# 介绍

* AngularJS是一款由Google公司开发维护的前端MVC框架，其克服了[HTML](http://baike.baidu.com/view/692.htm)在构建应用上的诸多不足，从而降低了开发成本提升了开发效率。
* 基于javascript开发的客户端应用框架，使我们可以更加快捷，简单的开发web应用。
* 诞生于2009年，后来被google收购，用在了很多项目中。
* 适用于CRUD应用或者SPA单页面网站的开发。

## 特点

AngularJS与jQuery是有一定的区别的，jQuery更准确来说只是一个类库（类库指的是一系列函数的集合）以DOM做为驱动（核心），而AngularJS则一个框架（诸多类库的集合）以数据和逻辑做为驱动（核心）。

框架对开发的流程和模式做了约束，开发者遵照约束进行开发，更注重的实际的业务逻辑。

AngularJS有着诸多特性，最为核心的是：模块化、双向数据绑定、语义化标签、依赖注入等。

与之类似的框架还有BackBone、KnockoutJS、Vue、React等。

## 下载

1、通过[AngularJS](https://angularjs.org/)官网下载，不过由于国内特殊的国情，需要翻墙才能访问。

2、通过npm下载，npm install angular

3、通过bower下载，bower install angular

bower是什么？

## 体验AngularJS

<div class="container" ng-app>

<input type="text" ng-model="xianjs">

<h1>{{xianjs}}</h1>

</div>

## MVC

MVC是一种开发模式，由模型（Model）、视图（View）、控制器（Controller）3部分构成，采用这种开发模式为合理组织代码提供了方便、降低了代码间的耦合度、功能结构清晰可见。

模型（Model）一般用来处理数据（读取/设置），一般指操作数据库。

视图（View）一般用来展示数据，比如通过HTML展示。

控制器（Controller）一般用做连接模型和视图的桥梁。



MVC更多应用在后端开发程序里，后被引入到前端开发中，由于受到前端技术的限制便有了一些细节的调整，进而出现了很多MVC的衍生版（子集）如MVVM、MVW、MVP、MV\*等。

## MVC和SPA

### MVC

MVC是一种优秀的模块化开发思想，全名是Model View Controller。

Model（模型）是应用程序中用于处理应用程序数据逻辑的部分。

View（视图）是应用程序中处理数据显示的部分。

Controller（控制器）是应用程序中处理用户交互的部分。

### SPA

1、single-page application是一种特殊的Web应用。它将所有的活动局限于一个Web页面中，仅在该Web页面初始化时加载相应的HTML、JavaScript、CSS。一旦页面加载完成，SPA不会因为用户的操作而进行页面的重新加载或跳转，而是利用JavaScript动态的变换HTML（采用的是div切换显示和隐藏），从而实现UI与用户的交互。

2、简单来说SPA的网页只有一个页面，而这个网页的实际方式要能够回应使用者所使用的各种装置并且赋值使用者在电脑上使用软件的体验，让使用者可以更容易和有效的使用网站。按照正常情况下，我们会在一个页面中链接到其他的很多个页面，进行页面的跳转，但是如果使用单页面应用的话，我们始终在一个页面中，通常使用a标签的描点来实现。

(1)好处

1、由于避免了页面的重新加载，SPA可以提供较为流畅的用户体验。得益于Ajax，可以实现无跳转刷新，由于与浏览器的history机制，可以使用hash的b变化从而可以实现推动界面变化。

2、只要使用支持HTML5和CSS3的浏览器就可以执行复杂的SPA,因此，开发人员不必为了写SPA网站而特别学习另一个开发方式，而使用者也不额外安装软件，所以，让开发SPA网页程序的入门和使用门槛降低不少。

(2)缺点

以SPA方式开发的网站不容易管理也不够安全。

因为没了一页一页的网页给搜索引擎的爬虫来爬，所以，在搜索引擎最佳化（SEO）的工作上，需要花费额外的功夫。

因为没有换页，需要自定义状态来取代传统网页程序以网址来做判断。

# 模块化

使用AngularJS构建应用（App）时是以模块化（Module）的方式组织的，即将整个应用划分成若干模块，每个模块都有各自的职责，最终组合成一个整体。

采用模块化的组织方式，可以最大程度的实现代码的复用，可以像搭积木一样进行开发。

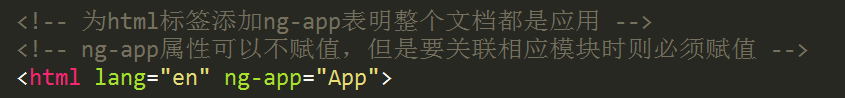


模块化设计，简单地说就是程序的编写不是开始就逐条录入计算机语句和指令，而是首先用主程序、子程序、子过程等框架**把软件的主要结构和流程描述出来**，**并定义和调试好各个框架之间的输入、输出链接关系**。逐步求精的结果是得到一系列**以功能块为单位的算法描述**。以功能块为单位进行程序设计，实现其求解算法的方法称为模块化。模块化的目的是为了降低程序复杂度，使程序设计、调试和维护等操作简单化。

**模块化开发有两个基本的模式：AMD、CMD。Angular是典型的AMD规范，NodeJS是典型的CMD规范**。

## 定义应用

通过为任一HTML标签添加ng-app属性，可以指定一个应用，表示此标签所包裹的内容都属于应用（App）的一部分。



## 定义模块

AngularJS提供了一个全局对象angular，在此全局对象下存在若干的方法，其中angular.module()方法用来定义一个模块。



注：应用（App）其本质也是一个模块（一个比较大的模块）。

## 定义控制器

控制器（Controller）作为连接模型（Model）和视图（View）的桥梁存在，所以当我们定义好了控制器以后也就定义好了模型和视图。

<!--控制器的调用-->

<div class="container" ng-controller="firstController">

<input type="text" ng-model="xianjs">

<!--{{}} xianjs视图调用数据进行展示-->

<h1>{{xianjs}}</h1>

</div>

<script type="text/javascript">

/\*定义模块

\*

\* 参数1：应用的名称

\* 参数2：模块的依赖

\* \*/

let app = angular.module("myApp", []);

/\* 定义控制器

\* 参数1：控制器的名称

\* 参数2：依赖参数，是一个数组

\* \*/

app.controller('firstController', ['$scope', function ($scope) {

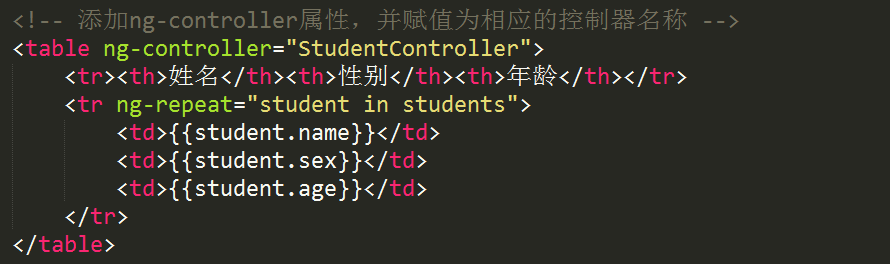
$scope.xianjs = "我是xianjs";

}]);

</script>



模型（Model）数据是要展示到视图（View）上的，所以需要将控制器（Controller）关联到视图（View）上，通过为HTML标签添加ng-controller属性并赋值相应的控制器（Controller）的名称，就确立了关联关系。



<div class="container" ng-controller="firstController">

<table class="table table-bordered">

<tr>

<td>序号</td>

<td>ID</td>

<td>Name</td>

<td>Age</td>

</tr>

<!--index为索引值，item为对象的key-->

<tr ng-repeat="(index,item) in users">

<td>{{index}}</td>

<td>{{item.id}}</td>

<td>{{item.name}}</td>

<td>{{item.age}}</td>

</tr>

</table>

</div>

<script type="text/javascript">

let app = angular.module("myApp", []);//模块

app.controller('firstController', ['$scope', function ($scope) {//控制器

$scope.users = [

{id: 'A01', name: "Tom", age: 25},

{id: 'A02', name: "jack", age: 16},

{id: 'A03', name: "pis", age: 22},

{id: 'A04', name: "kobe", age: 39}

];

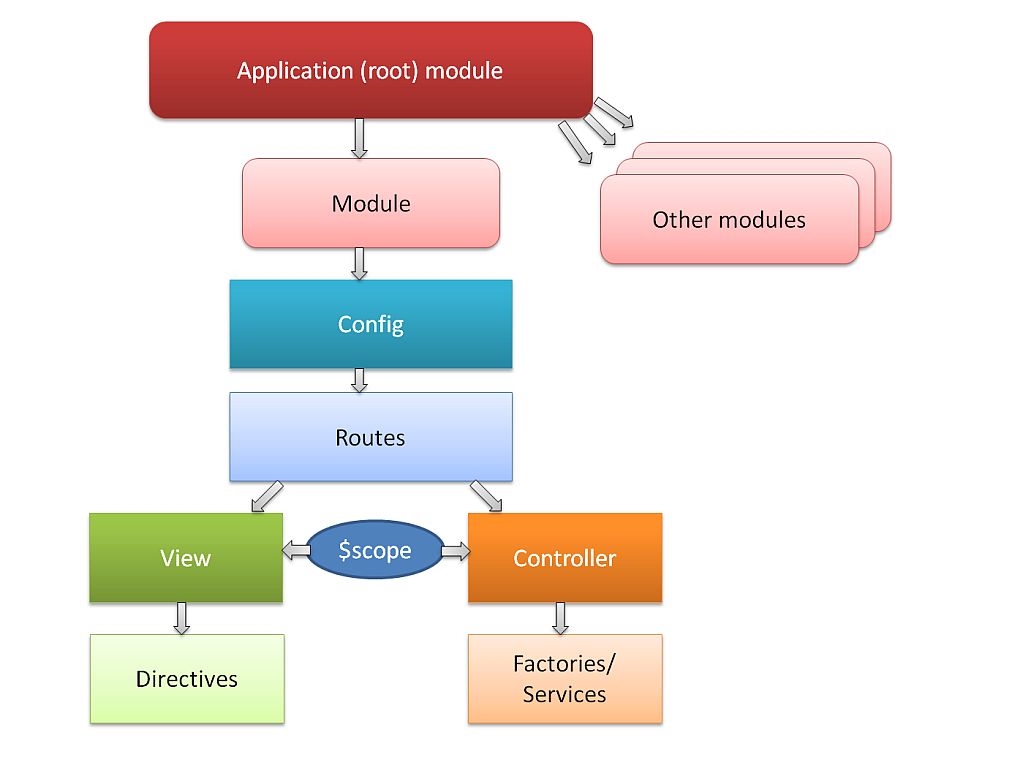
}]);

</script>



以上步骤就是AngularJS最基本的MVC工作模式。

下图是AngularJS的结构，学习AngularJS会围绕下图的结构展开。



## 视图实时更新

let app = angular.module( "app", [] );

//定义控制器

app.controller( "firstController", [ "$scope", function ( $scope ){

let timer = function (){

//视图实时更新

$scope.$apply( function (){

let now = new Date();

//当前时间格式化

$scope.date = `${ now.getFullYear() }-${ now.getMonth() + 1 }-${ now.getDate() } ${ now.getHours() }:${ now.getMinutes() }:${ now.getSeconds() }`;

console.log( $scope.date );

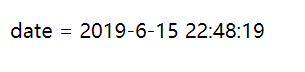
} );

};

setInterval( timer, 1\*1000 );

} ] );

时间实时更新的



app.controller( "firstController", [ "$scope", function ( $scope ){

$scope.xing = "";

$scope.ming = "";

$scope.fullName = "";

$scope.$watch( "xing", function (){

$scope.fullName = $scope.xing + $scope.ming;

} );

$scope.$watch( "ming", function (){

$scope.fullName = $scope.xing + $scope.ming;

} );

} ] );

# 指令

HTML在构建应用（App）时存在诸多不足之处，AngularJS通过扩展一系列的HTML属性或标签来弥补这些缺陷，所谓指令就是AngularJS自定义的HTML属性或标签，这些指令都是以ng-做为前缀的，例如ng-app、ng-controller、ng-repeat等。

## 内置指令

ng-app 指定应用根元素，至少有一个元素指定了此属性。

ng-controller 指定控制器

ng-show控制元素是否显示，true显示、false不显示

ng-hide控制元素是否隐藏，true隐藏、false不隐藏

ng-if控制元素是否“存在”，true存在、false不存在

ng-src增强图片路径

ng-href增强地址

ng-class控制类名

ng-include引入模板

ng-disabled表单禁用

ng-readonly表单只读

ng-checked单/复选框表单选中

ng-selected下拉框表单选中

<p>

<button ng-click="flag=!flag">显示隐藏切换</button>

</p>

<div>flag={{ flag }}</div>

<!--ng-show,ng-hide:控制的是display-->

<div ng-show="flag">show显示</div>

<div ng-hide="flag">hide测试</div>

<!--ng-if直接是控制的移除与创建DOM结构-->

<div ng-if="flag">if测试</div>

注：后续学习过程中还会介绍其它指令。

<style>

.red {

color: red;

}

</style>

<!--控制器的调用-->

<div class="container" ng-controller="firstController">

<li ng-if="a">ng-if控制元素是否“存在”，true存在、false不存在</li>

<li ng-hide="b">ng-hide控制元素是否隐藏，true隐藏、false不隐藏</li>

<li ng-show="b">ng-show控制元素是否显示，true显示、false不显示</li>

<!--使用src会报错-->

<li><a href="#"><img ng-src="{{src}}" alt=""></a></li>

<li><a ng-href="{{baidu}}">百度</a></li>

<!--ng-class的值为对象，对象的属性为存在的类名，值为布尔值，true添加类名，false不会添加-->

<li ng-class="{red:true}">red</li>

<!--表单禁用-->

<p><input type="text" ng-disabled="true" value="禁止使用"></p>

<!--只读属性-->

<p>money: <input type="text" ng-readonly="true" ng-value="money"></p>

<p><input type="checkbox" ng-checked="true">男</p>

<p><input type="checkbox" ng-checked="false">女</p>

</div>

<script type="text/javascript">

let app = angular.module("myApp", []);//模块

app.controller('firstController', ['$scope', function ($scope) {//控制器

$scope.a = "1";

// $scope.b = false;//显示

$scope.b = true;//隐藏

$scope.src = "./img/1.gif";

$scope.baidu = 'http://www.baidu.com';

$scope.money = "99994.12元";

}]);

</script>

### ng-options

在 AngularJS 中我们可以使用 **ng-option** 指令来创建一个下拉列表，列表项通过对象和数组循环输出;

ng-repeat 指令是通过数组来循环 HTML 代码来创建下拉列表，但 ng-options 指令更适合创建下拉列表，它有以下优势：

使用 ng-options 的选项是一个对象， ng-repeat 是一个字符串。

<select name="" id="">

<option **value="10"**>非常满意</option>

<option **value="8"**>比较满意</option>

<option **value="6"**>满意</option>

<option **value="4"**>比较不满意</option>

<option **value="2"**>稍微不满意</option>

<option **value="0"**>非常不满意</option>

</select>

**ng-options属性特别的好用，可以智能的从控制器中取值当做选项。注意，使用ng-options的下拉菜单必须有ng-model属性与控制器双向绑定了什么东西。**

1. 下拉菜单显示:普通数组

<select ng-model="selectedCity" class="form-control" ng-options="item for item in citys">

</select>

$scope.selectedCity = "";

$scope.citys = [ "西安", "南京", "北京", "西京" ];

必须绑定ng-model

2.数组对象

this.arr2 = [

{"phone" : "010" , "city" : "北京"},

{"phone" : "029" , "city" : "西安"},

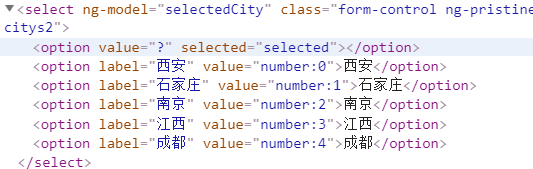
{"phone" : "0311" , "city" : "石家庄"}

];

ng-options指令这么写：

<select ng-model="mainctrl.zhi" ng-options="item.phone as item.city for item in mainctrl.arr2"></select>

格式就是 提交的值 as 显示的值 for 迭代变量 in 数组



3.key:value对象

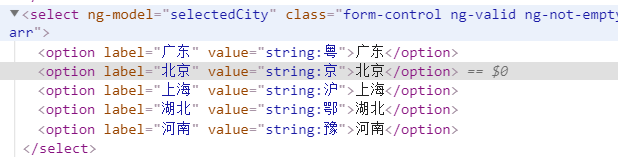
$scope.arr = {

"广东" : "粤", "北京" : "京", "上海" : "沪", "湖北" : "鄂", "河南" : "豫",

};

<select ng-model="mainctrl.zhi3" **ng-options="value as key for (key,value) in mainctrl.arr3"**></select>

格式就是 提交的值 as 显示的值 for (key,value) in 数组



4.三级联动

<form class=" form-inline" style="margin-top: 30px">

<div class="form-group">

<label>省份</label>

<select class="form-control" style="max-width: 120px"

ng-model="province"

ng-options="k as v.name for (k,v) in data"></select>

</div>

<div class="form-group">

<label>市</label>

<select class="form-control" ng-model="city"

ng-options="k as v.name for (k,v) in data[province].data"></select>

</div>

<div class="form-group">

<label>县</label>

<select class="form-control" ng-model="county"

ng-options="k as v.name for (k,v) in data[province].data[city].data"></select>

</div>

<div class="page-header">

<p>地址: {{ province }} :市 {{ city }} 县 :{{ county }}</p>

</div>

</form>



## 自定义指令

AngularJS允许根据实际业务需要自定义指令，通过angular全局对象下的directive方法实现。



E 代表元素指令 Element

A 代表属性执行 Attribute

M 代表注释指令 Mark

C 代表class执行 Class

<!--控制器的调用-->

<div class="container" ng-controller="firstController">

<div tag></div>

<tag></tag>

<p class="tag"></p>

<!-- directive:tag -->

</div>

<script type="text/javascript">

let app = angular.module("myApp", []);//模块

app.controller('firstController', ['$scope', function ($scope) {//控制器

}]);

/\*

\* 自定义指令

\* 参数1：指令的名称

\* 参数2：回调函数

\* \*/

app.directive("tag", function () {

return {

// 指令类型 E：element

// A：attribute

// M：mark

// C: class

restrict: 'ECMA',

replace: true,//替换

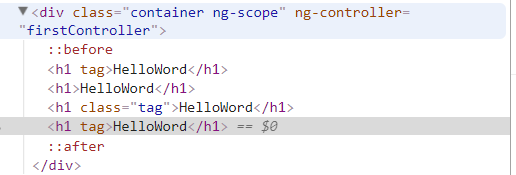
// template: "<h1>HelloWord</h1>",

templateUrl: "./head.html",//加载外部文件

};

});

</script>



## 最简单的用directive()定义一个指令

ng-if、ng-model都是指令，direcitve就是指令，允许我们自己创建一些HTML中能识别的“语法糖”。

Angular给世界最大的贡献，可以说就是创建了自定义指令这个东西，后面React、Vue将指令进行了简化，变为了“组件”。

最简单的一个案例，我们使用directive()函数来定义一个指令。

<!DOCTYPE html>

<html lang="en" ng-app="myapp">

<head>

<meta charset="UTF-8">

<title>指令学习</title>

</head>

<body>

<div **my-direct**>默认内容</div>

**<my-direct></my-direct>**

<script type="text/javascript" src="js/lib/angular/angular.min.js"></script>

<script type="text/javascript">

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

//定义指令的时候，不能用短横，而必须是驼峰风格，使用的时候驼峰自动变短横。

**myapp.directive("myDirect",[function(){**

//返回一个指令定义对象

**return {**

**template : "<h1>你好</h1>"**

**}**

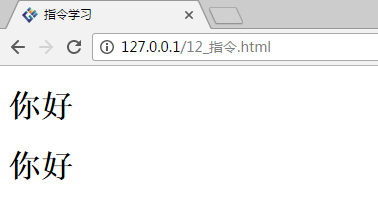
**}]);**

</script>

</body>

</html>

结果：



审查元素：



## restrict属性

上面的案例，使用指令的时候是两种形式：

**属性（Attribute）**：

<div my-direct>默认内容</div>

**元素（Element）**：

<my-direct></my-direct>

**默认情况就是A、E**两种形式的指令，还有两种：

**类名(Class)：**

<div class="my-direct"></div>

**注释(M)：**没人用

<!--directive:myDirect-->

四个类型的指令使用方法：AECM，其中AE是默认的，如果想用C的话，用restrict属性：

<script type="text/javascript">

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

//定义指令的时候，不能用短横，而必须是驼峰风格，使用的时候驼峰自动变短横。

myapp.directive("myDirect",[function(){

//返回一个指令定义对象

return {

**restrict : "AEC",**

template : "<h1>你好</h1>"

}

}]);

</script>

## link属性

先说一个小事儿，就是templateUrl属性可以将html模板放在外面，Angular将使用Ajax技术读取这个模板。此时页面必须运行在服务器环境中，不能直接双击运行。

//定义指令的时候，不能用短横，而必须是驼峰风格，使用的时候驼峰自动变短横。

myapp.directive("myDirect",[function(){

//返回一个指令定义对象

return {

restrict : "E",

**templateUrl : "./template/myDirect.html"**

}

}]);

link属性表示链接指令内部和外部的关系函数：

<script type="text/javascript">

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

//定义指令的时候，不能用短横，而必须是驼峰风格，使用的时候驼峰自动变短横。

myapp.directive("myDirect",[function(){

//返回一个指令定义对象

return {

restrict : "E",

templateUrl : "./template/myDirect.html",

link : function($scope,ele,attr){

$scope.a = 100;

}

}

}]);

</script>

外置模板，就有a的值是100。所以$scope表示外置模板的作用域。

<div>

<h1>---------------------------------------------</h1>

<h1>我是指令</h1>

<h1>a的值是:**{{ a }}**</h1>

<h1>---------------------------------------------</h1>

</div>

我们现在做一个按钮，可以让a的值加1：

myapp.directive("myDirect",[function(){

//返回一个指令定义对象

return {

restrict : "E",

templateUrl : "./template/myDirect.html",

link : function($scope,ele,attr){

$scope.a = 100;

$scope.add = function(){

$scope.a++;

}

}

}

}]);

同时我们在页面上放置两个组件：

<div ng-controller="MainCtrl as mainctrl">

<my-direct></my-direct>

<my-direct></my-direct>

</div>

在一个组件内部点击按钮，居然也影响了另一个组件



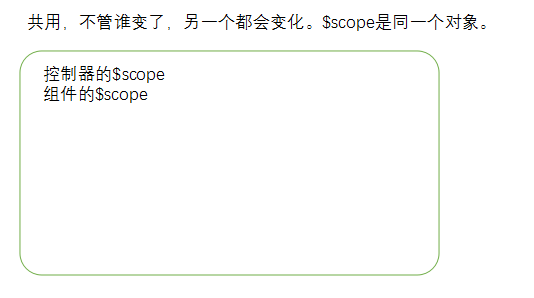
link中attr表示属性对象，ele表示添加指令的这个HTML元素。

ele是angular DOM对象，不是原生JS DOM对象，方法和jq特别相似：

addClass()-为每个匹配的元素添加指定的样式类名  
after()-在匹配元素集合中的每个元素后面插入参数所指定的内容，作为其兄弟节点  
append()-在每个匹配元素里面的末尾处插入参数内容  
attr() - 获取匹配的元素集合中的第一个元素的属性的值  
bind() - 为一个元素绑定一个事件处理程序  
children() - 获得匹配元素集合中每个元素的子元素，选择器选择性筛选  
clone()-创建一个匹配的元素集合的深度拷贝副本  
contents()-获得匹配元素集合中每个元素的子元素，包括文字和注释节点  
css() - 获取匹配元素集合中的第一个元素的样式属性的值  
data()-在匹配元素上存储任意相关数据  
detach()-从DOM中去掉所有匹配的元素  
empty()-从DOM中移除集合中匹配元素的所有子节点  
eq()-减少匹配元素的集合为指定的索引的哪一个元素  
find() - 通过一个选择器，jQuery对象，或元素过滤，得到当前匹配的元素集合中每个元素的后代  
hasClass()-确定任何一个匹配元素是否有被分配给定的（样式）类  
html()-获取集合中第一个匹配元素的HTML内容  
next() - 取得匹配的元素集合中每一个元素紧邻的后面同辈元素的元素集合。如果提供一个选择器，那么只有紧跟着的兄弟元素满足选择器时，才会返回此元素  
on() - 在选定的元素上绑定一个或多个事件处理函数  
off() - 移除一个事件处理函数  
one() - 为元素的事件添加处理函数。处理函数在每个元素上每种事件类型最多执行一次  
parent() - 取得匹配元素集合中，每个元素的父元素，可以提供一个可选的选择器  
prepend()-将参数内容插入到每个匹配元素的前面（元素内部）  
prop()-获取匹配的元素集中第一个元素的属性（property）值  
ready()-当DOM准备就绪时，指定一个函数来执行  
remove()-将匹配元素集合从DOM中删除。（同时移除元素上的事件及 jQuery 数据。）  
removeAttr()-为匹配的元素集合中的每个元素中移除一个属性（attribute）  
removeClass()-移除集合中每个匹配元素上一个，多个或全部样式  
removeData()-在元素上移除绑定的数据  
replaceWith()-用提供的内容替换集合中所有匹配的元素并且返回被删除元素的集合  
text()-得到匹配元素集合中每个元素的合并文本，包括他们的后代  
toggleClass()-在匹配的元素集合中的每个元素上添加或删除一个或多个样式类,取决于这个样式类是否存在或值切换属性。即：如果存在（不存在）就删除（添加）一个类  
triggerHandler() -为一个事件执行附加到元素的所有处理程序  
unbind() - 从元素上删除一个以前附加事件处理程序  
val()-获取匹配的元素集合中第一个元素的当前值  
wrap()-在每个匹配的元素外层包上一个html元素

## scope属性

scope天生是这样的，控制器和指令的$scope是同一个对象。



# 数据绑定

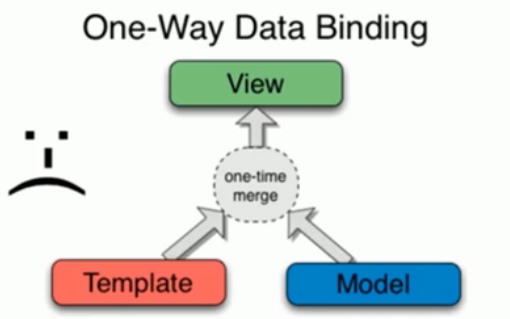
AngularJS是以数据做为驱动的MVC框架，所有模型（Model）里的数据经由控制器（Controller）展示到视图（View）中。

所谓数据绑定指的就是将模型（Model）中的数据与相应的视图（View）进行关联，分为单向绑定和双向绑定两种方式。

## 单向绑定

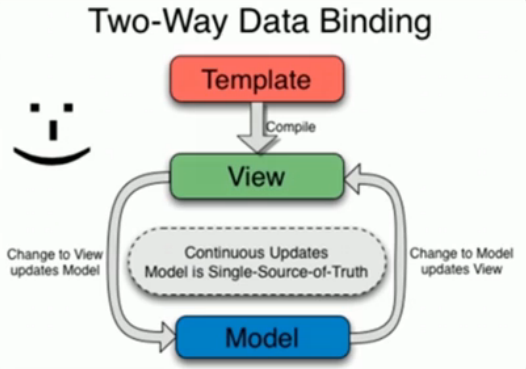
单向数据绑定是指将模型（Model）数据，按着写好的视图（View）模板生成HTML标签，然后追加到DOM中显示，如之前所学的artTemplate 模板引擎的工作方式。

如下图所示，只能模型（Model）数据向视图（View）传递。



## 双向绑定

双向绑定则可以实现模型（Model）数据和视图（View）模板的双向传递，如下图所示。



## 相关指令

在AngularJS中通过“{{}}”和ng-bind指令来实现模型（Model）数据向视图模板（View）的绑定，模型数据通过一个内置服务$scope来提供，这个$scope是一个空对象，通过为这个对象添加属性或者方法便可以在相应的视图（View）模板里被访问。

注：“{{}}”是ng-bind的简写形式，其区别在于通过“{{}}”绑定数据时会有“闪烁”现象，添加ng-cloak也可以解决“闪烁”现象。

<style type="text/css">[ng\:cloak],[ng-cloak],[data-ng-cloak],[x-ng-cloak],.ng-cloak,.x-ng-cloak,.ng-hide:not(.ng-hide-animate){display:none !important;}ng\:form{display:block;}.ng-animate-shim{visibility:hidden;}.ng-anchor{position:absolute;}</style>

<!--控制器的调用-->

<div class="container" ng-controller="firstController">

<h1 ng-cloak>{{data.name}}</h1>

<ul>

<!--<li ng-repeat="item in data.courcess">{{item}}</li>-->

<li ng-repeat="item in data.courcess" ng-bind="item"></li>

</ul>

</div>

<script type="text/javascript">

let app = angular.module("myApp", []);//模块

/\*

\* 单向数据绑定：数据=>视图，可以使用ng-bind

\* {{}}是ng-bind的简写,在网速不好时会出现闪烁现象，但是ng-bind不会出现闪烁现象

\* 可以使用ng-clock修复{{}}出现的闪烁现象

\* \*/

app.controller('firstController', ['$scope', function ($scope) {//控制器

$scope.data = {

name: "Tom",

courcess: ['java', 'css', 'php', 'go']

}

}]);

</script>

通过为表单元素添加ng-model指令实现视图（View）模板向模型（Model）数据的绑定。

通过ng-init可以初始化模型（Model）也就是$scope。

<**div** ng-init=**"name='lisi';age='26'"**>

<**p**>**name = {{name}}**</**p**>

<**p**>**age={{age}}**</**p**>

</**div**>

AngularJS对事件也进行了扩展，无需显式的获取DOM元素便可以添加事件，易用性变的更强。通过在原有事件名称基础上添加ng-做为前缀，然后以属性的形式添加到相应的HTML标签上即可。如ng-click、ng-dblclick、ng-blur等。

<**div** class=**"container"** ng-controller=**"firstController"**>

<**input** type=**"button"** value=**"增加"** ng-click=**" count = count + 1"**>

<**p**>**count = {{***count***}}**</**p**>

<**hr**>

<**button** ng-click=**"toggle()"**>**隐藏/显示**</**button**>

<**p** ng-hide=**"myVar"**>

**名:** <**input** type=**"text"** ng-model=**"firstName"**><**br**>

**姓名:** <**input** type=**"text"** ng-model=**"lastName"**><**br**>

<**br**>

**Full Name: {{firstName + " " + lastName}}**

</**p**>

</**div**>

<**script** type=**"text/javascript"**>

let *app* = *angular*.module("myApp", []);//模块

*app*.controller('firstController', ['$scope', function (*$scope*) {//控制器

*$scope*.firstName = "John";

*$scope*.lastName = "Doe";

*$scope*.myVar = false;

*$scope*.toggle = function () {

//实现显示和隐藏的切换

*$scope*.myVar = !*$scope*.myVar;

};

}]);

</**script**>

通过ng-repeat可以将数组或对象数据迭代到视图模板中，ng-switch、on、ng-switch-when可以对数据进行筛选。

<**div** class=**"container"** ng-controller=**"firstController"**>

<**ul**>

<!--(key,value)输出对象的key和value-->

<**li** ng-repeat=**"(***k***,***v***) in** *info***"**>**{{***k***}} -- {{***v***}}**</**li**>

</**ul**>

<**ul**>

<**li** ng-repeat=**"(***index***,***item***) in course"** ng-switch=**"item"**>

<**span** ng-switch-when=**"html"**>**{{***item***}}**</**span**>

<**span** ng-switch-when=**"css"**>**{{***item***}}**</**span**>

<**span** ng-switch-when=**"php"**>**{{***item***}}**</**span**>

</**li**>

</**ul**>

</**div**>

<**script** type=**"text/javascript"**>

let *app* = *angular*.module("myApp", []);//模块

*app*.controller('firstController', ['$scope', function (*$scope*) {//控制器

*$scope*.course = ['html', 'css', 'php'];

*$scope*.info = {

name: "Tom",

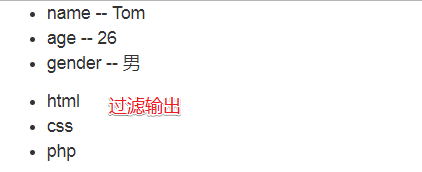
age: 26,

gender: '男'

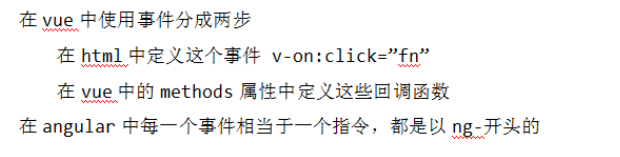
}

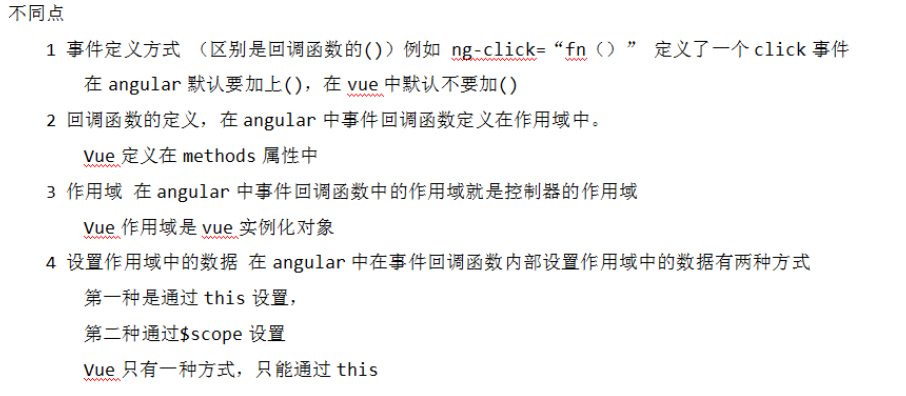
}]);

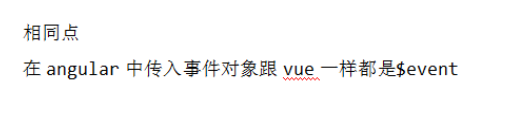
</**script**>



## 事件







# 作用域

通常AngularJS中应用（App）是由若干个视图（View）组合成而成的，而视图（View）又都是HTML元素，并且HTML元素是可以互相嵌套的，另一方面视图都隶属于某个控制器（Controller），进而控制器之间也必然会产生嵌套关系。

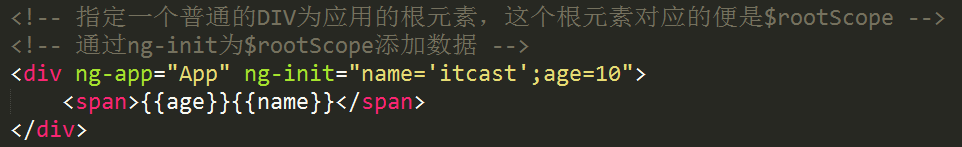
每个控制器（Controller）又都对应一个模型（Model）也就是$scope对象，不同层级控制器（Controller）下的$scope便产生了作用域。

## 根作用域

一个AngularJS的应用（App）在启动时会自动创建一个根作用域$rootScope，这个根作用域在整个应用范围（ng-app所在标签以内）都是可以被访问到的。

所有的应用都有一个 $rootScope，它可以作用在 ng-app 指令包含的所有 HTML 元素中。

$rootScope 可作用于整个应用中。是各个 controller 中 scope 的桥梁。用 rootscope 定义的值，可以在各个 controller 中使用。



## 子作用域

通过ng-controller指令可以创建一个子作用域，新建的作用域可以访问其父作用域的数据。

<**div** class=**"container"** ng-controller=**"firstController"**>

<**p**>**父name = {{***name***}}**</**p**>

<**p**>**全局Money= {{***money***}}**</**p**>

<**div** ng-controller=**"childController"**>

<!--内部作用域没有name的话会向上层查找

本层作用域有的话就会适应本层作用域的name

-->

<**p**> **name = {{***name***}}** </**p**>

</**div**>

</**div**>

<**div**>

**全局的 money = {{***$root***.money}}**

</**div**>

<**script** type=**"text/javascript"**>

let *app* = *angular*.module("myApp", []);//模块

*app*.controller('firstController', ['$scope', function (*$scope*) {//控制器

*$scope*.name = '父亲';

}]);

*app*.controller("childController", ['$scope', '$rootScope', function (*$scope*, *$rootScope*) {

*$scope*.name = "儿子";

*$rootScope*.money = 1000;

}]);

</**script**>

# 过滤器

在AngularJS中使用过滤器格式化展示数据，在“{{}}”中使用“|”来调用过滤器，使用“:”传递参数。

## 内置过滤器

1、currency[货币] 将数值格式化为货币格式

2、date日期格式化，年（y）、月（M）、日（d）、星期（EEEE/EEE）、时（H/h）、分（m）、秒（s）、毫秒（.sss），也可以组合到一起使用。

3、filter在给定数组中选择满足条件的一个子集，并返回一个新数组，其条件可以是一个字符串、对象、函数

4、json将Javascrip对象转成JSON字符串。

5、limitTo取出字符串或数组的前（正数）几位或后（负数）几位

6、lowercase将文本转换成小写格式

7、uppercase将文本转换成大写格式

8、number数字格式化，可控制小位位数

9、orderBy对数组进行排序，第2个参数是布尔值可控制方向(正序或倒序)

<**ul**>

<**li**>**小数点3位：{{***money***|number:3}}**</**li**>

<**li**>**货币：{{***money***|currency:'￥'}}**</**li**>

<**li**>**小写：{{***str***|lowercase}}**</**li**>

<**li**>**大写：{{***str***|uppercase}}**</**li**>

<**li**>**日期：{{currentDate | date:'yyyy-MM-dd HH:mm:ss Z'}}**</**li**>

<**li**>**json格式数据：{{person|json}}**</**li**>

<**li**>**字符串前三个字符：{{***str***|limitTo:3}}**</**li**>

<**li**>**数组前三项：{{***arr***|limitTo:3}}**</**li**>

</**ul**>

<!--小数点3位：95,984.260-->

<!--货币：￥95,984.26-->

<!--小写：abcdefxianjs-->

<!--大写：ABCDEFXIANJS-->

<!--日期：2018-03-15 18:24:01 +0800-->

<!--json格式数据：{ "name": "Tom", "age": "26岁", "gender": "男" }-->

<!--字符串前三个字符：Abc-->

<!--数组前三项：[1,2,3]-->

<**li**>**{{***color***|filter:'e'|filter:'o'}}**</**li**>

<!--

["red","green","yello","oragle"]

["yello","oragle"]

-->

<**li**>**{{***users***|orderBy:'age':true}}**</**li**>

<!--

true:倒叙

false:正序（默认）

-->

<**li**>**{{***users***|orderBy:'age':false}}**</**li**>

## 自定义过滤器

除了使用AngularJS内建过滤器外，还可以根业务需要自定义过滤器，通过模块对象实例提供的filter方法自定义过滤器。



<**div** class=**"container"** ng-controller=**"firstController"**>

<**p**>**xianjs反转 = {{***str***|revers}}**</**p**>

</**div**>

<**script** type=**"text/javascript"**>

let *app* = *angular*.module("myApp", []);//模块

*app*.controller('firstController', ['$scope', function (*$scope*) {//控制器

*$scope*.str = "Xianjs";

}]);

*app*.filter('revers', function () {//依赖注入

return function (*text*) {

return *text*.split("").reverse().join("");

};

});

*console*.log(*app*);

</**script**>

# 表单

## 表单基础

<**form** action=**""**>

<**div** class=**"form-group"**>

<**label** for=**"username"**>**user**</**label**>

<**input** type=**"text"** ng-model=**"user"** class=**"form-control"** id=**"username"**>

</**div**>

<**div** class=**"checkbox"**>

<**input** type=**"checkbox"** ng-model=**"myVal1"**>

</**div**>

<**div** class=**"form-group"**>

<**input** type=**"radio"** ng-model=**"myVar"** value=**"dogs"**>**Dogs**

<**input** type=**"radio"** ng-model=**"myVar"** value=**"tuts"**>**Tutorials**

<**input** type=**"radio"** ng-model=**"myVar"** value=**"cars"**>**Cars**

</**div**>

<**div** class=**"form-group"**>

<**select** ng-model=**"myVars"**>

<**option** value=**""**>

<**option** value=**"dogs"**>**Dogs**

<**option** value=**"tuts"**>**Tutorials**

<**option** value=**"cars"**>**Cars**

</**select**>

</**div**>

<**h3** class=**"page-header"**>**user = {{***user***}}**</**h3**>

<**h3** class=**"page-header"** ng-show=**"myVal1"**>**myVal = {{myVal1}}**</**h3**>

<**h3** class=**"page-header"** ng-show=**"myVar"**>**myVar = {{myVar}}**</**h3**>

<**h3** class=**"page-header"** ng-show=**"myVars"**>**myVar = {{myVars}}**</**h3**>

</**form**>

## 输入验证

Angular可以用当前最优雅的方式完成表单验证。

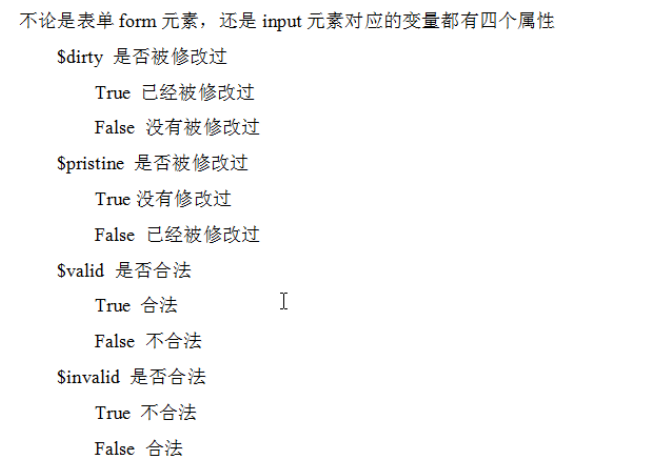
两个条件：

1）需要验证的控件必须有ng-model属性双向数据绑定

2）form必须有name属性。

ng-maxlength="6"：最大长度为6

ng-minlength="4"：最小的长度为4



<**form** action=**""** name=**"regist"** class=**"form-horizontal container-fluid"**>

<**div** class=**"form-group"**>

<**label** for=**"username"** class=**"col-xs-2 control-label"**>**用户名**</**label**>

<**div** class=**"col-xs-8"**>

<**input** type=**"text"** id=**"username"** name=**"username"** class=**"form-control"** ng-model=**"data.username"** ng-maxlength=**"6"**

ng-minlength=**"4"**>

</**div**>

</**div**>

<**div** class=**"form-group"**>

<**label** for=**"tel"** class=**"col-xs-2 control-label"**>**手机号**</**label**>

<**div** class=**"col-xs-8"**><**input** type=**"text"** id=**"tel"** name=**"tel"** ng-model=**"data.tel"** class=**"form-control"**

ng-pattern=**"/^1\d{10}$/"**></**div**>

</**div**>

<**div** class=**"form-group"**>

<**label** for=**"address"** class=**"col-xs-2 control-label"**>**地&emsp;址**</**label**>

<**div** class=**"col-xs-8"**><**input** type=**"text"** id=**"address"** name=**"address"** ng-model=**"data.address"** class=**"form-control"**

ng-required=**"true"**>

</**div**>

</**div**>

<**div** class=**"form-group"**>

<**label** for=**"email"** class=**"col-xs-2 control-label"**>**邮&emsp;箱**</**label**>

<**div** class=**"col-xs-8"**><**input** type=**"text"** id=**"email"** name=**"email"** ng-model=**"data.email"** class=**"form-control"**

ng-pattern=**"/^\w+@[\w\.]/"**></**div**>

</**div**>

<**div** class=**"warning"**>

<**p** class=**"text-danger"** ng-show=**"regist.username.$dirty && regist.username.$invalid"**>**输入正确的用户名称**</**p**>

<**p** class=**"text-danger"** ng-show=**"regist.tel.$dirty && regist.tel.$invalid"**>**输入正确的手机号**</**p**>

<**p** class=**"text-danger"** ng-show=**"regist.address.$dirty && regist.address.$invalid"**>**地址必须填写**</**p**>

<**p** class=**"text-danger"** ng-show=**"regist.email.$dirty && regist.email.$invalid"**>**邮箱格式**</**p**>

<**p** class=**"text-danger"** ng-show=**"showErr"**>**信息不完善**</**p**>

</**div**>

<**div** class=**"form-group col-sm-8 "**>

<**button** class=**"btn btn-danger btn-lg btn-block"** ng-click=**"sub()"** type=**"submit"**>**提交**</**button**>

</**div**>

</**form**>

</**div**>

<**script** type=**"text/javascript"**>

let *app* = *angular*.module('myApp', []);

*app*.controller('firstController', ['$scope', function (*$scope*) {//控制器

*$scope*.data = {};

*$scope*.showErr = false;

//点击提交按钮

*$scope*.sub = function () {

*$scope*.showErr = *$scope*.regist.$invalid;

*console*.log(*$scope*.regist.$invalid);

}

}]);

</**script**>

# 依赖注入

AngularJS采用模块化的方式组织代码，将一些通用逻辑封装成一个对象或函数，实现最大程度的复用，这导致了使用者和被使用者之间存在依赖关系。

所谓依赖注入是指在运行时自动查找依赖关系，然后将查找到依赖传递给使用者的一种机制。

通俗的讲就是通入注入的方式解决依赖关系。

常见的AngularJS内置服务有$http、$location、$timeout、$rootScope等

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。

## 行内注入

以数组形式明确声明依赖，数组元素都是包含依赖名称的字符串，数组最后一个元素是依赖注入的目标函数。



推荐使用这种方式声明依赖

见代码示例13 AngularJS依赖注入(行内式).html

## 推断式注入

没有明确声明依赖，AngularJS会将函数参数名称当成是依赖的名称。



这种方式会带来一个问题，当代码经过压缩后函数的参数被压缩，这样便会造成依赖无法找到。

见代码示例14 AngularJS依赖注入(推断式).html

**服务**

服务是一个对象或函数，对外提供特定的功能。

/\*行内式依赖注入\*/

/\*app.controller('firstController', ['$scope', function ($scope) {//控制器

$scope.str = "Xianjs";

}]);\*/

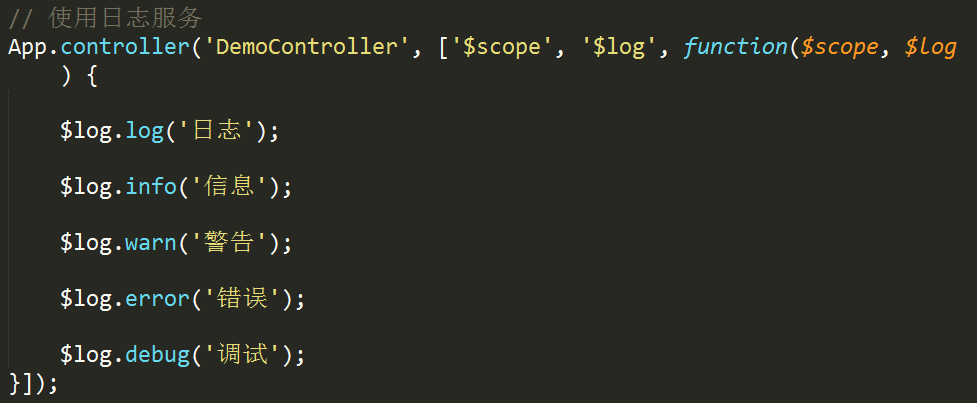
*app*.controller('firstController', '$http', function (*$scope*, *$http*) {

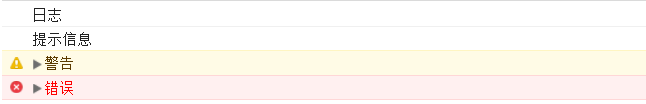
*$scope*.str = "xxxxxx";

});

# 内置服务

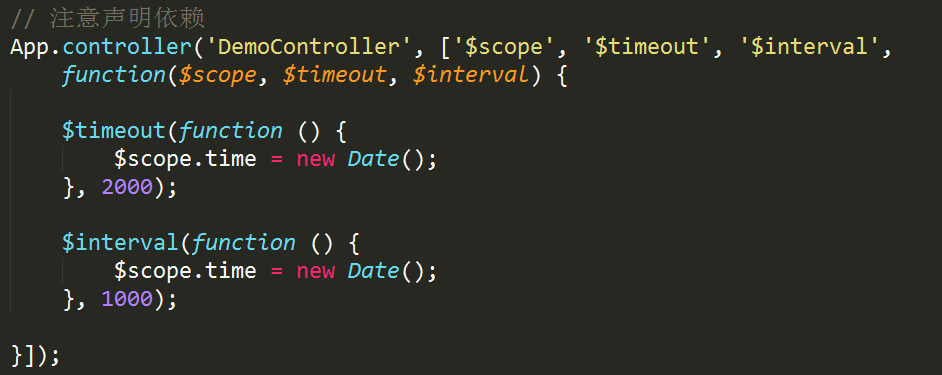
## $log打印调试信息





## angular定时器封装

$timeout&$interval对原生Javascript中的setTimeout和setInterval进行了封装。



*app*.controller('firstController', ['$scope', '$timeout', '$interval', function (*$scope*, *$timeout*, *$interval*) {

*$scope*.n = 0;

*$timeout*(function () {

*console*.log("3s后执行了");

}, 2002);

*$interval*(function () {

*console*.log('0.5秒执行一次', *$scope*.n);

*$scope*.n++;

}, 500);

}]);

let timer = *$interval*(function () {

*$scope*.now = new *Date*();

}, 1000);

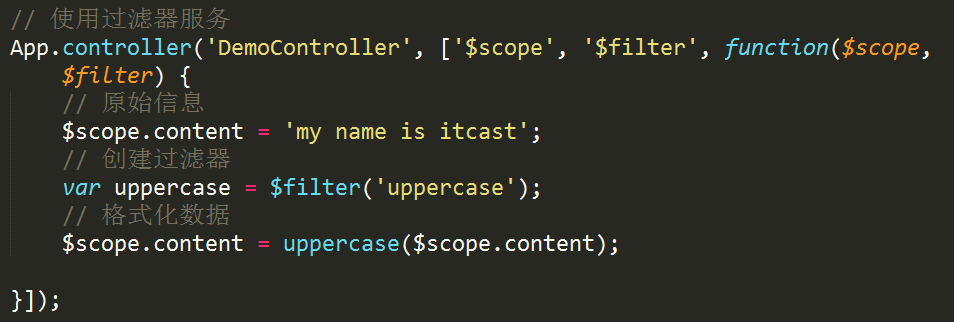
//停止定时器

*$scope*.stop = function () {

*$interval*.cancel(timer);

}

## $filter在控制器中格式化数据。



见代码示例17 AngularJS内置服务filter.html

## $http用于向服务端发起异步请求。



*app*.controller('firstController', ['$scope', '$http', function (*$scope*, *$http*) {

*$http*({

method: "get",//请求方法

url: "19.php",//请求地址

}).then(function success(*res*) {

*console*.log(*res*.data);

*$scope*.data = *res*.data;

}, function error() {

*console*.log("错误");

});

}]);

app.controller("fCtl", [ "$scope", "$http", function($scope, $http){

$http.get("./data.json").then(res => {

console.log(res);

}).catch(error => {

console.log(error);

});

} ]);

同时还支持多种快捷方式如$http.get()、$http.post()、$http.jsonp。

注：各参数含义见代码注释。

### $HTTP常用请求方法

此外还有以下简写方法：

* $http.get
* $http.head
* $http.post
* $http.put
* $http.delete
* $http.jsonp
* $http.patch

### get方式



*app*.controller('firstController', ['$scope', '$http', '$log', function (*$scope*, *$http*, *$log*) {

*$http*({

method: "get",//请求方法

url: "19.php",//请求地址

params: {

name: "李四",

age: 26,

gender: '男'

}

}).then(function success(*res*) {

*console*.log(*res*.data);

*$scope*.data = *res*.data;

}, function error() {

*console*.log("错误");

});

}]);

### pot方式

data: "name=李四&age=26&gender=男",

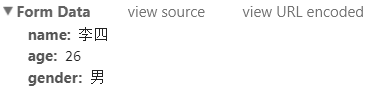
//设置请求头

headers: {

//Content-Type:application/x-www-form-urlencoded

'Content-Type': 'application/x-www-form-urlencoded'

}





## Promise数据请求

Promise.all([ $http.get("./data.json"), $http.get("./data.json"), $http.get("data.json") ]).then(res => {

let {res1, res2, res3} = res;

console.log(res1, res2, res3);

$scope.$apply();

}).catch(error => {

console.log(error);

});

## 获取服务端数据

App.controller('DemoCtrl', ['$scope', '$http', function ($scope, $http) {

$scope.get = function () {

$http({

url: './data.json',

method: 'get'

}).then(function (info) {

console.log(info);

$scope.stars = info.data;//返回的数据进行处理

});

}

}]);

## 获取天气预报

Weather.controller('WeatherCtrl', ['$scope', '$http', function ($scope, $http) {

// 使用jsonp

$http({

url: 'http://api.map.baidu.com/telematics/v3/weather',

method: 'jsonp',

params: {

ak: '0A5bc3c4fb543c8f9bc54b77bc155724',

location: '西安市',

output: 'json',

callback: 'JSON\_CALLBACK'

}

}).then(function (info) {

$scope.wether = info.data.results;

console.log(info)

})

}])

## 聊天机器人

<script>

let Chat = angular.module('Chat', []);

Chat.controller('ChatCtrl', ['$scope', '$http', function ($scope, $http) {

$scope.messages = [];

// [

// {text: '我说的话', role: '自已'},

// {text: '对方说的话', role: '对方'}

// ]

$scope.send = function () {

// alert($scope.msg);

$scope.messages.push({

text: $scope.msg,

role: '我说',

cls: 'self'

});

$scope.msg = '';

$http({

url: './chat.php'

}).then(function (info) {

console.log(info.data)

$scope.messages.push({

text: info.data,

role: '对方说',

cls: 'other'

});

})

}

}])

</script>

## 内置服务

### $location

有个 $location 服务，它可以返回当前页面的 URL 地址。

var app = angular.module('myApp', []);  
app.controller('customersCtrl', function($scope, $location) {  
    $scope.myUrl = $location.absUrl();  
});

### $http 服务

$http 是 AngularJS 应用中最常用的服务。 服务向服务器发送请求，应用响应服务器传送过来的数据。

var app = angular.module('myApp', []);  
app.controller('myCtrl', function($scope, $http) {  
    $http.get("welcome.htm").then(function (response) {  
        $scope.myWelcome = response.data;  
    });  
});

### $timeout 服务

AngularJS $timeout 服务对应了 JS window.setTimeout 函数。

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

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

$scope.myHeader = "Hello World!";

$timeout(function () {

$scope.myHeader = "How are you today?";

}, 2000);

});

### $interval 服务

AngularJS $interval 服务对应了 JS window.setInterval 函数。

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

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

$scope.theTime = new Date().toLocaleTimeString();

$interval(function () {

$scope.theTime = new Date().toLocaleTimeString();

}, 1000);

});

## 创建自定义服务

### 创建名为hexafy 的服务:

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

this.myFunc = function (x) {

return x.toString(16);

}

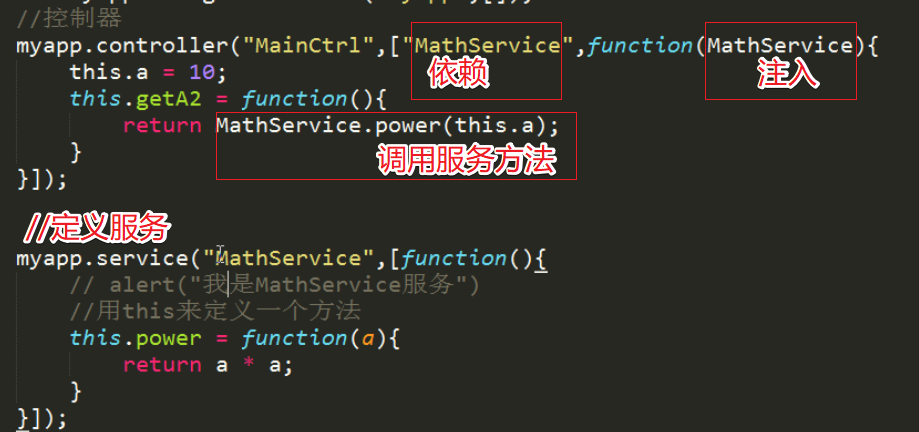
});

使用自定义的的服务 hexafy 将一个数字转换为16进制数:

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

$scope.hex = hexafy.myFunc(255);

});



### 过滤器中，使用自定义服务

当你创建了自定义服务，并连接到你的应用上后，你可以在控制器，指令，过滤器或其他服务中使用它。

在过滤器 myFormat 中使用服务 hexafy:

app.filter('myFormat',['hexafy', function(hexafy) {

return function(x) {

return hexafy.myFunc(x);

};

}]);

创建服务 hexafy:

<ul>

<li ng-repeat="x in counts">{{x | myFormat}}</li>

</ul>

## 自定义服务

通过上面例子得知，所谓服务是将一些通用性的功能逻辑进行封装方便使用，AngularJS允许将自定义服务。

### factory方法



见代码示例19 AngularJS自定义服务factory.html

<**script** type=**"text/javascript"**>

let *app* = *angular*.module("myApp", []);//模块

/\*\*

\* 返回值为函数

\*/

*app*.factory('demo1', [function () {

return function () {

*console*.log('我是自定义服务...');

}

}]);

/\*

\* 返回数据可以是任何的类型

\* \*/

*app*.factory('demo', [function () {

// return "Hello";

// return 1;

// return [1, 2, 43, 5];

return {

name: 'Tom',

age: 26

}

}]);

// 定义一个报时服务

*app*.factory('ShowTime', ['$filter', function (*$filter*) {

// 这里依赖了 $filter 这个服务

// 只要有依赖的地方都可以使用“依赖注入”

// 返回了一个对象，此对象下

// 包含了两个方法

return {

// 用来直接显示当前日期

now: function () {

return *$filter*('date')(new *Date*, 'yyyy-MM-dd');

},

// 根据用户参数显示特定格式日期

format: function (*format*) {

return *$filter*('date')(new *Date*, *format*);

}

}

}]);

*app*.controller('firstController', ['$scope', 'demo', 'ShowTime', function (*$scope*, *demo*, *ShowTime*) {

// demo();

// console.log(demo);

// 调用报时服务的 now 方法

*$scope*.now = *ShowTime*.now();

// 调用服时服务的 format 方法

*$scope*.time = *ShowTime*.format('hh:mm:ss');

}]);

</**script**>

### service方法



见代码示例20 AngularJS自定义服务service.html

在介绍服务时曾提到服务本质就是一个对象或函数，所以自定义服务就是要返回一个对象或函数以供使用。

<**script**>

var *App* = *angular*.module('App', []);

// controller 定义控制器

// directive 定义指令

// filter 定义过滤器

// factory 定义服务

// service 也能定义服务

// 需要两个参数

// 第1个参数 服务名称

// 第2个参数 数组（依赖）

*App*.service('sayHi', ['$http', function (*$http*) {

// 在此写当前服务的具体逻辑

// $http({

// url: ''

// })

// 不再使用 return 将服务结果返回了

// 而是使用 this 来返回

this.num = 1;

this.str = 'abc';

this.arr = ['html', 'js', 'css'];

this.fn = function () {

}

}]);

*App*.controller('DemoCtrl', ['$scope', 'sayHi', function (*$scope*, *sayHi*) {

*console*.log(*sayHi*)

}]);

// 报时服务

*App*.service('ShowTime', ['$filter', function (*$filter*) {

// 为服务添加一个方法

this.now = function () {

return *$filter*('date')(new *Date*, 'yyyy-MM-dd');

}

// 为服务添加一个方法

this.format = function (*format*) {

return *$filter*('date')(new *Date*, *format*);

}

}])

// 测试报时服务

*App*.controller('DemoCtrl2', ['$scope', 'ShowTime', function (*$scope*, *ShowTime*) {

// 将服务执行结果添加至模型上

*$scope*.now = *ShowTime*.now();

*$scope*.time = *ShowTime*.format('hh:mm:ss');

}])

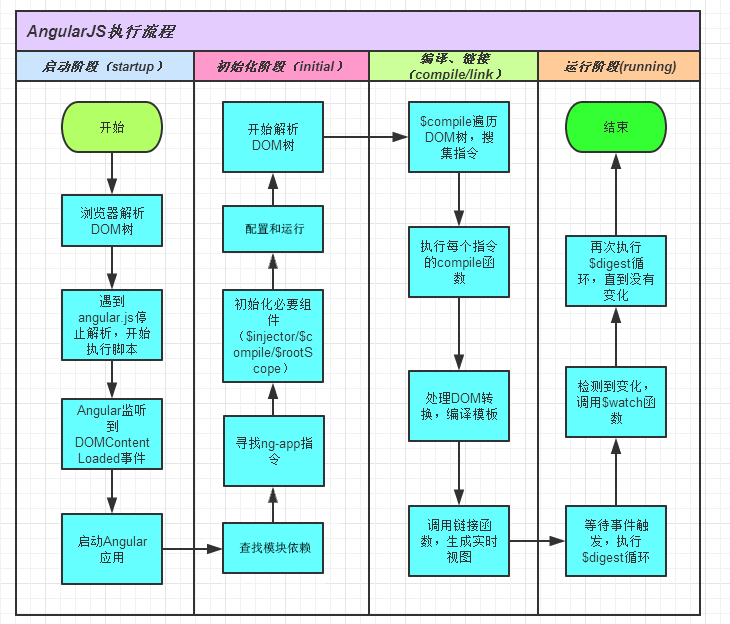
</**script**>

# 事件通信

let app = angular.module( "app", [] );  
/\*  
 \* 控制器模块间通信:  
 \* $scope.$emit:向上冒泡  
 \* $scope.$broadcast:向下传播  
 \* $scope.$on:监听指定的事件  
 \* \*/  
app.controller( "parent", [ "$scope", function ( $scope ){  
 $scope.$on( "Call", function ( params, user ){  
 console.log( arguments );  
 $scope.$broadcast( "CallMing", user );  
 } );  
  
} ] )  
 .controller( "xiaoli", [ "$scope", function ( $scope ){  
 $scope.name = "小丽";  
 $scope.status = "待机";  
 $scope.$on( "CallMing", function ( params, user ){  
 console.log( user, params );  
 $scope.status = `${ user } 来电了`;  
 } );  
 } ] )  
 .controller( "xiaoming", [ "$scope", function ( $scope ){  
 $scope.name = "小明";  
 $scope.status = "待机";  
 $scope.Call = function ( user ){  
 $scope.status = "呼叫" + user;  
 $scope.$emit( "Call", $scope.name );  
 };  
 } ] );

# 模块加载

AngularJS模块可以在被加载和执行之前对其自身进行配置。我们可以在应用的加载阶段配置不同的逻辑。



## 配置块

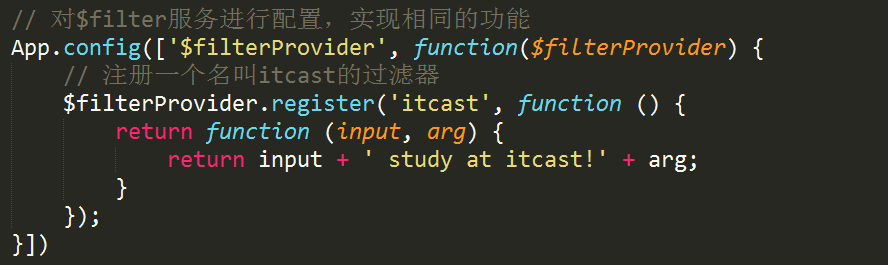
1、通过config方法实现对模块的配置，AngularJS中的服务大部分都对应一个“provider”，用来执行与对应服务相同的功能或对其进行配置。

比如$log、$http、$location都是内置服务，相对应的“provider”分别是$logProvider、$httpProvider、$locationPorvider。

下图以$log为例进行演示，修改了配置



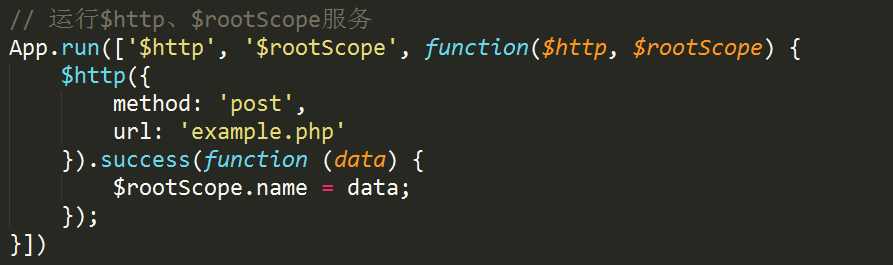
下图以$filter为例进行演示，实现相同功能



见代码示例21 AngularJS配置块.html

## 运行块

服务也是模块形式存在的对且对外提供特定功能，前面学习中都是将服务做为依赖注入进去的，然后再进行调用，除了这种方式外我们也可以直接运行相应的服务模块，AngularJS提供了run方法来实现。



不但如此，run方法还是最先执行的，利用这个特点我们可以将一些需要优先执行的功能通过run方法来运行，比如验证用户是否登录，未登录则不允许进行任何其它操作。

见代码示例22 AngularJS配置块.html

注：此知识点意在了解AngularJS的加载机制。

# 路由

# 二、前端路由

## 概述

我们做的是单页面应用（SPA），使用的是hash路由。#开头的前端路由。虽然看见的是一个页面，但是URL还是在变化。

注意地址栏：

|  |  |
| --- | --- |
| http://qiqu.uc.cn/#!/index/index | http://qiqu.uc.cn/#!/index/joke |

单页面应用的好处就是共享的顶部条不闪烁。古老的前端技术用frameset框架来解决，<frame>标签、<iframe>标签。

**现在，当你想共享页面的头部、侧边栏的时候，单页面应用是最好的选择。所以ipad、手机项目、dashboard（仪表盘、后台面板）项目都喜欢用SPA制作。**

我们会认为不管用户看什么栏目，都把资源已经下载了，实际上这样说不对！**事实上，当你点击到其他栏目的时候，使用Ajax在读取：数据、模板等等东西。**

任何MVC框架都提供了路由功能，**Angular中内置的是ngRoute模块，我们不介绍了，因为功能弱，已经被ui-router的第三方路由干掉了**，想了解的同学请自学：http://www.tuicool.com/articles/jqMveaB

## ui-router的使用

ui-router是第三方的Angular插件，现在基本上大家都用它，功能强大。

API文档：https://ui-router.github.io/ng1/

$ bower install -g ui-router

使用特别的简单，是和jQuery类似，先引用Angular然后引用ui-router

<script type="text/javascript" src="js/lib/angular/angular.min.js"></script>

<script type="text/javascript" src="js/lib/angular-ui-router/release/angular-ui-router.min.js"></script>

<script type="text/javascript">

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

</script>

路由清单：我们依赖的ui.router中提供了一个服务$state，此时可以用config来配置这个服务。用$stateProvider的state方法来设置路由清单。也就是说，定义一个个“状态”。

<script type="text/javascript">

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

//配置路由表，实际上在配置$state服务。

myapp.config(function($stateProvider) {

$stateProvider

**.state(**{

name: 'news',

url: '/news',

template: '<h3>新闻频道</h3>'

}**)**

**.state(**{

name: 'music',

url: '/music',

template: '<h3>音乐频道</h3>'

}**)**

**.state(**{

name: 'movie',

url: '/movie',

template: '<h3>电影频道</h3>'

}**);**

});

</script>

此时页面上不要忘记放置一个

<ui-view></ui-view>

的E级别指令。此时动态的内容（template里面的内容）都将呈现在ui-view里面。

制作超级链接，并不是直接连接到“地址”，而是“状态”上：

<a **ui-sref="news"** ui-sref-active="active">新闻</a>

<a **ui-sref="movie"** ui-sref-active="active">电影</a>

<a **ui-sref="music"** ui-sref-active="active">音乐</a>

ui-sref，s就是state状态的意思。

ui-sref-active自动检测当前匹配上了谁，匹配上了的a标签就自动加上active类名了。

## 引入控制器

myapp.config(function($stateProvider) {

$stateProvider

.state({

name: 'news',

url: '/news',

**controller : "NewsCtrl as newsctrl",**

**template : "<h3>新闻频道~~{{newsctrl.a}}</h3>"**

})

.state({

name: 'music',

url: '/music',

template: '<h3>音乐频道</h3>'

})

.state({

name: 'movie',

url: '/movie',

template: '<h3>电影频道</h3>'

});

});

控制器和服务还是原来的定义方式：

myapp.controller("NewsCtrl",["MathService",function(MathService){

this.a = MathService.m;

}]);

myapp.factory("MathService",[function(){

return {

m : 8

}

}]);

## 子状态

我们说/movie/123 是 /movie的子状态

//配置路由表，实际上在配置$state服务。

myapp.config(function($stateProvider) {

$stateProvider

.state({

name: 'movie',

url: '/movie',

template: '<h3>电影频道</h3>**<ui-view></ui-view>**'

})

.state({

name : "**movie.detail**",

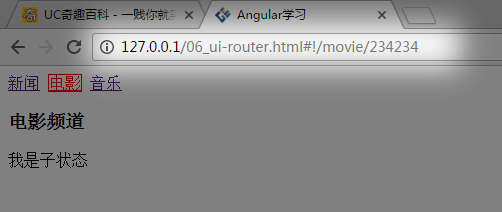
url : "**/:id**",

template : "我是子状态"

});

});

此时name是movie.detail，所以Angular就知道了movie.detail是movie子状态。所以URL会自动拼接，你的/:id实际上是/movie/:id。



制作超级链接的时候：

<a **ui-sref="movie.detail({id:1})"**>电影1</a>

如何得到这个id呢？此时控制器要依赖$state服务：

myapp.controller("MovieCtrl",["$state",function($state){

this.id = **$state.params.id;**

}]);

## 一个页面多个ui-view

Angular中ui-router插件是唯一一个MVC框架中一个页面能放置多个ui-view的插件。

React、vue中一个页面只能有一个view容器。

<div class="left" **ui-view="left"**></div>

<div class="right" **ui-view="right"**></div>

一个状态需要同时定义两个ui-view分别呈递什么：

myapp.config(function($stateProvider) {

$stateProvider

.state({

name: 'news',

url: '/news',

**views : {**

**"left" : {**

**template : "<h1>我是侧边栏</h1>"**

**},**

**"right" : {**

**template : "<h1>我是主要内容</h1>"**

**}**

**}**

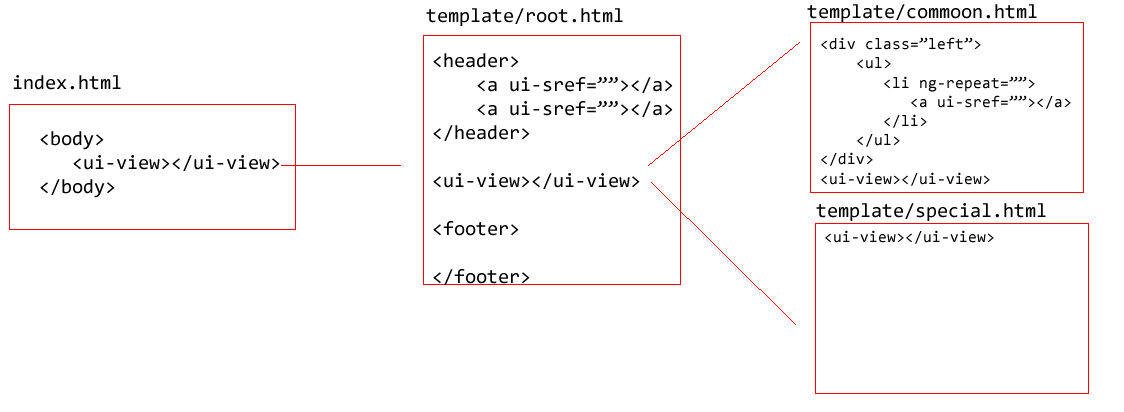
})

});

## ui-view设计

比如网站是这样的：

|  |  |  |  |
| --- | --- | --- | --- |
|  |  |  |  |



## =====分界线====

一个应用是由若个视图组合而成的，根据不同的业务逻辑展示给用户不同的视图，路由则是实现这一功能的关键。

## SPA

SPA（Single Page Application）指的是通单一页面展示所有功能，通过Ajax动态获取数据然后进行实时渲染，结合CSS3动画模仿原生App交互，然后再进行打包（使用工具把Web应用包一个壳，这个壳本质上是浏览器）变成一个“原生”应用。

在PC端也有广泛的应用，通常情况下使用Ajax异步请求数据，然后实现内容局部刷新，局部刷新的本质是动态生成DOM，新生成的DOM元素并没有真实存在于文档中，所以当再次刷新页面时新添加的DOM元素会“丢失”，通过单页面应可以很好的解决这个问题。

## 路由

在后端开发中通过URL地址可以实现页面（视图）的切换，但是AngularJS是一个纯前端MVC框架，在开发单页面应用时，所有功能都在同一页面完成，所以无需切换URL地址（即不允许产生跳转），但Web应用中又经常通过链接（a标签）来更新页面（视图），当点击链接时还要阻止其向服务器发起请求，通过锚点（页内跳转）可以实现这一点。

实现单页面应用需要具备：

a、只有一页面

b、链接使用锚点

c、history api

见代码实例23 锚点.html和24 单页面应用原理分析.html

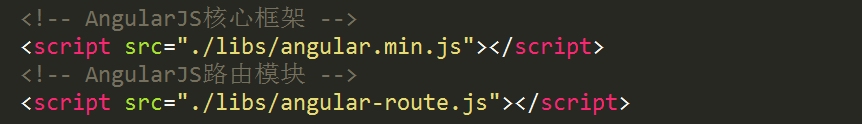
通过上面的例子发现在单一页面中可以能过hashchange事件监听到锚点的变化，进而可以实现为不同的锚点准不同的视图，单页面应用就是基于这一原理实现的。

AngularJS对这一实现原理进行了封装，将锚点的变化封装成路由（Route）,这是与后端路由的根本区别。

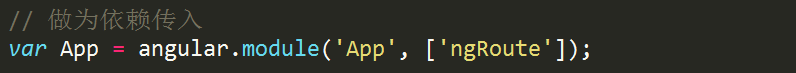
在1.2版前路由功能是包含在AngularJS核心代码当中，之后的版本将路由功能独立成一个模块，[下载angular-route.js](https://code.angularjs.org/)

## 使用

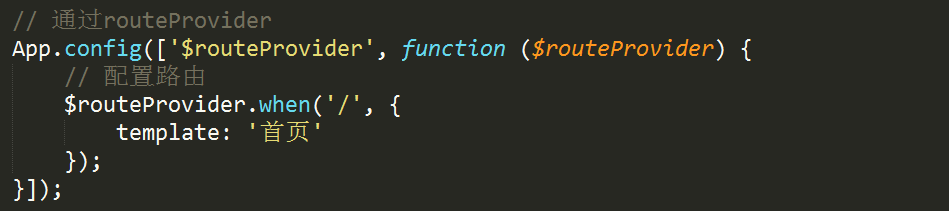
1、引入angular-route.js



2、实例化模块（App）时，当成依赖传进去（模块名称叫ngRoute）。

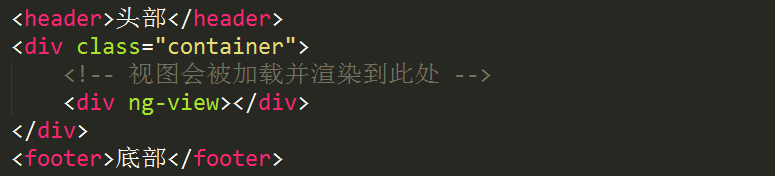


3、配置路由模块



4、布局模板

通过ng-view指令布局模板，路由匹配的视图会被加载渲染到些区域。



见代码实例 25 AngularJS路由.html

## 路由参数

1、提供两个方法匹配路由，分别是when和otherwise，when方法需要两个参数，otherwise方法做为when方法的补充只需要一个参数，其中when方法可以被多次调用。

2、第1个参数是一个字符串，代表当前URL中的hash值。

3、第2个参数是一个对象，配置当前路由的参数，如视图、控制器等。

a、template 字符串形式的视图模板

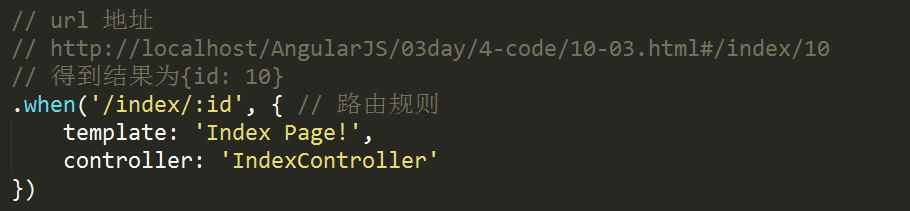
b、templateUrl 引入外部视图模板

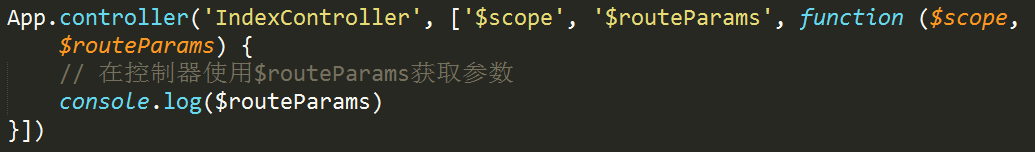
c、controller 视图模板所属的控制器

d、redirectTo跳转到其它路由

见代码实例26 AngularJS路由配置.html

4、获取参数，在控制中注入$routeParams可以获取传递的参数





# AMD规范和CMD规范

## 为什么需要模块化开发管理工具

我们之前做一个项目，会拆分很多js文件：

<script type="text/javascript" src="js/a.js"></script>

<script type="text/javascript" src="js/b.js"></script>

<script type="text/javascript" src="js/c.js"></script>

<script type="text/javascript" src="js/d.js"></script>

<script type="text/javascript" src="js/e.js"></script>

浏览器已经会非常智能的管理他们之间的“关系”，比如b.js中执行了一个函数

fun();

此时a.js文件还没有加载完毕（很可能，比如a.js文件比b.js文件大很多），此时任何浏览器都有一个机制，就是让b.js等待a.js，等待a.js加载完毕之后，再去执行b。果不其然，a.js中定义了一个fun函数。

function fun(){

}

此时就不会发生因为b先回来，所以b认为fun()没有定义而报错的情况。

浏览器天生有依赖管理的“感觉”，那么为什么人们还要发明依赖管理的方式呢？

因为浏览器天生不懒

比如e.js会死等a、b、c、d都加载完毕才执行，结果e.js就是一个语句alert("你好")谁都不依赖！此时干嘛要死等a、b、c、d呢！所以我们要让他们则需加载，就是说运行e的时候，发现需要a、b，此时你再加载a、b。如果一辈子不用a、b此时就一辈子不加载a、b了。也就是说，

我们要让浏览器变懒

换一个角度说，浏览器天生一视同仁，不知道a、b、c、d、e谁是主动调用别人的人，谁是苦工，谁是老大，分不清。

我们要让浏览器分清主次

AMD规范 Asynchronouse Module Defined，异步模块定义，AngularJS、RequireJS是符合AMD规范的

CMD规范 Common Modeule Defined，普通模块定义，NodeJS、SeaJS、commonJS、webpack是符合CMD的

## AMD规范

所有这个模块依赖的模块，都要通过异步来调用，语句放在回调函数里面。不依赖其他模块的语句，就不要放在回调函数里面了，不干扰其他模块的运行。AMD规范使用依赖注入的模式。

我们用RequireJS来讲解。



http://www.requirejs.cn/

┣ js

┃ ┣ require.min.js

┃ ┣ main.js

┣ demo.html

我们在demo.html中同时引用require.min.js，又同时指定main.js入口文件：

<!DOCTYPE html>

<html lang="en">

<head>

<meta charset="UTF-8">

<title>RequireJS学习</title>

</head>

<body>

<script data-main="js/main" src="js/require.min.js"></script>

</body>

</html>

js/main表示js/main.js。 拓展名可以省略。此时main.js中的代码可以执行。

再来一个案例：

┣ js

┃ ┣ require.min.js

┃ ┣ main.js

┃ ┣ math.js

┣ demo.html

main.js中可以声明我依赖math.js，注意.js必须省略拓展名。

/\*配置\*/

requirejs.config({

baseUrl: 'js' → 相对于demo.html的js文件夹路径

});

requirejs([**"math"**],function (**math**) {

alert(**math.pingfang(8)**);

});

你会发现math.js先执行，再执行main.js

math.js中，用define来定义模块，模块暴露的API用return来返回即可。

**define(**[],function(){

return {

pingfang : function(number){

return number \* number;

}

}

}**);**

此时能弹出8的平方，64

为什么是AMD规范呢？就是说如果不需要依赖别人的语句，此时可以不写在回调函数中：

requirejs.config({

baseUrl: 'js'

});

requirejs(["math"],function (math) {

alert(math.pingfang(8));

});

alert("你好");

先弹出你好，然后弹出64。

现在实际上很少有机会不再回调函数中写语句，所以AMD和CMD越来越像！

再来一个案例：

┣ js

┃ ┣ lib

┃ ┃ ┣ require.min.js

┃ ┣ const

┃ ┃ ┣ pi.js

┃ ┣ main.js

┃ ┣ math.js

┣ demo.html

相对于main.js，pi.js在其他文件夹中，所以要在main.js中的config中用paths字段来配置：

requirejs.config({

baseUrl: 'js',

**paths : {**

**"pi" : "const/pi"**

**}**

});

requirejs(["math"],function (math) {

alert(math.pingfang(8));

alert(math.mianji(8));

});

math模块：

define([**"pi"**],function(pi){

return {

pingfang : function(number){

return number \* number;

},

mianji : function(r){

return pi.pi \* r \* r;

}

}

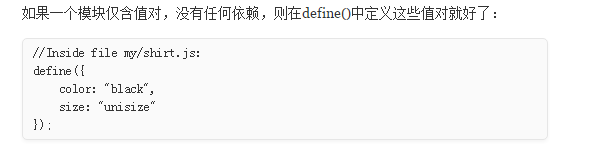
});

pi.js文件

define({

"pi" : 3.1415

});



## CMD规范

CMD太简单了，就是NodeJS的那些事儿，require、exports.\*\*=\*\*、module.exports = \*\*。

CMD在前端使用commonJS这个东西，在中国seajs远比commonjs火爆。

<!DOCTYPE html>

<html lang="en">

<head>

<meta charset="UTF-8">

<title>CMD规范</title>

</head>

<body>

<div id="box">你好</div>

<script type="text/javascript" src="js/lib/sea.min.js"></script>

<script type="text/javascript">

**seajs.config({**

**base: "./js", → 注意写./ js文件存放的根目录**

**alias: { → 别名**

**"jquery" : "lib/jquery"**

**}**

**});**

// 加载入口模块

**seajs.use("main"); → 入口模块，相对于base的路径**

</script>

</body>

</html>

模式是任何的模块文件都需要用define()来包裹一下，里面有一个函数，函数sea帮我们传入require、exports、module对象：

main.js文件：

**define(function(require,exports,module){**

//声明依赖

var math = require("math");

var People = require("People");

alert(math.pingfang(8));

var xiaoming = new People("小明",3);

xiaoming.sayHello();

$("#box").animate({"font-size":100},1000);

**});**

math文件：

**define(function(require,exports,module){**

//暴露

exports.pingfang = function(number){

return number \* number;

}

**});**

People.js文件：

define(function(require,exports,module){

function People(name,age){

this.name = name;

this.age = age;

}

People.prototype.sayHello = function(){

alert("我是" + this.name + this.age);

}

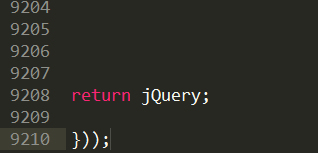
//暴露依赖

**module.exports = People;**

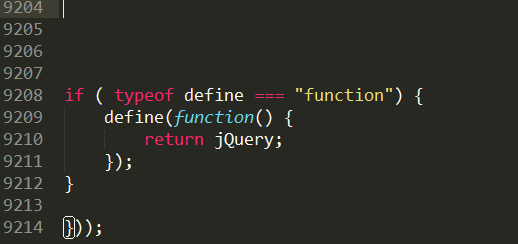
});

说一下使用jQuery

jQuery必须修改为CMD规范才能被引用，改变jQuery的最后的代码：



变为：



在模块里面就可以：

var $ = require("jquery");

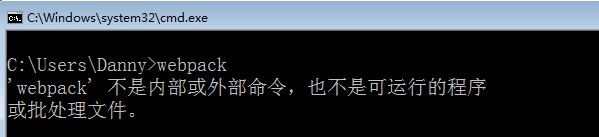
但是seajs慢慢不火了，因为有一个东西更牛，叫做webpack。

## webpack

webpack是CMD规范的构建工具，可以让我们裸写CMD规范的程序，帮我们自动打包成为一个js文件。

我们先简单学习webpack，而webpack非常复杂，功能极多，好多loader、好多插件、好多配置。

今天只看一下它编译CMD的能力。



$ cnpm install -g webpack

来一个简单案例：

┣ main.js

┣ math.js

这两个文件可以无脑裸写CMD规范代码，就是require、exports、module那些：

main.js:

var math = **require("./math");**

alert(math.pingfang(36));

math.js：

**exports.pingfang = function(number){**

return number \* number;

**}**

此时我们要用webpack将两个文件编译为一个文件，重要的不是说合并文件，而是用CMD规范编译出浏览器能够运行的程序。

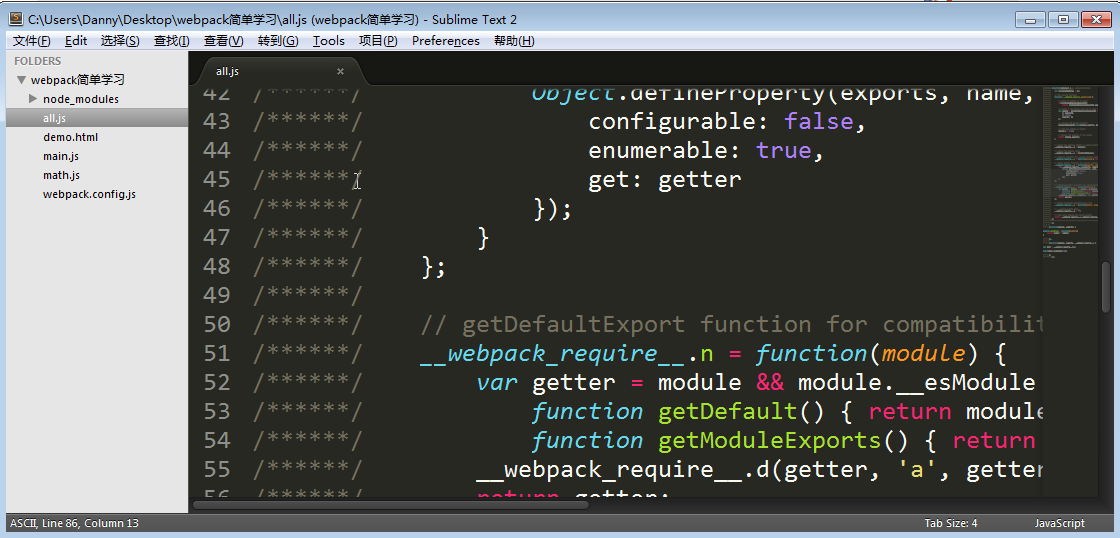
$ cd 项目文件夹

$ webpack main.js all.js

语法就是

$ webpack 入口文件 出口文件

此时all.js就是浏览器能够认识的程序：



demo.html仅仅需要引入all.js即可。

但是这样是不方便的，比如当我们修改了main.js或者math.js之后，就需要重新webpack main.js all.js一下。为了方便，我们可以用webpack.config.js文件来配置webpack如何工作。

所以，我们创建了webpack.config.js文件：

var webpack = require('webpack')

module.exports = {

entry: './main.js', //入口

output: { //出口

path: \_\_dirname,

filename: 'all.js'

},

watch : true //监控文件变化

}

这个文件的写法是从官网上抄下来的，不需要背诵。

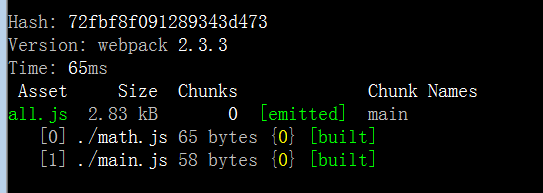
此时要在项目中重新安装一次webpack

$ cnpm install webpack

然后敲击webpack命令：

$ webpack

webpack将自动挂起，侦测你的文件是否发生了改变，如果改变了，自动保存为all.js：



今天我们先讲解到这里，webpack还有好多功能后面学习我们会遇见。

# angular工程化Angular-Async-Loader

## Angular在工程化上遇见的挑战

https://github.com/subchen/angular-async-loader

慢慢说这个东西是什么。

Angular是2009年发明的东西，它第一次提出了“依赖注入”思想，AMD风格由此流行开来。

思想特别好，但是Angular想错了一个事情，就是我们开发的时候程序是很大的，所以要把控制器、服务、指令、路由都要分开文件写。此时我们可以：

<script type="text/javascript" src="js/lib/angular/angular.min.js"></script>

<script type="text/javascript" src="js/app.js"></script>

<script type="text/javascript" src="js/MainCtrl.js"></script>

app.js：

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

MainCtrl.js

myapp.controller("MainCtrl",[function(){

this.a = 100;

}]);

此时问题是两个：

1） 偶尔性大，controller文件夹中没有体现出我依赖myapp这个事情。myapp一旦改名字，所有的js都要改变。

2） 浏览器又变成了“不懒”的模式，不管用户去不去News栏目，此时都会加载NewsService。依赖注入形同虚设。

我们想到了用requirejs。但是裸着使用requirejs也有问题：

比如一个控制器要依赖一个服务，此时

define([**MathService**],function(**MathService**){

myapp.controller("MainCtrl",[**"MathService"**,function(**MathService**){

this.a = 100;

}]);

});

代码重复性太高，所以RequireJS和Angular之间感觉少了一个“粘合剂”，此时有一个著名的解决方案叫做angular-async-loader，可以轻松优雅的解决RequireJS和Angular之间粘合问题。说一句，技术栈各有千秋，市面上甚至有用webpack去编译angular1的，但是不主流。我们的课程会介绍主流技术栈。不管用什么工程化的构建体系，业务代码是完全相同的！（想一下前天的消消乐）。不管哪种构建体系，都不会简化你的业务。

## 配置angular-async-loader 安装前端依赖

去官网：https://github.com/subchen/angular-async-loader

下方就有demo教程API。

大体思路就是用RequireJS去架构Angular项目。

进入项目文件夹，创建bower前端依赖文件：

$ bower init

创建一个.bowerrc文件（用rename奇淫技巧来创建），写入

{

"directory" : "assets"

}

安装angular

$ bower install angular --save

安装ui-router

$ bower install ui-router --save

安装requirejs

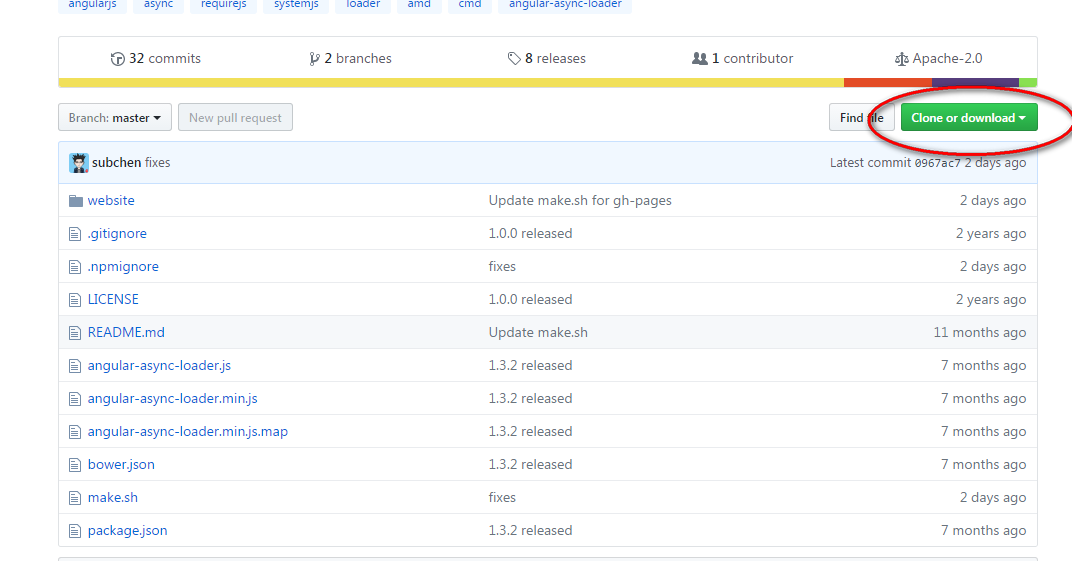
$ bower install requirejs --save

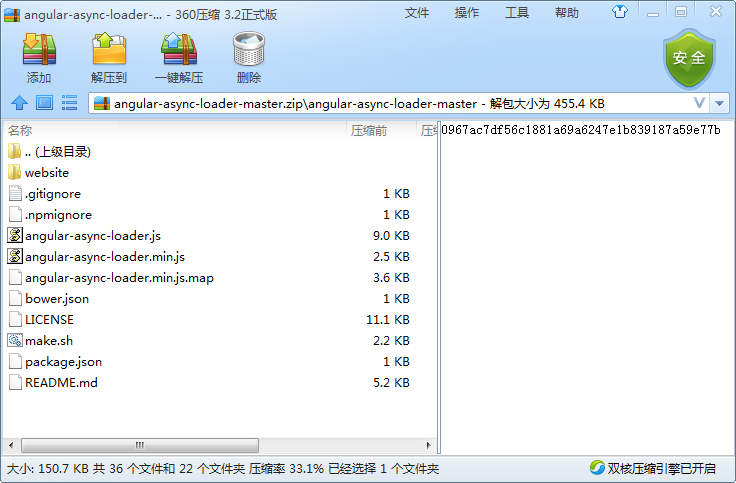
安装angular-async-loader（注意这里我们遇见了一个版本的问题，bower会下载1.0.x的东西，但实际上已经有了1.3.x），

$ bower install angular-async-loader --save

所以我们不从bower下载angular-async-loader了，直接去

https://github.com/subchen/angular-async-loader 点击绿色按钮下载，





将angular-async-loader.js和angular-async-loader.min.js文件拷贝进入

assets\angular-async-loader中就行了。

至此我们的bower.json文件（就是刚才bower init生成的）已经变为：

{

"name": "htdocs",

"authors": [

"shaoshanhuan <shaoshanhuan@163.com>"

],

"description": "",

"main": "",

"license": "MIT",

"homepage": "",

"ignore": [

"\*\*/.\*",

"node\_modules",

"bower\_components",

"js/lib",

"test",

"tests"

],

**"dependencies": {**

**"angular-async-loader": "^1.0.1",**

**"angular": "^1.6.4",**

**"requirejs": "^2.3.3",**

**"angular-ui-router": "ui-router#^0.4.2"**

**}**

}

## 书写三大文件

然后开始正式写“三大文件”。

bootstrap.js文件，这个文件是requirejs的入口文件：

require.config({

baseUrl: '/',

**//别名**

paths: {

'angular': 'assets/angular/angular.min',

'angular-ui-router': 'assets/angular-ui-router/release/angular-ui-router.min',

'angular-async-loader': 'assets/angular-async-loader/dist/angular-async-loader.min'

},

**//声明paths中列出的元素的暴露的接口和依赖**

shim: {

'angular': {exports: 'angular'}, //暴露的是angular

'angular-ui-router': {deps: ['angular']} //依赖的是angular

}

});

**//核心入口**

require(['angular', './app-routes'], function (angular) {

**//当整个文档就绪之后**

angular.element(document).ready(function () {

**//angular.bootstrap是一个方法，表示启动angular。之前就是人肉用ng-app指令来启动，这里更高级用程序来自动。**

**angular.bootstrap(document, ['app']);**

**//添加ng-app指令，通过类名添加，实际也可以通过attr**

angular.element(document).find('html').addClass('ng-app');

});

});

app.js

define(function (require, exports, module) {

**//这是一个CMD规范的模块，为什么突然是CMD了呢？因为这个模块的目的是向外暴露app整体**

**//而AMD只能暴露JSON形式的API，不方便，所以就借用了一下CMD的壳子**

**//引入依赖**

var angular = require('angular');

var asyncLoader = require('angular-async-loader');

require('angular-ui-router');

**//创建app对象！app对象诞生了！声明依赖ui-router：**

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

**// initialze app module for angular-async-loader，官网让写的**

asyncLoader.configure(app);

**//向外暴露！**

module.exports = app;

});

app-routes.js

define(function (require) {

//引入app对象

var app = require('./app');

//定义路由！

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

$urlRouterProvider.otherwise('/home');

$stateProvider

.state('home', {

url: '/home',

template: '<h1>我是首页！你已经成功起步！</h1>',

// controllerUrl: 'home/homeCtrl',

// controller: 'homeCtrl'

});

}]);

});

此时创建一个index.html（唯一的单页面），创建一个ui-view容器，然后用requirejs语法去引用我们的入口文件bootstrap.js即可。

<!DOCTYPE html>

<html lang="en">

<head>

<meta charset="UTF-8">

<title>Angular起步</title>

</head>

<body>

**<ui-view></ui-view>**

**<script src="/assets/requirejs/require.js" data-main="bootstrap.js"></script>**

</body>

</html>

## 标准壳

我们现在要创建控制器、服务、指令都要单独写在js文件中，此时这些js文件不能裸奔，因为是在requirejs模式下，此时就要有一个标准壳。

define(function (require) {

var app = require("app");

require("./rootService");

app.controller('RootCtrl', ["rootService",function (rootService) {

this.a = rootService.m;

}]);

});

define(function (require) {

var app = require("app");

app.factory("rootService",function(){

return {

m : 9

}

});

});

## 说一下jQuery

最简单的办法就是让index.html（唯一的页面）引用jQuery，就会在window对象上加上$函数。

<!DOCTYPE html>

<html lang="en">

<head>

<meta charset="UTF-8">

<title>Angular起步</title>

<link rel="stylesheet" href="css/css.css">

</head>

<body>

<ui-view></ui-view>

<**script type="text/javascript" src="/assets/jquery/dist/jquery.min.js"></script>**

<script src="/assets/requirejs/require.js" data-main="bootstrap.js"></script>

</body>

</html>

这种方法缺点就是不管页面用不用jquery，总是最先加载完毕了，有点不符合则须加载。

第二种方法就是用requirejs记载jquery，jquery文件不需要做任何的更改，和SeaJS不一样。

require.config({

baseUrl: '/',

//别名

paths: {

'angular': 'assets/angular/angular.min',

'angular-ui-router': 'assets/angular-ui-router/release/angular-ui-router.min',

'angular-async-loader': 'assets/angular-async-loader/angular-async-loader.min',

**'jquery' : 'assets/jquery/dist/jquery.min'**

},

//声明paths中列出的元素的暴露的接口和依赖

shim: {

'angular': {exports: 'angular'}, //暴露的是angular

**'jquery': {exports: 'jquery'}**, //暴露的是angular

'angular-ui-router': {deps: ['angular']} //依赖的是angular

}

});

//核心包

require(['angular', './app-routes'], function (angular) {

//当整个文档就绪之后

angular.element(document).ready(function () {

//angular.bootstrap是一个方法，表示启动angular。之前就是人肉用ng-app指令来启动，这里更高级用程序来自动。

angular.bootstrap(document, ['app']);

//添加ng-app指令，通过类名添加，实际也可以通过attr

angular.element(document).find('html').addClass('ng-app');

});

});

在需要使用jQuery的控制器中：

define(function(require){

var app = require('app');

**var $ = require("jquery");**

app.controller('HomeCtrl', [function () {

this.a = 100;

**$(".box").animate({"font-size":100},100,function(){**

**$(this).css("color","red");**

**})**

}]);

});

要引入jQuery的插件呢？此时bower下载jquery-ui，然后改变bootstrap.js：

require.config({

baseUrl: '/',

//别名

paths: {

'angular': 'assets/angular/angular.min',

'angular-ui-router': 'assets/angular-ui-router/release/angular-ui-router.min',

'angular-async-loader': 'assets/angular-async-loader/angular-async-loader.min',

'jquery' : 'assets/jquery/dist/jquery.min',

**'jquery-ui' : 'assets/jquery-ui/jquery-ui.min'**

},

//声明paths中列出的元素的暴露的接口和依赖

shim: {

'angular': {exports: 'angular'}, //暴露的是angular

'jquery': {exports: 'jquery'}, //暴露的是angular

'angular-ui-router': {deps: ['angular']}, //依赖的是angular

**'jquery-ui': {deps: ['jquery']}**  //依赖的是jquery

}

});

//核心包

require(['angular', './app-routes'], function (angular) {

.....

});

控制器：

define(function(require){

var app = require('app');

**require("jquery");**

**require("jquery-ui");**

app.controller('HomeCtrl', [function () {

this.a = 100;

$(".box").animate({"font-size":100},100,function(){

$(this).html("我现在可以被拖拽，试试看吧！")

$(this).css("color","red");

$(this).draggable();

})

}]);

});

至此项目结构是：

|  |
| --- |
| ┣ bootstrap.js  ┣ app.js  ┣ app-routes.js  ┣ index.html  ┣ ngApp  ┣ home  ┣ HomeCtrl.js  ┣ homeService.js  ┣ home.html  ┣ music  ┣ MusicCtrl.js  ┣ musicService.js  ┣ music.html  ┣ css  ┣ css.css  ┣ assets  jquery  jquery-ui  angular  aal |

## 结合Nodejs做项目

前端MVVM的，后端是MVC的（后端没有v了，不用ejs这种后端模板）。

此时Angular的层次就要放到www文件夹中了，我们用nodejs+express来跑静态路由，工作在3000端口。

|  |
| --- |
| ┣ app.js  ┣ models  ┣ controllers  ┣ node\_modules  ┣ www  ┣ bootstrap.js  ┣ app.js  ┣ app-routes.js  ┣ index.html  ┣ ngApp  ┣ home  ┣ HomeCtrl.js  ┣ homeService.js  ┣ home.html  ┣ music  ┣ MusicCtrl.js  ┣ musicService.js  ┣ music.html  ┣ css  ┣ css.css  ┣ assets  jquery  jquery-ui  angular  aal |

一会儿要进入一个3天的项目，技术栈：

Node.js + Express + MongoDB + Mongoose + Angular1 + RequireJS + ui-rooter + BootStrap + jQuery + jQueryUI + char.js

# 说说项目的开发-基本起步

# 其它

## jQuery

在没有引入jQuery的前提下AngularJS实现了简版的jQuery Lite，通过angular.element不能选择元素，但可以将一个DOM元素转成jQuery对象，如果引提前引入了jQuery则angular.element则完全等于jQuery。

见代码示例28AngularJS中使用jQuery.html

## bower

基于NodeJS的一个静态资源管理工具，由twitter公司开发维，解决大型网站中静态资源的依赖问题。

1、依赖NodeJS环境和git工具。

2、npm install -g bower安装bower

3、bower search 查找资源信息

4、bower install 安装（下载）资源，通过#号可以指定版本号

5、bower info 查看资源信息

6、bower uninstall 卸载（删除）资源