Contents

[DOM and BOM --< JavaScript Dom编程艺术> 1](#_Toc440624348)

[BOM (Browser Object Model，浏览器对象模型---浏览器窗口和桌面屏幕) 1](#_Toc440624349)

[DOM (Document Object Model, 文档对象模型---页面，文档) = Window.document or iframe.document 2](#_Toc440624350)

[文档元素查询 2](#_Toc440624351)

[元素样式查看和修改 3](#_Toc440624352)

[事件监听与传播 3](#_Toc440624353)

[动画js 4](#_Toc440624354)

[Ajax应用 5](#_Toc440624355)

[jQuery --< Learning jQuery > 7](#_Toc440624356)

[基本概念 7](#_Toc440624357)

[如何通过jQuery查询网页元素 7](#_Toc440624358)

[由JQuery处理事件 8](#_Toc440624359)

[修改样式与实现动画 10](#_Toc440624360)

[动态修改DOM 11](#_Toc440624361)

[通过Ajax与服务端交互数据 12](#_Toc440624362)

[加载html文件 (片段) 12](#_Toc440624363)

[加载js文件，自动执行js文件 13](#_Toc440624364)

[从第三方服务器加载 （跨域加载） 13](#_Toc440624365)

[Angularjs 15](#_Toc440624366)

[理解回合式和单页面应用程序 15](#_Toc440624367)

[AngularJS工具函数 15](#_Toc440624368)

[模块 16](#_Toc440624369)

[模型 16](#_Toc440624370)

[控制器Module.controller 16](#_Toc440624371)

[控制器继承 17](#_Toc440624372)

[控制器通信 17](#_Toc440624373)

[AngularJS与其他框架集成 18](#_Toc440624374)

[视图 19](#_Toc440624375)

[过滤器Module.filter 19](#_Toc440624376)

[内置过滤器 19](#_Toc440624377)

[服务 20](#_Toc440624378)

[内置服务 20](#_Toc440624379)

[自定义服务 21](#_Toc440624380)

[指令 24](#_Toc440624381)

[内置指令(包括数据绑定，表单验证，模板生成，事件处理和HTML元素操作) 24](#_Toc440624382)

[自定义指令: 扩展标准HTML词汇 28](#_Toc440624383)

[URL路由 32](#_Toc440624384)

[ngRoute (内置模块) 32](#_Toc440624385)

[UI-Router (参考google: angular-ui模块) 34](#_Toc440624386)

[处理危险数据 36](#_Toc440624387)

[$sce（strict contextual escaping SCE，严格上下文转义） 36](#_Toc440624388)

[$sanitize 37](#_Toc440624389)

[Ajax 37](#_Toc440624390)

[$http请求 37](#_Toc440624391)

[ngResource 41](#_Toc440624392)

[动画和触摸服务： 41](#_Toc440624393)

[单元测试 41](#_Toc440624394)

[angular-bootstrap: native angularJS directives for bootstrap 41](#_Toc440624395)

[*Accordion (ui.bootstrap.accordion):* 42](#_Toc440624396)

[*Alert (ui.bootstrap.alert):* 42](#_Toc440624397)

[*Buttons (ui.bootstrap.buttons):* 42](#_Toc440624398)

[*Carousel (ui.bootstrap.carousel):* 43](#_Toc440624399)

[*Collapse (ui.bootstrap.collapse): (hide/show element with css transition)* 43](#_Toc440624400)

[Datepicker(ui.bootstrap.datepicker): 有inline and popup两种形式 43](#_Toc440624401)

[Dropdown (ui.bootstrap.dropdown): 44](#_Toc440624402)

[Modal (ui.bootstrap.modal): 44](#_Toc440624403)

[Pagination (ui.bootstrap.pagination) 45](#_Toc440624404)

[Popover (ui.bootstrap.popover) 46](#_Toc440624405)

[Tooltip (ui.bootstrap.tooltip): 同popover 46](#_Toc440624406)

[Progressbar (ui.bootstrap.progressbar) 46](#_Toc440624407)

[Rating (ui.bootstrap.rating) 47](#_Toc440624408)

[Tabs (ui.bootstrap.tabs) 47](#_Toc440624409)

[Timepicker (ui.bootstrap.timepicker) 48](#_Toc440624410)

[Typeahead (ui.bootstrap.typeahead): create typeaheads with any form text input 48](#_Toc440624411)

[angular-ui-grid: angular表格 （含排序，过滤，表格头和单元内容可以采用定制模板，标签国际化） 50](#_Toc440624412)

[Angular-nvD3: (an angularjs directive for NVD3 re-usable charting library (based on D3) 52](#_Toc440624413)

# DOM and BOM --< JavaScript Dom编程艺术>

浏览器只是一个观察器，无法执行简单的计算任务（因为不能在本地机器上执行任何程序，万一程序有bug和病毒）。

如何解决浏览器的交互问题？客户端编程（Web浏览器执行任何它可以完成的工作）

Web浏览器就像一个功能受限的操作系统。客户端编程的几种方法:插件 （浏览器端下载插件，插件提供了一个后门，使得可创建新的客户端编程语言）; 脚本语言 （插件引发了浏览器脚本语言的开发）

通过使用某种脚本语言，你可以将客户端程序的源代码直接嵌入到HTML页面中，解释这种语言的插件在HTML页面被显示时自动激活

不需要插件的脚本语言是Javascript（所有浏览器都得实现和支持）;需要插件的语言：Java（通过applet以及使用Java web start来进行客户端编程,applet是只在Web浏览器中运行的小程序，它是作为网页的一部分而自动下载的）；Flex (通过flash player)；.NET (通过silverlight)

## BOM (Browser Object Model，浏览器对象模型---浏览器窗口和桌面屏幕)

Window.全局变量

Window.Javascript核心函数()

Window.navigator (浏览器相关的信息)

Window.location (存储当前载入页面URL信息)

Window.history (同一个浏览器会话中存储有限的访问记录)

Window.frames (当前页面中所有frame元素的集合)

Window.screen(浏览器之外的桌面信息)

Window.open() (打开新浏览器窗口即弹出窗, 现在一般用模态框即在页面上插入浮动的<div>实现)

Window.close()

Window.moveTo(x,y)

Window.moveBy(deltaX, deltaY)

Window.resizeTo(x, y)

Window.resizeBy(x, y)

Window.alert()

Window.prompt()

Window.confirm()

Window.setTimeout()

Window.setInterval()

## DOM (Document Object Model, 文档对象模型---页面，文档) = Window.document or iframe.document

将HTML文档解析成树，提供API访问页面任何元素，并进行元素的修改，删除和添加等操作。

DOM: document object model，由节点组成，**分元素节点，文本节点和属性节点**。

元素节点：<body>,<p>…，元素节点可以包含元素节点

文本节点：被包含在元素节点的内部，如元素节点<p title=”hi”>qzlin</p>包含着文本节点”qzlin”

属性节点：被包含在元素节点的内部，如元素节点<p title=”hi”>qzlin</p>包含着属性节点title=”hi”

### 文档元素查询

document.getElementById(“idName”) ;

document.getElementsByClassName(“className”) ;

var elementNode = document.getElementsByTagName(“p”)[0] ;

elementNode.childNodes; //获取元素的所有子元素

elementNode.firstChild; //等价于elementNode.childNodes[0]

elementNode.lastChild ; //等价于elementNode.childNodes[elementNode.childNodes.length-1]

elementNode.parentNode ; //获取元素节点的父元素

elementNode.previousSibling; //获取元素节点的上一个兄弟元素

elementNode.nextSibling ; //获取元素节点的下一个兄弟元素

elementNode.nodeType ; //判断节点类型，若1是元素节点，若2是属性节点，若3是文本节点。

elementNode.nodeName; //元素标签名如”p”

// 文本节点值的查看和修改

var textNode = elementNode.childNodes[0] ;

textNode.nodeValue;

textNode.nodeValue = “I am qzlin” ;

文档元素属性查看和修改

element.getAttribute(“title”) ; 等价于element.title

element.setAttribute(“title”, “qzlin”) ; 等价于element.title = “qzlin”

文档元素动态创建

// innerHTML属性查询和修改元素节点的HTML内容

elementNode.innerHTML = “<p>I inserted <em>this</em> content.</p>” ;

// 创建元素节点，并插入到另一个元素节点中

var txt = document.createTextNode(“qzlin”) ; //创建文本节点

var para = document.createElement(“p”) ; //创建元素节点

para.appendChild(txt) ; //元素节点包含文本节点

elementNode.appendChild(para) ; //文档元素节点 <- 新创建的元素节点

elementNode.parentNode.insertBefore(para, elementNode) ; //新创建的元素节点 插入 之前

form.childNodes ; //表单的所有子节点

form.elements ; //表单的所有表单字段

form.elements[0].value ;

**encodeURIComponent(..) ; 值编码成URL安全的字符串。**

### 元素样式查看和修改

:hover and :focus等伪类允许根据触发事件改变元素的呈现效果，**CSS正在利用伪类走进DOM的领地**，DOM通过样式给元素设定样式。(CSS与js技术重叠区域)

element.style; //style是个对象

//element.style.property ;

element.style.color

element.style.fontWeight ; //为什么不能用element.style.font-weight，因为”-“是保留字符，减号操作符，规定用驼峰命名法来替代。

**惯用法是在css里定义样式类，然后通过js更新的元素的class属性**。

.intro {

color: #fff ;

font-weight: bold ;

}

element.className += “ intro” ; //注意前面有空格，是添加类名,

### 事件监听与传播

事件传播分成三个阶段：先在标签上使用捕捉法，而后使用冒泡法。即**事件传播的路径应该是先从document到相关链接（或标签），然后回到document.**

事件类型：鼠标类，键盘类，载入/窗口类，表单类

window.onload = function() {

firstFun() ;

secondFun() ;

}

⬄

function addLoadEvent(func) {

var oldonload = window.onload ;

if (typeof window.onload != ‘function’) {

window.onload = func ;

} else {

window.onload = function() {

oldonload() ;

func() ;

}

}

}

addLoadEvent(firstFun) ;

addLoadEvent(secondFun) ;

### 动画js

**随时间变化的样式：按照预定的时间间隔重复调用一个函数，从而随着时间的推移而不断改变改变某个元素的样式。**

var timeout = setTimeout(‘fun’, delayTime) ; //在指定的毫秒数后调用函数

var interval = setInterval(“fun”, deltaTime) ; //周期性（毫秒）调用函数，直至clearInterval(interval)或窗口被关闭。

clearTimeout(timeout) ; //取消“等待执行”队列里的某个函数。

clearInterval(interval) ; //

图片滑动，每次只显示指定区域

所有预览图片生成一张“集体照”形式的照片。隐藏“集体照”图片的绝大部分。点击事件导致子元素欲显示部分滑动到父元素显示区域

#preview {

position: absolute ;

left: 0px ;

top: 0px ;

}

<div id=”slideshow”>

<img src=”images/topics.gif” id=”preview”/>

</div>

function prepareSlideshow() {

var preview = document.getElementById(“preview”) ;

var links = list.getElementsByTagName(“a”) ;

links[0].onmouseover = function() {

moveElement(“preview”, -100, 0, 10) ;

}

}

function moveElement(elementId, x1, y1, interval){

var elem = document.getElementId(elementId) ;

if (elem.movement) clearTimeout(elem.movement) ;

var xpos = parseInt(elem.style.left)；

var ypos = parseInt(elem.style.top) ;

if (xpos == x1 && ypos == y1) return true ;

xpos = xpos<x1 ? xpos++ : xpos-- ;

ypos = ypos<y1 ? ypos++ : ypos-- ;

elem.style.left = xpos + “px” ;

elem.style.top = ypos + “px” ;

var repeat = “moveElement(‘”+elementId+”’, “+x1+”, “+y1+”, “+interval+”)” ;

ele.movement = setTimeout(repeat, interval) ;

}

### Ajax应用

XMLHttpRequest()：一个用构建HTTP请求的Javascript对象（即构造器）

function request(url, callback) {

var xhr = new XMLHttpRequest() ;

xhr.onreadystatechange =

(function(myxhr){

return function() { callback(myxhr) ; }

})(xhr) ;

xhr.open(‘GET’, url, true) ;

xhr.send(‘’) ;

}

request(

‘data/content.json’,

function(o) {

document.getElementById(‘text’).innerHTML = o.responseText ;

}

) ;

<!DOCTYPE html>

<html>

<head>

<title>Bar Chart Display</title>

<link rel="stylesheet" href="script06.css">

<script src="script06.js"></script>

</head>

<body>

<div id="chartType">

Choose a chart<br>

<input type="radio" name="type" value="browser" checked="checked"> Browser Usage<br>

<input type="radio" name="type" value="platform"> JavaScript Usage<br>

</div>

</body>

</html>

如何获取标签元素节点(element node)？

var inputs = document.getElementsByTagName("input");

var radioButtons = new Array(inputs.lengh);

for (var i=0; i< inputs.length; i++) {

if (inputs [i].type == "radio") {

radioButtons.push(input[i]);

}

}

如何获取指定name and 打勾的element node?

function getButton(buttonSet) {

for (var i=0; i<radioButtons.length; i++) {

if (radioButtons[i].name == buttonSet && radioButtons[i].checked) {

return radioButtons[i].value;

}

}

return -1;

}

总结：javascript是基于对象编程，所以

**元素节点是一个对象，含属性和方法，访问属性和方法，并做出判断，如上，inputs是指元素节点input的数组，inputs[i]是指input对象，inputs[i].type是指访问input对象的属性type**. (同理, radioButtons是指radio的数组，radioButtons[i]是radioButton对象，radioButtons[i].name or radioButtons[i].checked是属性)

# jQuery --< Learning jQuery >

**jQuery利用CSS选择符创建jQuery对象，为这些对象提供丰富的方法，批量操作其中的DOM元素。**

## 基本概念

$(‘css选择符’): 返回jQuery对象实例，jQuery对象会封装0个或多个DOM元素，允许我们以多种不同的方式与这些DOM元素进行交互,如给jQuery对象绑定事件，添加样式。

$(document).ready(

//只有在DOM加载完毕后，查找和操作页面元素才有效

function(){

$('div.poem-stanza').addClass('highlight') ;

}

) ;

DOM(document object model文档对象模型)：充当JavaScript与网页之间的接口；它以对象网络而非纯文本的形式来表现型HTML的源代码。

## 如何通过jQuery查询网页元素

* 基本选择符:标签名，ID和类

$(‘p’)

$(‘#some-id’)

$(‘.some-class’)

//ID为selected-plays的子元素列表项

$(‘#selected-plays > li’)

* 属性选择符

//带有alt属性的所有图像元素

$(‘img[alt]’)

//属性选择符借鉴正则表达式挑选属性中满足条件的元素

$(‘a[href^=<mailto:>]’)

$(‘a[href$=”.pdf”]’)

$(‘a[href^=”http”][href\*=”henry”]’)

* 自定义选择符

类似与CSS中的伪类选择符语法，即选择符以冒号:开头。

特别注意：javascript数组采用从0开始的编号方式，所以eq(1)取得的是集合中的第2个元素，而CSS则是从1开始，则nth-child(1)取得的是第1个子元素

//$(‘集合:eq(1)’)

$(‘div.horizontal:eq(1)’)

$(‘tr:odd’)

$(‘tr:even’)

$(‘td:contains(henry)’)

// ID为selected-plays的子元素列表项, 当中没有horizontal类

$(‘#selected-plays li:not(.horizontal)’)

* 基于表单的选择符

//选择所有选择中的单选按钮

$(‘input[type=”radio”]:checked’)

* 遍历DOM树选择元素

//.filter()接受函数参数，执行复杂的测试，以决定相应元素是否应保留在匹配集合中。.filter()方法会迭代所有匹配的元素，对每个元素都调用传入的函数并测试函数的返回值。如果函数返回false,则从匹配集体中删除相应元素；如果返回true,则保留相应元素.

$(‘a’).filter(function(){ return this.hostname && this.hostname != location.hostname;})

$(‘a’).next()

$(‘a’).addBack()

$(‘a’).nextAll()

$(‘a’).prev()

$(‘a’).prevAll()

//选择处于相同DOM层次的所有其他元素

$(‘a’).siblings()

$(‘a’).parent().children()

* **jQuery对象转DOM对象**

***注意：以上返回元素是jQuery对象，若欲获得DOM对象，采用索引方式***

**var element = $(‘#my-element’)[i];**

**从而可以直接对DOM对象直接操作 （见javascript DOM编程艺术）**

## 由JQuery处理事件

* 页面加载时事件

$(document).ready(function(){

…;

}) ;

注意$(documend).ready()与window.onload()区别：onload事件设置界面，那么用户在能够使用这个页面之前必须要等到每一幅图像都下载完成; ready()也同样会在DOM完全就绪并可以使用时调用,也同样可以访问所有元素，但不意味着所有关联的文件都已经下载完毕，所以类似图像的高度和宽度这样的属性不一定有效

* 当触发任何事件处理程序时，关键字this引用的都是携带相应行为的DOM元素.$(this)为相应的元素创建jQuery元素.

//.CSS文件

.hidden {

display: none ;

}

.selected{

…;

}

$(‘#switcher button’).click(function(){

$(this).addClass(‘selected’) ;

//根据相应的类是否存在而添加或删除类

$(#switcher h3).toggleClass(‘hidden’) ;

…;

}) ;

* 事件传播

首先，事件要从一般元素到具体元素逐层捕获，然后，事件再通过冒泡返回DOM树的顶层. **jQuery始终会在模型的冒泡阶段注册事件处理程序，因此我们总是可以假定：最具体的元素会自先获得响应事件的机会。**

*关于事件传播的几个难点:*

* :hover伪类选择符，这个选择符可以让样式表在用户鼠标指针悬停在某个元素上时，影响元素的外观。

//jQuery hover()函数，第一个参数是鼠标指标进入时执行的函数，第二个参数是鼠标指标离开时执行的函数.

$(‘#switcher h3’).hover(function(){…}, function(){…}) ;

* 获得事件目标

//事件对象event是DOM结构，他会在元素获得处理事件的机会时传递给被调用的事件处理程序。这个对象中包含着与事件有关的信息。

//event.target属性保存着发生事件的目标元素.通过.target可以确定DOM中首先接收到事件的元素（即实际被单击的元素）

$(‘#switcher’).click(function(event){

if (event.target == this) { //仅实际被单击元素处理，其子元素不处理

$(‘#switcher button’).toggleClass(‘hidden’) ;

}

}) ;

* **停止事件传播 event.stopPropagation() ;**
* **阻止默认操作(默认操作如单击链接，浏览器自动加载新页面；编程表单后按回车，触发表单的submit事件。) event.preventDefault() ;**
* 事件委托(避免给多个子元素注册处理程序，内存影响，可以利用事件冒泡在其父元素注册)
* is() and .hasClass()

$(event.target).is(‘button’)测试被单击的元素是否包含button标签

集合.hasClass()

* 取消注册事件处理程序

$(‘#switcher’).off(‘click’) ;

* 模仿用户操作

$(‘#switcher’).trigger(‘click’) ; 或 $(‘#switcher’).click() ;

* 响应键盘事件

如果想知道用户按了哪个键，应该侦听keyup or keydown事件，如果想知道用户输入的是什么字符，应该侦听keypress事件.

//检查按键字符

$(document).keyup(function(event){

var key = String.fromCharCode(event.which) ;

… ;

}

## 修改样式与实现动画

* **HTML应用CSS的首选方式：先在单独的样式表中为类定义好样式，然后再通过jQuery来添加或者移除这些类。**

**.addClass(‘selected’) ;**

**.removeClass(‘hover’) ;**

* 另一种方法：.css(), 获取样式属性值，进行计算，然后设置属性值

Var num = parseFloat($(‘div.speech’).css(‘fontSize’)) ;

num \*= 1.4 ;

$(‘div.speech’).css(‘fontSize’, num+’px’) ;

* 动画特性

//.hide()将匹配的元素集合的内联style属性设置为display:none

//.show()将匹配的元素集全的display属性，恢复为应用display:none之前的可见属性(比如<li>元素具有display:list-item属性为了构建水平的导航菜单，可能会被修改为display:inline)

$(‘p’).eq(1).hide();

//.hide(‘duration’)方法，会同时减少元素的高度，宽度和不透明度，直到这3个属性的值为0，同时该元素应用CSS规则display:none.而.show(‘duration’)方法则会从上到下增大元素的高度，从左到右增长元素的宽度，同时从0到1增加元素的不透明度，直至其内容完全可见。

.show(‘slow’)会在600ms内完成效果。.show(‘fast’)则在200ms内完成效果。.show(850)则在850ms完成效果

$(‘p’).eq(1).show(‘slow’) ;

//.fadeIn(‘slow’) and .fadeOut(‘slow’)只改变元素的不透明度

$(‘p’).eq(1).fadeIn(‘slow’) ;

//.slideDown() and .slideUp()只改变元素的高度

$(‘p’).eq(1).slideDown(‘slow’) ;

//为检测和修改元素中包含的文本

$(this).text(‘read more’) ;

// 创建自定义动画

.animate(), 包含样式属性及值的对象，时长，缓动类型，回调函数

$firstPara.animate({Opacity: ‘toggle’,Height: ‘toggle’}, ‘slow’) ;

以上是并发运行动画

若排除运行动画

$firstPara.animate({Opacity: ‘toggle’}, ‘slow’)

.animate({Height:’toggle’}, ‘slow’)

## 动态修改DOM

操作HTML属性：（HTML属性是指页面标记中放在引号中的值）

.addClass() ;

.removeClass() ;

.toggleClass() ;

操作非类属性：

.attr({

property: value,

id: function(index, oldValue) {

return ‘wikilink-‘ + index ;

},

title: function() { //this是正在操作的DOM元素

return ‘Learn more about ‘ + $(this).text() + ‘ at Wikipedia.’ ;

}

}) ;

.removeAttr() ;

操作DOM元素属性：(DOM属性是指通过Javascrip能够存取的值)

某些DOM属性，例如nodeName,nodeType,selectedIndex,childNodes，在HTML中没有对应的属性，因此无法通过.attr()方法操作它们。在JQuery中，通过.prop()方法查询和修改DOM属性。

$(‘.my-checkbox’).prop(‘checked’, false) ; //支持回调函数如false是由函数计算出来

HTML属性与DOM属性差别最大的地方：表单控件。如文本输入框的value HTML属性，对应DOM属性是defaultValue, 选项列表的select HTML属性对应DOM属性是selectedIndex. 为避免差异，建议使用jQuery提供的.val()

//取得文本输入框的当前值

var inputValue = $(‘#my-input’).val() ;

//取得选项列表的当前值

var selectValue = $(‘#my-select’).val() ;

//设置单选列表的值

$(‘#my-single-select’).val(‘value3’) ;

//设置多选列表的值

$(‘#my-multi-select’).val([‘value1’, ‘value2’]) ;

//创建新元素

var $link = $(‘<a href=”#top”>back to top</a>’) ;

$link.insertAfter(‘div.chapter p’) ; //插入新元素

//元素中插入新元素，使用：

.append(), .appendTo(), .prepend(), .prependTo() ;

$(‘<p>Hello</p>’).appendTo(‘#container’) ; 等价于

$(‘#container’).append(‘<p>Hello</p>’) ;

//元素相邻的位置上插入新元素。

.after(), .insertAfter(), .before(), .insertBefore() ;

$(‘span.footnote’).insertBefore(‘#footer’) ; //移动元素，元素在DOM只有一份，插入到另一地方相当于移动的，因为只有一份，不是复制

//元素外部插入新元素

.wrap(), .wrapAll(), .wrapInner() ;

$(‘span.footnote’).insertBefore(‘#footer’)

.wrapAll(‘<ol></ol>’) //将所有脚注都包含在一个<ol>中

.wrap(‘<li></li>’) ; //将每个脚注分别包装在自己的<li>中

//复制元素

$(‘div.chapter p:eq(0)’).clone().insertBefore(‘div.chapter’) ;

在默认情况下，.clone()不会复制元素及其后代元素中绑定的事件。若需要事件一起复制，选.clone(true) ;

//要用新元素或文本替换元素

.html() ; 元素的内容

.text(); 元素的纯文本内容

.replaceAll(), .replaceWith() ;

//移除元素中的元素

.empty() ;

//从文档中移除元素及其后代过犹不及 ，但不实际删除它们

.remove(), .detach() ;

## 通过Ajax与服务端交互数据

### 加载html文件 (片段)

插入到<div id=”dictionary”></div>内部，当新html片段插入时，相应的css规则也会立即应用到它的标签上

$(‘#dictionary’).load(‘fragment.html’) ;

//仅加载html文档页面的部分文档

$(‘#dictionary’).load(‘whole.html .entry’) ;

//加载json文件,返回javascript对象

$.getJSON(‘data.json’, function(data){

var html = ‘’ ;

//若data是数组

$.each(data, function(index, item) {

html += ‘<div class=”entry”>’ ;

…

Html += ‘</div>’ ;

}) ;

$(‘#dictionary’).html(html) ;

}) ;

### 加载js文件，自动执行js文件

**$.getScript(‘c.js’) ;**

c.js文件中：

var entries = 数据模型；

var html = 由数据模型生成的视图；

$(‘#dictionary’).html(html) ; //js 插入片段

//加载xml文档，回调函数的数据参数是xml dom树，可以用jQuery DOM处理

$.get(‘d.xml’, function(data){

$(data) ; //data是dom, $(data)是jQuery对象

$(data).attr(‘term’) ; //获取属性值

$(data).find(‘definition’).text() ; //获取文本

}) ;

### 从第三方服务器加载 （跨域加载）

Javascript的安全模型会限制数据文件必须与网页保存在相同的服务器上，这样就可以保证数据的可靠性。可以跨域加载js文件，然后注入数据，一般采用JSONP技术（json with padding），JSONP的格式是把标准JSON文件包装在()中，()前置任意字符串，这个字符串即所谓的P,由请求数据的客户端来决定，因此返回的数据在客户端导致函数调用

服务端实现JSONP：

<?php

Print($\_GET[‘callback’] . ’(‘ . $data . ‘)‘ ) ；

?>

客户端实现JSONP, 这里?是填充

$.getJSON(url+’?callback=?’, function(data){

…

}) ;

提交数据给服务器

//get 请求

var requestData = {

name: “qzlin”

} ;

$.get(‘e.php’, requestData, function(data) {

$(‘#dictionary’).html(data) ; //若返回html片段

}) ;

// post 请求

$(‘#letter-f form’).submit(function(event){

event.preventDefault() ;

var formValues = $(this).serialize() ; //表单各字段落 -> 查询字符串对象

$.get(‘f.php’, formValues, function(data){

$(‘#dictionary’).html(data) ; //若返回html片段

}

}) ;

// 统一底层方法

//可以用来设置一些session信息

$.ajaxSetup({

url: ‘a.html’ ,

type: ‘POST’,

datatype: ‘html’

}) ;

var $loading = $(‘<div id=”loading”>Loading…</div>’).insertBefore(‘#dictionary’) ;

$(document).ajaxStart(function(){

$loading.show() ;

}).ajaxStop(function(){

$loading.hide() ;

}) ;

$.ajax({

type: ‘GET’,

success: function(data){

$(‘#dictionary’).hide().html(data, function(){

$(this).fadeIn() ; //加载html版本时，回调函数实现动画效果

}) ;

fail: function(jqXHR){}

}

}) ;

# Angularjs

## 理解回合式和单页面应用程序

回合式缺点：用户在一份HTML文档被请求并加载之前必须等待，它需要大型的服务器端基础设施来处理所有请求并管理所有的应用程序状态，需要许多带宽，因为每个HTML文档必须是自包含的（导致服务器产生的每个响应中包含许多相同内容）

单页面：一个初始的HTML文档被发送给浏览器，但是用户交互所产生的**Ajax请求只会请求较小的HTML片段，或者要插入到已有的显示给用户元素中的数据。**

对于较简单的项目，一般来说jQuery就可以，通过显式操作浏览器中的DOM来创建应用程序。

**AngularJS通过增加新元素、属性、CSS类和特殊注释的方法来扩展HTML**，AngularJS库动态编译一个文档中的HTML，以定位和处理这些附加品，并创建应用程序。

AngularJS的HTML文档每次被加载时，AngularJS会编译HTML元素，计算数据绑定，执行指令等等

## AngularJS工具函数

(通过**全局对象angular**,当html文档添加angular.js并加载时，angular对象就自动创建)

angular.isFunction, angular.isObject, angular.isDefined, angular.isUndefined, angular.isArray, angular.isNubmer,

var productCopy = product ? angular.copy(product) : {} ; //复制对象

angular.extend(myExtendedObject, myData) ; //复制对象myData至myExtendedObject中

angular.forEach(myData, function(value, key){}) ; //遍历对象的属性和方法

delete myData.name; //删除对象的属性和方法

var hasName = “name” in myData ; //检查对象是否拥有属性和方法

number(str), parseInt(str), parseFloat(str) ; //数值 <- 字符串

arr[idx]

arr.join(separator) ; //字符串 <- 数组

arr.slice(start, end) ; // 返回数组的一部分

arr.spice(index, count) ; //从数组中删除count个成员，从index指定的位置开始

arr.sort() ; //排序数组中的成员

$filter(“limitTo”)(data.splice(start\_index), size) ; //返回指定长度的数组，

//承诺对象，表明某项工作以异步方式执行并在未来某个点被完成,成功的话先调用success,然后调用then;失败的话先调用error,然后调用then.

//当通过Ajax请求JSON数据时，响应会被自动解析成JavaScript对象并传给回调函数。

var promise = $http.get(“todo.json”) ;

promise.success(function (data) {

$scope.todos = data ;

}).error(function(){

$scope.todos = [{action: “Error”}] ;

}).then(function(){

$scope.todos.push({action: “Request Complete”});

}) ;

## 模块

在AngularJS中有两个角色。第一是它使用ng-app指令定义应用于HTML元素中的应用程序的功能。第二是**使用模块来定义功能，比如服务、指令和过滤**，使之在不同的应用程序中某种程度上易于重用

若directive.js引用已定义模块todoApp，则directive.js应放在todoApp后面导入

angular.module(“todoApp”).directive(“cartSummary”, function(){}) ;

若模块依赖于其他模块，通过依赖注入方式，则无所谓模块文件顺序。（因为AngularJS在处理依赖前载入其所有模块）

//创建模块todoApp,依赖于其他模块列表

//获取已创建的模块：angular.module(“todoApp”) ;

**//AngularJS会加载定义在程序中的所有模块并解析依赖，将每个模块所包含的构件进行合并，从而无缝地使用来自其他模块的功能成为可能。**比如todoApp模块里的控制器依赖注入Services .cart模块里的cart服务，

var todoApp = angular.module(“todoApp”, [“Filters. customFilters”, “Services .cart”, “ngRoute”]);

// ng-app属性设置模块作用域，指定html元素包含一个被AngularJS编译和处理的模块

<html ng-app=”todoApp”>

控制器，过滤，服务的定义：方法(方法名，工厂函数)，工厂函数可以注入需要的组件

## 模型

处理存储或读取数据的逻辑，分视图模型和领域模型，视图模型只表示从控制器传往视图的功能。领域模型包含了业务领域的数据，以及用于创建、存储和操纵这些数据的各种操作，转换和规则，统称为模型逻辑。

var model = {

items: [{action:”Buy Flowers”, done: false},

{action:”Collect Tickets”, done: false}]

} ;

## 控制器Module.controller

对用户交互做出响应，更新模型中的数据并向视图提供所需要的数据。每个控制器都具有自己的关于整个应用程序的作用域的一部分

控制器依赖于$scope组件来执行工作， AngularJS会检查指定给函数的参数，并查找相应的组件。即所谓的依赖注入$scope,依赖注入简化了在组件之间处理依赖的过程（被称为是解决依赖），没有依赖注入，就不得不以某种方式自己查找$scope，很可能得使用全局变量。**AngularJS应用程序中的一个组件通过在工厂函数的参数上声明依赖，声明的名称要与所依赖的组件相匹配**。依赖注入改变了函数参数的用途，没有依赖注入，参数会被用接收调用者想传入的任何对象，但是**有了依赖注入后，函数使用参数来提出需求，告诉AngularJS它需要什么样的构件**。

在开发中使用依赖注入的主要好处是：AngularJS负责管理组件并在需要时提供给相应函数。依赖注入还能够为测试带来好处，因为它允许你能够使用假的或者模拟的对象来代替真实构件，从而让你专注于程序的特定部分。

todoApp.controller(“ToDoCtrl”, function($scope){

//$scope向对应的视图暴露数据和功能

$scope.todo = model ;

}) ;

ng-controller指定控制器所负责的视图,**AngularJS在文档中对HTML进行编译，发现ng-controller属性后，调用ToDoCtrl函数设置将被用于创建视图的作用域，当遇到每个数据绑定表达式后，AngularJS会查找$scope对象上的具体值，并向HTML文档中插入该值。**

### 控制器继承

作用域实际上是以层级结构的形式组织起来，**顶层是根作用域(root scope)，每个控制器都会被赋予一个新的作用域，该作用域是根作用域的一个子作用域**。子控制器productListCtrl的作用域继承父控制器ToDoCtrl的作用域（数据和行为），以及根作用域$rootScope的数据和行为

当读取一个直接的在作用域上定义的属性值时，AngularJS会检查该控制器作用域上是否有一个局部属性，如果没有，就会沿着作用域层次结构向上查找是否有一个被继承的属性。然而当使用ng-model指令来修改这个属性时，AngularJS会检查当前作用域是否有该名称属性，如果没有，就会假设你想隐式定义这个属性，结果便覆盖该属性值。若在作用域上定义一个对象，然后在对象上定义数据属性，就不会被覆盖，而是共享。结论：如果你想数据值在开始时被共享但在修改时会被复制一份，就直接在作用域上定义数据属性。如果想始终只有一份数据值，就通过一个对象来定义数据属性。一般情况，AngularJS开发中的习惯是使用接收参数的行为。每个主要视图创建新的控制器

$scope.dataValue = “hello” ; //数据刚开始被继承作用域共享，子作用域修改后独立

$scope.data = { //数据被继承作用域共享

dataValue: “hello” ;

}

<body ng-controller=”ToDoCtrl”>

<div ng-controller=productListCtrl>

### 控制器通信

使用服务和根作域$rootScope

//$scope or $rootScope的$broadcast(name, args)/$emit(name,args): 对所有子使用域/父作用域直至根作用域发送事件，（事件名称 和数据对象）；

$on(name, handler): 注册事件处理器

todoApp.service(“Mediator”, function($roopScope){

return {

setZipCode: function(type, zip) {

this[type] = zip ;

$rootScope.$boradcast(“zipCodeUpdated”, {type: type, zipCode: zip}) ;

}

}

}) ;

todoApp.controller(“serverCtrl”, function($scope, Mediator){

$scope.setAddress = function(type, zip) {

Mediator.setZipCode(type, zip) ;

}

}) ;

todoApp.controller(“clientCtrl”, function($scope, Mediator){

$scope.on(“zipCodeUpdated”, function(event, args){

//todo: args={type: type, zipCode: zip}

}) ;

}) ;

## AngularJS与其他框架集成

<div id=”angularRegion” ng-controller=”simpleCtrl”>

<label><input type=”checkbox” ng-model”buttonEnabled”>Enable jQuery button</label>

<div id=”jqui”>

<button>handle AngularJS content</button>

//$scope.$watch对外集成，$scope.buttonEnabled的变化触发另一个框架的动作

$scope.$watch(‘buttonEnabled’, function(newValue){

$(‘#jqui button’).button({disabled: !newValue});

}) ;

//$scope.apply对内集成，其化框架调用AngularJS的$scope.handleClick()

$(document).ready(function(){

$(‘#jqui button’).button().click(function(e){

angular.element(anguarRegion).scope().$apply(‘handleClick()’) ;

}

}) ;

若其他包也使用{{}}，可以配置内插.正规内联绑定是被AngularJS使用$interpolate服务处理，由于服务是单例的，任何配置的改变都将覆盖整个模块

angular.module(“todoApp”,[])

.config(function($interpolateProivder){

$interpolateProvider.startSymbol(“!!”) ;

$interpolateProvider.endSymbol(“!!”) ;

}) ;

## 视图

处理将数据格式化并是显示给用户的逻辑

过滤器函数unique在依赖模块customFilters中,AngularJS会在解析依赖之前，载入所有模块并解析依赖。即AngularJS的依赖注入。

过滤器通过”|”符号使用，过滤器函数orderBy表明按对象的哪个属性排序，过滤函数range实现过滤页面数据，参数seletedPageIdx和pageSize是变量（$scope.selectedPageIdx, $scope.pageSize）

<tr ng-repeat=”item in todo.items” | ordyBy: ‘category’ | range: selectePageIdx: pageSize>

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

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

</tr>

## 过滤器Module.filter

格式化，排序和过滤展现给用户的数据，一旦定义过滤器后，就可在整个模块中全面应用，也就意味着可以用来保证跨多个控制器和视图之间的数据展示的一致性。**过滤器在数据从作用域传递到了指令上时进行转换，但并不改变源数据。**

**使用过滤器，声明方式用|,代码方式用$filter**

### 内置过滤器

#### 过滤单个数据

currency/date/number: 货币值/日期/数值格式化 见<AngularJS高级程序设计>

本地化过滤器

<script src=”angular-local\_fr-fr.js”></script>

#### 过滤集合

limitTo: 从数组中取出来限定的数量

filter: 过滤数组，过滤条件可以为条件表达式，或用于匹配属性值的map对象；如p in products | filter:{category: ‘Fish’}” 找到category为Fish的子集

orderBy: 排序， 基于属性名，如orderBy: ‘-price’ 降序， 默认是升序；或基于属性名组依次排序

自定义过滤器

待过滤器值data和参数pageIdx, size

angular.modular(“Filters. customFilters”, [])

.filter(“range”, function($filter){

return function(data, pageIdx, size) {

if (angular.isArray(data) && angular.isNumber(pageIdx) && angular.isNumber(size)) {

var startIndex = (pageIdx-1) \* size ;

return $filter(“limitTo”)(data.splice(startIndex), size) ;

} else {

return data ;

}

} ;

}) ;

声明使用方式：

range: selectedPageIdx:pageSize

代码使用：

var rangeFilter = $filter(“range”) ;

var filteredData = rangeFilter(data, pageIdx, size) ;

//常量: 从而常量可以被注入到控制器中

todoApp.constant(“productListActiveClass”, “btn-primary”)

//创建局部视图,AngularJS遇到ng-include指令，会产生Ajax请求，载入src属性所指定的文件，并插入其内容以替换该元素

<ng-include src=”productList.html”></ng-include>

## 服务

**服务通常用于实现横切关注点（类似Spring AOP）**，比如日志，安全和联网；

AngularJS通过服务提供功能，**服务是贯穿整个应用程序的单例对象.**

### 内置服务

$scope，$filter，$http,

$route: 为基于浏览器URL路径的视图内容改变提供支持

$routeParams: 提供关于URL路由的信息

$exceptionHandler: 处理应用程序中出现的异常

**AngularJS使用$exceptionHandler服务处理任何在应用程序执行时出现的异常。默认实现是调用$log服务定义的error方法，其中调用了全局的console.error方法。一般情况下，它只是简单地写入异常的细节到Javascript控制台，且允许应用程序继续运行**

$sanitize: 将危险的HTML字符替换为与之相等的安全显示符

$window: 提供DOM window对象的引用 （Angularjs不增强或改变由全局对象提供的API）

$timeout: 提供围绕window.setTimeout函数的增强封装

$interval: window.setInterval函数的增强封装

$document: 提供jqLite对象，包括DOM window.document对象

$location: 浏览器location对象的封装，提供访问当前URL的入口

$location服务操作第一个#后面的URL部分，这意味着它可以用于当前文档的导航，而不导航到新文件中，通常你不会想要将用户导航出主文档，因为这会卸载你的web应用程序并抛弃你的数据和状态。

[http://mydomain.com/app.html#/cities/london?select=hotels#north](http://mydomain.com/app.html#/cities/london?select=hotels)

/cities/london：路径(path)， select=hotels: 搜索项(search term)，north:散列(hash)

$location.absUrl(), path()/path(target), hash()/hash(target), host(), port(), protocol(), search()/search(term,params), url()/url(target)

$location服务定义两个事件，当URL改变时或由于用户交互或编程方式改变，你可以使用它们接收通知。使用作用域中$on方法注册事件并传入事件对象，新URL以及旧URL

$scope.$on(“$locationChangeSuccess”, function(event, newUrl, oldUrl) {

$scope.url = newUrl ;

}) ;

$scope.deleteProduct = function(productId){

$location.path(“/cities/London”) ;

}

HTML5的History API提供了更优雅的方式改变URL,而不导致文档重载，所有主流浏览器的最新版本都支持History API，而且它的支持可以在AngularJS应用程序中通过$location服务的提供器，$locationProvider启用

angular.model(“todoApp”).config(function($locationProvider){

If (window.history && history.pushState) {

$locationProvider.html5Mode(true) ;

}

}) ;

$log: 全局console对象的封装

**$resource: 提供对RESTful API动作的支持**

$rootElement: DOM根元素的入口

$rootScope: 提供顶级作用域的入口

$anchorScroll: 滚动浏览器窗口至指定的的锚点(即$anchorScorll服务滚动浏览器窗口到显示id与$location.hash方法返回值一致的元素处。)

$animate: 使转换内容动画化

$swipe: 识别单击手势

### 自定义服务

#### 方法1: Module.factory方法

传入服务名称cart,和工厂函数。工厂函数将在AngularJS需要服务时被调用，**返回服务对象**，从一个服务对象被用于整个应用程序开始，工厂只被调用一次

angular.modular(“Services.cart”, [])

.factory(“cart”, function(){

var cartData = [] ;

return {

addProduct: function(id, name, price) {},

removeProduct: function(id) {}

getProducts: function(){ return cartData ; }

}

}) ;

#### 方法2: Module.service方法

* 1. 基于Jvascript原型特性创建service

//构造函数，功能模板

var baseLgger = function(){

this.messageCount = 0 ;

this.log = function(msg){

console.log(this.msgType + ”:” + (this.messageCount++) + “ “ + msg) ;

}

} ;

//new 关键字创建新对象并将属性和由构造函数所定义的函数复制到新对象中。Prototype属性用于改变模板

var debugLogger = function(){} ;

debugLogger.prototype = new baseLogger() ;

debugLogger.prototype.msgType = “Debug” ;

var errorLogger = function(){} ;

errorLogger.prototype = new baseLogger() ;

errorLogger.prototype.msgType = “Error” ;

//构造器传给service方法，AngularJS会主动调用new方法创建服务对象

angular.module(“todoApp”)

.service(“logService”, debugLogger)

.service(“errorService”, errorLogger);

2.2 直接作为服务对象：AngularJS调用工厂函数时，会**分配一个this对象**

todoApp.service(“days”, function(){

this.today = new Date().getDay() ;

this.tomorrow = this.today + 1 ;

}) ;

//使用服务days， AngularJS使用依赖注入来查找days服务并将其作为参数传给工厂函数

todoApp.controller(“tomorrowCtrl”, function($scope, days){

$scope.day = days.tomorrow ;

}) ;

//Module.value方法，创建值或对象的服务

todoApp.value(“nowValue”, new Date()) ;

todoApp.service(“days”, function(nowValue){

this.today = nowValue.getDay() ；

this.tomorrow = this.today + 1 ;

}) ;

#### 方法3: Module.provider，配置服务对象，然后返回服务对象

todoApp.provider(“logService”, function(){

var counter = true ;

var debug = true ;

return {

//功能函数，用于配置服务对象，一般在模块载入时设置

messageCounterEnabled: function(setting){

if (angular.isDefined(setting)){

counter = setting ;

return this ;

} else {

return this ;

}

},

debugEnabled: function(setting){

if (angular.isDefined(setting)) {

debug = setting ;

return this ;

} else {

return debug ;

}

},

//返回服务对象

$get: function() {

return {

messageCount = 0 ,

log: function(msg) {

if (debug) {

console.log(this.messageCount++ + msg) ;

}

}

};

}

}

});

//配置提供器logServiceProvider，注意，服务对象是单例的，一旦你对已启动的应用程序做出任何改变，所有正在使用该服务的组件都将受到影响

todoApp.config(function(logServiceProvider){

logServiceProvider.debugEnabled(true).messageCounterEnabled(false) ;

}) ;

//使用提供器返回的服务对象logService

todoApp.controller(“defaultCtrl”, function($scope, logService){

$scope.data = {totalClick: 0} ;

$scope.$watch(‘data.totalClicks’, function(newVal){

logService.log(“Total click count: “ + newVal) ;

}) ;

}) ;

## 指令

### 内置指令(包括数据绑定，表单验证，模板生成，事件处理和HTML元素操作)

#### 基本指令

**｛｛｝｝：单向绑定, 使用模型中的值，其将其插入到HTML文档中, AngularJS的绑定是动态的，也就是说当绑定所关联的值在数据模型中变化时，HTML元素也会被随之更新，显示新的值**。在请求绑定到一个不存在的模型属性时，AngularJS也不会报错，因为它假定这个属性将会在之后创建。所以它根本不会插入任何内容

ng-non-bindable: 阻止内联数据绑定。在混用JavaScript工具包并想在HTML的某部分上使用一些其化模板系统时（或者只是想在文本中使用双括号时）。解决方案是使用ng-non-bindable指令，可以阻止AngularJS处理内联绑定

<div ng-non-bindable>

…

</div>

**ng-model: 双向数据绑定**。从两个方向同时跟踪变化，允许元素从用户处收集数据以修改程序的状态。**仅能应用于那些允许用户输入数据值的元素上，也就是input, textarea and select元素**

ng-repeat: 生成重复内容， 对数组的每个元素或对象的每个属性重复生成一个元素及其内容。内置变量$index, $first, $last, $middle, $even, $odd

<tr ng-repeat=”item in todos” ng-class=”$odd ? ‘odd’ : ‘even’”>

<td>{{$index + 1}}</td>

<td ng-repeat=”(prop, propValue) in item”>

{{prop}}={{propValue}}

</td>

<td><span ng-if=”$first || $last”>{{item.complete}}</span></td>

</tr>

ng-include: 加载一个局部视图。当AngularJS遇到ng-include指令就会自动发出一个Ajax从服务器获取一段HTML片段，编译并处理其中包含的任何指令，并添加到DOM中去.局部视图也能访问对应的控制器的数据和行为. src可以是一个函数，只要返回局部视图

<ng-include src=”table.html” onload=”onLoadTable” autoscroll></ng-include>

ng-switch: 有条件地显示元素

<div ng-switch on=”data.mode” ng-cloak>

<div ng-switch-when=”Table”>

Table view…

<div ng-switch-when=”List”>

List view…

<div ng-switch-default>

Default view…

ng-cloak: 暂时隐藏含AngularJS的html, 等AngularJS处理完才显示。能够在AngularJS结束对内容的处理之前先将其隐藏。ng-cloak指令使用CCS对被应用到的元素进行隐藏，当内容被处理过后AngularJS库移除CSS样式，以保证用户永远不会看见模板表达的{{}}符号

元素指令

ng-if: 从DOM中添加和移除元素

ng-show, ng-hide: 从DOM中显示和隐藏元素

ng-class, ng-class-even, ng-class-odd: 对某个元素(或ng-repeat生成的偶数元素或ng-repeat生成的奇数元素)设置class属性

<td>

<span ng-hide=”item.complete”>(Incomplete)</span>

<span ng-show=”item.complete”>(Done)</span>

</td>

<td>

<span ng-if=”!item.complete”>(Incomplete)</span>

<span ng-if=”item.complete”>(Done)</span>

</td>

<tr np-repeat=”item in todos” ng-class-even=” productListActiveClass”>

#### 事件指令

ng-click/ng-dbclick: 用户点击/双击鼠标或光标时触发

ng-change: 表单元素的内容状态发生变化的时被触发（如复选框选中，输入框文本修改等）

ng-submit: 表单提交时触发

ng-blur/ng-focus, ng-copy, ng-cut, ng-paste, ng-keydown, ng-keypress, ng-keyup, ng-mousedown, ng-mouseenter, ng-mouseleave, ng-mousemove, ng-mouseup

<tr ng-repeat=”item in todos”

ng-mouseenter=”handleEvent($event)”

ng-mouseleave=handleEvent($event)” >

布尔属性指令

ng-checked: input元素

ng-diable: input and button

ng-open: details元素

ng-readonly: input元素

ng-selected: option

#### 表单

单向绑定，只读关联

双向数据绑定，作用域和绑定的数据之间双向更新。用于表单元素ng-model

**表单form会自动启用验证，novalidate属性禁用任何浏览器可能会执行的验证，name属性定义变量，用来报告表单有效性**

一旦form元素和验证属性被设好，AngularJS就会开始验证用户所提供的数据，**AngularJS会自动给form元素生成要么ng-valid和要么ng-invalid类。若有错误，$error对象会自动产生**，shippingForm.$invalid属性将设为true

<style>

.ng-invalid {background-color: …}

.ng-valid {background-color:…}

</style>

<form name=”shippingForm” novalidate>

<div class=”well”>

<h3>Ship to </h3>

<div class=”form-group”>

<label>Name</label>

<input name=”name” class=”form-control” ng-model=”data.shipping.name” required />

<span class=”error” ng-show=”shippingForm.name.$error.required>Please enter a name</span>

</div>

</div>

</form>

通过使用双向数据绑定来隐式地在数据模型中创建属性，即当你在使用表单元素收集用户输入数据以便在数据模型中创建一个新的对象或属性

<div class=”form-group row”>

<label for=”actionText”>Action:</label>

<input id=”actionText” class=”form-control” ng-model=”newTodo.action”>

</div>

<div class=”form-group row”>

<label for=”actionLocation”>Location:</label>

<select id=”actionLocation” class=”form-control” ng-model=”newTodo.location”>

<option>Home</option>

<option>Office</option>

</select>

</div>

<button class=”btn btn-primary btn-block” ng-click=”addNewItem(newItem)”>Add</button>

//当页面被浏览器第一次加载时，这个newItem对象及其action和location属性并不存在，当input or select元素改变时，AngularJS将自动创建newItem对象。**在取用不存在的对象或属性时并不会报错，而且当赋值给还不存在的对象或属性时，AngularJS将会简单创建一个出来，即制造出一个所谓的隐式定义属性或对象**

$scope.addNewItem = function(newItem) {

$scope.todos.push({action: newItem.action+newItem.location,

Complete: false}) ;

}

##### 使用表单指令属性

**当input type=”text”, “url”, “email”, “number” 或textarea时：**

**ng-model: 双向绑定**

**ng-change: 元素内容改变事件**

**ng-minlength/ng-maxlength: 元素最小/最大字符数**

**ng-pattern: 元素内容正则表达式**

**ng-required: required**

<input name=”sample” class=”form-control” ng-model=”inputValue” ng-required=”requireValue” ng-minlength=”3” ng-maxlength=”10” ng-pattern=”matchPattern”>

<textarea name=”sampleTextarea” cols=”40” rows=”3” ng-model=”textValue” ng-required=”requireValue” ng-minlength=”3” ng-maxlength=”10” ng-pattern=”matchPattern”

</textarea>

**当type属性为email, url or number,AngularJS会自动会设置ng-pattern属性为相应的正则表达式，并检查格式是否匹配。**

**当input type=”checkbox”时：ng-model, ng-change**

ng-true-value/ng-false-value: checkbox勾选/取消时绑定值，只有当复选框的勾选状态被改变时生效。也就是说模型属性不会被自动创建，直至有用户与元素的交互产生时才会被创建。

<input name=”sample” type=”checkbox” ng-model=”inputValue” ng-true-value=”A llo” ng-false-value=”Hello world”>

当select时：item.id为selectValue的值，item.action为显示标签, item.place为分组项

<select ng-model=”selectValue” ng-options=”item.id as item.action group by item.place for item in todos”>

<option value=””>(Pick One)</option> //默认的option元素

</select>

##### 执行表单验证

双向模型绑定，form元素不是必需，**form元素只有在表单校验时需要用到**。

要禁用浏览器所支持的校验并启用AngularJS校验功能，需要在自己的表单元素上增添novalidate属性，告诉浏览器不要自己校验表单

**当用户在与表单元素发生交互时，AngularJS在幕后就已在执行有效性的检查**，我们可以使用这些检查所提供的信息来给用户实时提供有意义的反馈信息，**AngularJS为报告实时校验信息所提供的两种机制：CSS类和变量。**

校验CCS类：

AngularJS通过在一个类的集合中增添或移除被校验的元素，来报告有效性检查的结果。

ng-pristine/ng-dirty: 用户与元素/表单交互否对应的类

ng-valid/ng-invalid: 校验结果有效否对应的类

校验变量：

**为各个想要验证的元素添加name属性是非常重要的，这样就可以访问到AngularJS所提供的校验变量，**如下：

$pristine/$dirty: 用户与元素/表单产生交互否？

$valid/$invalid: 元素/表单内容的校验结果是否有效？

$error: 校验错误的详情信息

<form **name=”myForm” novalidate ng-submit=”addUser(newUser)”**>

<div class=”form-group”>

<label>Email:</label>

<input **name=”userEmail” type=”email”** class=”form-control” ng-model=”newUser.email” required >

<div class=”error” **ng-show=”myForm.userEmail.$invalid && myForm.userEmail.$dirty”>**

**{{getError(myForm.userEmail.$error)}}**

</div>

</div>

**<button type=”submit”** class=”btn btn-primary btn-clock” **ng-disabled=”myForm.$invalid”**>

OK</button>

</form>

$scope.getError = function(error) {

if (error.required) return “Please enter a value” ;

else if (error.email) return “Please enter a valid email address” ;

} ;

### 自定义指令: 扩展标准HTML词汇

Module.directive: 指令名称unordered-list和工厂函数（返回链接函数），**当AngularJS建立指令的每个实例时，链接函数便被调用并接收三个参数：指令被应用到的视图的作用域scope，指令被应用到的HTML元素element, 以及HTML元素的属性attrs**

以链接函数里，要避免产生硬连接的依赖关系，比如数据获取，数据的schema等，只能通过html元素中的指令或属性获取

AngularJS内含jqLite（如element,用jqLite操作返回的对象都是jqLite）,即裁剪版本的jQuery, 最好**只处理传给链接函数的元素的子元素和后代元素**，因为对其化元素进行操作会引起混淆并且冒着干涉其化指令的操作的风险

导航DOM方法:

jqLite.children(): 返回一组子元素

eq(index): 返回元素集合中指定索引下的元素（jqLite对象），如果使用Javascript数组索引则返回HTMLElement对象，是浏览器用于表示DOM中的元素的对象

find(tag): 按照tag名称定位所有的后代元素

next(): 获取下一个兄弟元素

parent(): 返回父元素

创建和移除元素方法：

angular.element(html): 创建一个代表特定HTML字符串的元素的jqLite对象

after(elements): 在调用方法的元素后面插入特定内容

append(elements): 在调用方法的jqLite对象的每一个元素上，将特定元素作为最后一个子元素插入

clone(): 从方法调用的对象复制元素并作为一个新的jqLite对象返回

prepend(elements): 在调用方法的jqLite对象的每一个元素上，将特定元素作为第一个子元素插入

remove(): 从DOM中删除jqLite对象的元素

replaceWith(elements): 用指定元素替换调用方法的jqLite对象的元素

wrap(elements): 使用特定元素包装jqLite对象中的每个元素

修改元素方法：

jqLite.addClass(name), attr(name)/attr(name, value), css(name)/css(name, value), hasClass(name), prop(name)/prop(name, value), removeAttr(name), removeClass(name), text()/text(value), toggleClass(name), val()/val(value)

事件：

on(events, handler): 为jqLite对象所代表的元素发生的事件注册一个处理器

off(events, handler): 为jqLite对象所代表的元素发生的事件移除一个之前已注册的处理器

triggerHandler(event): 对jqLite对象所代表的元素上注册的的指定事件触发所有处理器

#### 自定义简单指令

##### 方法1: 通过jqLite代码html （难以阅读和维护）

todoApp.directive(“unorderedList”, function(){

return function(scope, element, attrs) {

//从attrs中属性unordered-list获取数据

var data =scope[attrs[“unorderedList”]] ;

//从attrs中属性list-property获取属性表达式

var propertyExpression = attrs[“listProperty”] ;

if (angular.isArray(data)) {

var listElem = angular.element(“<ul>”) ;

element.append(listElem) ;

for (var i = 0 ; i < data.length ; ++ i){

//Javascript支持闭包，允许函数引用其作用域之外的变量，没有闭包，就是确保为你的函数要访问的每个对象和值定义参数。容易混淆之外在于：函数所访问的变量是在函数被调用时进行计算的，而不是函数被定义时，所以采用IIFE使定义时就执行，从而使函数内访问到正确的作用域之外的变量。

(function(){

var itemElement = angular.element(‘<li>’) ;

listElem.append(itemElement) ;

var index = i ;

//监听器函数基于作用域watchScope，每次作用域发生变化调用

var watcherFn = function(watchScope) {

//属性值propertyExpression可能是个表达式”price | currency”

return watchScope.$eval(propertyExpression, data[index]) ;

}

//监听器函数watcherFn返回值变化，调用回调函数

scope.$watch(watcherFn, function(newValue, oldValue){

itemElement.text(newValue)；

}) ;

}()) ;

}

}

}

});

**AngularJS在遇到混合大小写的名称时有点特殊，html元素中unordered-list等价于javascript的unorderedList,即传给方法的参数中每个大写字母被认为是属性名中的一个独立的词，而每个词之间是以一个连字符分隔的。特例:如果属性名是以data-前缀，AngularJS会在生成传给链接函数的属性集合时移除这一前缀。即当属性名被规范化并传给链接函数时，属性data-list-property and list-property都会被表示为listProperty**

<div unordered-list=”products” list-property=”name”></div>

$scope.products = [{name: “Apples”, category: “Fruit”}] ;

##### 方法2: 通过字符串模板生成html

todoApp.directive(“unorderedList”, function(){

return {

link: function (scope, element, attrs) {

scope.data = scope[attrs[“unorderedList”]] ;

},

restrict: “A”,

template: “<ul><li ng-repeat=’item in data’>” +

“{{item.price | currentcy}}</li></ul>”

}

}) ;

##### 方法3: 通过ng-template

<script type=”text/template” id=”listTemplate.html”>

<ul>

<li ng-repeat=”item in data”>{{item.price | currency}}</li>

</ul>

</script>

todoApp.directive(“unorderedList”, function(){

return {

link: function (scope, element, attrs) {

scope.data = scope[attrs[“unorderedList”]] ;

},

restrict: “A”,

template: ‘listTemplate.html’

}

}) ;

##### 方法4: 通过外部模板 （Ajax加载）

itemTemplate.html

<ul>

<li ng-repeat=”item in data”>{{item.price | currency}}</li>

</ul>

todoApp.directive(“unorderedList”, function(){

return {

link: function (scope, element, attrs) {

scope.data = scope[attrs[“unorderedList”]] ;

},

restrict: “A”,

templateUrl: ‘listTemplate.html’

}

}) ;

#### 自定义复杂指令

restrict指定指令如何应用，若为E表示指令可被作为元素，若为A表示指令可被作为属性, templateUrl指定将被插入指令的元素内容的局部视图，controller指定向局部视图提供数据和行为的控制器

todoApp.directive(“cartSummary”, function(cart){

return {

restrict: “E”,

templateUrl: “components/cart/cartSummary.html”,

controller: function($scope) {

var cartData = cart.getProducts() ;

$scope.total = function(){} ;

}

} ;

}) ;

<cart-summary />

自定义事件指令：

jqLite的on方法为touchstart and touchend事件注册一个处理器函数。链接函数接收三个参数：指令被应用视图作用域scope,指令被应用到的HTML元素elem, 以及elem的属性attrs.

todoApp.directive(“tap”, function(){

return function(scope, elem, attrs) {

elem.on(“touchstart touchend”, function(){

scope.$apply(attrs[“tap”]) ;

}) ;

}

}) ;

## URL路由

### ngRoute (内置模块)

**Module.config在当前模块被加载后调用，Module.run在所有模块被加载后以及解析完他们的依赖后才会被调用**

导航：路由策略**$routeProvider.when/otherwise**提供URL与视图文件名称之间的映射。

显示来自激活路由的视图：**ng-view**告诉AngularJS应该将选择的视图插入到哪里。

变更激活的路由：使用**$location.path**或使用**href**属性匹配路由路径的元素

通过路径传入信息：使用**$routeParams**服务接收参数

将被激活路由显示的视图与控制器关联： 使用配置属性controller

使用URL路由的主要好处是组件可以改变ng-view指令所显示的布局，而不需要预先了解将被显示的视图的任何信息，ng-view的定位和部署，或向将被显示的视图分享的组件

路由匹配有两种：保守和贪婪的。保守的路由匹配一个，而贪婪的路由会匹配尽可能多的

比如：

$routeProvider.when(“/edit/:id”,{templateUrl: “editView.html”}); //只匹配类似/edit/1234

$routeProvider.when(“/edit/:id/:data\*”, {}) ; //匹配任何至少三个片段，第一个片段是edit,第二个片段将赋值给路由参数id,剩下的将赋值给路由参数data

<a href=”/edit/{{item.id}}” class=”btn btn-xs btn-primary”>Edit</a>

//监听路由变化，并获取url和路由参数

$scope.$on(“$routeChangeSuccess”, function(){

if ($location.path().indexOf(“/edit/”) == 0) {

var id = $routeParams[“id”] ;

}

}) ;

”#/edit/:id”导航到/edit/:id路径,**导航的改变会由AngularJS的路由服务所监测到**，从而会让ng-view指令显示views/editorView.html视图。

路由配置属性：

templateUrl: 路由匹配的视图文件url，可以是字符串或返回字符串的函数

（template: 视图内容，可以是字面量html字符串或返回html的函数; redirectTo: 当路由匹配时浏览器应重定向到的路径）

controller: 视图关联的控制器 (controllerAs:指定控制器别名；resolve: 控制器的依赖,一般用于初始化视图所必须执行的工作)

**控制器editCtrl将在每次editorView.html视图显示时创建**，这意味着不需要使用$route服务事件监听路由改变（因为视图创建即路由改变），可以只关注控制器函数是否执行

.controller(“editCtrl”, function($scope, $routeParams, $location){

if ($location.path().indexOf(“/edit/”) == 0) {

var id = $routeParams[“id”] ;

…

}

}) ;

todoApp.config(function($routeProvider){

$routeProvider.when(“/edit/:id”, {

templateUrl: “views/editorView.html”,

controller: “editCtrl”

}) ;

$routeProvider.otherwise({

templateUrl: “views/tableView.html”,

controller: “defaultCtrl”

}) ;

}) ;

<ng-view />

<a href=”/edit/{{item.id}}” class=”btn btn-xs btn-primary”>Edit</a>

// rest

// withCredentials=true, 会启用跨域请求(cross-origin requests)的支持，允许Ajax请求使用cookie处理验证

todoApp.constant(“authUrl”, <http://localhost:5500/users/login>)

todoApp.controller(“authCtrl”, function($scope, $http, $location, authUrl){

$scope.authenticate = function(user, pass) {

$http.post(authUrl, {username: user, password: pass}, {withCredentials: true}).

success(function(data){

$location.path(“/main”) ;

}).error(function(error){

$scope.authenticationError = error ;

}) ;

}

}

### UI-Router (参考google: angular-ui模块)

>bower install -- save angular-ui-router

State: chang application view based on application state, including nested states (for **nested views) and multiple named views**

**ui-sref:** create a link, point to a certain state of your application.

The controller will not be instantiated if template is not defined.

Parent.Child state:

* dot syntax to infer your hierarchy to the $stateProvider
* parent state must exist.
* No two states can have the same name.
* **When a state is “active”, all of its ancestor states are implicitly active as well**
* **Child state will load their templates into their parent’s ui-view** (When a state is activated,its templates are automatically inserted into the ui-view of its parent state’s template)

Abstract state can have child states but can not get activated itself, abstract state still need their own <ui-view/> for their children to plug into

shopping carts, signup forms, onboarding processes, and more have used the multi-step form to ease users through their online forms

**index.html**

<div ui-view></div>

**app.js**

angular.module('formApp', ['ngAnimate', **'ui.router'**])

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

**$stateProvider**

// route to show our basic form (/form)

**.state('form', {**

**url: '/form',**

**templateUrl: 'form.html',**

**controller: 'formController'**

})

// nested states

// each of these sections will have their own view

// url will be nested (/form/profile)

.state('form.profile', {

url: '/profile',

templateUrl: 'form-profile.html'

})

// url will be /form/interests

.state('form.interests', {

url: '/interests',

templateUrl: 'form-interests.html'

})

// catch all route

// send users to the form page

$urlRouterProvider.otherwise('/form/profile');

})

.controller('formController', function($scope) {

$scope.formData = {}; // we will store all of our form data in this object

$scope.processForm = function(formData) { // function to process the form

fun(formData);

};

});

**form.html**

<div id="status-buttons" class="text-center">

<a **ui-sref-active="active" ui-sref=".profile"**><span>1</span> Profile</a>

<a **ui-sref-active="active" ui-sref=".interests"**><span>2</span> Interests</a>

</div>

<form id="signup-form" ng-submit="processForm()">

<!-- our nested state views will be injected here -->

<div id="form-views" **ui-view**></div>

</form>

**form-profile.html**

<div class="form-group">

<label for="name">Name</label>

<input type="text" class="form-control" name="name" ng-model="formData.name">

</div>

<div class="form-group row">

<div class="col-xs-6 col-xs-offset-3">

<a **ui-sref="form.interests"** class="btn btn-block btn-info">

Next Section <span class="glyphicon glyphicon-circle-arrow-right"></span>

</a>

</div>

</div>

**form.interests.html**

Multiple Named Views:

multiple ui-views per template

relative names: **viewname**

absolute names: **viewname@statename**

index.html

<div ui-view=”tabledata”></div>

<div ui-view=”graph”></div>

spp.js

$stateProvider

.state(‘report’, {

views: {

‘tabledata’: {

templateUrl: ‘report-table.html’,

controller: function($scope) {}

}

‘graph’: {

templateUrl: ‘report-graph.html’,

controller: function($scope) {}

}

}

})

## 处理危险数据

常见的攻击：将恶意内容通过表单注入到应用程序，因此它将回显给攻击者或提交给其他用户。AngularJS有一些不错的内置服务降低这类攻击风险。

### $sce（strict contextual escaping SCE，严格上下文转义）

**将HTML字符串中的危险字符替换为与之对应的转义字符**，转义内容的过程不会影响scope中原有的值，只有被绑定显示的数据会影响。这意味着你可以继续暗中处理HTML数据并允许AngularJS安全地在浏览器中呈现它。**该特性是默认起作用**。

.controller(“defaultCtrl”, function($scope){

$scope.htmlData = “<p>This is <b onmouseover=alert(‘Attack!’)>dangerous</b> data</p>” ;

}) ;

<p>{{htmlData}}</p>

结果：<p>This is <b onmouseover=alert(‘Attack!’)>dangerous</b> data</p>

或者明确信任的数据：不需要转义或净化

.controller(“defaultCtrl”, function($scope, $sce){

$scope.htmlData = “<p>This is <b onmouseover=alert(‘Attack!’)>dangerous</b> data</p>” ;

$scope.$watch(“htmlData”, function(newValue){

$scope.trustedData = **$sce.trustAsHtml**(newValue) ;

}) ;

}) ;

<p ng-bind-html=”htmlData”></p>

结果：This is dangerous data （点击dangerous，显示alert框）

### $sanitize

从HTML中移除危险元素和属性,预防不安全的值通过数据绑定被展现出来.

angular.module(“todoApp”, [“ngSanitize”])

.controller(“defaultCtrl”, function($scope){

$scope.htmlData = “<p>This is <b onmouseover=alert(‘Attack!’)>dangerous</b> data</p>” ;

}) ;

<p **ng-bind-html=”htmlData”>**</p>

结果：This is dangerous data

或者采用立即净化：

angular.module(“todoApp”, [“ngSanitize”])

.controller(“defaultCtrl”, function($scope){

$scope.htmlData = “<p>This is <b onmouseover=alert(‘Attack!’)>dangerous</b> data</p>” ;

$scope.$watch(“dangerousData”, function(newValue){

$scope.htmlData = $sanitize(newValue) ;

}) ;

}) ;

<p ng-bind=”htmlData”></p>

结果：This is dangerous data

## Ajax

### $http请求

var promises = $http.get(url, config)/delete(url, config)/head(url,config)/ post(url, config)/put(url, config):

url是相对于被请求的主html文档的，不需要硬编码协议，主机名和端口号

GET请求是可寻址的------所有信息都被包含在了URL中，所以它适合放入书签，链接这些地址。

$http.jsonp(url, config):跨域执行GET请求,获取JavaScript代码片段然后执行该代码

$http请求的配置对象config含如下属性：

**data**: 设置发送到服务器的数据，如果你设置了该对象，AngularJS会将它序列化为JSON格式

**headers**: 常用于设置请求头部

**method**: http方法

**params**: url属性

**timeout**: 指定请求过期前的毫秒数

**transformRequest**: 转换请求

**transformResponse**: 转换响应

**url**: 设置url

**withCredentials**: 若为true,请求中验证cookies

var config = {

transformResponse: function(data, headers) {

if (headers(“content-type”) == “application/xml” && angular.isString(data)) {

products = [] ;

var productElems = angular.element(data.trim()).find(“product”) ;

for (var i = 0; i < productElems.length ; ++ i) {

var product = productElems.eq(i) ;

products.push({

name: product.attr(“name”),

category: product.attr(“category”) ;

price: product.attr(“price”) ;

}) ;

}

return products ;

} else {

return data ;

}

},

headers: { “content-type”: “application/xml”},

transformRequest: function(data, headers){

var rootElem = angular.element(“<xml>”) ;

for (var i = 0 ; i < data.length ; ++ i) {

var prodElem = angular.element(“<product>”) ;

prodElem.attr(“name”, data[i].name) ;

prodElem.attr(“category”, data[i].category) ;

prodElem.attr(“price”, data[i].price) ;

rootElem.append(prodElem) ;

}

rootElem.children().wrap(“<products>”) ;

return rootElem.html() ;

}

} ;

$http.get(“productData.xml”, config).success(function(data){

$scope.products = data ;

}) ;

$http.post(“ajax.html”, $scope.products, config) ;

通过$http服务的提供器$httpProvider为Ajax请求定义默认设置：

defaults.headers.common: 定义用于所有请求的默认头部

defaults.headers.post: 定义用于post请求的头部

defaults.headers.put: 定义用于put请求的头部

defaults.transformResquest: 应用于所有请求的转换函数数组

defaults.transformResponse: 应用于所有响应的转换函数数组

angular.module(“todoApp”,[])

.config(function($httpProvider){

$httpProvider.defaults.transformResponse.push(function(data,headers){}) ;

});

$http请求返回的承诺对象所定义的方法：

**promises.success(fn)/error(fn)/then(fn,fn): 当通过Ajax请求JSON数据时，响应会被自动解析成JavaScript对象并传给回调函数。**

承诺：事件的特殊类型，当指定的事件在未来发生时，它们允许组件被通知。但不同于事件：承诺表示一个活动的单一实例，一旦它们被解决或拒绝，承诺无法再次使用(比如：若3个button: Heads,Tails,Abort. 2个button Heads,Tails连上resolve, 那么当其中任一button点击发送事件，promise接收处理，之后再点击Heads,Tails无效)。而事件可以重复发出信号

.directive(“promiseWorker”, function($q){

**var deferred = $q.defer() ;**

return {

link: function(scope, element, attrs){

element.find(“button”).on(“click”, function(event){

var buttonText = event.target.innerText ;

//发布事件

if (buttonText == “Abort”) **deferred.reject(“Aborted”);**

else **deferred.resolve(buttonText);**

}) ;

},

controller: function($scope, $element, $attrs){

this.promise = **deferred.promise ;**

}

}

});

//promise.then(successFn, errorFn, notifyFn)

successFn响应deferred.resolve()

errorFn响应deferred.reject()

notifyFn响应deferred.notifyFn()

.directive(“promiseObserver”, function(){

return {

require: “^promiseWorker”,

link: function(scope, element, attrs, **ctrl**){

**ctrl.promise.then**(function(result){element.text(result);},

function(reason){element.text(“Fail (“ + reason + “)”);} );

}

} ;

}) ;

分组承诺：延迟活动直至其他几个结果可用时

.directive(“promiseWorker”, function($q){

**var deferred = [$q.defer(), $q.defer()];**

**var promises = [$deferred[0].promise, $deferred[1].promise] ;**

return {

link: function(scope, element, attrs){

element.find(“button”).on(“click”, function(event){

var buttonText = event.target.innerText ;

var buttonGroup = event.target.getAttribute(“data-group”) ;

//发布事件

if (buttonText == “Abort”) **deferred[buttonGroup].reject**(“Aborted”);

else **deferred[buttonGroup].resolve**(buttonText);

}) ;

},

controller: function($scope, $element, $attrs){

**this.promise = $q.all(promises).then(function(results){ return results.join() ; }) ;** ;

}

}

});

假定服务端RESTful endpoint:

GET /products

POST /products

GET /products/<id>

PUT /products/<id>

DELETE /products/<id>

.constant(“baseUrl”, <http://localhost:5500/products/>)

CRUD操作，注意后台与前台的同步

**$http.get(baseUrl).success**(function(data){ $scope.products = data ; }) ; //获取$scope.products

**$http.post(baseUrl, product).success**(function(newProduct){

$scope.products.push(newProduct) ; //添加newProduct于$scope.products里

}) ;

**$http({url: baseUrl+product.id, method:”PUT”, data:product}).success** (function(modifiedProduct){

//将$scope.products里的product更新为modifiedProduct

}) ;

**$http(url: baseUrl+product.id, method:”DELETE”).success**(function(){

$scope.products.splice($scope.products.indexOf(product), 1) ; //删除product

}) ;

### ngResource

$http存在的问题是同步本地数据和后台数据，需要手动处理。可以**采用$resource,确保任何本地数据的变化都将产生Ajax请求（通过把Ajax请求和URL格式的细节隐藏，使它更易于在应用程序中与RESTful数据打交道），即自动同步本地和后台**

注意：product是$scope.productsResource.query()获取到的集合的元素 (除了new). 存取对象$scope.productsResource提供了在服务器上查询和修改数据的手段，但不会自动执行任何动作本身。从query()返回的结果是最初为空的数组集合。$resource服务创建结果数组然后使用$http服务产生Ajax请求。当Ajax请求完成时，从服务器获得的数据就放到了集合中。Angular会使用数组绑定，自动更新界面

$scope.productsResource = **$resource(baseUrl+”:id”, {id: “@id”} )** ;

$scope.products = $scope.productsResource.**query()** ; // get products

$scope.productById = $scope.productsResource.**get(id)**; // get product or

$scope.productById.$get() ; //重新从服务器下载指定product

**new $scope.productsResource(product).$save().then**(function(newProduct){

$scope.products.push(newProduct) ; // post product

}) ;

**product.$save();** // update product

**product.$delete().then**(function(product){

$scope.products.splice($scope.products.indexOf(product), 1) ; // delete product

}) ;

## 动画和触摸服务：

当元素在DOM中添加，移除或移动时，$animate服务提供转变效果

## 单元测试

单元测试的一个重要方面是隔离一小段代码，并且测试它的行为而无需测试它所依赖的组件（本质上是创建焦点测试）

## angular-bootstrap: native angularJS directives for bootstrap

>bower install -- save angular-bootstrap (必须先安装bootstrap.css, 但不需要bootstrap.js)

angular.module("myModule', ['ui.bootstrap'])

具体参考：<https://github.com/angular-ui/bootstrap>

### *Accordion (ui.bootstrap.accordion):*

<**uib-accordion close-others="true"**>

<**uib-accordion-group heading=**"{{group.title}}" ng-repeat="group in groups">

**{{group.content}}**

</uib-accordion-group>

<uib-accordion-group heading="Custom template" template-url="group-template.html">

Hello

</uib-accordion-group>

</uib-accordion>

### *Alert (ui.bootstrap.alert):*

<**uib-alert** ng-repeat="alert in alerts" type="{{alert.type}}" **close="closeAlert($index)"**>{{alert.msg}}</uib-alert>

<uib-alert template-url="alert.html">A happy alert!</uib-alert>

### *Buttons (ui.bootstrap.buttons):*

//单按钮，通过ng-model绑定，按键与状态1<->0转换

<button type="button" class="btn btn-primary" **ng-model="singleModel" uib-btn-checkbox btn-checkbox-true="1" btn-checkbox-false="0"**> Single Toggle

</button>

//复选框，通过checkModel={left: false, middle:flase, right:false}.按键与状态true<->false转换

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

<label class="btn btn-primary" **ng-model="checkModel.left" uib-btn-checkbox**> Left </label>

<label class="btn btn-primary" **ng-model="checkModel.middle" uib-btn-checkbox**>Middle</label>

<label class="btn btn-primary" ng-model="checkModel.right" uib-btn-checkbox>Right </label>

</div>

// 单选按钮，radioModel值为点击其中一个button的值

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

<label class="btn btn-primary" **ng-model="radioModel" uib-btn-radio="'Left'"**>Left</label>

<label class="btn btn-primary" **ng-model="radioModel" uib-btn-radio="'Middle'"**>Middle

</label>

<label class="btn btn-primary" ng-model="radioModel" uib-btn-radio="'Right'">Right

</label>

</div>

### *Carousel (ui.bootstrap.carousel):*

**<uib-carousel** interval="myInterval" no-wrap="noWrapSlides">

**<uib-slide** ng-repeat="slide in slides" active="slide.active">

**<img** ng-src="{{slide.image}}" style="margin:auto;">

**<div class="carousel-caption">**

<h4>Slide {{$index}}</h4>

<p>{{slide.text}}</p>

</div>

</uib-slide>

</uib-carousel>

$scope.myInterval = 5000;

$scope.noWrapSlides = false;

$scope.slides = [{

image: '//placekitten.com/pics/300',

text: "cat"

}];

### *Collapse (ui.bootstrap.collapse): (hide/show element with css transition)*

<button type="button" class="btn btn-default" ng-click="isCollapsed = !isCollapsed">Toggle collapse </button>

<div **uib-collapse="isCollapsed"**>

<div class="well well-lg">Some content</div>

</div>

$scope.isCollapsed = false;

### Datepicker(ui.bootstrap.datepicker): 有inline and popup两种形式

Inline直接显示

<div style="display:inline-block; min-height:290px;">

**<uib-datepicker ng-model="dt"** min-date="minDate" show-weeks="true" class="well well-sm" custom-class="getDayClass(date, mode)"></uib-datepicker>

</div>

Popup弹出框显示

<p class="input-group">

<input type="text" class="form-control" **uib-datepicker-popup="{{format}}" ng-model="dt" is-open="status.opened"** min-date="minDate" max-date="maxDate" datepicker-options="dateOptions" date-disabled="disabled(date, mode)" ng-required="true" close-text="Close" />

<span class="input-group-btn"> <button type="button" class="btn btn-default" ng-click="open($event)"><i class="glyphicon glyphicon-calendar"></i></button>

</span>

</p>

$scope.open = function($event) {

$scope.status.opened = true;

};

### Dropdown (ui.bootstrap.dropdown):

<div class="btn-group" **uib-dropdown**>

<button id="button-template-url" type="button" class="btn btn-primary" **uib-dropdown-toggle ng-disabled="disabled"**>Dropdown using template <span class="caret"></span></button>

<ul class="uib-dropdown-menu" **role=”menu” aria-labelledby=**"button-template-url">

<li role="menuitem"><a href="#">Action in Template</a></li>

<li role="menuitem"><a href="#">Another action in Template</a></li>

<li class="divider"></li>

<li role="menuitem"><a href="#">Separated link in Template</a></li>

</ul>

</div>

### Modal (ui.bootstrap.modal):

主视图：

<div ng-controller="ModalDemoCtrl">

<button type="button" class="btn btn-default" ng-click="open()">Open me!</button>

</div>

ModalDemoCtrl（主控制器）：

controller('ModalDemoCtrl', function ($scope, $uibModal, $log) {

$scope.items = ['item1', 'item2', 'item3'];

$scope.animationsEnabled = true;

$scope.open = function (size) {

var modalInstance = $uibModal.open({

animation: $scope.animationsEnabled,

templateUrl: '**myModalContent.html**',

controller: '**ModalInstanceCtrl**',

size: size,

**resolve: { items**: function () { return $scope.items; }}

});

**modalInstance.result.then**(function (selectedItem) { $scope.selected = selectedItem;},

function () { $log.info('Modal dismissed at: ' + new Date()); });

};

$scope.toggleAnimation = function () {

$scope.animationsEnabled = !$scope.animationsEnabled;

};

});

myModalContent.html (弹出框视图)

<div class="modal-header"><h3 class="modal-title">I'm a modal!</h3></div>

<div class="modal-body">

<ul><li ng-repeat="item in items">

<a href="#" ng-click="$event.preventDefault(); selected.item = item">{{ item }}</a>

</li></ul>

Selected: <b>{{ selected.item }}</b>

</div>

<div class="modal-footer">

<button class="btn btn-primary" type="button" ng-click="ok()">OK</button>

<button class="btn btn-warning" type="button" ng-click="cancel()">Cancel</button>

</div>

ModalInstanceCtrl(弹出框控制器)

controller('ModalInstanceCtrl', function ($scope, **$uibModalInstance, items**) {

$scope.items = items;

$scope.selected = { item: $scope.items[0]};

**$scope.ok** = function () { $uibModalInstance.close($scope.selected.item);};

**$scope.cancel** = function () { $uibModalInstance.dismiss('cancel'); };

});

### Pagination (ui.bootstrap.pagination)

//项总数: total-items, 当前页：ng-model

//每页项数：items-per-page(默认为10), 导航页数：max-size (默认显示所有页)，总的页数：num-pages(只读)

**<uib-pagination boundary-links="true"** **total-items="**totalItems" **ng-model="**currentPage" **ng-change=**"pageChanged()" **num-pages="**numPages" **max-size="**maxSize" ></uib-pagination>

<uib-pager total-items="totalItems" ng-model="currentPage"></uib-pager>

$scope.totalItems = 64;

$scope.currentPage = 4;

$scope.pageChanged = function() {

$log.log('Page changed to: ' + $scope.currentPage);

};

### Popover (ui.bootstrap.popover)

//显示位置：popover-placement(=top/left/right/bottom), 控制显示和关闭：popover-trigger(=mouseenter/focus/click, mouseleave/blur…和tooltip一致)

//显示内容用模板uib-popover-template

<button **uib-popover=**"{{dynamicPopover.content}}" **popover-title=**"{{dynamicPopover.title}}" **popover-placement=**"top" **popover-trigger=**"mouseenter" type="button" class="btn btn-default">Dynamic Popover</button>

<button uib-popover-template="dynamicPopover.templateUrl" popover-title="{{dynamicPopover.title}}" type="button" class="btn btn-default">Popover With Template</button>

$scope.dynamicPopover = {

content: 'Hello, World!',

templateUrl: 'myPopoverTemplate.html',

title: 'Title'

};

### Tooltip (ui.bootstrap.tooltip): 同popover

<a href="#" **tooltip-placement="left" uib-tooltip="On the Left!"**>left</a>

<a href="#" **uib-tooltip-template="'**myTooltipTemplate.html'">Custom template</a>

myTooltipTemplate.html

<span>Special Tooltip with <strong>markup</strong> and {{ dynamicTooltipText }}</span>

### Progressbar (ui.bootstrap.progressbar)

//可以设置bar背景和内容，可以显示多个bar

<**uib-progressbar** **max="max" value="dynamic"**><span style="color:white; white-space:nowrap;">{{dynamic}} / {{max}}</span></uib-progressbar>

**<uib-progress><uib-bar** ng-repeat="bar in stacked track by $index" value="bar.value" type="{{bar.type}}"><span ng-hide="bar.value < 5">{{bar.value}}%</span></uib-bar></uib-progress>

$scope.max = 200;

$scope.dynamic = Math.floor((Math.random() \* 100) + 1);

$scope.randomStacked = function() {

$scope.stacked = [];

var types = ['success', 'info', 'warning', 'danger'];

for (var i = 0, n = Math.floor((Math.random() \* 4) + 1); i < n; i++) {

var index = Math.floor((Math.random() \* 4));

$scope.stacked.push({

value: Math.floor((Math.random() \* 30) + 1),

type: types[index]

});

}

};

$scope.randomStacked();

### Rating (ui.bootstrap.rating)

**<uib-rating ng-model="rate" max="max"** readonly="isReadonly" **on-hover="hoveringOver(value)"** on-leave="overStar = null" titles="['one','two','three']" aria-labelledby="default-rating"></uib-rating>

$scope.rate = 7;

$scope.max = 10;

$scope.isReadonly = false;

$scope.hoveringOver = function(value) {

$scope.overStar = value;

$scope.percent = 100 \* (value / $scope.max);

};

### Tabs (ui.bootstrap.tabs)

<uib-tabset>

<uib-tab ng-repeat="tab in tabs" heading="{{tab.title}}" active="tab.active" disable="tab.disabled">

{{tab.content}}

</uib-tab>

<uib-tab >

<uib-tab-heading>

<i class="glyphicon glyphicon-bell"></i> Alert!

</uib-tab-heading>

I've got an HTML heading, and a select callback. Pretty cool!

</uib-tab>

</uib-tabset>

<uib-tabset vertical="true" justified="true" type="pills">

<uib-tab heading="Vertical 1">Vertical content 1</uib-tab>

<uib-tab heading="Vertical 2">Vertical content 2</uib-tab>

</uib-tabset>

$scope.tabs = [

{ title:'Dynamic Title 1', content:'Dynamic content 1' },

{ title:'Dynamic Title 2', content:'Dynamic content 2', disabled: true }

];

### Timepicker (ui.bootstrap.timepicker)

**<uib-timepicker ng-model="mytime" ng-change="changed()" hour-step="hstep" minute-step="mstep" show-meridian="ismeridian"></uib-timepicker>**

$scope.mytime = new Date();

$scope.hstep = 1;

$scope.mstep = 15;

$scope.ismeridian = true;

$scope.changed = function () {

$log.log('Time changed to: ' + $scope.mytime);

};

### Typeahead (ui.bootstrap.typeahead): create typeaheads with any form text input

//第一个是内存查找, $viewValue是输入框的值

//第二个是异步RESTful获取

//第三个查找到的列表元素，可以定制显示

<input type="text" ng-model="selected" **uib-typeahead="**state for state in states | **filter:$viewValue |** limitTo:8" class="form-control">

<input type="text" ng-model="asyncSelected" placeholder="Locations loaded via $http" **uib-typeahead=**"address for address in **getLocation($viewValue)**" **typeahead-loading="**loadingLocations" typeahead-no-results="noResults" class="form-control">

<input type="text" ng-model="customSelected" placeholder="Custom template" uib-typeahead="state as state.name for state in statesWithFlags | filter:{name:$viewValue}" **typeahead-template-url="customTemplate.html"** class="form-control">

customTemplate.html

<a>

<img ng-src="http://upload.wikimedia.org/wikipedia/commons/thumb/{{match.model.flag}}" width="16">

<span ng-bind-html="match.label | uibTypeaheadHighlight:query"></span>

</a>

$scope.states = ['Alabama', 'Alaska',...];

$scope.getLocation = function(val) {

return $http.get('//maps.googleapis.com/maps/api/geocode/json', {

params: {address: val, sensor: false }

}).then(function(response){

return response.data.results.map(function(item){

return item.formatted\_address;

});

});

};

Highlight menu item for class=”nav navbar-nav” of bootstrap?

//导航

<ul class="nav navbar-nav" **bs-active-link**>

<li><a href="#/">Home</a></li>

<li><a ng-href="#/about">About</a></li>

</ul>

//自定义指令

angular.module('bootstrapExApp')

.directive('bsActiveLink', ['$location', function ($location) {

return {

restrict: 'A', replace: false,

link: function (scope, elem) {

scope.$on("$routeChangeSuccess", function () {

var hrefs = ['/#' + $location.path(), '#' + $location.path(), //html5: false

$location.path()]; //html5: true

angular.forEach(elem.find('a'), function (a) {

a = angular.element(a);

if (-1 !== hrefs.indexOf(a.attr('href'))) {

a.parent().addClass('active');

} else {

a.parent().removeClass('active');

};

});

});

}

}

}]);

## angular-ui-grid: angular表格 （含排序，过滤，表格头和单元内容可以采用定制模板，标签国际化）

>bower install – save angular-ui-grid

angular.module('app', ['ui.grid']);

.myGrid { width: 500px; height: 250px; } //table尺寸

$scope.gridOptions = { //table配置

enableSorting: true,

columnDefs: [

{ field: 'name' },

{ field: 'gender' },

{ field: 'company', enableSorting: false }

],

onRegisterApi: function( gridApi ) {

$scope.gridApi = gridApi;

},

data = [ //table矩阵

{

"firstName": "Cox",

"lastName": "Carney"...

};

<div ng-controller="MainCtrl">

<div ui-grid="{ data: myData }" class="myGrid"></div> //table声明

</div>

Table功能：

Sorting:

排序可以单列，点击列头即排序，**点击另一列头，先是关闭之前的排序，然后排序当前列**。在配置时，可以关闭（enableSorting: flase），默认是打开

**排序可以多列进行，shift + 点击多个列**，或者通过列菜单

编辑某个单元，会导致重新排序，也可以通过代码gridApi.core.notifyDataChange( uiGridConstants.dataChange.EDIT )

点击列头选择排序的顺序是：升->降->无，可以通过sortDirectionCycle配置，可以配置suppressRemoveSort使用该列排序只含升降

可以自定义排序算法sortingAlgorithm

Filtering:

整个表格或某列过滤，可以enable or disable via enableFiltering: false (默认是disable)

过滤条件: 选择行，匹配过滤项

过滤支持input框，下拉选择框，多个input框组合

i18n (本地化支持):

<div ui-i18n=”{{lang}}”>

<p ui-t=”groupPanel.description”></p>

<p>{{”groupPanel.description” | t}}</p>

<div ui-grid=”gridoptions” class=”grid”></div>

</div>

Footer: 显示表格底部，列尾（一行）和表格尾（一行）

$scope.gridOptions = {

showGridFooter: true,

showColumnFooter: true,

columnDefs: [

{ field: 'age', aggregationType: uiGridConstants.aggregationTypes.avg, aggregationHideLabel: true, width: '13%' },

{ name: 'customCellTemplate', field: 'age', width: '14%', footerCellTemplate: '<div class="ui-grid-cell-contents" style="background-color: Red;color: White">custom template</div>' },

],

map between table <-> data:

$scope.gridOptions = {

columnDefs: [

{ name:'1stFriend', field: 'friends[0]' },

{ name:'city', field: 'address.city'},

{ name:'getZip', field: 'getZip()', enableCellEdit:false}

],

data : [ {

"friends": ["friend0"],

"address": {street:"301 Dove Ave", city:"Laurel", zip:"39565"},

"getZip" : function() {return this.address.zip;}

}

]

};

Hidden Grids:

Method1: 基于确定的大小

<div ng-controller="MainCtrl">

<button type="button" class="btn btn-success" ng-click="hideGrid = !hideGrid">

{{ hideGrid && 'Show' || 'Hide' }} Grid </button>

<div class="well" ng-hide="hideGrid">

<div ui-grid="gridOptions" class="grid"></div>

</div>

</div>

**.grid { width: 500px; height: 150px; } //告诉具体的table大小**

$scope.hideGrid = true;

Method2： 基于ng-if，避免绘制table,因为此时还没有大小

<tabset>

<tab heading="No Grid"></tab>

<tab heading="Grid" **select="tabShown = !tabShown" deselect="tabShown = !tabShown"**>

<div id="grid1" ui-grid="gridOptions" **class="grid" ng-if="tabShown"**></div>

</tab>

</tabset>

.grid {

width: 100%;

}

Method3: 基于autoResize, 让table重绘自己，会产生250ms的flickering

<tabset>

<tab heading="No Grid"></tab>

<tab heading="Grid" >

<div id="grid1" ui-grid="gridOptions" **class="grid" ui-grid-auto-resize**></div>

</tab>

</tabset>

.grid {

width: 100%;

}

## Angular-nvD3: (an angularjs directive for NVD3 re-usable charting library (based on D3)

Line chart, box plot chart, pie chart, scatter chart, stacked area chart, multi bar chart, discrete bar chart, line with focus chart, parallel coordinates, candle stick chart, sun burst chart

>bower install angular-nvd3 - -save

控制器：

angular.module('myApp', ['nvd3'])

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

$scope.options = { /\* JSON data \*/ }; // Chart options

$scope.data = { /\* JSON data \*/ }; // Chart data

})

视图：

<div ng-controller="myCtrl">

<nvd3 options="options" data="data"></nvd3>

</div>

具体参考：<http://krispo.github.io/angular-nvd3/#/>