule.forChild（routes）;

### 

### 7.  在Angular 2应用中，我们应该注意哪些安全威胁？

就像任何其他客户端或Web应用程序一样，Angular 2应用程序也应该遵循一些基本准则来减轻安全风险。其中一些是：

1. 避免为你的组件使用/注入动态HTML内容。
2. 如果使用外部HTML，也就是来自数据库或应用程序之外的地方，那么就需要清理它。
3. 不要将外部网址放在应用程序中，除非它是受信任的。避免网址重定向，除非它是可信的。
4. 考虑使用AOT编译或离线编译。
5. 通过限制api，选择使用已知或安全环境/浏览器的app来防止XSRF攻击。

**扩展阅读**：[https://angular.io/docs/ts/latest/guide/security.HTML#!#best-practices](https://angular.io/docs/ts/latest/guide/security.html" \l "!)

### 

### 8.  如何优化Angular 2应用程序来获得更好的性能？

优化取决于应用程序的类型和大小以及许多其他因素。但一般来说，在优化Angular 2应用程序时，我会考虑以下几点：

1. 考虑AOT编译。
2. 确保应用程序已经经过了捆绑，uglify和tree shaking。
3. 确保应用程序不存在不必要的import语句。
4. 确保应用中已经移除了不使用的第三方库。
5. 所有dependencies 和dev-dependencies都是明确分离的。
6. 如果应用程序较大时，我会考虑延迟加载而不是完全捆绑的应用程序。

**扩展阅读：**

1. [https://medium.com/@areai51/the-4-stages-of-perf-tuning-for-your-angular2-app-922ce5c1b294#.pw4m2srmr](https://medium.com/@areai51/the-4-stages-of-perf-tuning-for-your-angular2-app-922ce5c1b294" \l ".pw4m2srmr)
2. <https://www.lucidchart.com/techblog/2016/05/04/angular-2-best-practices-change-detector-performance/>

### 

### 9.  如何实现不出现编辑器警告的自定义类型？

在大多数的情况下，第三方库都带有它的.d.ts 文件，用于类型定义。在某些情况下，我们需要通过向现有类型提供一些更多的属性来扩展现有类型，或者如果我们需要定义其它类型以避免TypeScript警告。

如果我们需要扩展外部库的类型定义，一个好的做法是，我们并非对node\_modules或现有的typings文件夹进行改动，而是创建一个命名为“自定义类型”的新文件夹，来存储所有的自定义类型。

要定义应用程序（JavaScript / Typescript）对象的类型，我们应该在应用程序相应模块的models文件夹中，定义接口和实体类。

对于这些情况，我们可以通过创建我们自己的“ .d.ts”文件来实现定义或扩展类型。

**扩展阅读：**

1. <https://www.typescriptlang.org/docs/handbook/declaration-merging.HTML>
2. [https://typescript.codeplex.com/wikipage?title=Writing%20Definition%20%28.d.ts%29%20Files](https://typescript.codeplex.com/wikipage?title=Writing Definition (.d.ts) Files)
3. <http://stackoverflow.com/questions/32948271/extend-interface-defined-in-d-ts-file>

### 

### 10. 什么是Shadow DOM？它如何帮助Angular 2更好地执行？

Shadow DOM是HTML规范的一部分，它允许开发人员封装自己的HTML标记，CSS样式和JavaScript。Shadow DOM以及其它一些技术，使开发人员能够像<audio>标签一样构建自己的一级标签，Web组件和API。总的来说，这些新的标签和API被称为Web组件。Shadow DOM通过提供了更好的关注分离，通过其它的HTML DOM元素实现了更少的样式与脚本的冲突。

因为shadow DOM本质上是静态的，同时也是开发人员无法访问的，所以它是一个很好的候选对象。因为它缓存的DOM将在浏览器中呈现得更快，并提供更好的性能。此外，还可以相对很好地管理shadow DOM，同时检测Angular 2应用的改变，并且可以有效地管理视图的重新绘制。

**扩展阅读：**

1. <https://developer.mozilla.org/en-US/docs/Web/Web_Components/Shadow_DOM>
2. <https://glazkov.com/2011/01/14/what-the-heck-is-shadow-dom/>
3. <https://code.tutsplus.com/tutorials/intro-to-shadow-dom--net-34966>

### 

### 11. 什么是AOT编译？它有什么优缺点？

AOT编译代表的是Ahead Of Time编译，其中Angular编译器在构建时，会将Angular组件和模板编译为本机JavaScript和HTML。编译好的HTML和JavaScript将会部署到Web服务器，以便浏览器可以节省编译和渲染时间。

**优点：**

1. 更快的下载：由于应用程序已经编译，许多Angular编译器相关库就不再需要捆绑，应用程序包变得更小，所以该应用程序可以更快地下载。
2. 更少的Http请求数：如果应用程序没有捆绑来支持延迟加载（或任何原因），对于每个关联的HTML和CSS，都会有一个单独的服务器请求。但是预编译的应用程序会将所有模板和样式与组件对齐，因此到服务器的Http请求数量会更少。
3. 更快的渲染：如果应用程序不是AOT编译，那么应用程序完全加载时，编译过程会发生在浏览器中。这需要等待下载所有必需的组件，然后等待编译器花费时间来编译应用程序。使用AOT编译，就能实现优化。
4. 在构建时检测错误：由于预先编译，可以检测到许多编译时错误，能够为应用程序提供更好的稳定性。

**缺点：**

1. 仅适用于HTML和CSS，其它文件类型需要前面的构建步骤
2. 没有watch模式，必须手动完成（bin / ngc-watch.js）并编译所有文件
3. 需要维护AOT版本的bootstrap文件（使用cli等工具时不需要）
4. 在编译之前，需要清理步骤

**扩展阅读**：[https://angular.io/docs/ts/latest/cookbook/aot-compiler.HTML](https://angular.io/docs/ts/latest/cookbook/aot-compiler.html)

### 

### 12. Observables和Promises的核心区别是什么？

从堆栈溢出就是一个区别：

当异步操作完成或失败时，Promise会处理一个单个事件。

Observable类似于（在许多语言中的）Stream，当每个事件调用回调函数时，允许传递零个或多个事件。通常Observable比Promise更受欢迎，因为它不但提供了Promise特性，还提供了其它特性。使用Observable可以处理0,1或多个事件。你可以在每种情况下使用相同的API。Observable是可取消的，这相比于Promise也具有优势。如果服务器的HTTP请求结果或其它一些异步操作不再需要，则Observable的订阅者可以取消订阅，而Promise将最终调用成功或失败的回调，即使你不需要通知或其提供的结果。Observable提供像map，forEach，reduce之类的类似于数组的**运算符**，还有强大的运算符，如retry（）或replay（）等，使用起来是相当方便的。

**Promises vs Observables**

* **Promises：**

1. 返回单个值
2. 不可取消

* **Observables：**

1. 可以使用多个值
2. 可取消
3. 支持map，filter，reduce和类似的操作符
4. ES 2016提议的功能
5. 使用反应式扩展（RxJS）
6. 根据时间的变化，数组成员可以异步获取