**angular1面试题**

1. **基础知识**

### 1.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：

1.减少监控项（比如对不会变化的数据采用单向绑定）

2.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）

3.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）

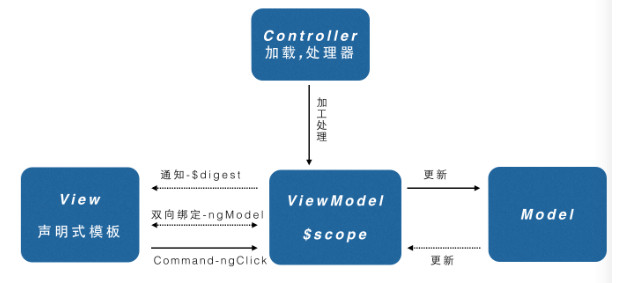
4.数据扁平化（比如对于树状结构，使用扁平化结构，构建一个 map 和树状数据，对树操作时，由于跟扁平数据同一引用，树状数据变更会同步到原始的扁平数据）

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。

1. **.angularjs 是mvc还是mvvm框架**  
     
   首先阐述下你对mvc和mvvm的理解  
   首先为什么我们会需要MVC？因为随着代码规模越来越大，切分职责是大势所趋，还有为了后期维护方便，修改一块功能不影响其他功能。还有为了复用，因为很多逻辑是一样的。而MVC只是手段，终极目标是模块化和复用。 **mvvm的优点**  
   低耦合：View可以独立于Model变化和修改，同一个ViewModel可以被多个View复用；并且可以做到View和Model的变化互不影响；  
   可重用性：可以把一些视图的逻辑放在ViewModel，让多个View复用；  
   独立开发：开发人员可以专注与业务逻辑和数据的开发（ViewModemvvmdi计人员可以专注于UI(View)的设计；  
   可测试性：清晰的View分层，使得针对表现层业务逻辑的测试更容易，更简单。  
   **在angular中MVVM模式主要分为四部分：**View：它专注于界面的显示和渲染，在angular中则是包含一堆声明式Directive的视图模板。  
   ViewModel：它是View和Model的粘合体，负责View和Model的交互和协作，它负责给View提供显示的数据，以及提供了View中Command事件操作Model的途径；在angular中$scope对象充当了这个ViewModel的角色；  
   Model：它是与应用程序的业务逻辑相关的数据的封装载体，它是业务领域的对象，Model并不关心会被如何显示或操作，所以模型也不会包含任何界面显示相关的逻辑。在web页面中，大部分Model都是来自Ajax的服务端返回数据或者是全局的配置对象；而angular中的service则是封装和处理这些与Model相关的业务逻辑的场所，这类的业务服务是可以被多个Controller或者其他service复用的领域服务。  
   Controller：这并不是MVVM模式的核心元素，但它负责ViewModel对象的初始化，它将组合一个或者多个service来获取业务领域Model放在ViewModel对象上，使得应用界面在启动加载的时候达到一种可用的状态。



mvc的界面和逻辑关联紧密，数据直接从数据库读取。mvvm的界面与viewmode是松耦合，界面数据从viewmodel中获取。所以angularjs更倾向于mvvm。

1. **在使用angularjs项目开发中 你使用过那些第三方的插件**  
   AngularUi ui-router oclazyload等等。
2. **在angular项目中你如何控制静态资源的合理加载**  
   oclazyload这个插件，很好的一个懒加载静态资源的第三方插件
3. **在写controlloer逻辑的时候 你需要注意什么？**  
   1.简化代码（这个是所有开发人员都要具备的）  
     
   2.坚决不能操作dom节点 这个时候可能会问 为什么不能啊  
     
   你的回答是：DOM操作只能出现在指令（directive）中。最不应该出现的位置就是服务（service）中。Angular倡导以测试驱动开发，在service或者controller中出现了DOM操作，那么也就意味着的测试是无法通过的。当然，这只是一点，重要的是使用Angular的其中一个好处是啥，那就是双向数据绑定，这样就能专注于处理业务逻辑，无需关系一堆堆的DOM操作。如果在Angular的代码中还到处充斥着各种DOM操作，那为什么不直接使用jquery去开发呢。
4. **测试驱动开发是什么呢？普及一下：**  
   测试驱动开发，英文全称Test-Driven Development，简称TDD，是一种不同于传统软件开发流程的新型的开发方法。它要求在编写某个功能的代码之前先编写测试代码，然后只编写使测试通过的功能代码，通过测试来推动整个开发的进行。这有助于编写简洁可用和高质量的代码，并加速开发过程。
5. **相关指令**

### **1.**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。

### **2.**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）。

### **3.**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);

}

})

### **4.{{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;

}

})

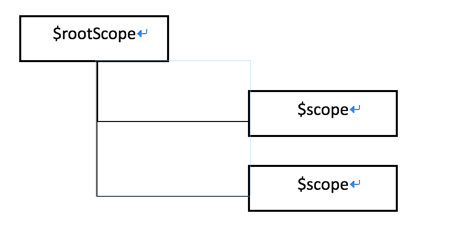
### 5.如果通过angular的directive规划一套全组件化体系，可能遇到哪些挑战？

## 6.$apply()和 $digest()的区别

安全性：$apply()可以接收一个参数作为function()，这个 function 会被包装到一个 try … catch 块中，所以一旦有异常发生，该异常会被 $exceptionHandler service 处理。$apply会使ng进入 $digest cycle , 并从$rootScope开始遍历(深度优先)检查数据变更。$digest仅会检查该scope和它的子scope，当你确定当前操作仅影响它们时，用$digest可以稍微提升性能。

### 7..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会继续解析，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。

### 8.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。

### **9.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();

...

}

});

1. **路由**

### **1.**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'

})

# **2.**AngularJS 路由和模板实例及路由地址简化方法

**1.AngularJS路由介绍**

AngularJS路由与后端MVC的路由不一样。AngularJS的前端路由，需要在模板页提前对指定的(ng-app)，定义路由规则(routeProvider)，然后通过不同的URL，告诉(ng-app)加载哪个页面(HTML)，再渲染到(ng-app)视图(ng-view)中。而MVC路由是通过不同的Url请求，然后根据路由规则请求控制器的Action，并返回View。AngularJS是一个纯前端的路由。后面我们会结合配置文件简化路由地址。

**2.AngularJS路由实例**

我们新建一个JS文件，此处我命名为app.js，在文件中我们写入如下代码：

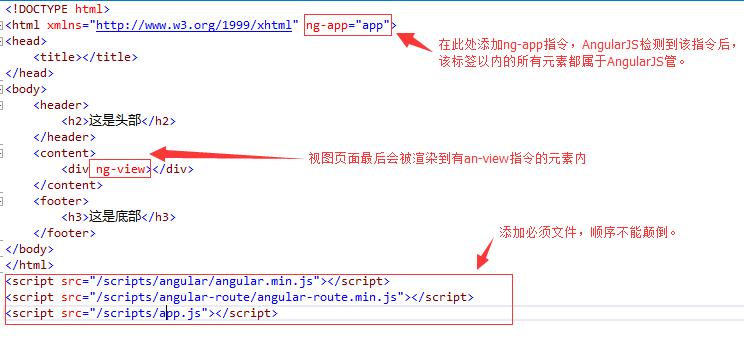
angular.module("app", [ 'ngRoute' ])

.config(['$routeProvider', function ($routeProvider) { $routeProvider.when("/list", { template: "这是列表页"})

.when("/detail", { template: "这是详情页" })

.otherwise({redirectTo: "/list"})}])

我们新建一个html文件命名为index.html，代码及说明如下：



<!DOCTYPE html>

<html xmlns="http://www.w3.org/1999/xhtml" ng-app="app">

<head>

<title></title>

</head>

<body>

<header>

<h2>这是头部</h2>

</header>

<content>

<div ng-view></div>

</content>

<footer>

<h3>这是底部</h3>

</footer>

</body>

</html>

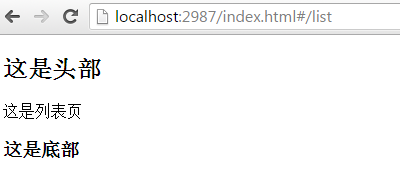
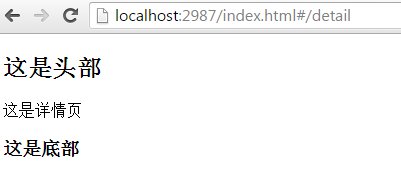
<script src="/scripts/angular/angular.min.js"></script>

<script src="/scripts/angular-route/angular-route.min.js"></script>

<script src="/scripts/app.js"></script>

运行实例，效果如下。

浏览器打开输入地址：http://localhost:2987/index.html#/detail和http://localhost:2987/index.html#/list分别可以看到如下图所示页面：



**3.AngularJS采用HTML5模式的路由结合WebServer简化路由地址**

上图中我们可以看到，请求地址很长，对于有强迫症的我来说，很难接受这种类型的地址，于是我决定作出这样的改变：

1）AngulaJS开启HTML5模式路由，去掉#号：

对于ANgulaJS来说，其默认是没有打开Html5路由模式，我们通过如下做法开启HTML5路由模式：

angular.module("app", [ 2 'ngRoute' 3 ])

.config(['$routeProvider', '$locationProvider', function ($routeProvider, $locationProvider) {

//开启html5路由模式

$locationProvider.html5Mode(true)

$routeProvider.when("/list", { template: "这是列表页" })

.when("/detail", { template: "这是详情页" })

.otherwise({13 redirectTo: "/404.html" })}])

　这样按照上面的请求地址去掉#号后刷新页面，发现提示404找不到页面，原因是这样的地址请求到后台IIS会找不到对应的文件，因此直接返回404错误页面。因此我们需要在配置文件中添加WebServer配置，重定向文件如下：

<system.webServer>

<rewrite>

<rules>

<rule name="name" >

<match url="^list|detail\*" ignoreCase="true"/>

<conditions logicalGrouping="MatchAll">

<add input="{REQUEST\_FILENAME}" matchType="IsFile" negate="true"/>

<add input="{REQUEST\_FILENAME}" matchType="IsDirectory" negate="true"/>

</conditions>

<action type="Rewrite" url="/index.html"/>

</rule>

</rules>

</rewrite>

</system.webServer>

如上面代码显示，其中<match url="^list|detail\*" ignoreCase="true"/> 的意思是，如果请求地址中包含list或detail（根据需求自己书写正则表达式），那么会被重定向到index.html页面，这样在页面上输入http://localhost:2987/detail的时候IIS会先返回模板页（index.html），然后AngulaJS路由才开始执行。

在浏览器输入http://localhost:2987/detail回车，发现并没有得到我们想要的结果，也就是说，路由没有匹配上。

网上查了很久，终于找到解决方案，就是在html页面中添加<base>标签如下：

<!DOCTYPE html>

<html xmlns="http://www.w3.org/1999/xhtml" ng-app="app">

<head>

<title></title>

<base href="/" />

</head>

<body>

<header>

<h2>这是头部</h2>

</header>

<content>

<div ng-view></div>

</content>

<footer>

<h3>这是底部</h3>

</footer>

</body>

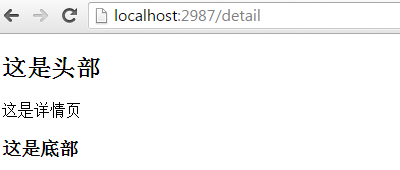
</html>

<script src="/scripts/angular/angular.min.js"></script>

<script src="/scripts/angular-route/angular-route.min.js"></script>

<script src="/scripts/app.js"></script>

这样，在浏览器上输入：http://localhost:2987/detail回车就会看到我们想要的结果，截图如下：



1. **服务**

### **1.**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。

1. **其他**

### **1.**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）。

### **2.**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。

### **3.**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）。

### **4.**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。

### **5.**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。

### **6.**详述 angular 的 “依赖注入”

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){

...

}])

## 7.compile和link的区别： 看到一个比较6的答案。性能力（性能和能力）

编译的时候，compile转换dom，碰到绑定监听器的地方就先存着，有几个存几个，到最后汇总成一个link函数，一并执行，提升了性能。

function compile(tElement, tAttrs, transclude) { ... }tElement为编译前的element

function link(scope, iElement, iAttrs, controller) { ... } iElement为编译后的element，已经与作用域关联起来，所以可以数据绑定

如果指令只进行DOM的修改，不进行数据绑定，那么配置在compile函数中，如果指令要进行数据绑定，那么配置在link函数中。

**8.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。

### 9.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。

### **10.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

### **11.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](http://lib.csdn.net/base/jquery" \o "jQuery知识库" \t "http://blog.csdn.net/donggx/article/details/_blank)**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**。

### **12.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);

});

});

});

1. **框架之间的比较**

# AngularJS进行性能调优的7个建议

AnglarJS作为一款优秀的Web框架，可大大简化前端开发的负担。近日Sebastian Fröstl在一篇博文《AngularJS Performance Tuning for Long Lists》中表示AnglarJS在处理包含复杂数据结构的大型列表时，其运行速度会非常慢。他在文中同时分享了解决方案。下面为该文的译文。

　　AnglarJS很棒，但当处理包含复杂数据结构的大型列表时，其运行速度就会非常慢。这是我们将核心管理页面迁移到AngularJS过程中遇到的问题。这些页面在显示500行数据时本应该工作顺畅，但首个方法的渲染时间竟花费了7秒，太可怕了。

　　后来，我们发现了在实现过程中存在两个主要性能问题。一个与“ng-repeat ”指令有关，另一个与过滤器有关。

　　下文将分享我们通过不同的方法解决性能问题的经验，希望可以给你带来启示。

　　一、AngularJS 中的ng-repeat在处理大型列表时，速度为什么会变慢？

　　AngularJS中的ng-repeat在处理2500个以上的双向数据绑定时速度会变慢。这是由于AngularJS通过“dirty checking”函数来检测变化。每次检测都会花费时间，所以包含复杂数据结构的大型列表将降低你应用的运行速度。

　　二、提高性能的先决条件

　　时间记录指令

　　为了测量一个列表渲染所花费的时间，我们写了一个简单的程序，通过使用“ng-repeat”的属性“$last”来记录时间。时间存放在TimeTracker服务中，这样时间记录就与服务器端的数据加载分开了。

　　// Post repeat directive for logging the rendering time angular.module('siApp.services').directive('postRepeatDirective', ['$timeout', '$log', 'TimeTracker', function($timeout, $log, TimeTracker) { return function(scope, element, attrs) { if (scope.$last){ $timeout(function(){ var timeFinishedLoadingList = TimeTracker.reviewListLoaded(); var ref = new Date(timeFinishedLoadingList); var end = new Date(); $log.debug("## DOM rendering list took: " + (end - ref) + " ms"); }); } }; } ]); // Use in HTML: …

　　Chrome开发者工具的时间轴(Timeline)属性

　　在Chrome开发者工具的时间轴标签中，你可以看见事件、每秒内浏览器帧数和内存分配。“memory”工具用来检测内存泄漏，及页面所需的内 存。当帧速率每秒低于30帧时就会出现页面闪烁问题。“frames”工具可帮助了解渲染性能，还可显示出一个JavaScript任务所花费的CPU时 间。

　　三、通过限制列表的大小进行基本的调优

　　缓解该问题，最好的办法是限制所显示列表的大小。可通过分页、添加无限滚动条来实现。

　　分页

　　分页，我们可以使用AngularJS的“limitTo”过滤器(AngularJS1.1.4版本以后)和“startFrom”过滤器。可以通过限制显示列表的大小来减少渲染时间。这是减少渲染时间最高效的方法。

　　// Pagination in controller $scope.currentPage = 0; $scope.pageSize = 75; $scope.numberOfPages = function() { return Math.ceil($scope.displayedItemsList.length/ $scope.pageSize); }; // Start from filter angular.module('app').filter('startFrom', function() { return function(input, start) { return input.slice(start); }; // Use in HTML // Pagination buttons{{$index + 1}}

　　如果你不能/不想使用分页，但过滤过程又很慢，这时一定要检查前五步，并使用“ng-show”隐藏掉多余的列表元素。

　　无限滚动条

　　如果你希望进一步了解该方法，可访问 <http://binarymuse.github.io/ngInfiniteScroll/>

　　四、七大调优法则

　　1. 渲染没有数据绑定的列表

　　这是最明显的解决方案，因为数据绑定是性能问题最可能的根源。如果你只想显示一次列表，并不需要更新、改变数据，放弃数据绑定是绝佳的办法。不过可惜的是，你会失去对数据的控制权，但除了该法，我们别无选择。进一步了解： <https://github.com/Pasvaz/bindonce>。

　　2.不要使用内联方法计算数据

　　为了在控制器中直接过滤列表，不要使用可获得过滤链接的方法。“ng-repeat”会评估每个 [$digest([http://docs.angularjs.org/api/ng.$rootScope.Scope#$digest)%5D](http://docs.angularjs.org/api/ng.$rootScope.Scope" \l "$digest)])表达式。在我们的案例中，“filteredItems()”返回过滤链接。如果评估过程很慢，它将迅速降低整个应用的速度。

　　//这并不是一个好方法，因为要频繁地评估。

　　//这是要采用的方法

　　3.使用两个列表(一个用来进行视图显示，一个作为数据源)

　　将要显示的列表与总的数据列表分开，是非常有用的模型。你可以对一些过滤进行预处理，并将存于缓存中的链接应用到视图上。下面案例展示了基本实现过程。filteredLists变量保存着缓存中的链接，applyFilter方法来处理映射。

　　/\* Controller \*/ // Basic list var items = [{name:"John", active:true }, {name:"Adam"}, {name:"Chris"}, {name:"Heather"}]; // Init displayedList $scope.displayedItems = items; // Filter Cache var filteredLists['active'] = $filter('filter)(items, {"active" : true}); // Apply the filter $scope.applyFilter = function(type) { if (filteredLists.hasOwnProperty(type){ // Check if filter is cached $scope.displayedItems = filteredLists[type]; } else { /\* Non cached filtering \*/ } } // Reset filter $scope.resetFilter = function() { $scope.displayedItems = items; } /\* View \*/Select active

　　{{item.name}}

　　4.在其他模板中使用ng-if来代替ng-show

如果你用指令、模板来渲染额外的信息，例如通过点击来显示列表项的详细信息，一定要使用 ng-if(AngularJSv. 1.1.5以后)。ng-if可阻止渲染(与ng-show相比)。所以其它DOM和数据绑定可根据需要进行评估。

### 2.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");

# 3.解决angular的$http.post()提交数据时后台接收不到参数值问题的方法

前端：html，jquery，angular  
后端：java，springmvc  
**一、平常使用的post提交和接收方式**前端使用jquery提交数据。

[?](http://www.jb51.net/article/76147.htm)

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9 | $.ajax({    url:'/carlt/loginForm',    method: 'POST',    data:{"name":"jquery","password":"pwd},    dataType:'json',    success:function(data){      //...    }  }); |

后端java接收：

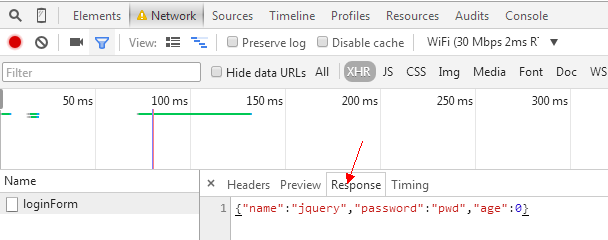
[?](http://www.jb51.net/article/76147.htm)

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9  10  11  12  13  14  15  16  17  18  19 | @Controller  public class UserController {    @ResponseBody    @RequestMapping(value="/loginForm",method=RequestMethod.POT)    public User loginPost(User user){      System.out.println("username:"+user.getName());      System.out.println("password:"+user.getPassword());      return user;    }  }  model（不要忘记get、set方法）：  public class User {    private String name;    private String password;    private int age;      //setter getter method    } |

后台打印：

**username:jquery  
password:pwd**

调用接口查看到的前端返回结果：



**二、使用angularJs的post方法提交**

[?](http://www.jb51.net/article/76147.htm)

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9  10 | <div ng-app="myApp" ng-controller="formCtrl">   <form novalidate>   UserName:<br>  <inputtype="text" ng-model="user.username"><br>   PassWord:<br>   <input type="text" ng-model="user.pwd">   <br><br>   <button ng-click="login()">登录</button>   </form>  </div> |

js代码：

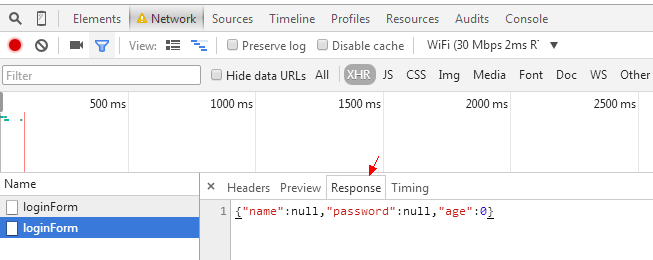
[?](http://www.jb51.net/article/76147.htm)

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9  10  11  12  13  14 | var app = angular.module('myApp', []);  app.controller('formCtrl', function($scope,$http) {   $scope.login = function() {    $http({     url:'/carlt/loginForm',     method: 'POST',     data: {name:'angular',password:'333',age:1}    }).success(function(){     console.log("success!");    }).error(function(){     console.log("error");    })   };  }); |

后台打印结果：

**username:null  
password:null：**

查看前端：

[](http://files.jb51.net/file_images/article/201512/2015121090904937.png?201511109911" \t "http://www.jb51.net/article/_blank)

**三、解决angular提交post问题。**相信看过上面提到的哪怕文章的人已经知道怎么解决问题了吧。文中是更改了angular的提交方式，使得angular的提交数据方式更像jquery的。

我试过，也是行得通的。然后我又试了另外一种方式。如下：

前端不变，依然是：

[?](http://www.jb51.net/article/76147.htm)

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9  10  11  12  13  14 | var app = angular.module('myApp', []);  app.controller('formCtrl', function($scope,$http) {    $scope.login = function() {      $http({        url:'/carlt/loginForm',        method: 'POST',     data: {name:'angular',password:'333',age:1}      }).success(function(){       console.log("success!");      }).error(function(){        console.log("error");      })    };  }); |

后台变了，只是在User前加上@RequstBody，因为angular提交的是json对象：

[?](http://www.jb51.net/article/76147.htm)

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9  10  11 | @Controller  public class UserController {    @ResponseBody    @RequestMapping(value="/loginForm",method=RequestMethod.POST)    public User loginPost(@RequestBody User user){      System.out.println("username:"+user.getName());      System.out.println("password:"+user.getPassword());      return user;    }  }  @RequestBody |

**作用：**

      i) 该注解用于读取Request请求的body部分数据，使用系统默认配置的HttpMessageConverter进行解析，然后把相应的数据绑定到要返回的对象上；

      ii) 再把HttpMessageConverter返回的对象数据绑定到 controller中方法的参数上。

**使用时机：**

A) GET、POST方式提时， 根据request header Content-Type的值来判断:

    application/x-www-form-urlencoded， 可选（即非必须，因为这种情况的数据@RequestParam, @ModelAttribute也可以处理，当然@RequestBody也能处理）；  
    multipart/form-data, 不能处理（即使用@RequestBody不能处理这种格式的数据）；  
    其他格式， 必须（其他格式包括application/json, application/xml等。这些格式的数据，必须使用@RequestBody来处理）；  
B) PUT方式提交时，根据**request header Content-Type**的值来判断:

    application/x-www-form-urlencoded， 必须；  
    multipart/form-data, 不能处理；  
    其他格式，必须；  
说明：request的body部分的数据编码格式由header部分的Content-Type指定；

**四、解决了angular问题之后，发现jquery按照原来的方式提交post请求会报错（错误码415）。**

如下方式可以解决jquery提交问题：

[?](http://www.jb51.net/article/76147.htm)

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9  10 | $.ajax({    url:'/carlt/loginForm',    method: 'POST',    contentType:'application/json;charset=UTF-8',    data:JSON.stringify({"name":"jquery","password":"pwd"}),    dataType:'json',    success:function(data){      //...    }  }); |

json对象转json字符串：JSON.stringify(jsonObj);