目录

[1. 常见设计原则 1](#_Toc492033311)

[2. 常见的设计模式（了解） 2](#_Toc492033312)

[ angularJS 2](#_Toc492033313)

[1、ng的基本语法 3](#_Toc492033314)

[2、ng的指令 3](#_Toc492033315)

[3. MVC 4](#_Toc492033316)

[4. 自定义指令app.directive 5](#_Toc492033317)

[5. 在webStrom去创建一个指定的模板 6](#_Toc492033318)

[6. 双向数据绑定 7](#_Toc492033319)

[7. 模型数据的监听-$scope.$watch 9](#_Toc492033320)

[8. 过滤器 9](#_Toc492033321)

[9. 自定义过滤器app.filter 11](#_Toc492033322)

[10. ng中内置的常用函数 12](#_Toc492033323)

[11. 服务的基本用法 12](#_Toc492033324)

[12. 控制器之间通信$broadcast/$emit 13](#_Toc492033325)

[13. 常见的内置服务 14](#_Toc492033326)

[14. $interval $timeout 15](#_Toc492033327)

[15. $http服务 16](#_Toc492033328)

[16. 自定义服务的创建和使用 factory/service/constant/value 17](#_Toc492033329)

[17. 依赖注入 19](#_Toc492033330)

[18. 依赖注入的工作机制(了解) 20](#_Toc492033331)

[19. 模块化设计多模块之间的相互引用 20](#_Toc492033332)

[20. ng内置模块 ngRoute基本用法 21](#_Toc492033333)

[21. ng常见的表单验证方式 24](#_Toc492033334)

[22. animate 动画模块。 25](#_Toc492033335)

[23. ngInclude 动态引入代码片段 25](#_Toc492033336)

**js：DOM驱动**

**--》**

**ng：数据驱动**

**ng的四大特征：**

①基于MVC的设计模式

②双向数据绑定

③依赖注入

④模块化设计

### 常见设计原则

**1、YAGNI 不写不需要的代码**

You aren't gonna need it

不要把未来才会出现的需求或者你认为会出现的需求，添加现有的工程中

**2、KISS 代码越简单越好**

keep it simple and stupid

项目开发 --》迭代和维护

**3、DRY 不要重复自己**

Don't Repeat Yourself

善于对代码进行封装和重用

**4、高内聚 低耦合**

内聚：一个组件内部聚合的关系。

耦合：不同组件之间的关系；将耦合度降低的过程：解耦

**5、SRP 单一责任原则**

single responsibility principle

每一个文件、模块、组件尽量做到只干一件事情

**6、OCP 开闭原则**

open closed principle

对于代码的升级、扩展是开放的是允许的；

对于原有代码的修改是关闭

**7、LKP 最少知识法则**

封装的模块要小，尽量做到只与自己相关。

### 常见的设计模式（了解）

（23种设计模式）

总体来说设计模式分为三大类：

①创建型模式 共五种：工厂方法模式、抽象工厂模式、单例模式、建造者模式、原型模式。

②结构型模式，共七种：适配器模式、装饰器模式、代理模式、外观模式、桥接模式、组合模式、享元模式。

③行为型模式，共十一种：策略模式、模板方法模式、观察者模式、迭代子模式、责任链模式、命令模式、备忘录模式、状态模式、访问者模式、中介者模式、解释器模式。

**了解常用的设计模式：**

**工厂方法模式、单例模式、适配器模式、组合模式、迭代子模式**

## angularJS

AngularJS是一个Google开发的开源的js框架

①是基于MVC的设计模式进行开发

②双向数据绑定

③依赖注入

④模块化设计

### 1、ng的基本语法

//双花括号（叫做mustache、interpolation）

<any>{{表达式}}</any>

### 2、ng的指令

* + **ngApp指令：**

①启动AngularJS

②限定作用范围

③ngApp在一个ng应用中只调用一次，第一次调用是生效的

* + **ngInit指令：初始化变量**

①初始化变量(无需加上var关键字)

②初始化多个变量

<any ng-init="name='zhangsan';age=10"></any>

* + **ngRepeat**

语法1：

<any ng-repeat="tmp in myList"></any>

语法2：

<any ng-repeat="(key,value) in myList"></any>

特殊属性：$index/$first/$middle/$last/$even/$odd

**注意事项：**

当有重复数据时，指定track by $index

<ul ng-init="list=[1,2,3,3,5]">

<li **ng-repeat="tmp in list track by $index"**> {{tmp}}</li>

</ul>

* + **ngIf**

语法：

<any ng-if="表达式"></any>

作用：

根据表达式的结果来判断 是要挂载到DOM还是要从DOM中移除掉。

* + **其他常见指令**

ngApp ngInit ngRepeat ngIf ngShow ngHide

ngChecked ngDisabled...

**ngIf和ngShow/ngHide的区别什么？**

ngIf控制的DOM元素时插入还是删除，

ngSHow/ngHide控制显示或者隐藏

ngSrc

<img ng-src="img/{{imgUrl}}"/>

ngDisabled

禁用/非使能

ngChecked

控制单选复选 默认是否被选中

* + **ngBind**

将表达式的结果输出当前元素的innerHtml中，可以解决双花括号带来的闪烁问题。

<**div ng-controller="myCtrl"**>  
 *<!-- 会有一个潜在的问题，就是在ng未启动称之前会显示字符串，会带来一个闪烁情况-->* <**p**>{{myAge}}</**p**>  
 *<!-- ngBind可以解决：将表达式的结果输出当前元素的innerHtml中-->* <**p ng-bind="myAge"**></**p**>  
</**div**>  
<**script**>  
 *//创建模块* **var *app*** = ***angular***.module(**'myApp'**, [**'ng'**]);  
 *//创建一个控制器* ***app***.**controller**(**'myCtrl'**, **function** ($scope) {  
 *//指定一个测试语句* ***console***.log(**'in ctrl func'**);  
 $scope.**myAge**=20;  
 }  
 );

### MVC

MVC的工作原理？？

在视图中（View）进行用户的交互，会到控制器（controller）中对数据（Model）进行一些增删改查的操作，由于将Model绑定到了View，所以数据发生了变化，视图就会跟着更新

M：model 模型数据

V：view 视图

C：controller 控制器

<**html ng-app="myModule"**>  
<**head lang="en"**>  
 <**meta charset="UTF-8"**>  
 <**script src="js/angular.js"**></**script**>  
 <**title**></**title**>  
</**head**>  
<**body**>  
*<!--调用控制器-->*<**div ng-controller="myCtrl"**>  
 <**ul**>  
 <**li ng-repeat="dish in dishList"**>  
 <**img ng-src="img/{{dish.imgUrl}}" alt=""**/>  
 <**h1**>{{**dish**.dishName}}</**h1**>  
 <**p**>{{**dish**.**price**}}</**p**>  
 </**li**>  
 </**ul**>  
 <**button ng-click="addToList()"**>  
 加载更多  
 </**button**>  
</**div**>  
<**script**>  
 *//创建模块 module* **var *app*** = ***angular***.module(**'myModule'**,[**'ng'**]);  
 *//创建控制器* ***app***.controller(**'myCtrl'**, **function** ($scope) {  
 *//初始化一个对象数组* $scope.**dishList** = [  
 {**imgUrl**:**'1.jpg'**,**dishName**:**'鱼香肉丝'**,**price**:20},  
 {**imgUrl**:**'2.jpg'**,**dishName**:**'宫保鸡丁'**,**price**:21},  
 {**imgUrl**:**'3.jpg'**,**dishName**:**'尖角肉丝'**,**price**:22},  
 ];  
 *//能够向菜品列表中添加一条数据* $scope.addToList = **function** () {  
 $scope.dishList.push(  
 {**imgUrl**:**'4.jpg'**,**dishName**:**'肉夹馍'**,**price**:10}  
 )  
 }  
 })

### 自定义指令app.directive

ng内置的指令偏向于工具化，为了满足企业开发的需求，如何完成指令的自定义和使用。

**自定义指令的创建：**

app.directive('指令名称'，function(){

return {

配置

}

});

**创建自定义指令在配置对象都有哪些属性**？

template：'要去显示的模板内容'

restrict：

‘EMCM’(E:Element A:Attribute C:Class M:Comment)

replace:true

**在使用指令 参数的传递**

①在指令内部完成参数的接收和调用

app.directive('myDirective',function(){

return {

template:'<h1>{{myValue}}</h1>'

scope:{

myValue:'@' } }})

②调用指令时 发送参数<my-directive my-value="值"></my-directive>

*<!--作为element去使用自定义指令-->  
<!--通过指定的testName传递值-->*<**div ng-controller="myCtrl"**>  
 <**my-directive my-name="{{name}}"**></**my-directive**>  
</**div**>  
<**script**>  
 *// 创建模块* **var *app*** = ***angular***.module(**'myApp'**, [**'ng'**]);  
 *// 创建一个控制器、* ***app***.**controller**(**'myCtrl'**, **function** ($scope) {  
*// 初始化变量* $scope.**name**=**"zhangsan"** });  
 *// 创建一个自定义指令* ***app***.directive(**'myDirective'**,**function**(){  
 **return**{  
 **template**:**'<p>Hello {{myName}}</p>'**,  
 **scope**:{  
*// @意味着该指令在被调用时，读取test-name属性所对应的值保存在tesName中* **myName**:**'@'** }}  
 });  
</**script**>

### 在webStrom去创建一个指定的模板

file->new ->Edit File Templates -> + ->指定模板的名称以及结尾，把内容粘贴到文本区域

### 双向数据绑定

**ng的四大特征：**

①基于MVC的设计模式

②双向数据绑定

③依赖注入

④模块化设计

**1、什么是双向数据绑定**

方向1：

将数据绑定到视图

方向2：

将视图中用户操作的结果绑定到数据

2、方向1

将数据绑定到视图

有两大类方式

①双花括号

②常见的ng指令：

ngRepeat ngIf ngSrc ngShow ngHide ngChcked ngDiasbaled ngBind ngClass ngStyle...

3、方向2

①将视图中表单控件用户操作的结果 绑定到数据

只有一种实现方向2的绑定的方式：ngModel

②ng中提供了一个$watch的方法，可以来实现对于模型数据的监听，从而来实现当数据变化时，执行指定的操作

**$scope.$watch('要监听的模型数据的名字',**

function(newValue,oldValue){

//当监听的模型数据发生变化时 会调用的方法

})

双向数据绑定：在一个ng的应用中，既有方向1的绑定 又有方向2的绑定。

例子1：

<**div ng-controller="myCtrl"**>  
 *<!-- 当我们选择某一选项时，能够获取操作的结果，  
 改变img标签要显示的图片-->* <**select ng-model="myImage"**>  
 <**option value="1.jpg"**>水仙花</**option**>  
 <**option value="2.jpg"**>百合</**option**>  
 <**option value="3.jpg"**>郁金香</**option**>  
 </**select**>  
 *<!--将变量绑定到视图：方向1的绑定 -->* <**img ng-src="img/{{myImage}}" alt=""**/>  
</**div**>  
<**script**>  
 *//创建模块* **var *app*** = ***angular***.module(**'myApp'**, [**'ng'**]);  
 *//创建一个控制器* ***app***.**controller**(  
 **'myCtrl'**,  
 **function** ($scope) {  
 *//指定一个测试语句* ***console***.log(**'in ctrl func'**);  
 *//初始化默认要显示的图片* $scope.**myImage** = **'1.jpg'**;  
 }  
 );  
</**script**>

例子二：

<**div ng-controller="myCtrl"**>  
 *<!-- 当勾选了复选框，按钮就可以点击，否则禁用-->  
 <!-- 方向2的绑定：将复选框是否被选中的结果 绑定到 模型数据-->* <**input  
 ng-model="isAgree"  
 type="checkbox"**/> 是否同意本站协议  
 <**br**/>  
 *<!-- 方向1的绑定：将数据绑定到视图-->* <**button ng-disabled="!isAgree"**>注册</**button**>  
 *<!-- ngModel 双向数据绑定指令-->* <**input type="text" ng-model="myInput"**/>  
</**div**>  
<**script**>  
 *//创建模块* **var *app*** = ***angular***.module(**'myApp'**, [**'ng'**]);  
 *//创建一个控制器* ***app***.**controller**(  
 **'myCtrl'**,  
 **function** ($scope) {  
 *//指定一个测试语句* ***console***.log(**'in ctrl func'**);  
 *//对于方向2绑定的数据* $scope.**isAgree** = **true**;  
 $scope.**myInput** = **"zhangsan"** }  
 );  
</**script**>

### 模型数据的监听-$scope.$watch

<**div ng-controller="myCtrl"**>  
 *<!--将视图当中用户操作的结果绑定到数据：方向2实现方式：ngModel  
 ngModel指令可以用在select/chekbox/textarea  
 -->* <**input type="text" ng-model="myText"**/>  
 *<!--将数据绑定到视图：方向1的绑定-->  
 <!--①花括号 ②常见ng指令-->* <**h1**>{{**myText**}}</**h1**>  
 <**textarea ng-model="myMsg"**></**textarea**>  
 <**p**>{{"当前用户输入的是："+myMsg}}</**p**>  
</**div**>  
<**script**>  
 *// 创建模块* **var *app*** = ***angular***.module(**'myApp'**, [**'ng'**]);  
 *// 创建一个控制器、* ***app***.controller(**'myCtrl'**, **function** ($scope) {  
*// 对要是使用的任何数据都做初始化、* $scope.**myText**=**""**;  
*// 监听模型数据变化* $scope.$watch(**"myText"**,**function**(newValue,oldValue){  
 **console**.log(**""**+newValue+oldValue)  
 **console**.log($scope.**myText**)  
 });  
 });  
</**script**>

### 过滤器

* + **过滤器：实现对于数据的筛选、过滤和格式化。**

支持多重过滤

{{表达式 | 过滤器：参数 | 过滤器}}

currency 货币样式的格式化

date 日期样式

uppercase/lowercase 大小写转换

orderBy:'要按照谁排序':true

默认是升序的，指定了第二个参数为true 就是降序处理

number：限定显示小数点后几位

{{ myValue | number:2}}

3.1415926

3.14

limitTo：限定数组元素的个数

支持多重过滤

{{表达式 | 过滤器：参数 | 过滤器}}

<**div ng-controller="myCtrl"**>  
 <**p**>{{**price**}}</**p**>  
 *<!-- 调用过滤器-->* <**p**>{{**price** | currency}}</**p**>  
 *<!-- 调用过滤器并指定参数-->* <**p**>{{**price** | currency:'￥'}}</**p**>  
  
 <**p**>{{**nowDate** | date:'MM-dd'}}</**p**>  
  
 <**p**>{{**userName** | uppercase}}</**p**>  
 *<!-- 把myList数组中数据 绑定到列表去显示-->* <**ul**>  
 <**li ng-repeat="tmp in myList | orderBy:'score':true"**>  
 {{**tmp**.name+":"+**tmp**.**score**}}  
 </**li**>  
 </**ul**>  
 *<!-- number 限定显示小数点后几位-->* <**p**>{{**myValue**}}</**p**>  
 <**p**>{{**myValue** | number:2}}</**p**>  
</**div**>  
<**script**>  
 *//创建模块* **var *app*** = ***angular***.module(**'myApp'**, [**'ng'**]);  
 *//创建一个控制器* ***app***.**controller**(  
 **'myCtrl'**,  
 **function** ($scope) {  
 *//指定一个测试语句* ***console***.log(**'in ctrl func'**);  
 $scope.**price** = 1234;  
 $scope.**nowDate** = **new** Date();  
 $scope.**userName** = **"Reese"**;  
 $scope.**myList** = [  
 {**age**:10,**name**:**'Michael'**,**score**:80},  
 {**age**:20,**name**:**'Lincoln'**,**score**:90},  
 {**age**:30,**name**:**'Tbag'**,**score**:82}  
 ];  
 $scope.**myValue** = 3.1415926;  
 }  
 );  
</**script**>

### 自定义过滤器app.filter

过滤器本质其实就是一个有参数有返回方法。

3.2.1 创建过滤器

app.filter('过滤器的名称',function(){

return function(myInput,arg){

return '处理后的数据'

}

})

myInput指的是通过管道所传递来的数据

myFilter:true ,arg指是就是冒号跟上的参数

<**div ng-controller="myCtrl"**>  
 <**p**>{{**price** | **myCurrency**:'￥'}}</**p**>  
</**div**>  
<**script**>  
 *//创建模块* **var *app*** = ***angular***.module(**'myApp'**, [**'ng'**]);  
 *//创建一个控制器* ***app***.**controller**(  
 **'myCtrl'**,  
 **function** ($scope) {  
 *//指定一个测试语句* ***console***.log(**'in ctrl func'**);  
 $scope.**price** = 30;  
 }  
 );  
 *//创建一个自定义的过滤器* ***app***.filter(  
 **'myCurrency'**, **function** () {  
 *//返回的方法就是过滤器（过滤器的本质就是一个方法有参数有返回值）* **return function** (myInput,arg1) {  
 **return** arg1+myInput;  
 }  
 }  
 );  
</**script**>

### ng中内置的常用函数

Json是客户端与服务器进行数据交互的载体。

angular.toJson 序列化：将一个对象或者数组转换成一个json格式的字符串，方便发给服务器端

angular.fromJson 反序列化：将一个json格式的字符串转换成一个对象或者数组

<**script**>  
 *//创建模块* **var *app*** = ***angular***.module(**'myApp'**, [**'ng'**]);  
 *//创建一个控制器* ***app***.**controller**(  
 **'myCtrl'**,  
 **function** () {  
 *//指定一个测试语句* ***console***.log(**'in ctrl func'**);  
 **var** car = {**name**:**'BBA'**,**price**:20};  
 *//序列化:将一个对象或者数组转换为一个json格式的字符串* **var** jsonStr = ***angular***.*toJson*(car);  
 ***console***.log(jsonStr);  
 *//反序列化：将一个json格式的字符串转换为对象或者数组，方便进行操作* **var** myObj =  
 ***angular***.*fromJson*(jsonStr);  
 ***console***.log(myObj);  
 ***console***.log(***angular***.*uppercase*(**'Michael'**));  
 }  
 );  
</**script**>

### 服务的基本用法

以$location port方法。

步骤1：将服务注入到ng对象

app.controller('myCtrl',function($scope,$location){})

步骤2：按照手册去调用 该服务 所提供的数据或者方法

app.controller('myCtrl',function($scope,$location){

//调用服务中所封装好的方法或者数据

$location.port();

})

<**div ng-controller="myCtrl"**>  
 <**h1**>{{**nowPort**}}</**h1**>  
 <**button ng-click="getPort()"**>  
 获取端口号  
 </**button**>  
</**div**>  
<**script**>  
 *//创建模块* **var *app*** = ***angular***.module(**'myApp'**, [**'ng'**]);  
  
 *//创建一个控制器* ***app***.**controller**(  
 **'myCtrl'**,  
 **function** ($scope,$location) {  
 *//指定一个测试语句* ***console***.log(**'in ctrl func'**);  
 $scope.**nowPort** = 0;  
 $scope.getPort = **function** () {  
 *//通过$location服务的port方法得到端口号* $scope.**nowPort** =  
 $location.**port**();  
 }  
 }  
 );  
</**script**>

### 控制器之间通信$broadcast/$emit

$rootScope:是在ng启动时就已经初始化好了，id是1，所有的$scope都是子作用域对象

通信方式1：

借助于$rootScope：子能够读父的东西

通信方式2：

控制器之间的嵌套 来手工的实现父与子的关系

注意事项：子读父的数据和方法，但是父并不能直接读子的数据和方法。

通信方式3：

借助于事件：事件的绑定和触发

//绑定事件（在需要接受数据的地方去绑定）

$scope.$on('myEvent',function(event,data){

//data是传过来的数据

})

//触发事件（在需要发送参数去触发事件）

$scope.$broadcast('myEvent',data)//父-》子

$scope.$emit('myEvent',data)// 子-》父

<**div ng-controller="parentCtrl"** >  
 <**h1** >这是小头爸爸</**h1**>  
 <**button ng-click="tellSon()"**>告诉儿子一句话</**button**>  
 <**hr**/>  
 <**div ng-controller="sonCtrl"**></**div**>  
 <**h1**>这是大头儿子</**h1**>  
 <**p**>{{"方式1 "+**fatherName**}}</**p**>  
</**div**>  
<**script**>  
 *// 创建模块* **var *app*** = ***angular***.module(**'myApp'**, [**'ng'**]);  
 *// 创建一个控制器、* ***app***.controller(**'parentCtrl'**, **function** ($scope) {  
*// 指定一个测试语句* **console**.log(**"in parenctrl func"**);  
 $scope.**fatherName**=**"小头爸爸"***// 通过$broadcast 向儿子触发对应的事件、* $scope.tellSon=**function**(){  
 $scope.$broadcast(**'recvFromFather'**,**'你小子在干嘛呢'**)  
 }  
 });  
 ***app***.controller(**'sonCtrl'**, **function** ($scope) {  
*// 指定一个测试语句* **console**.log(**"in sonCtrl func"**);  
 $scope.$on(**'recvFromFather'**,**function**(event,data){  
 **console**.log(**"接收的父亲所传递来的事件 "**);  
 **console**.log(arguments);  
 **console**.log(data)  
 **console**.log(event)  
 });  
 });  
</**script**>

### 常见的内置服务

$scope $rootScope $location $interval $timeout $http ....

双向数据绑定背后的工作机制：

每一次进行数据绑定，ng框架都会自动的生成一个监听的处理函数（修改DOM）；ng框架是如何知道数据发生了变化呢？？ 背后有一个循环（$digest循环），在不断的判断数据是否变化，如果变化执行处理函数，完成DOM的更新。

$scope.$digest();//当前$scope中数据是否发生了变化

$scope.$apply();//直接触发$rootScope开始所有的作用域范围对象，每一个$scope中数据是否发生了变化

### $interval $timeout

$interval = setInterval + $scope.$digest();

//启动定时器

promise = $interval(func,delay)

//结束掉定时器

$interval.cancel(promise);

//启动定时器

promise = $timeout(func,delay)

//结束掉定时器

$timeout.cancel(promise);

<**div ng-controller="myCtrl"**>  
 <**h1**>{{"count is "+**count**}}</**h1**>  
</**div**>  
<**script**>  
 *// 创建模块* **var *app*** = ***angular***.module(**'myApp'**, [**'ng'**]);  
 *// 创建一个控制器、* ***app***.controller(**'myCtrl'**, **function** ($scope,$interval) {  
*// 指定一个测试语句* **console**.log(**"in ctrl func"**);  
 $scope.**count**=0;  
*// setInterval(function () {  
// console.log('in interval func');  
// $scope.count++;  
// //$scope.$digest();  
// $scope.$apply();  
// },300)* $interval(**function** () {  
 $scope.**count**++  
 },300)  
 });  
</**script**>

### $http服务

普通的用法：

$http({method:'get',url:''}).success().error()

快捷用法：

$http.get().success().error()

$http.post().success().error()

**发起post请求时，如果要传递参数，请记得设置请求头：（全局设置）**

app.run(function ($http) {

$http.defaults.headers.post =

{'Content-Type':'application/x-www-form-urlencoded'};

})

**局部设置：**

<**div ng-controller="myCtrl"**>  
 <**input type="text" ng-model="userName" placeholder="请输入用户名"**/>  
 <**button ng-click="sendToServer()"**>提交给服务器</**button**>  
 <**p**>{{'服务器返回的数据是：'+***result***.tip}}</**p**>  
</**div**>  
<**script**>  
 *// 创建模块* **var *app*** = ***angular***.module(**'myApp'**, [**'ng'**]);  
 *// 创建一个控制器、* ***app***.controller(**'myCtrl'**, **function** ($scope,$http) {  
*// 指定一个测试语句* **console**.log(**"in ctrl func"**);  
  **$http.defaults.headers.post={'Content-Type':'application/x-www-form-urlencoded'};** *// 定义向服务器端发请求的方法* $scope.sendToServer=**function**(){  
 //传递多个参数用&符号链接，如：”name=” +$scope.**userName &”upwd=1234”** $http.post(**'data/server.php'**,**"name="**+$scope.**userName**).success(**function**(data){  
 **console**.log(data)  
 $scope.**result**=data;  
 }).error(**function**(){  
 **console**.log(**'post failed'**)  
 });  
 }  
 });

### 自定义服务的创建和使用 factory/service/constant/value

服务存在的意义???

需要反复用到的功能封装成一个服务，

（方便代码的复用，提高代码的编写速度，降低测试难度）

（指令存在的意义？？是扩展html标签）

自定义指令app.directive()

自定义过滤器 app.filter()

自定义控制器 app.controller()

自定义服务： 服务的本质就是一个单例对象，自定义服务的过程就是配置对象的过程

自定义服务与内置服务 用法一样：①在ng对象中注入该服务 ②按照服务所提供的数据或者方法来调用

* 1. **factory 必须返回对象**
  2. **service 构造函数**
  3. **constant 保存常量服务**
  4. **value 保存变量服务**
  + **通过factory方法创建服务**

app.factory('服务名称',function(){

return {

isDev:true

log:function(){}

}

})

*//创建模块***var *app*** = **angular**.*module*(**'myApp'**, [**'ng'**]);  
*//通过factory创建自定义服务****app***.**factory**(  
 **'$LogService'**,  
 **function** () {  
 **return** {  
 **isDev**:**false**,  
 log: **function** (msg) {  
 **if**(**this**.**isDev**)  
 {  
 **console**.log(msg);  
 }  
 }  
 }  
 });  
*//创建一个控制器****app***.**controller**(  
 **'myCtrl'**,  
 **function** ($scope,$LogService) {  
 *//指定一个测试语句* **console**.log(**'in ctrl func'**);  
 *//给按钮指定一个单击处理函数  
 //在该处理函数中通过调用服务实现控制台的输出* $scope.handleClick = **function** () {  
 $LogService.log(  
 **"Hello $logService"**)  
 }  
 }  
);

* + **通过service方法创建一个服务**

app.service('服务名称',function(){

this.变量名称 = 值/方法

})

***app***.**service**(  
 **'$myWindow'**, **function** () {  
 *//通过service创建服务的时候，func是一个构造函数* **this**.showAlert = **function** (msg) {  
 alert(msg);  
 }  
 }  
)

* + **value创建一个变量服务 constant创建一个常量服务**

//创建一个变量服务，经常变化的数据

app.value('$myValues',{

token:'AOSKDHJKSD',

isLogin:true

});

* + **//创建一个常量服务，保存不会变化的数据**

app.constant('$myConfig',{

serverUrl:'54.34.24.14',

devPort:8081,

proPort:80

})

### 依赖注入

文件压缩是在项目进行服务器端部署必不可少的环节：①删除不必要的注释 ②删除没有的空白字符 ③尽量简化变量的名称

方式1：

java -jar \*\*\*.jar \*\*.js > \*\*.min.js

方式2：

借助于webStorm

依赖注入的3种方式：

4.1 推断式（猜测）依赖注入（no）

app.controller('',function(1e,$http){})

推断式依赖注入在文件压缩时 是有问题的。

4.2 标记式依赖注入

//给需要依赖的服务做一个标记

ctrlFunc.$inject = ["$scope","$http"];

注意事项：数组中服务的名字的先后顺序，得和在创建ng对象时function注入的顺序一致

//var ctrlFunc = function ($scope,$http){}

4.3 行内式依赖注入（强制必须按照该方式）

在创建一个ng对象时，ng允许将一个字符串构成的数组做一个参数，同时在数组中的最后一个元素的位置必须是一个func

*//创建模块***var** app = angular.module(**'myApp'**, [**'ng'**]);  
*//创建一个控制器*app.controller(  
 **'myCtrl'**,  
 [**"$scope"**,**"$http"**,**function** ($scope,$http) {  
 *//指定一个测试语句* console.log(**'in ctrl func'**);  
 $scope.count = 100;  
 }]  
);

### 依赖注入的工作机制(了解)

当创建一个ng对象（比如说控制器对象），如果需要$scope,将$scope服务直接注入进来就可以使用该服务，注入进来为什么就可以去用这个对象？

$injector 注入器，当需要注入服务的时候，ng这个框架会通过$injector在当前的服务的注册列表中去寻找该服务，如果该服务的名字存在列表中，ng框架就会通过实例化该服务，拿到该服务的对象，注入到对应的ng对象中，使服务可以去使用

简化：

查找-》创建

$injector常见API：

has('服务名称') 判断指定的服务是否存在

get('服务名称') 得到指定的服务的实例

### 模块化设计多模块之间的相互引用

面试题：

一个ng的模块都可以包含哪些？

①控制器：处理业务逻辑

②服务：把需要反复用到的功能封装，方便复用

③指令：扩展html标签

④过滤器：实现对于数据的筛选、过滤、格式化

在创建一个基于ng项目时候，创建多个模块，比如userModule,orderModule,deliveryModule..,多模块之间难免需要数据的通信，如何在一个模块中调用另外模块中的东西？？？

如果A模块需要调用B模块中的某些东西，在创建A模块，在它的第二个参数所对应的数组中写上B模块的名字。

angular.module('orderModule',['ng','userModule'])

<**div ng-controller="orderCtrl"**>  
 <**button ng-click="submitOrder()"**>下单</**button**>  
</**div**>  
<**script**>  
 **var *userModule***= ***angular***.module(**'userModule'**,[**'ng'**]);  
*// 创建一个$userLogin的服务模块* ***userModule***.**factory**(**'$userLogin'**,**function**(){  
 **return** {  
 isUserLogin: **function** () {  
 **return true**;  
 }  
 }  
 });  
 **var *orderModule***=***angular***.module(**'orderModule'**,[**'ng'**,**'userModule'**]);  
 ***orderModule***.controller(**'orderCtrl'**,[**'$scope'**,**'$userLogin'**, **function** ($scope,$userLogin) {  
 $scope.submitOrder= **function** () {  
*// 判断用户是否登录* **var** isLogin= $userLogin.isUserLogin();  
 **console**.log(isLogin);  
 }  
 }]);  
</**script**>

### ng内置模块 ngRoute基本用法

ngRoute模块时属于angularJS众多模块中的一个，通过ngRoute来实现SPA（Single Page Application）：ngRoute帮助实现路由地址和代码片段的绑定关系、多个页面的跳转、传参

3.1 SPA的介绍

SPA:单一完整页面的应用程序，整个应用程序中只有一个完成的页面，其余的都是代码片段

典型的SPA：邮箱、后台管理。。

http://localhost:63342/day4/demo03\_SPA.html#/myRegister

SPA的工作原理：

http://127.0.0.1/index.html#/inbox

①解析完整的页面index.html

②解析#后所对应的inbox（路由地址）

③从叫做路由词典的配置找到inbox所对应的配置对象，找到真正要加载的代码片段的地址

④发起异步请求

⑤将请求来的代码片段插入完整的页面的容器中

* + **使用ngRoute来实现SPA的基本步骤**

①在项目中引入angular-route.js

②在创建自定义模块，指定依赖于ng、ngRoute

③通过ngView指定一个容器 用来盛放代码片段

<div ng-view></div>

④创建代码片段

⑤配置路由词典

app.config(function($routeProvider){

//通过when方法添加一条路由信息，

//第一个参数是路由地址（必须是/开头的）

//第二个参数是配置对象

$routeProvider

.when('/myLogin',{

templateUrl:'tpl/login.html'

})

.otherwise({redirectTo:'/myLogin'})

})

⑥测试路由地址配置是否正确

可以通过修改地址栏中的路由地址测试配置是否生效

* + **ngRoute实现页面的跳转**

方式1：直接修改地址栏中的路由地址

方式2：js--》依赖于$location

$location.path('路由地址');

举个例子：

$location.path('/elmMain');

方式3：超链接

<a href="#/路由地址"></a>

举个例子

<a href="#/elmLogin">退出登录</a>

* + **ngRoute跳转时传递参数**

①明确发送方、接收方

②接收方做准备工作

2.1 配置接收方的路由地址

$routeProvider.when('/myOrder',{})

-->

$routeProvider.when('/myOrder/:id',{})

2.2 给接收方的页面 指定控制器，注入$routeParams(注入进来就保存了传递来的数据)

③发送参数

a href="#/myOrder/10"

$location.path('/myOrder/10')

<**div ng-view**></**div**>  
  
<**script**>  
 **var *app*** = ***angular***.module(**'myApp'**, [**'ng'**, **'ngRoute'**]);  
 *//配置路由词典* ***app***.config(  
 **function** ($routeProvider) {  
 $routeProvider  
 .when(**'/list'**, {  
 **templateUrl**: **'tpl/myList.html'**,  
 **controller**:**'listCtrl'** })  
 .*when*(**'/detail/:index'**, {  
 **templateUrl**: **'tpl/myDetail.html'** })  
 .otherwise({**redirectTo**: **'/list'**})  
 }  
 )  
 ***app***.controller(**'detailCtrl'**,  
 [**'$scope'**, **'$routeParams'**,  
 **function** ($scope, $routeParams) {  
 ***console***.log($routeParams);  
 $scope.**result** =  
 $routeParams.index  
 }  
 ]  
 );  
 ***app***.controller(**'listCtrl'**,  
 [**'$scope'**, **'$location'**,  
 **function** ($scope, $location) {  
 $scope.jump =  
 **function** (desPath) {  
 $location.path(desPath);  
 }  
 }  
 ]  
 )  
</**script**>

### ng常见的表单验证方式

ng提供了属性帮助完成表单验证。

$dirty

$pristine

$valid

$invalid

$error

<**div ng-controller="myCtrl"**>  
 <**form name="myForm"**>  
 <**input  
 placeholder="输入用户名"  
 name="userName"  
 ng-model="uName"  
 required  
 type="text"**/>  
 <**span ng-show=  
 "myForm.userName.$invalid &&  
 myForm.userName.$dirty"**>  
 请输入用户名  
 </**span**>  
 <**br**/>  
 <**input  
 name="userPwd"  
 ng-model="uPwd"  
 placeholder="请输入密码"  
 type="text" required**/>  
 <**span ng-show=  
 "myForm.userPwd.$invalid &&  
 myForm.userPwd.$dirty"**>  
 请输入密码  
 </**span**>  
 <**br**/>  
 <**button ng-disabled="myForm.$invalid"**>  
 登录  
 </**button**>  
 </**form**>  
</**div**>  
<**script**>  
 *//创建模块* **var *app*** = ***angular***.module(**'myApp'**, [**'ng'**]);  
 *//创建一个控制器* ***app***.controller(  
 **'myCtrl'**,  
 **function** () {  
 *//指定一个测试语句* ***console***.log(**'in ctrl func'**);  
 }  
 );  
</**script**>

### animate 动画模块。

①引入对应的angular-animate.js文件

②创建模块指定依赖于 ngAnimate

③使用模块。

ng-enter

ng-enter-active

ng-leave

ng-leave-active

### ngInclude 动态引入代码片段

*<!--页头-->*<**div ng-include="'tpl/include/header.html'"**></**div**>  
<**h1**>这是详情页面detail</**h1**>  
*<!--页尾-->*<**div ng-include="'tpl/include/footer.html'"**></**div**>