Contents

[Contents 1](#_Toc137625617)

[jQuery 3](#_Toc137625618)

[基本概念 3](#_Toc137625619)

[如何通过jQuery查询网页元素 3](#_Toc137625620)

[由JQuery处理事件 4](#_Toc137625621)

[修改样式与实现动画 7](#_Toc137625622)

[动态修改DOM 8](#_Toc137625623)

[通过Ajax与服务端交互数据 9](#_Toc137625624)

[加载html文件 (片段) 9](#_Toc137625625)

[加载js文件，自动执行js文件 10](#_Toc137625626)

[从第三方服务器加载 （跨域加载） 10](#_Toc137625627)

[Angular 11](#_Toc137625628)

[integrate bootstrap 13](#_Toc137625629)

[国际化 15](#_Toc137625630)

[doc 17](#_Toc137625631)

[Angularjs 26](#_Toc137625632)

[MVC模型，控制器和视图 27](#_Toc137625633)

[模块 29](#_Toc137625634)

[过滤器Module.filter 29](#_Toc137625635)

[服务（内置服务，自定义服务） 30](#_Toc137625636)

[指令（内置指令和自定义指令） 34](#_Toc137625637)

[URL路由（ngRoute, UI-Router） 45](#_Toc137625638)

[Ajax 49](#_Toc137625639)

[工具函数，危险数据，与其他框架集成，动画，单元测试 53](#_Toc137625640)

[angular-bootstrap: native angularJS directives for bootstrap 55](#_Toc137625641)

[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） 62](#_Toc137625642)

[Angular-nvD3: (an angularjs directive for NVD3 re-usable charting library (based on D3) 65](#_Toc137625643)

[AngularJS最佳实践 65](#_Toc137625644)

[Sencha Touch 70](#_Toc137625645)

[Extjs5.0.1 72](#_Toc137625646)

[Bootstrap 3.X 79](#_Toc137625647)

[概览 79](#_Toc137625648)

[栅格系统 80](#_Toc137625649)

[全局 CSS 样式 84](#_Toc137625650)

[组件 99](#_Toc137625651)

[JavaScript插件 118](#_Toc137625652)

[React 133](#_Toc137625653)

[Why react? 133](#_Toc137625654)

[Using props 133](#_Toc137625655)

[Reactive state 134](#_Toc137625656)

[Displaying Data: JSX 134](#_Toc137625657)

[Attribute Expressions 134](#_Toc137625658)

[Boolean Attributes 135](#_Toc137625659)

[Child Expressions 135](#_Toc137625660)

[Interactivity and Dynamic UIs 135](#_Toc137625661)

[Components are Just State Machines 135](#_Toc137625662)

[Multiple Components 135](#_Toc137625663)

[Data Flow 136](#_Toc137625664)

[Reusable Components 136](#_Toc137625665)

[Mixins 136](#_Toc137625666)

[ES6 Classes 136](#_Toc137625667)

[Stateless Functions 137](#_Toc137625668)

[Transferring Props: A Shortcut 137](#_Toc137625669)

[destructuring 138](#_Toc137625670)

[Forms 138](#_Toc137625671)

[Working With the Browser 139](#_Toc137625672)

[Component Lifecycle 139](#_Toc137625673)

[Refs to Components 140](#_Toc137625674)

[Tools 141](#_Toc137625675)

[RxJS <- Promises <- Callback 141](#_Toc137625676)

[可观察数据类型 142](#_Toc137625677)

[观察者(Observer) 143](#_Toc137625678)

[Subscription订阅 143](#_Toc137625679)

[Subject主题 144](#_Toc137625680)

[Operators操作符 146](#_Toc137625681)

[Scheduler调度者 148](#_Toc137625682)

[Redux 148](#_Toc137625683)

[主要概念 149](#_Toc137625684)

# jQuery

**Books: 《**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。

**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）

## 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’)

$(‘#selected-plays > li’) //ID为selected-plays的子元素列表项

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

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

* 属性选择符

$(‘img[alt]’) //带有alt属性的所有图像元素

//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

$(‘input[type=”radio”]:checked’) //选择所有选择中的单选按钮

* 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()

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 myTag = $(‘#my-element’)[0].tagName ;

## 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’) ;

$(#switcher h3).toggleClass(‘hidden’) ; //根据相应的类是否存在而添加或删除类

…;

}) ;

* **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:*

1. :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(){…}) ;

1. 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’) ;

}

}) ;

1. 停止事件传播 event.stopPropagation() ;
2. 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() ;
3. 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)
4. 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()

1. 取消注册事件处理程序

$(‘#switcher’).off(‘click’) ;

1. 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.

$(document).keyup(function(event){ //检查按键字符

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

… ;

}

1. 修改样式与实现动画
2. 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’) ;

1. 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’)

## 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’) ;

$(‘p’).eq(1).fadeIn(‘slow’) ; //.fadeIn(‘slow’) and .fadeOut(‘slow’)只改变元素的不透明度

$(‘p’).eq(1).slideDown(‘slow’) ; //.slideDown() and .slideUp()只改变元素的高度

$(this).text(‘read more’) ; //为检测和修改元素中包含的文本

// 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){}

}

}) ;

# Angular

* install nodejs from <https://nodejs.org/en> // install angular-cli for window

含nodejs (js服务端编程框架)以及NPM(Node包管理器, 安装升级和卸载包), NPM两种模式：全局和本地。本地模式是NPM的默认操作模式，在工作目录下管理包。全局模式用于全局加载的公共模块，比如命令行工具Angular CLI，NPM会把包安装到/usr/local/lib/node\_modules.

* download node-v8.1.0-linux-x64 & untar // install angular-cli for linux

$cd node-v-8.1.0-linux-x64/bin/node -v //测试可执行文件可以直接使用

软链接，从而放入命令行搜索路径中

ln -s /home/q/Downloads/node-v8.1.0-linux-x64/bin/node /usr/local/bin/node

ln -s /home/q/Downloads/node-v8.1.0-linux-x64/bin/npm /usr/local/bin/npm

> node -v && npm -v // Node.js JavaScript runtime and npm (the Node.js package manager)

> npm install -g @angular/cli //安装Angular CLI

> ng help

若无法下载，需要设置代理> npm config set proxy <http://161.92.64.42:8080>

>npm --registry=https://registry.npm.taobao.org //使用国内淘宝镜像

> ng new project-name // create angular project

> cd project-name

> ng serve --open // 启动开发服务器、监视文件，并在这些文件发生更改时重建应用

* **Debug by Chrome**

Chrome浏览器F12打开开发者工具

Source -> top/webpack://, 该目录与实际开发目录一致，打开文件添加断点比如

src/app/app.component.ts

刷新页面，代码运行到断点处就会停住，mouse hover会显示变量值，F10下一步，F11进入函数，也可以在右侧Watch窗口添加变量监视

* IDE for angular: Install **visual studio code**, npm, angular cli

<https://code.visualstudio.com/docs/nodejs/angular-tutorial>

The Visual Studio Code editor supports TypeScript IntelliSense and code navigation out of the box, so you can do Angular development without installing any other extension.

> cd project-name

> code . // 打开vscode

设置断点, debug

Angular 是一个基于 TypeScript 构建的开发平台。它包括：**基于组件的框架**/**库** (包括路由、表单管理、客户端-服务器通信等) **和开发工具**（开发、构建、测试和更新代码）

组件：提供用户体验

组件应该表达用于数据绑定的属性和方法，以在视图（由模板渲染）和应用逻辑（通常包括某个模型的一些概念）之间进行居中协调。

组件可以将某些任务委托给服务，例如从服务器获取数据、验证用户输入或直接把日志记录到控制台。

> ng generate component <component >

生成目录及4个文件

<component >

<component >.component.ts

<component >.component.html

<component >.component.css

<component >.component.spec.ts //测试文件

@Component({

selector: 'app-component-overview',

templateUrl: './component-overview.component.html',

styleUrls: ['./component-overview.component.css']

})

export class ComponentOverviewComponent {

}

模板

在模板中使用 Angular 的特有语法来扩展应用程序的 HTML 词汇表。比如，Angular 可以通过内置的模板函数、变量、事件监听和数据绑定等功能来帮助你动态获取和设置 DOM（文档对象模型）中的值

<h3>Current customer: {{ currentCustomer }}</h3>

<button type="button" (click)="deleteHero()">Delete hero</button> //click事件

<button type="button" (click)="onSave($event)">Save</button>

<form #heroForm (ngSubmit)="onSubmit(heroForm)"> ... </form>

指令： 改变外观

属性型指令 （更改元素、组件或其他指令的外观或行为的指令）

//NgClass：添加和删除一组 CSS 类。

<!-- toggle the "special" class on/off with a property -->

<div [**ngClass**]="isSpecial ? 'special' : ''">This div is special</div>

NgStyle：添加和删除一组 HTML 样式。

NgModel ：将双向数据绑定添加到 HTML 表单元素。

NgIf：有条件地从模板创建或销毁子视图。

NgFor：为列表中的每个条目重复渲染一个节点。

NgSwitch：一组在备用视图之间切换的指令。

依赖注入: 创建可注入的服务和注入服务

Angular 使用一种称为 Injector 的抽象来促进依赖消费者和依赖提供者之间的互动。当有人请求依赖项时，注入器会检查其注册表以查看那里是否已有可用的实例。如果没有，就会创建一个新实例并将其存储在注册表中。Angular 会在应用的引导过程中创建一个应用范围的注入器（也称为“根”注入器），并会根据需要创建任何其它注入器。

依赖提供者（创建可注入的服务）

>ng generate service heroes/hero

生成heroes/hero.service.ts

import { Injectable } from '@angular/core';

import { HEROES } from './mock-heroes';

import { Logger } from '../logger.service'; //使用Logger服务

@Injectable({ //@Injectable() 装饰器指出 Angular 可以在 DI 体系中使用此类

providedIn: 'root', //元数据 providedIn: 'root' 表示服务在整个应用程序中可见的

})

export class HeroService {

constructor(private logger: Logger) { }

getHeroes() { return HEROES; }

}

依赖使用者 （注入服务）

Heroes/hero-list.component.ts

@Component({

selector: 'hero-list',

template: '...',

providers: [HeroService]

})

class HeroListComponent {

constructor(private service: HeroService) {}

}

## doc

Angular modules vs. JavaScript modules

The Angular module — a class decorated with @NgModule — is a fundamental feature of Angular

The @NgModule decorator identifies AppModule as an Angular module class (also called an NgModule class). **@NgModule takes a metadata object that tells Angular how to compile and launch the application.**

**src/app/app.module.ts**

import { NgModule } from '@angular/core';

import { BrowserModule } from '@angular/platform-browser';

import { AppComponent } from './app.component';

**@NgModule**({

imports: [ BrowserModule ],

declarations: [ AppComponent ],

bootstrap: [ AppComponent ]

})

export class AppModule { }

* imports — the **BrowserModule** that this and every application needs to run in a browser.

Many features of Angular itself are organized as Angular modules. HTTP services are in the **HttpModule**. The router is in the **RouterModule**. Eventually you may create a feature module.

* declarations — the application's lone component, which is also ...

You must declare every component in an NgModule class, Only **declarables — components, directives and pipes** — belong in the declarations array

* bootstrap — the root component that Angular creates and inserts into the index.html host web page.

Each bootstrapped component is the base of its own tree of components. Inserting a bootstrapped component usually triggers a cascade of component creations that fill out that tree.

The bootstrapping process **sets up the execution environment**, **digs the root AppComponent** out of the module's bootstrap array, **creates an instance of the component** and **inserts it within the element tag** identified by the component's selector.

(In JavaScript each file is a module and all objects defined in the file belong to that module. The module declares some objects to be public by marking them with the export key word. Other JavaScript modules use import statements to access public objects from other modules.)

Components

A component controls a patch of screen called a view

**Angular creates, updates, and destroys components as the user moves through the application**. Your app can take action at each moment in this lifecycle through optional lifecycle hooks, like ngOnInit() declared above.

Templates

A template is a form of HTML that tells Angular how to render the component.

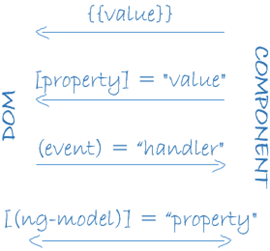
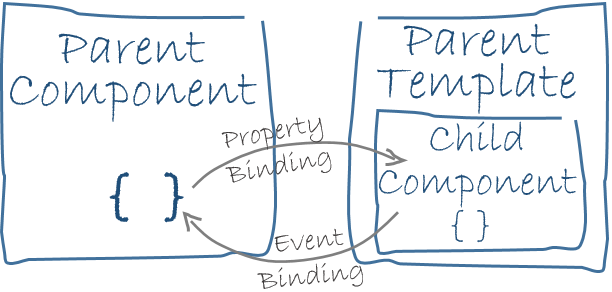
Metadata

**Metadata tells Angular how to process a class**.

@Component tell Angular that XXXComponent is a component, attach metadata to the class.

Data binding

four forms of data binding syntax. Each form has a direction — to the DOM, from the DOM, or in both directions.

<li>{{hero.name}}</li> interpolation

<hero-detail [hero]="selectedHero"></hero-detail> property binding

<li (click)="selectHero(hero)"></li> event binding

<input [(ngModel)]="hero.name"> two-way data binding

Directives

A directive is a class with a @Directive decorator. **A component is a directive-with-a-template**;

<li **\*ngFor**="let hero of heroes"></li>

<hero-detail **\*ngIf**="selectedHero"></hero-detail>

<input **[(ngModel)]**="hero.name">

Services

Service is a broad category encompassing any value, function, or feature that your application needs

A service is typically a class with a narrow, well-defined purpose. It should do something specific and do it well.

Examples include:

* logging service
* data service
* message bus
* tax calculator
* application configuration

Yet services are fundamental to any Angular application. **Components are big consumers of services.**

Component classes should be lean. They don't fetch data from the server, validate user input, or log directly to the console. They delegate such tasks to services.

A component's job is to enable the user experience and nothing more. It mediates between the view (rendered by the template) and the application logic (which often includes some notion of a model). **A good component presents properties and methods for data binding. It delegates everything nontrivial to services.**

Dependency injection

Most dependencies are services. Angular uses dependency injection to provide new components with the services they need.

**Angular can tell which services a component needs by looking at the types of its constructor parameters**

constructor(private service: HeroService) {

}

When Angular creates a component, it first asks an injector for the services that the component requires.

**An injector can create a new service instance from a provider**. Then injector maintains a container of service instances that it has previously created.



In general, add providers to the root module so that the same instance of a service is available everywhere.

src/app/app.module.ts (module providers)

providers: [

BackendService,

HeroService,

Logger

],

Alternatively, Registering at a component level means you get a new instance of the service with each new instance of that component.

src/app/hero-list.component.ts (component providers)

@Component({

selector: 'hero-list',

templateUrl: './hero-list.component.html',

providers: [ HeroService ]

})

Angular权威教程

表单

formex.component.html

<form **[formGroup]="form"** (ngSubmit)="onSubmit(form.value)">

<div class="form-group">

<label for="skuInput">SKU</label>

<input class="form-control" id="skuInput" placeholder="SKU" **[formControl]="sku" [(ngModel)]="skuValue"**>

</div>

<div *\*ngIf="sku.touched && sku.hasError('invalidSku')"* class="alert alert-danger" role="alert">SKU must begin with <span>123</span></div>

<p \*ngIf="skuValue">the input value is {{ skuValue }}</p>

<button type="submit" class="btn btn-default">Submit</button>

</form>

formex.component.ts

import { Component, OnInit } from '@angular/core';

import { FormBuilder, FormGroup, Validators, AbstractControl } from '@angular/forms';

**import { FormValidator } from './form-validator';**

@Component({

selector: 'app-form-validate',

templateUrl: './form-validate.component.html'

})

export class FormValidateComponent implements OnInit {

**form: FormGroup;**

**sku: AbstractControl;**

constructor(fb: FormBuilder) {

**this.form = fb.group({**

**'sku': ['', Validators.compose([Validators.required, FormValidator.skuValidator])]**

});

**this.sku = this.form.controls['sku'];**

this.sku.valueChanges.subscribe(

(value: string) => {

console.log('sku changed to: ', value);

}

);

}

onSubmit(value: String): void {

console.log('you submitted value: ', value);

}

}

form-validator.ts

import { FormControl } from '@angular/forms';

export class FormValidator {

**static skuValidator(control: FormControl): { [s: string]: boolean }** {

if ( !control.value.match(/^123/) ) {

return { invalidSku: true };

}

}

}

Dependency Injector

The constructor parameter type, the @Component decorator, and the parent's providers information combine to tell the Angular injector to inject an instance of HttpService whenever it creates a new HttpExComponent.

The injector relies on providers to create instances of the services that the injector injects into components and other services.You must register a service provider with the injector, or it won't know how to create the service.

You can either register a provider within an NgModule or in application components

Dependencies are singletons within the scope of an injector

范例1: 基于http的服务

https://codecraft.tv/courses/angular/http/core-http-api/

http-service.ts

import { Http, Response } from '@angular/http';

import { Injectable } from '@angular/core';

**@Injectable()**

export class HttpService {

data: object;

loading: boolean;

constructor(private http: Http) {

this.loading = true;

this.http.request('http://wthrcdn.etouch.cn/weather\_mini?city=%E4%B8%8A%E6%B5%B7')

.subscribe( (res: Response) => {

this.data = res.json();

this.loading = false;

});

}

}

http-ex.component.ts

import { Component, OnInit } from '@angular/core';

import { HttpService } from './http-service';

@Component({

selector: 'app-http-ex',

**providers: [ HttpService]**

})

export class HttpExComponent implements OnInit {

constructor(**private httpService: HttpService**) {}

}

范例二：自定义服务依赖于其他自定义服务

logger.ts

import { Injectable } from '@angular/core';

**@Injectable()**

export class Logger {…}

hero.service.ts

constructor(**private logger: Logger**, private isAuthorized: boolean) { }

hero.service.provider.ts

let heroServiceFactory = (**logger: Logger, userService: UserService**) => {

return new HeroService(logger, userService.user.isAuthorized);

};

export let heroServiceProvider =

**{ provide: HeroService,**

**useFactory: heroServiceFactory,**

**deps: [Logger, UserService]**

**}**;

heroes.component.ts

import { heroServiceProvider } from './hero.service.provider';

@Component({

**providers: [heroServiceProvider]**

})

export class HeroesComponent {

constructor(**heroService: HeroService**)

}

范例三：配置文件注入

app-config.ts

export interface AppConfig {

weatherApi: string;

}

export const HERO\_DI\_CONFIG: AppConfig = {

weatherApi: 'http://wthrcdn.etouch.cn/weather\_mini'

};

**import { InjectionToken } from '@angular/core';**

**export let APP\_CONFIG = new InjectionToken<AppConfig>('app.config');**

app.module.ts

import { APP\_CONFIG, HERO\_DI\_CONFIG } from './app-config';

providers: [

**{ provide: APP\_CONFIG, useValue: HERO\_DI\_CONFIG }**

],

http-service.ts

import { APP\_CONFIG, AppConfig } from '../app-config';

constructor(private http: Http, **@Inject(APP\_CONFIG) config: AppConfig**) {

config.weatherApi

路由

The router is a combination of multiple provided services (RouterModule), multiple directives (RouterOutlet, RouterLink, RouterLinkActive), and a configuration (Routes)

Redux

ngrx/store

npm install @ngrx/core @ngrx/store --save

counter.ts

import { ActionReducer, Action } from '@ngrx/store';

export const INCREMENT = 'INCREMENT';

export const DECREMENT = 'DECREMENT';

export const RESET = 'RESET';

export function counterReducer(state: number = 0, action: Action) {

switch (action.type) {

case INCREMENT: return state + 1;

case DECREMENT: return state - 1;

case RESET: return 0;

default: return state;

}

}

app.module.ts

import { StoreModule } from '@ngrx/store';

import { counterReducer } from './counter';

@NgModule({

imports: [ StoreModule.provideStore({ counter: counterReducer }) ]

})

component.ts

import { Store } from '@ngrx/store';

import { INCREMENT, DECREMENT, RESET } from './counter';

interface AppState {

counter: number;

}

@Component({

selector: 'my-app',

template: `

<button (click)="increment()">Increment</button>

<div>Current Count: {{ counter | async }}</div>

<button (click)="decrement()">Decrement</button>

<button (click)="reset()">Reset Counter</button>

`

})

class MyAppComponent {

counter: Observable<number>;

constructor(private store: Store<AppState>){

this.counter = store.select('counter'); //Use store.select to select slice(s) of state

}

increment(){ this.store.dispatch({ type: INCREMENT }); }

decrement(){ this.store.dispatch({ type: DECREMENT }); }

reset(){ this.store.dispatch({ type: RESET }); }

}

## integrate bootstrap

http://tutorialsdojo.com/blog/2017/03/03/intro-bootstrap-4-integrating-bootstrap-4-angular-cli-angular-tutorial/

$ npm install --save bootstrap@4.0.0-alpha.6 font-awesome[bootstrap@next](mailto:bootstrap@next) font-awesome

angular-cli.json

"styles": [

"../node\_modules/bootstrap/dist/css/bootstrap.css",

"../node\_modules/font-awesome/css/font-awesome.css",

"styles.css"

],

"scripts": [

"../node\_modules/jquery/dist/jquery.js",

"../node\_modules/tether/dist/js/tether.js",

"../node\_modules/bootstrap/dist/js/bootstrap.js"

],

可以使用Bootstrap 4

https://v4-alpha.getbootstrap.com

https://github.com/ng-bootstrap/ng-bootstrap

$npm install --save @ng-bootstrap/ng-bootstrap

Icon网站：

http://fontawesome.io/icons/

<i class="fa fa-arrow-up"></i>

$ng generate component product-list 自动更新app.module.ts的依赖

$ng generate component product-list/product-item

$ng generate class product-list/product-item/product

app.module.ts

import {NgbModule} from '@ng-bootstrap/ng-bootstrap';

@NgModule({

declarations: [AppComponent, ...],

imports: [NgbModule.forRoot(), ...],

bootstrap: [AppComponent]

})

export class AppModule {

}

app.component.html

<div class="container">

<ngb-accordion [closeOthers]="true" activeIds="static-1">

<ngb-panel id="static-1" title="Simple">

<ng-template ngbPanelContent>… </ng-template>

</ngb-panel>

<ngb-panel id="static-2">

<ng-template ngbPanelTitle>

<span>&#9733; <b>Fancy</b> title &#9733;</span>

</ng-template>

<ng-template ngbPanelContent>… </ng-template>

</ngb-panel>

</ngb-accordion>

</div>

## 国际化

https://github.com/ngx-translate/core

npm install @ngx-translate/core --save

npm install @ngx-translate/http-loader --save

1. Import the TranslateModule:

在项目的根模块app.module.ts中引入该模块

app.module.ts

**import {TranslateModule, TranslateLoader} from '@ngx-translate/core';**

**import {TranslateHttpLoader} from '@ngx-translate/http-loader';**

// AoT requires an exported function for factories

**export function createTranslateLoader(http: Http) {**

**return new TranslateHttpLoader(http, './assets/i18n/', '.json');**

**}**

@NgModule({

imports: [

**TranslateModule.forRoot({**

**loader: {**

**provide: TranslateLoader,**

**useFactory: (createTranslateLoader),**

**deps: [Http]**

**}**

**})**

]

})

1. Define the translations:

在assets文件夹下新建i18n文件夹，并且新建语言的json文件

en.json

{

"HOME": {

"HELLO": "hello {{value}}"

},

“HELLO\_HTML”: "<br><strong>latest technologies!</strong>",

"HELLOWORLD": {

"TITLE": "hello-world works! {{ value }}"

}

}

zh-CN.json

{

"HOME": {

"HELLO": "您好 {{value}}"

},

“HELLO\_HTML”: "<br><strong>最新技术!</strong>",

"HELLOWORLD": {

"TITLE": "您好世界，工作! {{ value }}"

}

}

1. Init the TranslateService for your application:

在根组件中使用Translate服务

app.component.html

<p>**{{ 'HELLO' | translate: param }}**</p>

<div class="form-group">

<label for="langSelect"> **{{ 'LANGUAGE' | translate }}**</label>

<select id="langSelect" #langSelect (change)="changeLang(langSelect.value)">

<option \*ngFor="let lang of **translate.getLangs()**" [value]="lang"

[selected]="lang === **translate.currentLang**">{{ lang }}

</option>

</select>

</div>

app.component.ts

**import {TranslateService} from '@ngx-translate/core';**

export class AppComponent {

**param = {value: 'world'};**

constructor(**private translate: TranslateService**) {

translate.addLangs(['zh-CN', 'en']); //添加语言支持

translate.setDefaultLang('en'); //默认语言

//translate.use('zh-CN');

//获取当前浏览器环境的语言比如en、 zh

let broswerLang = translate.getBrowserLang();

translate.use(broswerLang.match(/en|zh-CN/) ? broswerLang : 'zh-CN');

}

changeLang(lang) {

this.translate.use(lang);

}

}

在子组件可以直接使用Translate服务，而无需依赖注入translate

比如在app/hello-world/hello-world.component.html

<p>

{{ 'HELLOWORLD.TITLE' | translate: param }}

</p>

app/hello-world/hello-world.component.tx

export class HelloWorldComponent implements OnInit {

param = {value: 'hi'};

}

# Angularjs



**Scope**： Scope is the glue between application controller and the view

**Controllers**: Controllers are the behavior behind the DOM elements. AngularJS lets you express the behavior in a clean readable form without the usual boilerplate of updating the DOM, registering callbacks or watching model changes.

**Directives**: Let you invent new HTML syntax, specific to your application. Directives are markers on a DOM element (such as an attribute, element name, comment or CSS class) that tell AngularJS's HTML compiler ($compile) to attach a specified behavior to that DOM element or even transform the DOM element and its children Directives allow us to create reusable components. A component allows you to hide complex DOM structure, CSS, and behavior. This lets you focus either on what the application does or how the application looks separately.

**Services**:Value / Constant,Factory,Service,Provider

• Use to organize and share code across application

• Lazily instantiated – only instantiates when a component depends on it

• Singletons – Each component dependent on a service gets a reference to the single instance generated by the service factory

JSDO(use if you don’t want to spend any time on DA framework) provides support for a complex data model and API to manipulate that data while maintaining data integrity. The JSDO catalog defines the logical schema and mapping to a remote data source

## MVC模型，控制器和视图

* 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 （继承和通信）

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}

}) ;

}) ;

* 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>

## 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

## 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（内置服务，自定义服务）

**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.module(“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.module(“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（内置指令和自定义指令）

* 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，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

ng-class的用法

* ng-class="row.point >=0?'style1':'style2'" //point>0的时候执行style1，否则style2
* ng-class="{true: 'style1', false: 'style2'}[isActive]" //isactive是ture的时候执行style1，否则style2
* ng-class {'selected': isSelected, 'car': isCar}" //当 isSelected = true 则增加selected class，当isCar=true,则增加car class，

<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

参考：<https://docs.angularjs.org/guide/directive>

自定义指令：创建可复用于不同应用程序的自包含的功能

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

normalization: Angular normalizes an element's tag and attribute name to determine which elements match which directives

The normalization process is as follows:

* strip x- and data- from the front of the element/attributes.
* convert the :, -, or \_-delimited name to camelCase.

For example, the following forms are all equivalent and match the ngBind directive:

<div ng-controller="Controller">

Hello <input ng-model='name'> <hr/>

<span ng-bind="name"></span> <br/>

<span ng:bind="name"></span> <br/>

<span ng\_bind="name"></span> <br/>

<span data-ng-bind="name"></span> <br/>

<span x-ng-bind="name"></span> <br/>

</div>

生成html片段方式

方法1: 通过jqLite代码html （难以阅读和维护）

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**，因为对其他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

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)；

}) ;

}()) ;

}

}

}

});

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’

}

}) ;

通过链接函数方式(采用默认值的选项)

一般用来操纵DOM or 绑定事件监听

客户端使用

<span my-draggable>Drag Me</span>

自定义指令

.directive('myDraggable', myDraggable);

myDraggable.$inject = ['$document'];

function myDraggable($document) {

return function(scope, element, attrs, controller, transcludeFn) {

var startX = 0, startY = 0, x = 0, y = 0;

element.css({

position: 'relative',

border: '1px solid red',

backgroundColor: 'lightgrey',

cursor: 'pointer'

});

element.on('mousedown', function(event) {

// Prevent default dragging of selected content

event.preventDefault();

startX = event.pageX - x;

startY = event.pageY - y;

$document.on('mousemove', mousemove);

$document.on('mouseup', mouseup);

});

function mousemove(event) {

y = event.pageY - startY;

x = event.pageX - startX;

element.css({

top: y + 'px',

left: x + 'px'

});

}

function mouseup() {

$document.off('mousemove', mousemove);

$document.off('mouseup', mouseup);

}

};

}

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**

在链接函数里，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**

通过返回对象（定义选项属性：restrict, templateUrl, scope, controller）

客户端使用

状态

$stateProvider

.state('home', {

url: '/',

templateUrl: 'app/main/main.html',

controller: 'MainController as main'

});

视图main.html

<unordered-list **title="Hello World" products="main.products" get-detail="main.getCode (main.getCode)**"></unordered-list>

控制器

function MainController($timeout, webDevTec, toastr) {

var vm = this;

vm.products = [

{ name: 'Apples', category: 'Fruit' },

{ name: 'Banana', category: 'Fruit' },

{ name: 'Orange', category: 'Fruit' }

];

vm.getCode = getCode;

function getCode(fn) {

return "The Code of the function: " + fn;

}

}

自定义指令

unorderedList

unorderedList.directive.js

unorderedList.html

指令unorderedList.directive.js

.directive('unorderedList', unorderedList);

function unorderedList() {

var directive = {

restrict: 'E',

templateUrl: 'app/components/unorderedList/unorderedList.html',

**scope: {**

**title: '@',**

**products: '=',**

**getDetail: '&'**

**},**

//controller: NavbarController,

//controllerAs: 'vm',

//bindToController: true

}

return directive;

function NavbarController(moment) {

var vm = this;

vm.title

vm.products

vm.getDetail

}

}

指令视图unorderedList.html

<h3>{{ title }}</h3>

<ul>

<li ng-repeat="item in products">{{ item.name }} - {{ item.category }}</li>

</ul>

<p> {{ getDetail() }}</p>

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

**scope: 创建一个隔离的作用域，angularjs为指令的每个实例创建一个独立的作用域，该作用域不继承作用域层次**。也就是说，隔离的作用域完全从作用域层次结构上被隔绝出来。当创建可复用指令时，隔离的作用域可以防止控制器作用域和指令之间出现意外的交互。但是完全隔绝一个指令会使得难以输入和输出数据？**通过指令的属性绑定**

如：

title=”Hello World” [通过**@单向绑定**数据scope.title](mailto:通过@单向绑定数据scope.title)

products="main.products"通过**=双向绑定**数据scope.products

get-detail="main.getCode (main.getCode)"通过**&绑定函数**scope.getDetail

注意：在隔离作用域上的单向绑定总是被计算作字符串值。如果你想访问一个数组，就必须使用双向绑定，即使你不打算修改它。

包装元素

以上指令，客户端只能传播对象给自定义指令

如果传视图给自定义指令？

客户端使用

视图main.html

<my-panel>

<h3>sub title</h3>

<p>The data value comes from the :{{ main.dataSource }}</p>

</my-panel>

控制器

function MainController($timeout, webDevTec, toastr) {

var vm = this;

vm.dataSource = 'controller';

}

注意:被嵌入在自定义指令里的视图，其作用域是控制器作用域，这是合理的

自定义指令

myPanel

myPanel.directive.js

myPanel.html

myPanel.directive.js

.directive('myPanel', myPanel);

function myPanel() {

return {

restrict: 'E',

transclude: true,

templateUrl: 'app/components/myPanel/myPanel.html'

};

}

myPanel.html

<div class="panel panel-default">

<div class="panel-heading"><h4>This is the panel</h4></div>

<div class="panel-body" ng-transclude></div>

</div>

指令嵌套

客户端

<my-tabs>

<my-pane title="Hello"></my-pane>

<my-pane title="World"></my-pane>

</my-tabs>

子指令访问父指令的控制器

angular.module('docsTabsExample', [])

.directive('myTabs', function() {

return {

restrict: 'E',

controller: ['$scope', function($scope) {

this.addPane = function(pane) {…};

}],

templateUrl: 'my-tabs.html'

};

})

.directive('myPane', function() {

return {

require: '^myTabs',

restrict: 'E',

link: function(scope, element, attrs, tabsCtrl) {

tabsCtrl.addPane(scope);

},

templateUrl: 'my-pane.html'

};

});

子指令myPane通过require:’^myTabs’声明对父指令myTabs的依赖，通过link函数第四个参数tabsCtrl访问父指令myTabs的控制器

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, UI-Router）

* 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) {}

}

}

})

## 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

}) ;

## 工具函数，危险数据，与其他框架集成，动画，单元测试

* 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”});

}) ;

* 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

* AngularJS与其他框架集成

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

<label><input type=”checkbox” ng-mode=”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(“!!”) ;

}) ;

* 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');

};

});

});

}

}

}]);

AngularJS-Toaster

An angularJS port of the Toastr non-blocking notification jQuery library

## 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/#/](http://krispo.github.io/angular-nvd3/" \l "/)

## AngularJS最佳实践

1. 单一职责：一个文件只定义一个组件
2. IIFE

(function(){

‘use strict’ ;

angular.module(‘app’).factory(‘storage’, storage) ;

function storage() {}

})() ;

1. Controller

**controller被构建的时候，就会有一个新的实例，controllerAs的语法比经典的$scope语法更接近Javascript的构造函数**,促进在view中对绑定到“有修饰”的对象的使用（例如用customer.name代替name）, 这将更有语境，更容易阅读，也避免了任何没有“修饰”而产生的引用问题

controller中的$scope被绑定到了this上，避免在controller中使用$scope,最好不用它们或是把它们移动一个factory中，factory中可以考虑使用$scope, controller中只在需要时候才使用$scope,例如当使用$emit, $boradcast, $on…

在activate函数中解决controller的启动逻辑：把启动逻辑放在一个controller中固定的位置可以方便定位，有利用保持测试的一致性，并能够避免controller中到处都是激活逻辑

Controller是构造函数，对应view, 每个view每个controller实例化对象

(function(){

‘use strict’ ;

angular.module(‘app’).controller(‘Customer’, Customer) ;

function Customer(){

var vm = this ;

//接口

vm.name = {} ;

vm.sendMessage = sendMessage ;

//初始化

activate() ;

//实现细节

function activate(){} ;

function sendMessage(){}

}

})() ;

<div ng-controller=”Customer as customer”>

{{customer.name}}

</div>

保持controller的专一性：一个view定义一个controller,尽量不要在其它view中使用这个controller,把可重用的逻辑放到factory中，保证controller只服务于当前视图

1. Services (所有的Angular services都是单例)

推荐用factory，factory单一职责，

接口置顶

(function(){

‘use strict’ ;

angular.module(‘app’).factory(‘dataService’, dataService) ;

function dataService(){

var someValue = ‘’ ;

var service = {

save: save,

someValue: someValue,

validate: validate

};

return service ;

function save(){}

function validate(){}

}

})() ;

独立的数据调用：把进行数据操作和数据交互的逻辑放到factory中，数据服务负责XHR请求，本地存储，内存存储和其它任何数据操作。数据服务的实现可能有非常明确的代码来处理数据仓库，这可能包含headers，如何与数据交互或是其它service,例如$http,把逻辑封装到单独的数据服务中，这隐藏了外部调用者(例如controller)对数据的直接操作，这样更加容易执行变更

(function(){

‘use strict’ ;

angular.module(‘app’).factory(‘dataService’, dataService) ;

dataService.$inject = [‘$http’, ‘logger’] ;

function dataService($http, logger){

return {

getAvengers: getAvengers

} ;

function getAvengers(){

return $http.get(‘/api/maa’).then(getAvengersComplete)

.catch(getAvengersFailed) ;

function getAvengersComplete(respone) { return response.data.results; }

function getAvengersFailed(error) { logger.error(error.data); }

}

}

})() ;

数据调用返回一个promise

(function(){

‘use strict’ ;

angular.module(‘app’).controller(‘Avengers, Avengers) ;

Avengers.$inject = [‘dataService’, ‘logger’] ;

function Avengers(dataService, logger) {

var vm = this ;

vm.avengers = [] ;

activate() ;

function activate(){

return dataService.getAvengers().then(function(data){

vm.avengers = data ;

return vm.avengers ;

}) ;

}

}

})() ;

1. Directives

一个directive一个文件，并依照directive来命名文件

calendarRange.directive.js

.directive(‘acmeOrderCalendarRange’, orderCalendarRange) ;

function orderCalendarRange(){}

<div acme-order-calendar-range></div>

当需要直接操作dom的时候，使用directive,如果有替代方法可以使用，例如，使用css来设置样式，animation services, Angular模板，ngShow或ngHide, 那么就直接使用这些即可

1. Route Revolve Promises

当一个controller在激活之前，需要依赖一个promise的完成时，那么就在controller的逻辑执行之前在$routeProvider中解决这些依赖。

controller在加载前可能需要一些数据，这些数据可能是从一个通过自定义factory或是$http的promise而来的。**route resove允许promise在controller的逻辑执行之前解决，因此它可能对从promise中来的数据做一些处理。**

在路由之前通过一个promise来执行，拒绝了承诺就会取消路由，接受了就会等待路由跳转到新视图，如果你想更快地进入视图，并且无需验证是否可以进入视图，你可以考虑用控制器activate技术

angular.module(‘app’).config(function($routeProvider){

$routeProvider.when(‘/avengers’, {

templateUrl: ‘avengers.html’,

controller: ‘Avengers’,

controllerAs: ‘vm’,

resolve: {

moviesPrepService: function(movieService){

return movieService.getMovies() ; //返回promise

}

}

}) ;

}) ;

angular.module(‘app’).controller(‘Avengers’, [‘moviesPrepService’, function(moviesPrepService){

var vm = this ;

vm.movies = moviesPrepService.movies ;

}]) ;

1. 注入依赖：手动或数组方式？

angular.module(‘app’).controller(‘Dashboard’, [‘$location’, ‘$routeParams’, ‘dataservice’,

function($location, $routeParams, dataservice){

}]) ;

(function(){

angular.module(‘app’).controller(‘Dashboard’, Dashboard) ;

Dashboard.$inject = [‘$location’, ‘$routeParams’, ‘dataservice’] ;

function Dashboard($location, $routeParams, dataservice){

}

})();

1. 功能文件命名

avengers.controller.js AvengersController

logger.service.js logger

app.module.js app

admin.module.js admin

app.config.js //模块app的配置文件

admin.config.js //模块admin的配置文件

app.route.js //模块app的路由文件

admin.route.js //模块admin的路由文件

为什么controller命名是首字母大写，因为**controller是作为构造函数**

为什么service and factory首字母小写，因为**单例对象**

1. 模块化

应用程序的**根模块**的作用：把其它模块都绑定到一起，变成了一个描述哪些模块有助于定义应用程序的清单

**自包含的模块**可以无缝地被添加到应用程序中，项目进行迭代时，可以专注于功能，在开发完成启用它们即可

**通过service创建代表可重用的应用程序块的模块，例如异常处理，日志记录，诊断，安全性或本地数据储藏等模块**

1. 启动逻辑

angular.module(‘app’).config([‘routerHelperProvider’, ‘exceptionHandlerProvider’, ‘toaster’, function([‘routerHelperProvider’, ‘exceptionHandlerProvider’, ‘toaster’){

routerHelperProvider.config() ;

exceptionHandlerProvider.config() ;

toaster.options.timeOut = 4000 ;

toaster.options.positionClass = ‘toast-bottom-right’;

}]) ;

angular.module(‘app’).run([‘authenticator’, ‘translator’,

function(authenticator, translator){

authenticator.initialize() ;

translator.initialize() ;

}]);

1. 常量

//全局常量 constants.js, Constants used by the entire app

(function(){

‘use strict’,

angular.module(‘app.core’)

.constant(‘moment’, false) ;

})();

// Constants used only by the sales module

angular.module(‘app.sales’)

.constant(‘events’ {

ORDER\_CREATED: ‘event\_order\_created’

}) ;

1. 使用angular $包装服务

$document, $window, $timeout, $interval

1. 测试

单元测试：Jasmine or Mocha

测试运行器：Karma （代码发生修改时自动运行，容易hook到持续集成）

1. 动画

在页面过渡时使用angular动画，ngAnimate模块，短持续时间一般为300ms,传统动画使用animate.css, css提供的动画是快速的，流畅的，易于添加到应用程序中的

Gulp: 代码优先

Grunt: 配置优先

# Sencha Touch

1. Install SenchaCmd (for creating, building and deploy application, like grunt for other javascript framework, or like maven/gradle for java application)

Referrence: <http://docs.sencha.com/touch/2.4/getting_started/getting_started.html>

1. Create application

cmd> sencha -sdk sencha\_touch\_sdk\_path generate app MyApp

D:\qzlin\work\language\senchaTouch23\MyApp

cmd> cd D:\qzlin\work\language\senchaTouch23\MyApp

the generated application has following file structure



3. watch file with any modification

D:\qzlin\work\language\senchaTouch23\MyApp> sencha app watch

you have created an application and deployed a server, then you could open browser: localhost:1841

features:

* rich UI controls
* over 300 built-in icons
* full support for theming
* MVC
* data bindings and a data package
* eight complete examples and more
* a rich set of documentation
* Native Packaging

Sencha Cmd includes our native packager, which lets you package and build your application as an Android .apk or an iOS .ipa. Native device APIs that work on both iOS and Android, which includes Camera, Orientation, Network Connectivity, native confirmation dialogs, and more

Sencha Touch now supports Apache Cordova APIs for Accelerometer, Camera, Capture, Compass, Connection, Contacts, Device, Events, File, Geolocation, Globalization, InAppBrowser, Media, Notification, Splashscreen, and Storage. Sencha Touch also supports the Adobe PhoneGap Build from within Sencha Command which can package your application for multiple devices with a single command. Sencha SDK Tools give you the best of both worlds, providing a way to seamlessly “wrap” your web app in a native shell and utilize all the device features.

# Extjs5.0.1

1. setup environment

install ruby

install SenchaCmd

//强烈建议用sencha cmd来创建应用程序模板，文件改变，服务器会自动更新，从而刷新浏览器即可

2. create application

cmd> sencha -sdk extjs\_path generate app MyApp

D:\qzlin\work\language\ExtJS5\MyApp

cmd> cd D:\qzlin\work\language\ExtJS5\MyApp

3. watch file with any modification

D:\qzlin\work\language\ExtJS5\MyApp> sencha app watch

you have created an application and deployed a server, then you could open broswer: localhost:1841

4. add class (view,controller,model)

D:\qzlin\work\language\ExtJS5\MyApp> sencha generate view foo.Thing

Note:

foo is folder corresponding to main, Thing is equal to Thing.is, ThingController.js, ThingModel.js, by default, MyApp.view.foo.Thing is derived from Ext.panel.Panel, but could derived from others, for example:

D:\qzlin\work\language\ExtJS5\MyApp> sencha generate view –base Ext.tab.Panel foo.Thing

4. build as production

D:\qzlin\work\language\ExtJS5\MyApp> sencha app build

For each class in your application, Sencha Cmd checks for a corresponding SCSS file in sass/var/ for variables, and sass/src/ for rules. Since the application has a class named MyApp.view.main.Main, the sass/var/view/main/Main.scss file gets included in the build if you add it. You could also target any other view by creating a {ViewName}.scss file under sass/var/view/{namespace}/{ViewName}.scss

bootstrap.js。这是一个神奇的文件，他根据配置文件bootstrap.json来设置extjs的类的调用路径、引入css，最后再根据配置文件中的信息，加载app.js

MVVM中的事件绑定和属性值绑定都用到

一个按钮“Button”的handler(即click)事件绑定到MainController.js中的函数“OnClickButton”之上。蓝色部分表示将panel的title属性，绑定到了MainModel的属性data的name值上，如果你要修改该Panel的值，则只要修改MainModel中的data.name就可以。

this.getView().getViewModel().set('name' , "修改后的title");

this表示的是当前的类，也就是MainController的实例。

this.getView() 表示当前控制器控制下的View类实例。

this.getView().getViewModel()表示该view实例绑定的Model。

set('name','修改后的title')，将Model的值name修改，修改后panel的title会自动的进行更新。

注意其类名的前面和其路径是一致的。extjs的类加载机制就是根据类名来找到具体的类文件在哪里的

为了找到这个数字，你先要去[Font Awesome](http://www.bootcss.com/p/font-awesome/#icons-new) 网站上找到你需要的图标，记下名称，然后打开 css 目录下的 font-awesome.css,从中找到该名称的.class值，然后记下content的值

自定义一个控件最简单的办法就是继承一个已有的控件

MVC + MVVM (standing for Model-View-ViewModel



Data Binding

Using bind, you can be sure that the appropriate component config will have its setter method called whenever the bound value changes

//View.js

xtype: 'grid',

bind: {

title: 'Summary for {fullName}',

store: '{summary}'

}

// viewmodel.js

Ext.define('App.view.summary.SummaryModel', {

extend: 'Ext.app.ViewModel',

formulas: {

fullName: function (get) {

return get('firstName') + ' ' + get('lastName');

}

},

stores: {

summary: {

source: 'allSummaries', // chains to a global store

filters: [{

property: 'fullName',

value: '{fullName}'

}]

}

}

});

ViewModels

Formulas

Stores

Configuration

In Ext JS 4, we introduced a dedicated config property that gets processed by the powerful Ext.Class pre-processors before the class is created. Features include:

Configurations are completely encapsulated from other class members

Getter and setter methods for every config property are automatically generated into the class prototype during class creation if methods are not already defined.

An apply method is also generated for every config property. The auto-generated setter method calls the apply method internally before setting the value. You may override the apply method for a config property if you need to run custom logic before setting the value. If apply does not return a value, the setter will not set the value.

Ext.define('My.own.Window', {

/\*\* @readonly \*/

isWindow: true,

config: {

title: 'Title Here',

bottomBar: {

height: 50,

resizable: false

}

},

applyTitle: function(title) {

if (!Ext.isString(title) || title.length === 0) {

alert('Error: Title must be a valid non-empty string');

}

else {

return title;

}

},

applyBottomBar: function(bottomBar) {

if (bottomBar) {

if (!this.bottomBar) {

return Ext.create('My.own.WindowBottomBar', bottomBar);

}

else {

this.bottomBar.setConfig(bottomBar);

}

}

}

});

/\*\* A child component to complete the example. \*/

Ext.define('My.own.WindowBottomBar', {

config: {

height: undefined,

resizable: true

}

});

Statics

Static members can be defined using the statics config

Ext.define('Computer', {

statics: {

instanceCount: 0,

factory: function(brand) {

// 'this' in static methods refer to the class itself

return new this({brand: brand});

}

},

config: {

brand: null

}

});

var dellComputer = Computer.factory('Dell');

var appleComputer = Computer.factory('Mac');

alert(appleComputer.getBrand()); // using the auto-generated getter to get the value of a config property. Alerts "Mac"

XTypes and Lazy Instantiation

Every Component has a symbolic name called an [xtype](http://docs.sencha.com/extjs/5.0/apidocs/" \l "!/api/Ext.Component-cfg-xtype). For example [Ext.panel.Panel](http://docs.sencha.com/extjs/5.0/apidocs/" \l "!/api/Ext.panel.Panel) has an xtype of ‘panel’. The above example showed how to add already instantiated [Component](http://docs.sencha.com/extjs/5.0/apidocs/#!/api/Ext.Component)s to a [Container](http://docs.sencha.com/extjs/5.0/apidocs/#!/api/Ext.container.Container). In a large application, however, this is not ideal since not all of the Components need to be instantiated right away, and some Components might never be instantiated depending on how the application is used. For example an application that uses a [Tab Panel](http://docs.sencha.com/extjs/5.0/apidocs/#!/api/Ext.tab.Panel) will only need the contents of each tab to be rendered if and when each tab is clicked on by the user. This is where xtypes come in handy by allowing a Container’s children to be configured up front, but not instantiated until the Container determines it is necessary.

Showing and Hiding

var panel = Ext.create('Ext.panel.Panel', {

renderTo: Ext.getBody(),

title: 'Test',

html: 'Test Panel',

hideMode: 'visibility' // use the CSS visibility property to show and hide this

component

});

panel.hide(); // hide the component

panel.show(); // show the component

Floating Components

Floating [Component](http://docs.sencha.com/extjs/5.0/apidocs/#!/api/Ext.Component) are positioned outside of the document flow using CSS absolute positioning, and do not participate in their Containers’ layout. Some Components such as [Window](http://docs.sencha.com/extjs/5.0/apidocs/#!/api/Ext.window.Window)s are floating by default, but any Component can be made floating using the [floating](http://docs.sencha.com/extjs/5.0/apidocs/#!/api/Ext.Component-cfg-floating) configuration.

var panel = Ext.create('Ext.panel.Panel', {

width: 200,

height: 100,

floating: true, // make this panel an absolutely-positioned floating component

title: 'Test',

html: 'Test Panel'

});

The above code instantiates a [Panel](http://docs.sencha.com/extjs/5.0/apidocs/#!/api/Ext.panel.Panel) but does not render it. Normally a Component either has a [renderTo](http://docs.sencha.com/extjs/5.0/apidocs/" \l "!/api/Ext.Component-cfg-renderTo)configuration specified, or is added as a child Component of a [Container](http://docs.sencha.com/extjs/5.0/apidocs/#!/api/Ext.container.Container), but in the case of floating Components neither of these is needed. Floating Components are automatically rendered to the document body the first time their [show](http://docs.sencha.com/extjs/5.0/apidocs/#!/api/Ext.Component-method-show) method is called:

panel.show(); // render and show the floating panel

Template Methods

Ext JS uses the [Template method pattern](http://en.wikipedia.org/wiki/Template_method_design_pattern) to delegate to subclasses, behavior which is specific only to that subclass.

It is important to note that many of the template methods also have a corresponding event. For example the [render](http://docs.sencha.com/extjs/5.0/apidocs/" \l "!/api/Ext.Component-event-render)event is fired after the Component is rendered. When subclassing, however, it is it is essential to use template methods to perform class logic at important phases in the lifecycle and not events.

Below are the template methods that can be implemented by subclasses of Component:

initComponent This method is invoked by the constructor. It is used to initialize data, set up configurations, and attach event handlers.

beforeShow This method is invoked before the Component is shown.

onShow Allows addition of behavior to the show operation. After calling the superclass’s onShow, the Component will be visible.

afterShow This method is invoked after the Component is shown.

onShowComplete This method is invoked after the afterShow method is complete

onHide Allows addition of behavior to the hide operation. After calling the superclass’s onHide, the Component will be hidden.

afterHide This method is invoked after the Component has been hidden

onRender Allows addition of behavior to the rendering phase.

afterRender Allows addition of behavior after rendering is complete. At this stage the Component’s Element will have been styled according to the configuration, will have had any configured CSS class names added, and will be in the configured visibility and the configured enable state.

onEnable Allows addition of behavior to the enable operation. After calling the superclass’s onEnable, the Component will be enabled.

onDisable Allows addition of behavior to the disable operation. After calling the superclass’s onDisable, the Component will be disabled.

onAdded Allows addition of behavior when a Component is added to a Container. At this stage, the Component is in the parent Container’s collection of child items. After calling the superclass’s onAdded, the ownerCt reference will be present, and if configured with a ref, the refOwner will be set.

onRemoved Allows addition of behavior when a Component is removed from its parent Container. At this stage, the Component has been removed from its parent Container’s collection of child items, but has not been destroyed (It will be destroyed if the parent Container’s autoDestroy is true, or if the remove call was passed a truthy second parameter). After calling the superclass’s onRemoved, the ownerCt and the refOwner will not be present.

onResize Allows addition of behavior to the resize operation.

onPosition Allows addition of behavior to the position operation.

onDestroy Allows addition of behavior to the destroy operation. After calling the superclass’s onDestroy, the Component will be destroyed.

beforeDestroy This method is invoked before the Component is destroyed.

afterSetPosition This method is invoked after the Components position has been set.

afterComponentLayout This method is invoked after the Component is laid out.

beforeComponentLayout This method is invoked before the Component is laid out.

Component

If the required UI Component does not need to contain any other Components, that is, if it just to encapsulate some form of HTML which performs the requirements, then extending [Ext.Component](http://docs.sencha.com/extjs/5.0/apidocs/" \l "!/api/Ext.Component) is appropriate. For example, the following class is a Component that wraps an HTML image element, and allows setting and getting of the image’s src attribute. It also fires a load event when the image is loaded:

Ext.define('Ext.ux.Image', {

extend: 'Ext.Component', // subclass Ext.Component

alias: 'widget.managedimage', // this component will have an xtype of 'managedimage'

autoEl: {

tag: 'img',

src: Ext.BLANK\_IMAGE\_URL,

cls: 'my-managed-image'

},

// Add custom processing to the onRender phase.

// Add a 'load' listener to the element.

onRender: function() {

this.autoEl = Ext.apply({}, this.initialConfig, this.autoEl);

this.callParent(arguments);

this.el.on('load', this.onLoad, this);

},

onLoad: function() {

this.fireEvent('load', this);

},

setSrc: function(src) {

if (this.rendered) {

this.el.dom.src = src;

} else {

this.src = src;

}

},

getSrc: function(src) {

return this.el.dom.src || this.src;

}

});

Components

<http://docs.sencha.com/extjs/5.0/core_concepts/components.html>

Scope Listener Option

Scope sets the value of this inside your handler function. By default, this is set to the instance of the class firing the event. This is often, but not always, the functionality that you want. This functionality allows us to call this.hide()to hide the button in the second example earlier in this guide. In the following example, we create a Button and a Panel. We then listen to the Button’s click event with the handler running in Panel’s scope. In order to do this, we need to pass in an object instead of a handler function. This object contains the function AND the scope:

var panel = Ext.create('Ext.Panel', {

html: 'Panel HTML'

});

var button = Ext.create('Ext.Button', {

renderTo: Ext.getBody(),

text: 'Click Me'

});

button.on({

click: {

scope: panel,

fn: function() {

Ext.Msg.alert(this.getXType());

}

}

});

Listening to an Event Once

You may want to listen to one event only once. The event itself might fire any number of times, but we only want to listen to it once. The following codes illustrates this situation:

var button = Ext.create('Ext.Button', {

renderTo: Ext.getBody(),

text: 'Click Me',

listeners: {

click: {

single: true,

fn: function() {

Ext.Msg.alert('I will say this only once');

}

}

}

});

# Bootstrap 3.X

## 概览

* HTML5 文档类型

Bootstrap 使用到的某些 HTML 元素和 CSS 属性需要将页面设置为 HTML5 文档类型。在你项目中的每个页面都要参照下面的格式进行设置。

<!DOCTYPE html>

<html lang="zh-CN">

...

</html>

* 移动设备优先

在 Bootstrap 2 中，我们对框架中的某些关键部分增加了对移动设备友好的样式。而在 Bootstrap 3 中，我们重写了整个框架，使其一开始就是对移动设备友好的。这次不是简单的增加一些可选的针对移动设备的样式，而是直接融合进了框架的内核中。也就是说，Bootstrap 是移动设备优先的。针对移动设备的样式融合进了框架的每个角落，而不是增加一个额外的文件。为了确保适当的绘制和触屏缩放，需要在 <head> 之中添加 viewport 元数据标签。在移动设备浏览器上，通过为视口（viewport）设置 meta 属性为 user-scalable=no 可以禁用其缩放（zooming）功能。这样禁用缩放功能后，用户只能滚动屏幕，就能让你的网站看上去更像原生应用的感觉。注意，这种方式我们并不推荐所有网站使用，还是要看你自己的情况而定！

<meta name="viewport" content="width=device-width, initial-scale=1, maximum-scale=1, user-scalable=no">

* 排版与链接

Bootstrap 排版、链接样式设置了基本的全局样式。分别是：

* 为 body 元素设置 background-color: #fff;
* 使用 @font-family-base、@font-size-base 和 @line-height-base a变量作为排版的基本参数
* 为所有链接设置了基本颜色 @link-color ，并且当链接处于 :hover 状态时才添加下划线

这些样式都能在 scaffolding.less 文件中找到对应的源码。

* Normalize.css

为了增强跨浏览器表现的一致性，我们使用了 Normalize.css，这是由 Nicolas Gallagher 和 Jonathan Neal 维护的一个CSS 重置样式库。

* 布局容器

Bootstrap **需要为页面内容和栅格系统包裹一个 .container 容器**。我们提供了两个作此用处的类。注意，由于 padding 等属性的原因，这两种 容器类不能互相嵌套。

固定宽度响应式布局容器

.container 类用于固定宽度并支持响应式布局的容器。

<div class="container">

...

</div>

流式布局容器

.container-fluid 类用于 100% 宽度，占据全部视口（viewport）的容器。

<div class="container-fluid">

<div class="row">

...

</div>

</div>

## 栅格系统

1. 简介

Bootstrap 提供了一套响应式、移动设备优先的流式栅格系统，随着屏幕或视口（viewport）尺寸的增加，系统会自动分为最多12列,栅格系统用于通过一系列的行（row）与列（column）的组合来创建页面布局，你的内容就可以放入这些创建好的布局中

工作原理：

* **“行（row）”必须包含在 .container （固定宽度）或 .container-fluid （100% 宽度）中，以便为其赋予合适的排列（aligment）和内补（padding）。**
* 通过“行（row）”在水平方向创建一组“列（column）”。
* 你的内容应当放置于“列（column）”内，并且，只有“列（column）”可以作为行（row）”的直接子元素。
* 类似 .row 和 .col-xs-4 这种预定义的类，可以用来快速创建栅格布局。Bootstrap 源码中定义的 mixin 也可以用来创建语义化的布局。
* **通过为“列（column）”设置 padding 属性，从而创建列与列之间的间隔（gutter）。通过为 .row 元素设置负值 margin 从而抵消掉为 .container 元素设置的 padding，也就间接为“行（row）”所包含的“列（column）”抵消掉了padding。**
* 负值的 margin就是下面的示例为什么是向外突出的原因。在栅格列中的内容排成一行。
* 栅格系统中的列是通过指定1到12的值来表示其跨越的范围。例如，三个等宽的列可以使用三个 .col-xs-4 来创建。
* **如果一“行（row）”中包含了的“列（column）”大于 12，多余的“列（column）”所在的元素将被作为一个整体另起一行排列。**
* **Grid classes apply to devices with screen widths greater than or equal to the breakpoint sizes, and override grid classes targeted at smaller devices**. Therefore, e.g. applying any .col-md-\* class to an element will not only affect its styling on medium devices but also on large devices if a .col-lg-\* class is not present.

1. 媒体查询

在栅格系统中，我们在 Less 文件中使用以下媒体查询（media query）来创建关键的分界点阈值。

/\* 超小屏幕（手机，小于 768px） \*/

/\* 没有任何媒体查询相关的代码，因为这在 Bootstrap 中是默认的（还记得 Bootstrap 是移动设备优先的吗？） \*/

/\* 小屏幕（平板，大于等于 768px） \*/

@media (min-width: @screen-sm-min) { ... }

/\* 中等屏幕（桌面显示器，大于等于 992px） \*/

@media (min-width: @screen-md-min) { ... }

/\* 大屏幕（大桌面显示器，大于等于 1200px） \*/

@media (min-width: @screen-lg-min) { ... }

我们偶尔也会在媒体查询代码中包含 max-width 从而将 CSS 的影响限制在更小范围的屏幕大小之内。

@media (max-width: @screen-xs-max) { ... }

@media (min-width: @screen-sm-min) and (max-width: @screen-sm-max) { ... }

@media (min-width: @screen-md-min) and (max-width: @screen-md-max) { ... }

@media (min-width: @screen-lg-min) { ... }

1. 栅格参数

通过下表可以详细查看 Bootstrap 的栅格系统是如何在多种屏幕设备上工作的。



1. 实例：手机、平板、桌面

使用针对超小屏幕定义的类 .col-xs-\*

使用针对平板屏幕定义的类 .col-sm-\*

使用针对中等屏幕定义的类 .col-md-\*

  手机

 <- 平板

 <- 中等屏幕 （电脑）

<div class="row">

<div class="col-xs-12 col-sm-6 col-md-8">.col-xs-12 .col-sm-6 .col-md-8</div>

<div class="col-xs-6 col-md-4">.col-xs-6 .col-md-4</div>

</div>

<div class="row">

<div class="col-xs-6 col-sm-4">.col-xs-6 .col-sm-4</div>

<div class="col-xs-6 col-sm-4">.col-xs-6 .col-sm-4</div>

<!-- Optional: clear the XS cols if their content doesn't match in height -->

<div class="clearfix visible-xs-block"></div>

<div class="col-xs-6 col-sm-4">.col-xs-6 .col-sm-4</div>

</div>

1. 列偏移

使用 .col-md-offset-\* 类可以将列向右侧偏移。这些类实际是通过使用 \* 选择器为当前元素增加了左侧的边距（margin）。例如，.col-md-offset-4 类将 .col-md-4 元素向右侧偏移了4个列（column）的宽度。



<div class="row">

<div class="col-md-4">.col-md-4</div>

<div class="col-md-4 col-md-offset-4">.col-md-4 .col-md-offset-4</div>

</div>

<div class="row">

<div class="col-md-3 col-md-offset-3">.col-md-3 .col-md-offset-3</div>

<div class="col-md-3 col-md-offset-3">.col-md-3 .col-md-offset-3</div>

</div>

<div class="row">

<div class="col-md-6 col-md-offset-3">.col-md-6 .col-md-offset-3</div>

</div>

1. 嵌套列

为了使用内置的栅格系统将内容再次嵌套，可以通过添加一个新的 .row 元素和一系列 .col-sm-\* 元素到已经存在的 .col-sm-\* 元素内。被嵌套的行（row）所包含的列（column）的个数不能超过12（其实，没有要求你必须占满12列）。



<div class="row">

<div class="col-sm-9">

Level 1: .col-sm-9

<div class="row">

<div class="col-xs-8 col-sm-6">

Level 2: .col-xs-8 .col-sm-6

</div>

<div class="col-xs-4 col-sm-6">

Level 2: .col-xs-4 .col-sm-6

</div>

</div>

</div>

</div>

1. Less mixin 和变量

除了用于快速布局的预定义栅格类，Bootstrap 还包含了一组 Less 变量和 mixin 用于帮你生成简单、语义化的布局。

变量

通过变量来定义列数、槽（gutter）宽、媒体查询阈值（用于确定合适让列浮动）。我们使用这些变量生成预定义的栅格类，如上所示，还有如下所示的定制 mixin。

@grid-columns: 12;

@grid-gutter-width: 30px;

@grid-float-breakpoint: 768px;

mixin

mixin 用来和栅格变量一同使用，为每个列（column）生成语义化的 CSS 代码。

## 全局 CSS 样式

排版

基本的 HTML 元素均可以通过 class 设置样式并得到增强效果，variables.less 文件中定义的两个 Less 变量决定了排版尺寸：@font-size-base 和 @line-height-base。第一个变量定义了全局 font-size 基准，第二个变量是 line-height 基准。我们使用这些变量和一些简单的公式计算出其它所有页面元素的 margin、 padding 和 line-height。自定义这些变量即可改变 Bootstrap 的默认样式。

标题

HTML 中的所有标题标签，<h1> 到 <h6> 均可使用。另外，还提供了 .h1 到 .h6 类，为的是给内联（inline）属性的文本赋予标题的样式。在标题内还可以包含 <small> 标签或赋予 .small 类的元素，可以用来标记副标题。

页面主体

Bootstrap 将**全局 font-size 设置为 14px，line-height 设置为 1.428**。这些属性直接赋予 <body> 元素和所有段落元素。另外，**<p> （段落）元素还被设置了等于 1/2 行高（即 10px）的底部外边距（margin）。**

中心内容

通过添加 .lead 让段落突出显示: <p class="lead">...</p>

内联文本元素

Marked text

You can use the mark tag to <mark>highlight</mark> text.

被删除的文本:对于被删除的文本使用 <del> 标签。

<del>This line of text is meant to be treated as deleted text.</del>

无用文本: 对于没用的文本使用 <s> 标签。

<s>This line of text is meant to be treated as no longer accurate.</s>

插入文本: 额外插入的文本使用 <ins> 标签。

<ins>This line of text is meant to be treated as an addition to the document.</ins>

带下划线的文本: 为文本添加下划线，使用 <u> 标签。

<u>This line of text will render as underlined</u>

小号文本

对于不需要强调的inline或block类型的文本，使用 <small> 标签包裹，其内的文本将被设置为父容器字体大小的 85%。标题元素中嵌套的 <small> 元素被设置不同的 font-size 。你还可以为行内元素赋予 .small 类以代替任何 <small> 元素。

<small>This line of text is meant to be treated as fine print.</small>

着重: 通过增加 font-weight 值强调一段文本。

<strong>rendered as bold text</strong>

斜体: 用斜体强调一段文本。

<em>rendered as italicized text</em>

在 HTML5 中可以放心使用 <b> 和 <i> 标签。<b> 用于高亮单词或短语，不带有任何着重的意味；而 <i> 标签主要用于发言、技术词汇等。

对齐

通过文本对齐类，可以简单方便的将文字重新对齐。

<p class="text-left">Left aligned text.</p>

<p class="text-center">Center aligned text.</p>

<p class="text-right">Right aligned text.</p>

<p class="text-justify">Justified text.</p>

<p class="text-nowrap">No wrap text.</p>

改变大小写

通过这几个类可以改变文本的大小写。

<p class="text-lowercase">Lowercased text.</p>

<p class="text-uppercase">Uppercased text.</p>

<p class="text-capitalize">Capitalized text.</p>

缩略语

当鼠标悬停在缩写和缩写词上时就会显示完整内容，Bootstrap 实现了对 HTML 的 <abbr> 元素的增强样式。缩略语元素带有 title 属性，外观表现为带有较浅的虚线框，鼠标移至上面时会变成带有“问号”的指针。如想看完整的内容可把鼠标悬停在缩略语上（对使用辅助技术的用户也可见）, 但需要包含 title 属性。

基本缩略语

<abbr title="attribute">attr</abbr>

首字母缩略语: 为缩略语添加 .initialism 类，可以让 font-size 变得稍微小些。

<abbr title="HyperText Markup Language" class="initialism">HTML</abbr>

地址

让联系信息以最接近日常使用的格式呈现。在每行结尾添加 <br> 可以保留需要的样式。

<address>

<strong>Twitter, Inc.</strong><br>

795 Folsom Ave, Suite 600<br>

San Francisco, CA 94107<br>

<abbr title="Phone">P:</abbr> (123) 456-7890

</address>

<address>

<strong>Full Name</strong><br>

<a href="mailto:#">first.last@example.com</a>

</address>

引用

在你的文档中引用其他来源的内容。

默认样式的引用: 将任何 HTML 元素包裹在 <blockquote> 中即可表现为引用样式。对于直接引用，我们建议用 <p> 标签。

<blockquote>

<p>Lorem ipsum dolor sit amet, consectetur adipiscing elit. Integer posuere erat a ante.</p>

</blockquote>

多种引用样式

对于标准样式的 <blockquote>，可以通过几个简单的变体就能改变风格和内容。

命名来源

添加 <footer> 用于标明引用来源。来源的名称可以包裹进 <cite>标签中。

<blockquote>

<p>Lorem ipsum dolor sit amet, consectetur adipiscing elit. Integer posuere erat a ante.</p>

<footer>Someone famous in <cite title="Source Title">Source Title</cite></footer>

</blockquote>

另一种展示风格

通过赋予 .blockquote-reverse 类可以让引用呈现内容右对齐的效果。

<blockquote class="blockquote-reverse">

...

</blockquote>

列表

无序列表:排列顺序无关紧要的一列元素。

<ul>

<li>...</li>

</ul>



有序列表: 顺序至关重要的一组元素

<ol>

<li>...</li>

</ol>



无样式列表

移除了默认的 list-style 样式和左侧外边距的一组元素（只针对直接子元素）。这是针对直接子元素的，也就是说，你需要对所有嵌套的列表都添加这个类才能具有同样的样式。

<ul **class="list-unstyled"**>

<li>...</li>

</ul>



内联列表

通过设置 display: inline-block; 并添加少量的内补（padding），将所有元素放置于同一行。

<ul **class="list-inline"**>

<li>...</li>

</ul>



描述:带有描述的短语列表。

<dl>

<dt>...</dt>

<dd>...</dd>

</dl>



水平排列的描述

.dl-horizontal 可以让 <dl> 内的短语及其描述排在一行。开始是像 <dl> 的默认样式堆叠在一起，随着导航条逐渐展开而排列在一行。

<dl **class="dl-horizontal"**>

<dt>...</dt>

<dd>...</dd>

</dl>



自动截断

通过 text-overflow 属性，水平排列的描述列表将会截断左侧太长的短语。在较窄的视口（viewport）内，列表将变为默认堆叠排列的布局方式。

* 代码

内联代码

通过 <code> 标签包裹内联样式的代码片段

For example, <code>&lt;section&gt;</code> should be wrapped as inline.



用户输入

通过 <kbd> 标签标记用户通过键盘输入的内容。=

To switch directories, type <kbd>cd</kbd> followed by the name of the directory.<br>

To edit settings, press <kbd><kbd>ctrl</kbd> + <kbd>,</kbd></kbd>



代码块

多行代码可以使用 <pre> 标签。为了正确的展示代码，注意将尖括号做转义处理。

<pre>&lt;p&gt;Sample text here...&lt;/p&gt;</pre>

变量

通过 <var> 标签标记变量。

<var>y</var> = <var>m</var><var>x</var> + <var>b</var>

程序输出

通过 <samp> 标签来标记程序输出的内容。

<samp>This text is meant to be treated as sample output from a computer program.</samp>

* 表格

基本实例

为任意 <table> 标签添加 .table 类可以为其赋予基本的样式 — 少量的内补（padding）和水平方向的分隔线。这种方式看起来很多余！？但是我们觉得，表格元素使用的很广泛，如果我们为其赋予默认样式可能会影响例如日历和日期选择之类的插件，所以我们选择将此样式独立出来。

<table **class="table"**>

...

</table>



条纹状表格

通过 .table-striped 类可以给 <tbody> 之内的每一行增加斑马条纹样式。

<table **class="table table-striped"**>

...

</table>



带边框的表格

添加 .table-bordered 类为表格和其中的每个单元格增加边框。

<table **class="table table-bordered"**>

...

</table>



鼠标悬停

通过添加 .table-hover 类可以让 <tbody> 中的每一行对鼠标悬停状态作出响应。

<table **class="table table-hover"**>

...

</table>

紧缩表格

通过添加 .table-condensed 类可以让表格更加紧凑，单元格中的内补（padding）均会减半。

<table **class="table table-condensed"**>

...

</table>

状态类

通过这些状态类可以为行或单元格设置颜色。

Class 描述

.active 鼠标悬停在行或单元格上时所设置的颜色

.success 标识成功或积极的动作

.info 标识普通的提示信息或动作

.warning 标识警告或需要用户注意

.danger 标识危险或潜在的带来负面影响的动作

<!-- On rows -->

<tr class="active">...</tr>

<tr class="success">...</tr>

<tr class="warning">...</tr>

<tr class="danger">...</tr>

<tr class="info">...</tr>

<!-- On cells (`td` or `th`) -->

<tr>

<td class="active">...</td>

<td class="success">...</td>

<td class="warning">...</td>

<td class="danger">...</td>

<td class="info">...</td>

</tr>

响应式表格

将任何 .table 元素包裹在 .table-responsive 元素内，即可创建响应式表格，其会在小屏幕设备上（小于768px）水平滚动。当屏幕大于 768px 宽度时，水平滚动条消失。垂直方向的内容截断: 响应式表格使用了 overflow-y: hidden 属性，这样就能将超出表格底部和顶部的内容截断。特别是，也可以截断下拉菜单和其他第三方组件。

<div **class="table-responsive"**>

<table class="table">

...

</table>

</div>

* 表单

基本实例

单独的表单控件会被自动赋予一些全局样式。所有设置了 .form-control 类的 <input>、<textarea> 和 <select> 元素都将被默认设置宽度属性为 width: 100%;。 将 label 元素和前面提到的控件包裹在 .form-group 中可以获得最好的排列。

<form>

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

<label for="exampleInputEmail1">Email address</label>

<input type="email" **class="form-control"** id="exampleInputEmail1" placeholder="Email">

</div>

<div class="form-group">

<label for="exampleInputPassword1">Password</label>

<input type="password" class="form-control" id="exampleInputPassword1" placeholder="Password">

</div>

<div class="form-group">

<label for="exampleInputFile">File input</label>

<input type="file" id="exampleInputFile">

<p class="help-block">Example block-level help text here.</p>

</div>

<div class="checkbox">

<label>

<input type="checkbox"> Check me out

</label>

</div>

<button type="submit" class="btn btn-default">Submit</button>

</form>



内联表单

为 <form> 元素添加 .form-inline 类可使其内容左对齐并且表现为 inline-block 级别的控件。只适用于视口（viewport）至少在 768px 宽度时（视口宽度再小的话就会使表单折叠）



<form name=*"searchForm"* **class=*"form-inline"***>

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

<input type=*"text"* **class=*"form-control"*** placeholder=*"Search"*>

</div>

<button **class=*"btn btn-info btn-sm"*** ng-click=*"search()"* tooltip=*"…"*><i class=*"glyphicon glyphicon-search"*></i>

</button>

</form>

一定要添加 label 标签

如果你没有为每个输入控件设置 label 标签，屏幕阅读器将无法正确识别。对于这些内联表单，你可以通过为 label 设置 .sr-only 类将其隐藏。还有一些辅助技术提供label标签的替代方案，比如 aria-label、aria-labelledby 或 title 属性。如果这些都不存在，屏幕阅读器可能会采取使用 placeholder 属性，如果存在的话，使用占位符来替代其他的标记，但要注意，这种方法是不妥当的。

<form class="form-inline">

<div class="form-group">

<label **class="sr-only"** for="exampleInputEmail3">Email address</label>

<input type="email" class="form-control" id="exampleInputEmail3" placeholder="Email">

</div>

<div class="form-group">

<label class="sr-only" for="exampleInputPassword3">Password</label>

<input type="password" class="form-control" id="exampleInputPassword3" placeholder="Password">

</div>

<div class="checkbox">

<label>

**<input type="checkbox">** Remember me

</label>

</div>

<button type="submit" class="btn btn-default">Sign in</button>

</form>



水平排列的表单

通过为表单添加 .form-horizontal 类，并联合使用 Bootstrap 预置的栅格类，可以将 label 标签和控件组水平并排布局。这样做将改变 .form-group 的行为，使其表现为栅格系统中的行（row），因此就无需再额外添加 .row 了。

<**form class="form-horizontal"**>

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

<label for="inputEmail3" class="col-sm-2 control-label">Email</label>

<div class="col-sm-10">

<input type="email" **class="form-control"** id="inputEmail3" placeholder="Email">

</div>

</div>

<div class="form-group">

<div class="col-sm-offset-2 col-sm-10">

<div class="checkbox">

<label><input type="checkbox"> Remember me</label>

</div>

</div>

</div>

<div class="form-group">

<div class="col-sm-offset-2 col-sm-10">

<button type="submit" class="btn btn-default">Sign in</button>

</div>

</div>

</form>



被支持的控件

输入框

包括大部分表单控件、文本输入域控件，还支持所有 HTML5 类型的输入控件： text、password、datetime、datetime-local、date、month、time、week、number、email、url、search、tel 和 color。

必须添加类型声明

只有正确设置了 type 属性的输入控件才能被赋予正确的样式。

<input **type="text" class="form-control"** placeholder="Text input">

文本域

支持多行文本的表单控件。可根据需要改变 rows 属性。

<textarea class="form-control" rows="3"></textarea>

多选和单选框

默认外观（堆叠在一起）

<div class="checkbox">

<label>

<input type="checkbox" value="">

Option one is this and that&mdash;be sure to include why it's great

</label>

</div>

<div class="checkbox disabled">

<label>

<input type="checkbox" value="" disabled>

Option two is disabled

</label>

</div>

<div class="radio">

<label>

<input type="radio" name="optionsRadios" id="optionsRadios2" value="option2">

Option two can be something else and selecting it will deselect option one

</label>

</div>

<div class="radio disabled">

<label>

<input type="radio" name="optionsRadios" id="optionsRadios3" value="option3" disabled>

Option three is disabled

</label>

</div>

内联单选和多选框

通过将 .checkbox-inline 或 .radio-inline 类应用到一系列的多选框（checkbox）或单选框（radio）控件上，可以使这些控件排列在一行。

<label **class="checkbox-inline"**>

<input **type="checkbox"** id="inlineCheckbox1" value="option1"> 1

</label>

<label **class="checkbox-inline"**>

<input **type="checkbox"** id="inlineCheckbox2" value="option2"> 2

</label>

<label class="radio-inline">

<input type="radio" name="inlineRadioOptions" id="inlineRadio1" value="option1"> 1

</label>

<label class="radio-inline">

<input type="radio" name="inlineRadioOptions" id="inlineRadio2" value="option2"> 2

</label>

下拉列表（select）

<select class="form-control">

<option>1</option>

<option>2</option>

<option>3</option>

<option>4</option>

<option>5</option>

</select>

静态控件

如果需要在表单中将一行纯文本和 label 元素放置于同一行，为 <p> 元素添加 .form-control-static 类即可。

<div class="form-group">

<label class="col-sm-2 **control-label**">Email</label>

<div class="col-sm-10">

<p **class="form-control-static"**>email@example.com</p>

</div>

</div>



校验状态

Bootstrap 对表单控件的校验状态，如 error、warning 和 success 状态，都定义了样式。使用时，添加 .has-warning、.has-error 或 .has-success 类到这些控件的父元素即可。任何包含在此元素之内的 .control-label、.form-control 和 .help-block 元素都将接受这些校验状态的样式。

添加额外的图标

你还可以针对校验状态为输入框添加额外的图标。只需设置相应的 .has-feedback 类并添加正确的图标即可。反馈图标（feedback icon）只能使用在文本输入框 <input class="form-control"> 元素上。

<div class="form-group has-success has-feedback">

<label class="control-label" for="inputGroupSuccess1">Input group with success</label>

<div class="input-group">

<span class="input-group-addon">@</span>

<input type="text" class="form-control" id="inputGroupSuccess1" aria-describedby="inputGroupSuccess1Status">

</div>

<span class="glyphicon glyphicon-ok form-control-feedback" aria-hidden="true"></span>

<span id="inputGroupSuccess1Status" class="sr-only">(success)</span>

</div>



为水平排列的表单和内联表单设置可选的图标

<form class="form-horizontal">

<div class="form-group has-success has-feedback">

<label class="control-label col-sm-3" for="inputSuccess3">Input with success</label>

<div class="col-sm-9">

<input type="text" class="form-control" id="inputSuccess3" aria-describedby="inputSuccess3Status">

<span class="glyphicon glyphicon-ok form-control-feedback" aria-hidden="true"></span>

<span id="inputSuccess3Status" class="sr-only">(success)</span>

</div>

</div>

<div class="form-group has-success has-feedback">

<label class="control-label col-sm-3" for="inputGroupSuccess2">Input group with success</label>

<div class="col-sm-9">

<div class="input-group">

<span class="input-group-addon">@</span>

<input type="text" class="form-control" id="inputGroupSuccess2" aria-describedby="inputGroupSuccess2Status">

</div>

<span class="glyphicon glyphicon-ok form-control-feedback" aria-hidden="true"></span>

<span id="inputGroupSuccess2Status" class="sr-only">(success)</span>

</div>

</div>

</form>



控件尺寸

通过 .input-lg 类似的类可以为控件设置高度，通过 .col-lg-\* 类似的类可以为控件设置宽度。

高度尺寸

创建大一些或小一些的表单控件以匹配按钮尺寸。

<input class="form-control input-lg" type="text" placeholder=".input-lg">

<input class="form-control" type="text" placeholder="Default input">

<input class="form-control input-sm" type="text" placeholder=".input-sm">



调整列（column）尺寸

用栅格系统中的列（column）包裹输入框或其任何父元素，都可很容易的为其设置宽度。

<div class="row">

<div class="col-xs-2">

<input type="text" class="form-control" placeholder=".col-xs-2">

</div>

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

<input type="text" class="form-control" placeholder=".col-xs-3">

</div>

<div class="col-xs-4">

<input type="text" class="form-control" placeholder=".col-xs-4">

</div>

</div>



* 按钮

可作为按钮使用的标签或元素

为 <a>、<button> 或 <input> 元素添加按钮类（button class）即可使用 Bootstrap 提供的样式。

**<a class="btn btn-default" href="#" role="button">Link</a>**

<button class="btn btn-default" type="submit">Button</button>

<input class="btn btn-default" type="button" value="Input">

<input class="btn btn-default" type="submit" value="Submit">

链接被作为按钮使用时的注意事项

如果 <a> 元素被作为按钮使用 -- 并用于在当前页面触发某些功能 -- 而不是用于链接其他页面或链接当前页面中的其他部分，那么，务必为其设置 role="button" 属性。

跨浏览器展现

我们总结的最佳实践是：强烈建议尽可能使用 <button> 元素来获得在各个浏览器上获得相匹配的绘制效果。

预定义样式

<!-- Standard button -->

<button type="button" class="btn btn-default">（默认样式）Default</button>

<!-- Provides extra visual weight and identifies the primary action in a set of buttons -->

<button type="button" class="btn btn-primary">（首选项）Primary</button>

<!-- Indicates a successful or positive action -->

<button type="button" class="btn btn-success">（成功）Success</button>

<!-- Contextual button for informational alert messages -->

<button type="button" class="btn btn-info">（一般信息）Info</button>

<!-- Indicates caution should be taken with this action -->

<button type="button" class="btn btn-warning">（警告）Warning</button>

<!-- Indicates a dangerous or potentially negative action -->

<button type="button" class="btn btn-danger">（危险）Danger</button>

<!-- Deemphasize a button by making it look like a link while maintaining button behavior -->

<button type="button" class="btn btn-link">（链接）Link</button>



尺寸

需要让按钮具有不同尺寸吗？使用 .btn-lg、.btn-sm 或 .btn-xs 就可以获得不同尺寸的按钮。

<p>

<button type="button" class="btn btn-primary btn-lg">（大按钮）Large button</button>

<button type="button" class="btn btn-default btn-lg">（大按钮）Large button</button>

</p>

<p>

<button type="button" class="btn btn-primary">（默认尺寸）Default button</button>

<button type="button" class="btn btn-default">（默认尺寸）Default button</button>

</p>

<p>

<button type="button" class="btn btn-primary btn-sm">（小按钮）Small button</button>

<button type="button" class="btn btn-default btn-sm">（小按钮）Small button</button>

</p>

<p>

<button type="button" class="btn btn-primary btn-xs">（超小尺寸）Extra small button</button>

<button type="button" class="btn btn-default btn-xs">（超小尺寸）Extra small button</button>

</p>



通过给按钮添加 .btn-block 类可以将其拉伸至父元素100%的宽度，而且按钮也变为了块级（block）元素。

<button type="button" class="btn btn-primary btn-lg btn-block">（块级元素）Block level button</button>

<button type="button" class="btn btn-default btn-lg btn-block">（块级元素）Block level button</button>



激活状态

当按钮处于激活状态时，其表现为被按压下去（底色更深、边框夜色更深、向内投射阴影）。**对于 <button> 元素，是通过 :active 状态实现的。对于 <a> 元素，是通过 .active 类实现的**。然而，你还可以将 .active 应用到 <button> 上（包含 aria-pressed="true" 属性)），并通过编程的方式使其处于激活状态。

可以为基于 <a> 元素创建的按钮添加 .active 类。

<a href="#" class="btn btn-primary btn-lg active" role="button">Primary link</a>

<a href="#" class="btn btn-default btn-lg active" role="button">Link</a>



* 图片

响应式图片

在 Bootstrap 版本 3 中，通过为图片添加 .img-responsive 类可以让图片支持响应式布局。其实质是为图片设置了 max-width: 100%;、 height: auto; 和 display: block; 属性，从而让图片在其父元素中更好的缩放。

如果需要让使用了 .img-responsive 类的图片水平居中，请使用 .center-block 类，不要用 .text-center。

<img src="..." class="img-responsive" alt="Responsive image">

图片形状

通过为 <img> 元素添加以下相应的类，可以让图片呈现不同的形状。

<img src="..." alt="..." **class="img-rounded"**>

<img src="..." alt="..." **class="img-circle"**>

<img src="..." alt="..." **class="img-thumbnail"**>



* 辅助类

情境文本颜色

<p class="text-muted">...</p>

<p class="text-primary">...</p>

<p class="text-success">...</p>

<p class="text-info">...</p>

<p class="text-warning">...</p>

<p class="text-danger">...</p>



情境背景色

和情境文本颜色类一样，使用任意情境背景色类就可以设置元素的背景。链接组件在鼠标经过时颜色会加深，就像上面所讲的情境文本颜色类一样。

<p class="bg-primary">...</p>

<p class="bg-success">...</p>

<p class="bg-info">...</p>

<p class="bg-warning">...</p>

<p class="bg-danger">...</p>

关闭按钮

通过使用一个象征关闭的图标，可以让模态框和警告框消失。

<button type="button" class="close" aria-label="Close"><span aria-hidden="true">&times;</span></button>



三角符号

通过使用三角符号可以指示某个元素具有下拉菜单的功能。注意，向上弹出式菜单中的三角符号是反方向的。

<span class="caret"></span>



快速浮动

将任意元素向左或向右浮动

**<div class="pull-left">...</div>**

**<div class="pull-right">...</div>**

让内容块居中

为任意元素设置 display: block 属性并通过 margin 属性让其中的内容居中

<div **class="center-block"**>...</div>

清除浮动

通过为父元素添加 .clearfix 类可以很容易地清除浮动（float）。这里所使用的是 Nicolas Gallagher 创造的 micro clearfix 方式。此类还可以作为 mixin 使用。

<!-- Usage as a class -->

<div **class="clearfix"**>...</div>

显示或隐藏内容

.show 和 .hidden 类可以强制任意元素显示或隐藏(对于屏幕阅读器也能起效)。这些类通过 !important 来避免 CSS 样式优先级问题，就像 quick floats 一样的做法。注意，这些类只对块级元素起作用，另外，还可以作为 mixin 使用。

.hide 类仍然可用，但是它不能对屏幕阅读器起作用，并且从 v3.0.1 版本开始就不建议使用了。请使用 .hidden 或 .sr-only 。

另外，.invisible 类可以被用来仅仅影响元素的可见性，也就是所，元素的 display 属性不被改变，并且这个元素仍然能够影响文档流的排布。

<div class="show">...</div>

<div class="hidden">...</div>

屏幕阅读器和键盘导航

.sr-only 类可以对屏幕阅读器以外的设备隐藏内容。.sr-only 和 .sr-only-focusable 联合使用的话可以在元素有焦点的时候再次显示出来（例如，使用键盘导航的用户）。对于遵循 可访问性的最佳实践 很有必要。这个类也可以作为 mixin 使用。

<a class="sr-only sr-only-focusable" href="#content">Skip to main content</a>

响应式工具

为了加快对移动设备友好的页面开发工作，利用媒体查询功能并使用这些工具类可以方便的针对不同设备展示或隐藏页面内容。另外还包含了针对打印机显示或隐藏内容的工具类。

有针对性的使用这类工具类，从而避免为同一个网站创建完全不同的版本。相反，通过使用这些工具类可以在不同设备上提供不同的展现形式。

通过单独或联合使用以下列出的类，可以针对不同屏幕尺寸隐藏或显示页面内容。



## 组件

* Glyphicons 字体图标

**出于性能的考虑，所有图标都需要一个基类和对应每个图标的类**

为了设置正确的内补（padding），务必在图标和文本之间添加一个空格。

**图标类不能和其它组件直接联合使用**。它们不能在同一个元素上与其他类共同存在。应该创建一个嵌套的 <span> 标签，并将图标类应用到这个 <span> 标签上。

<span class="glyphicon glyphicon-star"></span> 或

<i class=*"glyphicon glyphicon-search"*></i>

只对内容为空的元素起作用

图标类只能应用在不包含任何文本内容或子元素的元素上。

图标的可访问性

现代的辅助技术能够识别并朗读由 CSS 生成的内容和特定的 Unicode 字符。为了避免 屏幕识读设备抓取非故意的和可能产生混淆的输出内容（尤其是当图标纯粹作为装饰用途时），我们为这些图标设置了 aria-hidden="true" 属性。

如果你使用图标是为了表达某些含义（不仅仅是为了装饰用），请确保你所要表达的意思能够通过被辅助设备识别，例如，包含额外的内容并通过 .sr-only 类让其在视觉上表现出隐藏的效果。

如果你所创建的组件不包含任何文本内容（例如， <button> 内只包含了一个图标），你应当提供其他的内容来表示这个控件的意图，这样就能让使用辅助设备的用户知道其作用了。这种情况下，你可以为控件添加 aria-label 属相。

<span class="glyphicon glyphicon-search" aria-hidden="true"></span>

<button type="button" class="btn btn-default btn-lg">

<span class="glyphicon glyphicon-star" aria-hidden="true"></span> Star

</button>

alert 组件中所包含的图标是用来表示这是一条错误消息的，通过添加额外的 .sr-only 文本就可以让辅助设备知道这条提示所要表达的意思了。

<div class="alert alert-danger" role="alert">

<span class="glyphicon glyphicon-exclamation-sign" aria-hidden="true"></span>

<span class="sr-only">Error:</span>

Enter a valid email address

</div>



* 下拉菜单

实例

用于显示链接列表的可切换、有上下文的菜单。下拉菜单的 JavaScript 插件让它具有了交互性。

将下拉菜单触发器和下拉菜单都包裹在 .dropdown 里，或者另一个声明了 position: relative; 的元素。然后加入组成菜单的 HTML 代码。（通过为下拉菜单的父元素设置 .dropup 类，可以让菜单向上弹出（默认是向下弹出的））

<div class="dropdown">

<button class="btn btn-default dropdown-toggle" type="button" id="dropdownMenu1" data-toggle="dropdown" aria-haspopup="true" aria-expanded="true">

Dropdown

<span class="caret"></span>

</button>

<ul class="dropdown-menu" aria-labelledby="dropdownMenu1">

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

<li><a href="#">Another action</a></li>

<li><a href="#">Something else here</a></li>

<li><a href="#">Separated link</a></li>

</ul>

</div>



对齐

默认情况下，下拉菜单自动沿着父元素的上沿和左侧被定位为 100% 宽度。 为 .dropdown-menu 添加 .dropdown-menu-right 类可以让菜单右对齐。

<ul class="dropdown-menu dropdown-menu-right" aria-labelledby="dLabel">

...

</ul>

标题

在任何下拉菜单中均可通过添加标题来标明一组动作。

<ul class="dropdown-menu" aria-labelledby="dropdownMenu3">

...

<li class="dropdown-header">Dropdown header</li>

...

</ul>



分割线

为下拉菜单添加一条分割线，用于将多个链接分组。

<ul class="dropdown-menu" aria-labelledby="dropdownMenuDivider">

...

<li **role="separator" class="divider"**></li>

...

</ul>



禁用的菜单项

为下拉菜单中的 <li> 元素添加 .disabled 类，从而禁用相应的菜单项。

<ul class="dropdown-menu" aria-labelledby="dropdownMenu4">

<li><a href="#">Regular link</a></li>

<li class="disabled"><a href="#">Disabled link</a></li>

<li><a href="#">Another link</a></li>

</ul>



* 按钮组

按钮工具栏

通过按钮组容器把一组按钮放在同一行里

确保设置正确的 role 属性并提供一个 label 标签

为了向使用辅助技术 - 如屏幕阅读器 - 的用户正确传达一正确的按钮分组，需要提供一个合适的 role 属性。对于按钮组合，应该是 role="group"，对于toolbar（工具栏）应该是 role="toolbar"。

<div class="btn-group" role="group" aria-label="...">

<button type="button" class="btn btn-default">Left</button>

<button type="button" class="btn btn-default">Middle</button>

<button type="button" class="btn btn-default">Right</button>

</div>



<div **class="btn-toolbar" role="toolbar"** aria-label="...">

<div class="btn-group" role="group" aria-label="...">...</div>

<div class="btn-group" role="group" aria-label="...">...</div>

<div class="btn-group" role="group" aria-label="...">...</div>

</div>



尺寸

只要给 .btn-group 加上 .btn-group-\* 类，就省去为按钮组中的每个按钮都赋予尺寸类了，如果包含了多个按钮组时也适用。

嵌套

想要把下拉菜单混合到一系列按钮中，只须把 .btn-group 放入另一个 .btn-group 中。

<div class="btn-group" role="group" aria-label="...">

<button type="button" class="btn btn-default">1</button>

<button type="button" class="btn btn-default">2</button>

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

<button type="button" class="btn btn-default dropdown-toggle" data-toggle="dropdown" aria-haspopup="true" aria-expanded="false">

Dropdown

<span class="caret"></span>

</button>

<ul class="dropdown-menu">

<li><a href="#">Dropdown link</a></li>

<li><a href="#">Dropdown link</a></li>

</ul>

</div>

</div>



* 按钮式下拉菜单

把任意一个按钮放入 .btn-group 中，然后加入适当的菜单标签，就可以让按钮作为菜单的触发器了。

单按钮下拉菜单

只要改变一些基本的标记，就能把按钮变成下拉菜单的开关。

<!-- Single button -->

<div class="btn-group">

<button type="button" class="btn btn-default dropdown-toggle" data-toggle="dropdown" aria-haspopup="true" aria-expanded="false">

Action <span class="caret"></span>

</button>

<ul class="dropdown-menu">

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

<li><a href="#">Another action</a></li>

<li><a href="#">Something else here</a></li>

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

<li><a href="#">Separated link</a></li>

</ul>

</div>



分裂式按钮下拉菜单

相似地，分裂式按钮下拉菜单也需要同样的改变一些标记，但只是多一个分开的按钮

<!-- Split button -->

<div class="btn-group">

<button type="button" class="btn btn-danger">Action</button>

<button type="button" class="btn btn-danger dropdown-toggle" data-toggle="dropdown" aria-haspopup="true" aria-expanded="false">

<span class="caret"></span>

<span class="sr-only">Toggle Dropdown</span>

</button>

<ul class="dropdown-menu">

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

<li><a href="#">Another action</a></li>

<li><a href="#">Something else here</a></li>

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

<li><a href="#">Separated link</a></li>

</ul>

</div>



* 输入框组

通过在文本输入框 <input> 前面、后面或是两边加上文字或按钮，可以实现对表单控件的扩展。**为 .input-group 赋予 .input-group-addon 类，可以给 .form-control 的前面或后面添加额外的元素。**

在输入框的任意一侧添加额外元素或按钮。你还可以在输入框的两侧同时添加额外元素。我们不支持在输入框的单独一侧添加多个额外元素。我们不支持在单个输入框组中添加多个表单控件。

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

**<span class="input-group-addon">$</span>**

<input type="text" class="form-control" aria-label="Amount (to the nearest dollar)">

**<span class="input-group-addon">.00</span>**

</div>

尺寸

为 .input-group 添加相应的尺寸类，其内部包含的元素将自动调整自身的尺寸。不需要为输入框组中的每个元素重复地添加控制尺寸的类。

**为输入框组添加按钮需要额外添加一层嵌套，不是 .input-group-addon，而是添加 .input-group-btn 来包裹按钮元素**。由于不同浏览器的默认样式无法被统一的重新赋值，所以才需要这样做。

<div class="input-group">

<input type="text" class="form-control" placeholder="Search for...">

<span class="input-group-btn">

<button class="btn btn-default" type="button">Go!</button>

</span>

</div><!-- /input-group -->



* 导航

标签页

Bootstrap 中的导航组件都依赖同一个 .nav 类，状态类也是共用的。改变修饰类可以改变样式。

确保导航组件的可访问性

如果你在使用导航组件实现导航条功能，务必在 <ul> 的最外侧的逻辑父元素上添加 role="navigation" 属性，或者用一个 <nav> 元素包裹整个导航组件。不要将 role 属性添加到 <ul> 上，因为这样可以被辅助设备（残疾人用的）上被识别为一个真正的列表。

标签页

注意 .nav-tabs 类依赖 .nav 基类。

<ul class="nav nav-tabs">

<li role="presentation" class="active"><a href="#">Home</a></li>

<li role="presentation"><a href="#">Profile</a></li>

<li role="presentation"><a href="#">Messages</a></li>

</ul>



胶囊式标签页

HTML 标记相同，但使用 .nav-pills 类：

<ul **class="nav nav-pills"**>

<li **role="presentation" class="active"**><a href="#">Home</a></li>

<li role="presentation"><a href="#">Profile</a></li>

<li role="presentation"><a href="#">Messages</a></li>

</ul>



胶囊是标签页也是可以垂直方向堆叠排列的。只需添加 .nav-stacked 类。

<ul class="nav nav-pills nav-stacked">

...

</ul>



添加下拉菜单

<ul class="nav nav-tabs">

...

<li role="presentation" class="dropdown">

<a class="dropdown-toggle" data-toggle="dropdown" href="#" role="button" aria-haspopup="true" aria-expanded="false">

Dropdown <span class="caret"></span>

</a>

<ul class="dropdown-menu">

...

</ul>

</li>

...

</ul>



* 导航条

默认样式的导航条

导航条是在您的应用或网站中作为导航页头的响应式基础组件。它们在移动设备上可以折叠（并且可开可关），且在视口（viewport）宽度增加时逐渐变为水平展开模式。

导航条内所包含元素溢出

由于 Bootstrap 并不知道你在导航条内放置的元素需要占据多宽的空间，你可能会遇到导航条中的内容折行的情况（也就是导航条占据两行）。解决办法如下：

减少导航条内所有元素所占据的宽度。

在某些尺寸的屏幕上（利用 响应式工具类）隐藏导航条内的一些元素。

修改导航条在水平排列和折叠排列互相转化时，触发这个转化的最小屏幕宽度值。可以通过修改 @grid-float-breakpoint 变量实现，或者自己重写相关的媒体查询代码，覆盖 Bootstrap 的默认值。

导航条的可访问性

务必使用 <nav> 元素，或者，如果使用的是通用的 <div> 元素的话，务必为导航条设置 role="navigation" 属性，这样能够让使用辅助设备的用户明确知道这是一个导航区域。

**<nav class="navbar navbar-default">**

<div class="container-fluid">

**<div class="navbar-header">**

<button type="button" **class="navbar-toggle collapsed" data-toggle="collapse" data-target="#bs-example-navbar-collapse-1"** aria-expanded="false">

**<span class="sr-only">Toggle navigation</span>**

**<span class="icon-bar"></span>**

**<span class="icon-bar"></span>**

**<span class="icon-bar"></span>**

</button>

<a class="navbar-brand" href="#">Brand</a>

</div>

**<div class="collapse navbar-collapse" id="bs-example-navbar-collapse-1">**

**<ul class="nav navbar-nav">**

<li class="active"><a href="#">Link <span class="sr-only">(current)</span></a></li>

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

**<li class="dropdown">**

**<a href="#" class="dropdown-toggle" data-toggle="dropdown"** role="button" aria-haspopup="true" aria-expanded="false">Dropdown <span class="caret"></span></a>

**<ul class="dropdown-menu">**

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

...

</ul>

</li>

</ul>



表单

将表单放置于 .navbar-form 之内可以呈现很好的垂直对齐，并在较窄的视口（viewport）中呈现折叠状态。 使用对齐选项可以规定其在导航条上出现的位置。

注意，.navbar-form 和 .form-inline 的大部分代码都一样，内部实现使用了 mixin。 某些表单组件，例如输入框组，可能需要设置一个固定宽度，从而在导航条内有合适的展现。

<form **class="navbar-form navbar-left" role="search"**>

<div class="form-group">

<input type="text" class="form-control" placeholder="Search">

</div>

<button type="submit" class="btn btn-default">Submit</button>

</form>

按钮

对于不包含在 <form> 中的 <button> 元素，加上 .navbar-btn 后，可以让它在导航条里垂直居中。有一些对于为辅助设备提供可识别标签的方法，例如， aria-label、aria-labelledby 或者 title 属性。如果这些方法都没有，屏幕阅读器将使用 placeholder 属性（如果这个属性存在的话），但是请注意，使用 placeholder 代替其他识别标签的方式是不推荐的。

<button type="button" class="btn btn-default navbar-btn">Sign in</button>



文本

把文本包裹在 .navbar-text中时，为了有正确的行距和颜色，通常使用 <p> 标签。

<p class="navbar-text">Signed in as Mark Otto</p>



非导航的链接

或许你希望在标准的导航组件之外添加标准链接，那么，使用 .navbar-link 类可以让链接有正确的默认颜色和反色设置。

<p class="navbar-text navbar-right">Signed in as <a href="#" class="navbar-link">Mark Otto</a></p>



组件排列

通过添加 .navbar-left 和 .navbar-right 工具类让导航链接、表单、按钮或文本对齐。两个类都会通过 CSS 设置特定方向的浮动样式。例如，要对齐导航链接，就要把它们放在个分开的、应用了工具类的 <ul> 标签里。

向右侧对齐多个组件

导航条目前不支持多个 .navbar-right 类。

固定在顶部

添加 .navbar-fixed-top 类可以让导航条固定在顶部，还可包含一个 .container 或 .container-fluid 容器，从而让导航条居中，并在两侧添加内补（padding）。

需要为 body 元素设置内补（padding）

p>这个固定的导航条会遮住页面上的其它内容，除非你给 <body> 元素底部设置了 padding。用你自己的值，或用下面给出的代码都可以。提示：导航条的默认高度是 50px。

body { padding-top: 70px; }

固定在底部

添加 .navbar-fixed-bottom 类可以让导航条固定在底部，并且还可以包含一个 .container 或 .container-fluid 容器，从而让导航条居中，并在两侧添加内补（padding）。

反色的导航条

通过添加 .navbar-inverse 类可以改变导航条的外观。/p>

<nav **class="navbar navbar-inverse"**>

...

</nav>



路径导航

在一个带有层次的导航结构中标明当前页面的位置。

各路径间的分隔符已经自动通过 CSS 的 :before 和 content 属性添加了。

<ol class="breadcrumb">

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

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

<li class="active">Data</li>

</ol>



* 分页

默认分页

受 Rdio 的启发，我们提供了这个简单的分页组件，用在应用或搜索结果中超级棒。组件中的每个部分都很大，有点事容易点击、易缩放、点击区域大。

<nav>

<ul class="pagination">

<li>

<a href="#" aria-label="Previous">

<span aria-hidden="true">&laquo;</span>

</a>

</li>

<li><a href="#">1</a></li>

<li><a href="#">2</a></li>

<li><a href="#">3</a></li>

<li><a href="#">4</a></li>

<li><a href="#">5</a></li>

<li>

<a href="#" aria-label="Next">

<span aria-hidden="true">&raquo;</span>

</a>

</li>

</ul>

</nav>



禁用和激活状态

链接在不同情况下可以定制。你可以给不能点击的链接添加 .disabled 类、给当前页添加 .active 类。

<nav>

<ul class="pagination">

<li class="disabled"><a href="#" aria-label="Previous"><span aria-hidden="true">&laquo;</span></a></li>

<li class="active"><a href="#">1 <span class="sr-only">(current)</span></a></li>

...

</ul>

</nav>

尺寸

想要更小或更大的分页？.pagination-lg 或 .pagination-sm 类提供了额外可供选择的尺寸。

翻页

用简单的标记和样式，就能做个上一页和下一页的简单翻页。用在像博客和杂志这样的简单站点上棒极了。

默认实例

在默认的翻页中，链接居中对齐。

<nav>

<ul class="pager">

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

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

</ul>

</nav>



对齐链接

你还可以把链接向两端对齐：

<nav>

<ul class="pager">

<li class="previous"><a href="#"><span aria-hidden="true">&larr;</span> Older</a></li>

<li class="next"><a href="#">Newer <span aria-hidden="true">&rarr;</span></a></li>

</ul>

</nav>



* 徽章

给链接、导航等元素嵌套 <span class="badge"> 元素，可以很醒目的展示新的或未读的信息条目。

<a href="#">Inbox **<span class="badge">42</span>**</a>

<button class="btn btn-primary" type="button">

Messages <span class="badge">4</span>

</button>



Self collapsing

如果没有新的或未读的信息条目，也就是说不包含任何内容，徽章组件能够自动隐藏（通过CSS的 :empty 选择符实现)

适配导航元素的激活状态

Bootstrap 提供了内置的样式，让胶囊式导航内处于激活状态的元素所包含的徽章展示相匹配的样式。

<ul class="nav nav-pills" role="tablist">

<li role="presentation" class="active"><a href="#">Home <span class="badge">42</span></a></li>

<li role="presentation"><a href="#">Profile</a></li>

<li role="presentation"><a href="#">Messages <span class="badge">3</span></a></li>

</ul>



* 巨幕

这是一个轻量、灵活的组件，它能延伸至整个浏览器视口来展示网站上的关键内容。

<div class="jumbotron">

<h1>Hello, world!</h1>

<p>...</p>

<p><a class="btn btn-primary btn-lg" href="#" role="button">Learn more</a></p>

</div>



* 页头

页头组件能够为 h1 标签增加适当的空间，并且与页面的其他部分形成一定的分隔。它支持 h1 标签内内嵌 small 元素的默认效果，还支持大部分其他组件（需要增加一些额外的样式）。

<div class="page-header">

<h1>Example page header <small>Subtext for header</small></h1>

</div>

* 缩略图

通过缩略图组件扩展 Bootstrap 的 栅格系统，可以很容易地展示栅格样式的图像、视频、文本等内容。

如果你想实现一个类似 Pinterest 的页面效果（不同高度和/宽度的缩略图顺序排列）的话，你需要使用一个第三方插件，比如 Masonry、Isotope 或 Salvattore。

默认样式的实例

Boostrap 缩略图的默认设计仅需最少的标签就能展示带链接的图片。

<div class="row">

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

<a href="#" class="thumbnail">

<img src="..." alt="...">

</a>

</div>

...

</div>

自定义内容

添加一点点额外的标签，就可以把任何类型的 HTML 内容，例如标题、段落或按钮，加入缩略图组件内。

<div class="row">

<div class="col-sm-6 col-md-4">

<div **class="thumbnail"**>

<img src="..." alt="...">

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

<h3>Thumbnail label</h3>

<p>...</p>

<p>

<a href="#" **class="btn btn-primary" role="button"**>Button</a>

<a href="#" class="btn btn-default" role="button">Button</a>

</p>

</div>

</div>

</div>

</div>



警告框

警告框组件通过提供一些灵活的预定义消息，为常见的用户动作提供反馈消息。

实例

将任意文本和一个可选的关闭按钮组合在一起就能组成一个警告框，.alert 类是必须要设置的，另外我们还提供了有特殊意义的4个类（例如，.alert-success），代表不同的警告信息。

没有默认类

警告框没有默认类，只有基类和修饰类。默认的灰色警告框并没有多少意义。所以您要使用一种有意义的警告类。目前提供了成功、消息、警告或危险。

<div class="alert alert-success" role="alert">...</div>

<div class="alert alert-info" role="alert">...</div>

<div class="alert alert-warning" role="alert">...</div>

<div class="alert alert-danger" role="alert">...</div>

可关闭的警告框

为警告框添加一个可选的 .alert-dismissible 类和一个关闭按钮。

<div class="alert alert-warning alert-dismissible" role="alert">

<button type="button" class="close" data-dismiss="alert" aria-label="Close"><span aria-hidden="true">&times;</span></button>

<strong>Warning!</strong> Better check yourself, you're not looking too good.

</div>



进度条

带有提示标签的进度条

将设置了 .sr-only 类的 <span> 标签从进度条组件中移除类，从而让当前进度显示出来。

<div class="progress">

<div class="progress-bar" role="progressbar" aria-valuenow="60" aria-valuemin="0" aria-valuemax="100" style="width: 60%;">

60%

</div>

</div>



在展示很低的百分比时，如果需要让文本提示能够清晰可见，可以为进度条设置 min-width 属性。

<div class="progress">

<div class="progress-bar" role="progressbar" aria-valuenow="2" aria-valuemin="0" aria-valuemax="100" **style="min-width: 2em; width: 2%;"**>

2%

</div>

</div>



动画效果

为 .progress-bar-striped 添加 .active 类，使其呈现出由右向左运动的动画效果。

<div class="progress">

<div class="progress-bar progress-bar-striped active" role="progressbar" aria-valuenow="45" aria-valuemin="0" aria-valuemax="100" style="width: 45%">

<span class="sr-only">45% Complete</span>

</div>

</div>



堆叠效果

把多个进度条放入同一个 .progress 中，使它们呈现堆叠的效果。

<div class="progress">

<div class="progress-bar progress-bar-success" style="width: 35%">

<span class="sr-only">35% Complete (success)</span>

</div>

<div class="progress-bar progress-bar-warning progress-bar-striped" style="width: 20%">

<span class="sr-only">20% Complete (warning)</span>

</div>

<div class="progress-bar progress-bar-danger" style="width: 10%">

<span class="sr-only">10% Complete (danger)</span>

</div>

</div>



* 媒体对象

这是一个抽象的样式，用以构建不同类型的组件，这些组件都具有在文本内容的左或右侧对齐的图片（就像博客评论或 Twitter 消息等）。

默认样式

默认样式的媒体对象组件允许在一个内容块的左边或右边展示一个多媒体内容（图像、视频、音频）。

对齐

图片或其他媒体类型可以顶部、中部或底部对齐。默认是顶部对齐。

<div class="media">

<div **class="media-left media-middle"**>

<a href="#">

<img **class="media-object"** src="..." alt="...">

</a>

</div>

<div **class="media-body"**>

<h4 **class="media-heading"**>Middle aligned media</h4>

...

</div>

</div>



媒体对象列表

用一点点额外的标记，就能在列表内使用媒体对象组件（对评论或文章列表很有用）。

<ul class="media-list">

<li class="media">

<div class="media-left">

<a href="#">

<img class="media-object" src="..." alt="...">

</a>

</div>

<div class="media-body">

<h4 class="media-heading">Media heading</h4>

...

</div>

</li>

</ul>

* 列表组

列表组是灵活又强大的组件，不仅能用于显示一组简单的元素，还能用于复杂的定制的内容。

基本实例

最简单的列表组仅仅是一个带有多个列表条目的无序列表，另外还需要设置适当的类。我们提供了一些预定义的样式，你可以根据自身的需求通过 CSS 自己定制

<ul class="list-group">

<li class="list-group-item">Cras justo odio</li>

<li class="list-group-item">Dapibus ac facilisis in</li>

<li class="list-group-item">Morbi leo risus</li>

<li class="list-group-item">Porta ac consectetur ac</li>

<li class="list-group-item">Vestibulum at eros</li>

</ul>

徽章

**给列表组加入徽章组件，它会自动被放在右边。**

<ul class="list-group">

<li class="list-group-item">

<span class="badge">14</span>

Cras justo odio

</li>

</ul>



链接

用 <a> 标签代替 <li> 标签可以组成一个全部是链接的列表组（还要注意的是，我们需要将 <ul> 标签替换为 <div> 标签）。没必要给列表组中的每个元素都加一个父元素。

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

<**a href="#" class="list-group-item active"**>Cras justo odio</a>

<a href="#" class="list-group-item">Dapibus ac facilisis in</a>

<a href="#" class="list-group-item">Morbi leo risus</a>

<a href="#" class="list-group-item">Porta ac consectetur ac</a>

<a href="#" class="list-group-item">Vestibulum at eros</a>

</div>

按钮

列表组中的元素也可以直接就是按钮（也同时意味着父元素必须是 <div> 而不能用 <ul> 了），并且无需为每个按钮单独包裹一个父元素。注意不要使用标准的 .btn 类！

<div class="list-group">

<button type="button" class="list-group-item">Cras justo odio</button>

<button type="button" class="list-group-item">Dapibus ac facilisis in</button>

<button type="button" class="list-group-item">Morbi leo risus</button>

<button type="button" class="list-group-item">Porta ac consectetur ac</button>

<button type="button" class="list-group-item">Vestibulum at eros</button>

</div>



定制内容

**列表组中的每个元素都可以是任何 HTML 内容**，甚至是像下面的带链接的列表组。

<div class="list-group">

<a href="#" class="list-group-item active">

<h4 class="list-group-item-heading">List group item heading</h4>

<p class="list-group-item-text">...</p>

</a>

</div>



* 面版

虽然不总是必须，但是某些时候你可能需要将某些 DOM 内容放到一个盒子里。对于这种情况，可以试试面板组件。

基本实例

默认的 .panel 组件所做的只是设置基本的边框（border）和内补（padding）来包含内容。

<div **class="panel panel-default"**>

<div **class="panel-heading"**>Panel heading without title</div>

<div **class="panel-body"**>

Panel content

</div>

<div **class="panel-footer"**>Panel footer</div>

</div>

情境效果

像其他组件一样，可以简单地通过加入有情境效果的状态类，给特定的内容使用更针对特定情境的面版。

<div class="panel panel-primary">...</div>

<div class="panel panel-success">...</div>

<div class="panel panel-info">...</div>

<div class="panel panel-warning">...</div>

<div class="panel panel-danger">...</div>

带表格的面版

为面板中不需要边框的表格添加 .table 类，是整个面板看上去更像是一个整体设计。如果是带有 .panel-body 的面板，我们为表格的上方添加一个边框，看上去有分隔效果。如果没有 .panel-body ，面版标题会和表格连接起来，没有空隙。

<div class="panel panel-default">

<!-- Default panel contents -->

<div class="panel-heading">Panel heading</div>

<div class="panel-body">

<p>...</p>

</div>

<!-- Table -->

<table class="table">

...

</table>

</div>



* Well

默认效果:把 Well 用在元素上，能有嵌入（inset）的的简单效果。

<div class="well">...</div>

<div class="well well-lg">...</div>

<div class="well well-sm">...</div>

## JavaScript插件

* 概览

jQuery 插件为 Bootstrap 的组件赋予了“生命”。可以简单地一次性引入所有插件，或者逐个引入到你的页面中。

data 属性

你可以仅仅通过 data 属性 API 就能使用所有的 Bootstrap 插件，无需写一行 JavaScript 代码。这是 Bootstrap 中的一等 API，也应该是你的首选方式。

编程方式的 API

我们为所有 Bootstrap 插件提供了纯 JavaScript 方式的 API。**所有公开的 API 都是支持单独或链式调用方式，并且返回其所操作的元素集合（注：和jQuery的调用形式一致）**。

$('.btn.danger').button('toggle').addClass('fat')

所有方法都可以接受一个可选的 option 对象作为参数，或者一个代表特定方法的字符串，或者什么也不提供（在这种情况下，插件将会以默认值初始化）：

$('#myModal').modal() // 以默认值初始化

$('#myModal').modal({ keyboard: false }) // initialized with no keyboard

$('#myModal').modal('show') // 初始化后立即调用 show 方法

每个插件还通过 Constructor 属性暴露了其原始的构造函数：$.fn.popover.Constructor。如果你想获取某个插件的实例，可以直接通过页面元素获取：$('[rel="popover"]').data('popover')。

默认设置

每个插件都可以通过修改其自身的 Constructor.DEFAULTS 对象从而改变插件的默认设置：

复制

$.fn.modal.Constructor.DEFAULTS.keyboard = false // 将模态框插件的 `keyboard` 默认选参数置为 false

避免命名空间冲突

某些时候可能需要将 Bootstrap 插件与其他 UI 框架共同使用。在这种情况下，命名空间冲突随时可能发生。如果不幸发生了这种情况，你可以通过调用插件的 .noConflict 方法恢复其原始值。

复制

var bootstrapButton = $.fn.button.noConflict() // return $.fn.button to previously assigned value

$.fn.bootstrapBtn = bootstrapButton // give $().bootstrapBtn the Bootstrap functionality

事件

**Bootstrap 为大部分插件所具有的动作提供了自定义事件**。一般来说，这些事件都有不定式和过去式两种动词的命名形式，例如，不定式形式的动词（例如 show）表示其在事件开始时被触发；而过去式动词（例如 shown ）表示在动作执行完毕之后被触发。

**从 3.0.0 版本开始，所有 Bootstrap 事件的名称都采用命名空间方式**。

所有以不定式形式的动词命名的事件都提供了 preventDefault 功能。这就赋予你在动作开始执行前将其停止的能力。

复制

$('#myModal').on('show.bs.modal', function (e) {

if (!data) return e.preventDefault() // 阻止模态框的展示

})

* 过渡效果 transition.js

关于过渡效果

对于简单的过渡效果，只需将 transition.js 和其它 JS 文件一起引入即可。如果你使用的是编译（或压缩）版的 bootstrap.js 文件，就无需再单独将其引入了。

包含的内容

Transition.js 是针对 transitionEnd 事件的一个基本辅助工具，也是对 CSS 过渡效果的模拟。它被其它插件用来检测当前浏览器对是否支持 CSS 的过渡效果。

禁用过度效果

通过下面的 JavaScript 代码可以在全局范围禁用过渡效果，并且必须将此代码放在 transition.js （或 bootstrap.js 或 bootstrap.min.js）后面，确保在 js 文件加载完毕后再执行下面的代码：

复制

$.support.transition = false

* 模态框 modal.js

模态框经过了优化，更加灵活，以弹出对话框的形式出现，具有最小和最实用的功能集。

不支持同时打开多个模态框

千万不要在一个模态框上重叠另一个模态框。要想同时支持多个模态框，需要自己写额外的代码来实现。

模态框的 HTML 代码放置的位置

务必将模态框的 HTML 代码放在文档的最高层级内（也就是说，尽量作为 body 标签的直接子元素），以避免其他组件影响模态框的展现和/或功能。

$('#myModal').on('shown.bs.modal', function () {

$('#myInput').focus()

})

静态实例

以下模态框包含了模态框的头、体和一组放置于底部的按钮。

增强模态框的可访问性

务必为 .modal 添加 role="dialog" 和 aria-labelledby="..." 属性，用于指向模态框的标题栏；为 .modal-dialog 添加 aria-hidden="true" 属性。

另外，你还应该通过 aria-describedby 属性为模态框 .modal 添加描述性信息。

<!-- Button trigger modal -->

<button type="button" class="btn btn-primary btn-lg" **data-toggle="modal" data-target="#myModal"**>Launch demo modal</button>

<!-- Modal -->

<div **class="modal fade"** **id="myModal"** tabindex="-1" **role="dialog"** aria-labelledby="myModalLabel">

<div **class="modal-dialog"** role="document">

<div **class="modal-content"**>

<div **class="modal-header"**>

<button type="button" class="close" data-dismiss="modal" aria-label="Close"><span aria-hidden="true">&times;</span></button>

<h4 class="modal-title" id="myModalLabel">Modal title</h4>

</div>

<div **class="modal-body"**>

...

</div>

<div **class="modal-footer"**>

<button type="button" class="btn btn-default" data-dismiss="modal">Close</button>

<button type="button" class="btn btn-primary">Save changes</button>

</div>

</div>

</div>

</div>



通过 data 属性

不需写 JavaScript 代码也可激活模态框。通过在一个起控制器作用的元素（例如：按钮）上添加 data-toggle="modal" 属性，或者 data-target="#foo" 属性，再或者 href="#foo" 属性，用于指向被控制的模态框。

<button type="button" data-toggle="modal" data-target="#myModal">Launch modal</button>

通过 JavaScript 调用

只需一行 JavaScript 代码，即可通过元素的 id myModal 调用模态框：

$('#myModal').modal(options)

参数

可以将选项通过 data 属性或 JavaScript 代码传递。对于 data 属性，需要将参数名称放到 data- 之后，例如 data-backdrop=""。



$('#myModal').modal('toggle')； //手动打开或关闭模态框

$('#myModal').modal('show');

$('#myModal').modal('hide');

事件

Bootstrap 的模态框类提供了一些事件用于监听并执行你自己的代码。

$('#myModal').on('hidden.bs.modal', function (e) {

// do something...

})



* Dropdowns dropdown.js

Add dropdown menus to nearly anything with this simple plugin, including the navbar, tabs, and pills.

Note: The data-toggle="dropdown" attribute is relied on for closing dropdown menus at an application level, so it's a good idea to always use it.

<div **class="dropdown"**>

<button id="dLabel" type="button" **data-toggle="dropdown"** aria-haspopup="true" aria-expanded="false">

Dropdown trigger

<span class="caret"></span>

</button>

<ul **class="dropdown-menu"** aria-labelledby="dLabel">

...

</ul>

</div>

* 滚动监听 scrollspy.js

导航条实例

滚动监听插件是用来根据滚动条所处的位置来自动更新导航项的。

Resolvable ID targets required

Navbar links must have resolvable id targets. For example, a <a href="#home">home</a> must correspond to something in the DOM like <div id="home"></div>.

* Togglable tabs tab.js

Add quick, dynamic tab functionality to transition through panes of local content, even via dropdown menus. Nested tabs are not supported.

Enable tabbable tabs via JavaScript (each tab needs to be activated individually):

$('#myTabs a').click(function (e) {

e.preventDefault()

$(this).tab('show')

})

Markup

You can activate a tab or pill navigation without writing any JavaScript by simply specifying data-toggle="tab" or data-toggle="pill" on an element. Adding the nav and nav-tabs classes to the tab ul will apply the Bootstrap tab styling, while adding the nav and nav-pills classes will apply pill styling.

<div>

<!-- Nav tabs -->

<ul **class="nav nav-tabs" role="tablist"**>

<li role="presentation" class="active"><a **href="#home" aria-controls="home" role="tab" data-toggle="tab"**>Home</a></li>

<li role="presentation"><a href="#profile" aria-controls="profile" role="tab" data-toggle="tab">Profile</a></li>

<li role="presentation"><a href="#messages" aria-controls="messages" role="tab" data-toggle="tab">Messages</a></li>

<li role="presentation"><a href="#settings" aria-controls="settings" role="tab" data-toggle="tab">Settings</a></li>

</ul>

<!-- Tab panes -->

<div **class="tab-content"**>

<div **role="tabpanel" class="tab-pane active" id="home"**>...</div>

<div role="tabpanel" class="tab-pane" id="profile">...</div>

<div role="tabpanel" class="tab-pane" id="messages">...</div>

<div role="tabpanel" class="tab-pane" id="settings">...</div>

</div>

</div>

Methods

$().tab

Activates a tab element and content container. Tab should have either a data-target or an href targeting a container node in the DOM. In the above examples, the tabs are the <a>s with data-toggle="tab" attributes.

.tab('show')

Selects the given tab and shows its associated content. Any other tab that was previously selected becomes unselected and its associated content is hidden.

$('#someTab').tab('show')

* Tooltips tooltip.js

Inspired by the excellent jQuery.tipsy plugin written by Jason Frame; Tooltips are an updated version, which don't rely on images, use CSS3 for animations, and data-attributes for local title storage.

Tooltips with zero-length titles are never displayed.

<button type="button" class="btn btn-default" **data-toggle="tooltip" data-placement="left" title="Tooltip on left"**>Tooltip on left</button>

<button type="button" class="btn btn-default" data-toggle="tooltip" data-placement="top" title="Tooltip on top">Tooltip on top</button>

<button type="button" class="btn btn-default" data-toggle="tooltip" data-placement="bottom" title="Tooltip on bottom">Tooltip on bottom</button>

<button type="button" class="btn btn-default" data-toggle="tooltip" data-placement="right" title="Tooltip on right">Tooltip on right</button>



$().tooltip(options); //Attaches a tooltip handler to an element collection.

$('#element').tooltip('show');

$('#element').tooltip('hide');

$('#element').tooltip('toggle');

$('#element').tooltip('destroy');

* 弹出框 popover.js

为任意元素添加一小块浮层，就像 iPad 上一样，用于存放非主要信息。

弹出框的标题和内容的长度都是零的话将永远不会被显示出来。

初始化

由于性能的原因，工具提示和弹出框的 data 编程接口（data api）是必须要手动初始化的。

$(function () {

$('[data-toggle="popover"]').popover()

})

<button type="button" class="btn btn-lg btn-danger" **data-toggle="popover" title="Popover title" data-content="And here's some amazing content. It's very engaging. Right?"**>点我弹出/隐藏弹出框</button>



$('#example').popover(options);

$('#element').popover('show');

$('#element').popover('hide');

$('#element').popover('toggle');

$('#element').popover('destroy');

* 警告信息 alert.js

通过此插件可以为警告信息添加点击并消失的功能。

当使用 .close 按钮时，它必须是 .alert-dismissible 的第一个子元素，并且在它之前不能有任何文本内容。

为关闭按钮添加 data-dismiss="alert" 属性就可以使其自动为警告框赋予关闭功能。关闭警告框也就是将其从 DOM 中删除。

<button type="button" **class="close" data-dismiss="alert" aria-label="Close"**>

<span aria-hidden="true">&times;</span>

</button>

为了让警告框在关闭时表现出动画效果，请确保为其添加了 .fade 和 .in 类。

方法

$().alert()

让警告框监听具有 data-dismiss="alert" 属性的后裔元素的点击（click）事件。（如果是通过 data 属性进行的初始化则无需使用）

$().alert('close')

关闭警告框并从 DOM 中将其删除。如果警告框被赋予了 .fade 和 .in 类，那么，警告框在淡出之后才会被删除。

* 按钮 button.js

按钮的功能很丰富。通过控制按钮的状态或创建一组按钮并形成一些新的组件，例如工具条。

状态

通过添加 data-loading-text="Loading..." 可以为按钮设置正在加载的状态。

Single toggle

Add data-toggle="button" to activate toggling on a single button.

Checkbox / Radio

Add data-toggle="buttons" to a .btn-group containing checkbox or radio inputs to enable toggling in their respective styles.

* Collapse collapse.js

Flexible plugin that utilizes a handful of classes for easy toggle behavior.

<a class="btn btn-primary" role="button" **data-toggle="collapse" href="#collapseExample" aria-expanded="false" aria-controls="collapseExample"**>

Link with href

</a>

<div **class="collapse" id="collapseExample"**>

<div class="well">

...

</div>

</div>





Accordion example

Extend the default collapse behavior to create an accordion with the panel component.

<div class="panel-group" id="accordion" role="tablist" aria-multiselectable="true">

<div class="panel panel-default">

<div class="panel-heading" role="tab" id="headingOne">

<h4 class="panel-title">

<a role="button" data-toggle="collapse" data-parent="#accordion" href="#collapseOne" aria-expanded="true" aria-controls="collapseOne">

Collapsible Group Item #1

</a>

</h4>

</div>

<div id="collapseOne" class="panel-collapse collapse in" role="tabpanel" aria-labelledby="headingOne">

<div class="panel-body">

...

</div>

</div>

</div>

<div class="panel panel-default">

<div class="panel-heading" role="tab" id="headingTwo">

<h4 class="panel-title">

<a class="collapsed" role="button" data-toggle="collapse" data-parent="#accordion" href="#collapseTwo" aria-expanded="false" aria-controls="collapseTwo">

Collapsible Group Item #2

</a>

</h4>

</div>

<div id="collapseTwo" class="panel-collapse collapse" role="tabpanel" aria-labelledby="headingTwo">

<div class="panel-body">

...

</div>

</div>

</div>

</div>



Usage

The collapse plugin utilizes a few classes to handle the heavy lifting:

.collapse hides the content

.collapse.in shows the content

.collapsing is added when the transition starts, and removed when it finishes

These classes can be found in component-animations.less.

Via data attributes

Just add data-toggle="collapse" and a data-target to the element to automatically assign control of a collapsible element. The data-target attribute accepts a CSS selector to apply the collapse to. Be sure to add the class collapse to the collapsible element. If you'd like it to default open, add the additional class in.

To add accordion-like group management to a collapsible control, add the data attribute data-parent="#selector". Refer to the demo to see this in action.

Via JavaScript

Enable manually with:

$('.collapse').collapse()

.collapse('toggle'); Toggles a collapsible element to shown or hidden.

.collapse('show'); //Shows a collapsible element.

.collapse('hide'); //Hides a collapsible element.

* Carousel carousel.js

A slideshow component for cycling through elements, like a carousel. Nested carousels are not supported.

<div id="carousel-example-generic" class="carousel slide" data-ride="carousel">

**<!-- Indicators -->**

<ol class="carousel-indicators">

<li data-target="#carousel-example-generic" data-slide-to="0" class="active"></li>

<li data-target="#carousel-example-generic" data-slide-to="1"></li>

<li data-target="#carousel-example-generic" data-slide-to="2"></li>

</ol>

**<!-- Wrapper for slides -->**

<div class="carousel-inner" role="listbox">

<div class="item active">

<img src="..." alt="...">

<div class="carousel-caption">

...

</div>

</div>

<div class="item">

<img src="..." alt="...">

<div class="carousel-caption">

...

</div>

</div>

...

</div>

**<!-- Controls -->**

<a class="left carousel-control" href="#carousel-example-generic" role="button" data-slide="prev">

<span class="glyphicon glyphicon-chevron-left" aria-hidden="true"></span>

<span class="sr-only">Previous</span>

</a>

<a class="right carousel-control" href="#carousel-example-generic" role="button" data-slide="next">

<span class="glyphicon glyphicon-chevron-right" aria-hidden="true"></span>

<span class="sr-only">Next</span>

</a>

</div>



<Bootstrap实战>

HTML5 Boilerplate样板文件

.htaccess文件：主要作用是保证站点性能最优

基于HTML最佳实践，考虑ARIA(Accessible Rich Internet Appliations, 可访问富因特网应用)的role属性（即banner,navigation,main,contentinfo）

响应式导航条

<div class=”navbar-header”>

<button type=”button” **class=”navbar-toggle” data-toggle=”collapse” data-target=”.navbar-collapse”**>

<span class=”icon-bar”></span>

<span class=”icon-bar”></span>

<span class=”icon-bar”></span>

</button>

<a class=”navbar-brand” href=”index.html”>Project name</a>

</div>

navbar-toggle类用于应用CSS样式

数据属性data-toggle and data-target是Bootstrap的JavaScript插件要用的，分别表示预期行为和预期目标

类名为Icon-bar的span元素是CSS用来创建按钮中的三道杠按钮用的

<div **class=”navbar-collpase collapse”**>

<ul class=”nav navbar-nav”>

<li class=”active”><a href=”index.html”>Home</a></li>

<li ><a href=”#”>Link</a></li>

</ul>

</div>

导航项包装在一个收起的div中，注意目标是navbar-collpase，行为是collapse

把链接变成按钮

btn类用于把链接变成按钮的样式

btn-primary类用于把按钮变成主品牌颜色

pull-right类用于把链接浮动到右侧，使其占据更大的空间

<p><a class=”btn btn-primary pull-right” href=”#”>See our portfollo</a></p>

让banner-brand只在中、大型视口显示，而让navbar-brand只在小和超小型视口中显示

<a class=”banner-brand visible-md visible-lg” href=”index.html”>…</a>

<a class=”navbar-brand visible-xs visible-sm” href=”index.html”>…</a>

将导航栏收缩按钮从右边放到左边

把浮动值由right改为left,把margin-right改为margin-left

.navbar-toggle {

position: relative;

float:left;

margin-left: @navbar-padding-horontal;

}

小屏幕只显示大图标，大屏幕显示图标+文字

<li><a href=”#”>

<i class=”icon fa fa-shopping-cart fa-lg”></i>

<span>Log In or Register</span>

</a>

</li>

@media (max-width: @grid-float-breakpoint – 1) {

span { .sr-only(); } //对除屏幕阅读器这外的所有设备隐藏文本

.icon { //增大图标

font-size: 2em;

line-height: 1.2;

}

}

<section class=”col-sm-4 col-md-5”>

类col-sm-4表示当前栏是父元素宽度的三分之一，从小视口(764px)及以上宽度开始

类col-md-5表示当前栏是父元素宽度的5/12, 从中型视口及以上宽度开始

<div class=”clearfix visible-sm”></div>

类clearfix会强制当前元素清除上方的浮动。而类visible-sm则控制这个div仅在小屏幕，也就是我们指定的断点范围内可见。

将列表转化为面包屑

Home / Parent Category / Current Category

<ul class=“breadcrumb”>

<li><a href=”#”>Home</a></li>

<li><a href=”#”>Parent Category</a></li>

<li class=”active”>Current Category</li>

</ul>

分布链接 （将分页导航在网格下方居中）

首把，把它包围在一个div标签中，给这个标签一个row类以保证它清除上方内容，然后添加一个恰如其分的自定义类pagination-wrap

<div class=”row pagination-wrap”>

<ul class=”pagination”>

<li><a href=”#”><span class=”fa fa-chevron-left”></span>Prev</a></li>

<li><a href=”#”>1</a></li>

<li><a href=”#”>2</a></li>

<li><a href=”#”>3</a></li>

<li><a href=”#”>Prev<span class=”fa fa-chevron-right”></span></a></li>

</ul>

</div>

.pagination-wrap {

text-align: center;

}

若每个浮动元素高度不确定，左浮动时，浮动元素可能插入到前面的浮动元素中，解决问题的关键是找到最高的浮动元素

.product-item {

height:360px;

overflow:hidden;

}

使用less等于html中添加类标记

<ul class=”options-list row”>

<li class=”col-xs-6”><a href=”#”>Option 1</a></li>

</ul>

使用less代码实现同样的效果

.options-list {

.make-row() ;

}

li {

.make-xs-column(6) ;

}

<section id=”features”>

<div class=”container”>

<h1>Features</h1>

<div class=”row>

<div class=”features-item col-md-4”>

<span class=”icon fa fa-cloud”></span>

<h2>Feature 1</h2>

<p>Done …</p>

</div>

</div>

</div>

</section>

居中文本，添加内边距，并设定高度以避免浮动的功能项互相交错，同时将.icon字体增大为90px; 为.features-item设置最大宽度，同时居中

#features{

.features-item {

text-align: center;

padding: 20px;

height: 270px;

.icon { font-size: 90px; }

@media (max-width: @screen-xs-max) {

max-wdith: 320px;

.center-block();

}

}

}

用户评论区

<div class=”hreview review-item-1 thumbnail”>

<img src=”img/smiling1.jpg” alt=”Customer Photo1”>

<div class=”caption”>

<blockquote class=”description”><p>…</p></blockquote>

<p class=”reviewer”>Smiling customer</p>

</div>

</div>

.hreview{

position:relative;

padding:0 10px;

boder:none;

.caption {

position:absolute;

left:10px;

right:10px; 置于底部，并且拉伸离左右10px

top:auto;

bottom:0;

background: hsla(0, 0, 10%, 0.55) ; //半透明背景

}

.reviewer{

text-align: right; //置右

}

}

.caption也可以置于右上角，左上角等等

要实现图片墙效果，就得利用JavaScript计算可用空间，然后用最合适的图片去填充相应空间，最终让高度不同的块形成整齐的拼贴效果

下载Masonry插件

给div class=”reviews”添加js-masonry类，这是所有评价的父元素。这样插件就会知道要在哪里起作用。

然后在同一个元素上，添加一个数据属性，指定要拼贴的项

<div class=”reviews js-masory” data-masonry-options-‘{“itemSelector”:”.hreview”}’>

<thead>

<tr>

<th colspan=”..”>

<h2>..

<div>…

</th>

</tr>

</thead>

<tbody>…

</tody>

<tfoot>

</tfoot>

将某些字缩小，且置于align top

<em class=”price”<span>$</span>19</em>

.price {

span{

font-size: .5em;

vertical-align: super;

}

}

一行三列中，扩大当中列，缩小其余两列，且三列紧挨着

#signup .package-basic table,

#signup .package-pro table{

width: 90%; 减小其他栏的宽度

margin-top: 36px;

}

#signup .package-premium table{ //按比例增大第二栏的字体

thead th{

font-size: 1.5em;

h2 { font-size: 1.5em;}

}

}

//使用负的右外边距把左侧的列向右推，同时用等量的右外边距抵消它，以保持三列的相对位置不变

#signup .package-basic {

margin-right: -58px;

margin-left: 58px;

z-index: 1;

}

//使用负的左外边距把第三列向左推

#signup .package-pro{

margin-left: -30px;

z-index;1;

}

//调整所有列的z-index值，将中间一列置于所有列之上

#signup .package-premium{

z-index: 1000;

}

# React

## Why react?

think of React as the V in MVC.

building large applications with data that changes over time.

React is all about modular, composable components

HTML components are regular React components, just like the ones you define, with one difference. The JSX compiler will automatically rewrite HTML tags to React.createElement(tagName) expressions

### Using props

Data passed in from a parent component is available as a 'property' on the child component. These 'properties' are accessed through this.props.

By surrounding a JavaScript expression in braces inside JSX (as either an attribute or child), you can drop text or React components into the tree. We access named attributes passed to the component as keys on this.props and any nested elements as this.props.children.

### Reactive state

based on its props, each component has rendered itself once. props are immutable: they are passed from the parent and are "owned" by the parent. To implement interactions, we introduce mutable state to the component. this.state is private to the component and can be changed by calling this.setState(). When the state updates, the component re-renders itself.

render() methods are written declaratively as functions of this.props and this.state.

## Displaying Data: JSX

is a JavaScript syntax extension that looks similar to XML

React JSX transforms from an XML-like syntax into native JavaScript. XML elements, attributes and children are transformed into arguments that are passed to React.createElement.

var Nav;

// Input (JSX):

var app = <Nav color="blue" />;

// Output (JS):

var app = React.createElement(Nav, {color:"blue"});

namespaced components let you use one component that has other components as attributes:

var Form = MyFormComponent;

var App = (

<Form>

<Form.Row>

<Form.Label />

<Form.Input />

</Form.Row>

</Form>

);

JavaScript Expressions

### Attribute Expressions

var person = <Person name={window.isLoggedIn ? window.name : ''} />;

### Boolean Attributes

This often comes up when using HTML form elements, with attributes like disabled, required, checked and readOnly.

<input type="button" disabled />;

### Child Expressions

var content = <Container>{window.isLoggedIn ? <Nav /> : <Login />}</Container>;

## Interactivity and Dynamic UIs

### Components are Just State Machines

React thinks of UIs as simple state machines. By thinking of a UI as being in various states and rendering those states, it's easy to keep your UI consistent.

A common way to inform React of a data change is by calling setState(data, callback). This method merges data into this.state and re-renders the component. When the component finishes re-rendering, the optional callback is called.

Most of your components should simply take some data from props and render it.Try to keep as many of your components as possible stateless.

A common pattern is to create several stateless components that just render data, and have a stateful component above them in the hierarchy that passes its state to its children via props. The stateful component encapsulates all of the interaction logic, while the stateless components take care of rendering data in a declarative way.

State should contain data that a component's event handlers may change to trigger a UI update. In real apps this data tends to be very small and JSON-serializable. When building a stateful component, think about the minimal possible representation of its state, and only store those properties in this.state.

### Multiple Components

In React, an owner is the component that sets the props of other components. More formally, if a component X is created in component Y's render() method, it is said that X is owned by Y. As discussed earlier, a component cannot mutate its props — they are always consistent with what its owner sets them to. This fundamental invariant leads to UIs that are guaranteed to be consistent.

<Parent><Child /></Parent>

Parent can read its children by accessing the special this.props.children prop.

#### Dynamic Children

when the children are shuffled around (as in search results) or if new components are added onto the front of the list (as in streams). In these cases where the identity and state of each child must be maintained across render passes, you can uniquely identify each child by assigning it a key:

render: function() {

var results = this.props.results;

return (

<ol>

{results.map(function(result) {

return <li key={result.id}>{result.text}</li>;

})}

</ol>

);

}

### Data Flow

one-way data binding: owners bind their owned component's props to some value the owner has computed based on its props or state. Since this process happens recursively, data changes are automatically reflected everywhere they are used.

the bottleneck is almost always the DOM mutation and not JS execution. React will optimize this for you by using batching and change detection.

## Reusable Components

### Mixins

sometimes very different components may share some common functionality. These are sometimes called cross-cutting concerns. React provides mixins to solve this problem.

### ES6 Classes

export class Counter extends React.Component {

constructor(props) {

super(props);

this.state = {count: props.initialCount};

//We recommend that you bind your event handlers in the constructor so they are only bound once for every instance:

this.tick = this.tick.bind(this);

}

tick() {

this.setState({count: this.state.count + 1});

}

render() {

return (

<div onClick={this.tick}>

Clicks: {this.state.count}

</div>

);

}

}

Counter.propTypes = { initialCount: React.PropTypes.number };

Counter.defaultProps = { initialCount: 0 };

React.PropTypes exports a range of validators that can be used to make sure the data you receive is valid. When an invalid value is provided for a prop, a warning will be shown in the JavaScript console. Note that for performance reasons propTypes is only checked in development mode.

this.props.initialCount will have a value if it was not specified by the parent component. This allows you to safely just use your props without having to write repetitive and fragile code to handle that yourself.

### Stateless Functions

function HelloMessage(props) {

return <div>Hello {props.name}</div>;

}

ReactDOM.render(<HelloMessage name="Sebastian" />, mountNode);

//Or using the new ES6 arrow syntax:

const HelloMessage = (props) => <div>Hello {props.name}</div>;

ReactDOM.render(<HelloMessage name="Sebastian" />, mountNode);

In an ideal world, most of your components would be stateless functions because in the future we’ll also be able to make performance optimizations specific to these components by avoiding unnecessary checks and memory allocations. This is the recommended pattern,

### Transferring Props: A Shortcut

var CheckLink = React.createClass({

render: function() {

// This takes any props passed to CheckLink and copies them to <a>

return <a {...this.props}>{'√ '}{this.props.children}</a>;

}

});

ReactDOM.render(

<CheckLink href="/checked.html">

Click here!

</CheckLink>,

document.getElementById('example')

);

### destructuring

you can use destructuring assignment with rest properties to extract a set of unknown properties.

function FancyCheckbox(props) {

var { checked, ...other } = props;

var fancyClass = checked ? 'FancyChecked' : 'FancyUnchecked';

// `other` contains { onClick: console.log } but not the checked property

return (

<div {...other} className={fancyClass} />

);

}

ReactDOM.render(

<FancyCheckbox checked={true} onClick={console.log.bind(console)}>

Hello world!

</FancyCheckbox>,

document.getElementById('example')

);

Rest properties allow you to extract the remaining properties from an object into a new object. It excludes every other property listed in the destructuring pattern.

var { x, y, ...z } = { x: 1, y: 2, a: 3, b: 4 };

x; // 1

y; // 2

z; // { a: 3, b: 4 }

## Forms

Form components support a few props that are affected via user interactions:

value, supported by <input> and <textarea> components.

checked, supported by <input> components of type checkbox or radio.

selected, supported by <option> components.

Form components allow listening for changes by setting a callback to the onChange prop. The onChange prop works across browsers to fire in response to user interactions

Controlled Components

A Controlled component does not maintain its own internal state; the component renders purely based on props.

getInitialState: function() {

return {value: 'Hello!'};

},

handleChange: function(event) {

this.setState({value: event.target.value});

},

render: function() {

return (

<input

type="text"

value={this.state.value}

onChange={this.handleChange}

/>

);

}

## Working With the Browser

React provides powerful abstractions that free you from touching the DOM directly in most cases. React is very fast because it never talks to the DOM directly. React maintains a fast in-memory representation of the DOM. render() methods actually return a description of the DOM, and React can compare this description with the in-memory representation to compute the fastest way to update the browser.

Most of the time you should stay within React's "faked browser" world since it's more performant and easier to reason about. However, sometimes you simply need to access the underlying API, perhaps to work with a third-party library like a jQuery plugin. React provides escape hatches for you to use the underlying DOM API directly.

To interact with the browser, you'll need a reference to a DOM node. You can attach a ref to any element, which allows you to reference the backing instance of the component. This is useful if you need to invoke imperative functions on the component, or want to access the underlying DOM nodes

### Component Lifecycle

Components have three main parts of their lifecycle:

Mounting: A component is being inserted into the DOM.

getInitialState(): object is invoked before a component is mounted. Stateful components should implement this and return the initial state data.

componentWillMount() is invoked immediately before mounting occurs.

componentDidMount() is invoked immediately after mounting occurs. Initialization that requires DOM nodes should go here.

Updating: A component is being re-rendered to determine if the DOM should be updated.

Unmounting: A component is being removed from the DOM.

componentWillUnmount() is invoked immediately before a component is unmounted and destroyed. Cleanup should go here.

### Refs to Components

React provides an escape hatch known as refs. These refs (references) are especially useful when you need to: find the DOM markup rendered by a component (for instance, to position it absolutely), use React components in a larger non-React application, or transition your existing codebase to React.

ReactDOM.render() will return a reference to your component's backing instance (or null for stateless components).

var myComponent = ReactDOM.render(<MyComponent />, myContainer);

Keep in mind, however, that the JSX doesn't return a component instance! It's just a ReactElement: a lightweight representation that tells React what the mounted component should look like.

#### The ref Callback Attribute

The ref attribute can be a callback function, and this callback will be executed immediately after the component is mounted. The referenced component will be passed in as a parameter, and the callback function may use the component immediately, or save the reference for future use (or both).

render: function() {

return <TextInput ref={(c) => this.\_input = c} />;

},

componentDidMount: function() {

this.\_input.focus();

},

When attaching a ref to a DOM component like <div />, you get the DOM node back; when attaching a ref to a composite component like <TextInput />, you'll get the React class instance. In the latter case, you can call methods on that component if any are exposed in its class definition.

Also note that when writing refs with inline function expressions as in the examples here, React sees a different function object each time so on every update, ref will be called with null immediately before it's called with the component instance.

#### The ref String Attribute

<input ref="myInput" />

//In some other code (typically event handler code), access the backing instance via this.refs as in:

var input = this.refs.myInput;

var inputValue = input.value;

var inputRect = input.getBoundingClientRect();

Refs are a great way to send a message to a particular child instance in a way that would be inconvenient to do via streaming Reactive props and state. They should, however, not be your go-to abstraction for flowing data through your application. By default, use the Reactive data flow and save refs for use cases that are inherently non-reactive.

## Tools

<https://github.com/kriasoft/react-starter-kit>

>git clone –o react-starter-kit –b master –single-branch https://github.com/kriasoft/react-starter-kit.git ex

ex>npm install

ex>npm start

This command will build the app from the source files (/src) into the output /build folder. As soon as the initial build completes, it will start the Node.js server (node build/server.js) and Browsersync with HMR on top of it.

Whenever you modify any of the source files inside the /src folder, the module bundler (Webpack) will recompile the app on the fly and refresh all the connected browsers.

# RxJS <- Promises <- Callback

http://reactivex.io/rxjs/manual/overview.html#introduction

回调函数:对于多重异步，导致“末日金字塔”

Promises: 处理这种单值（或单个错误）时非常高效,没有事件的删除、分配、重试等等的语法定义

RxJS: 观察者模式 + 函数式编程

应用场景：处理嵌套回调的异步事件，复杂的列表过滤和变换，或者时间相关问题

响应式的思路：把随时间不断变化的数据、状态、事件等等转成可被观察的序列(Observable Sequence)，然后订阅序列中那些Observable对象的变化，一旦变化，就会执行事先安排好的各种转换和操作。

RxJS中解决异步事件管理的基本概念如下：

* Observable可观察对象：表示一个可调用的未来值或者事件的集合。
* Observer观察者：一个回调函数集合,它知道怎样去监听被Observable发送的值
* Subscription订阅： 表示一个可观察对象的执行，主要用于取消执行。
* Operators操作符： 纯粹的函数，使得以函数编程的方式处理集合比如:map,filter,contact,flatmap。
* Subject(主题)：等同于一个事件驱动器，是将一个值或者事件广播到多个观察者的唯一途径。
* Schedulers(调度者)： 用来控制并发，当计算发生的时候允许我们协调，比如setTimeout,requestAnimationFrame。

Observable，四个生命周期：创建 、订阅 、 执行 、销毁

1. Creating Observables
2. Subscribing to Observables
3. Executing the Observable
4. Disposing Observables

## 可观察数据类型

用于包装一个数据片段（按钮事件，键盘事件，鼠标事件，数字，字符串或者队列）

//转换为可观察对象

* of(arg) 一个或多个值->可观察对象
* from(iterable) 迭代的 -> 可观察对象
* fromPromise(promise) promise -> 可观察对象
* fromEvent(element, eventName) 事件->可观察对象

(Dom元素，jQuery元素，Zepto元素，Angular元素，Ember.js元素或者EventEmitter)

* var exists = Rx.Observable.bindCallback(callback) 回调函数->可观察对象

//创建可观察对象

外部产生新事件

var myObservable = new Rx.Subject();

myObservable.subscribe(value => console.log(value));

myObservable.next('foo');

内部产生新事件

var myObservable = Rx.Observable.create(observer => {

observer.next('foo');

setTimeout(() => observer.next('bar'), 1000);

});

myObservable.subscribe(value => console.log(value));

var observable = Rx.Observable.create((observer) => {

observer.next(1);

observer.next(2);

observer.next(3);

var intervalID = setInterval(() => {

observer.next('hi');

}, 1000);

// Provide a way of canceling and disposing the interval resource

return function unsubscribe() {

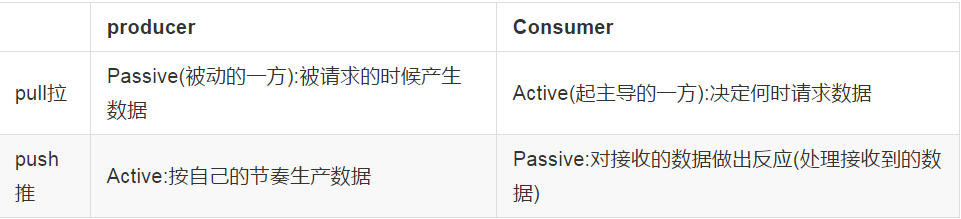
clearInterval(intervalID);

};

});

注意：当我们使用create()创建可观察对象，每一个可观察对象必须确定怎样去处理该执行的资源

Pull拉取 VS Push推送



Pull: 每一个JS函数,iterator迭代器和Generator生成器

Push: Promise, Observables

## 观察者(Observer)

观察者代表模型的消费者一端。观察者简单而言是一组回调函数 ， 分别对应一种被可观察对象发送的通知的类型:next, error和complete

var unsubscribe = observable.subscribe(**{**

**next: x => console.log('got value ' + x),**

**error: err => console.error('something wrong occurred: ' + err),**

**complete: () => console.log('done'),**

**}**);

unsubscribe();

也可以仅用一个函数作为参数，表示执行next

observable.subscribe(**x => console.log('got value ' + x)**);

## Subscription订阅

var subscription = Rx.Observable.interval(1000).subscribe(x => console.log(x));

subscription.unsubscribe();

//多个订阅进行取消。做法是:把一个订阅"加"进另一个订阅

var subscription = Rx.Observable.interval(400).subscribe(x => console.log('first: ' + x));

var childSubscription = Rx.Observable.interval(300).subscribe(x => console.log('second: ' + x));

subscription.add(childSubscription);

subscription.unsubscribe();

## Subject主题

* 每一个Subject都是一个observable可观察对象

Subject就是一个可观察对象，只不过可以被多播至多个观察者。同时Subject也类似于EventEmitter:维护者着众多事件监听器的注册表。

var subject = new Rx.Subject();

subject.subscribe({ next: (v) => console.log('observerA: ' + v) });

subject.subscribe({ next: (v) => console.log('observerB: ' + v) });

subject.next(1);

subject.next(2);

* 每一个Subject都是一个Observer观察者对象

Subject也是一个观察者，这就意味着你可以提供一个Subject当做observable.subscribe()的参数

//通过将一个单播的可观察对象转化为多播

var subject = new Rx.Subject();

subject.subscribe({ next: (v) => console.log('observerA: ' + v) });

subject.subscribe({ next: (v) => console.log('observerB: ' + v) });

var observable = Rx.Observable.from([1, 2, 3]);

observable.subscribe(subject); // You can subscribe providing a Subject

* 多播的可观察对象

var multicasted = Rx.Observable.from([1,2,3]).multicast(new Rx.Subject());

multicasted.subscribe({ next:(v)=>console.log('observerA:' +v); });

multicasted.subscribe({ next: (v) => console.log('observerB: ' + v) });

multicasted.connect(); //决定何时开始分享可观察对象的执行

* reference counting引用计数

在第一个观察者到达时自动的链接，并且在最后一个观察者取消订阅后自动的取消可观察对象的执行

* BehaviorSubject

它储存着要发射给消费者的最新的值。无论何时一个新的观察者订阅它，都会立即接受到这个来自BehaviorSubject的"当前值"。

BehaviorSubject对于表示"随时间的值"是很有用的。举个例子，人的生日的事件流是一个Subject,然而人的年龄的流是一个BehaviorSubject。

var subject = new Rx.BehaviorSubject(0); // 0 is the initial value

//接收当前值0和将来所有值1, 2

subject.subscribe({ next: (v) => console.log('observerA: ' + v) });

subject.next(1);

subject.next(2);

//接收当前值2和将来所有值3

subject.subscribe({ next: (v) => console.log('observerB: ' + v) });

subject.next(3);

* ReplaySubject

缓存值（以个数和逝去时间毫秒，来决定过去多久出现的值可以被重发）

var subject = new Rx.ReplaySubject(3, 500); //缓存了三个值, 指定窗500ms

//接收将来所有值1, 2, 3, 4, 5

subject.subscribe({ next: (v) => console.log('observerA: ' + v) });

subject.next(1);

subject.next(2);

subject.next(3);

subject.next(4);

//接收历史值2,3,4和将来所有值5

subject.subscribe({ next: (v) => console.log('observerB: ' + v) });

subject.next(5);

* AsyncSubject

只发送给观察者可观察对象执行的最新值，并且仅在执行结束时。

var subject = new Rx.AsyncSubject();

//仅接收complete()时的最新值5

subject.subscribe({ next: (v) => console.log('observerA: ' + v) });

subject.next(1);

subject.next(2);

subject.next(3);

subject.next(4);

//仅接收complete()时的最新值5

subject.subscribe({ next: (v) => console.log('observerB: ' + v) });

subject.next(5);

subject.complete();

## Operators操作符

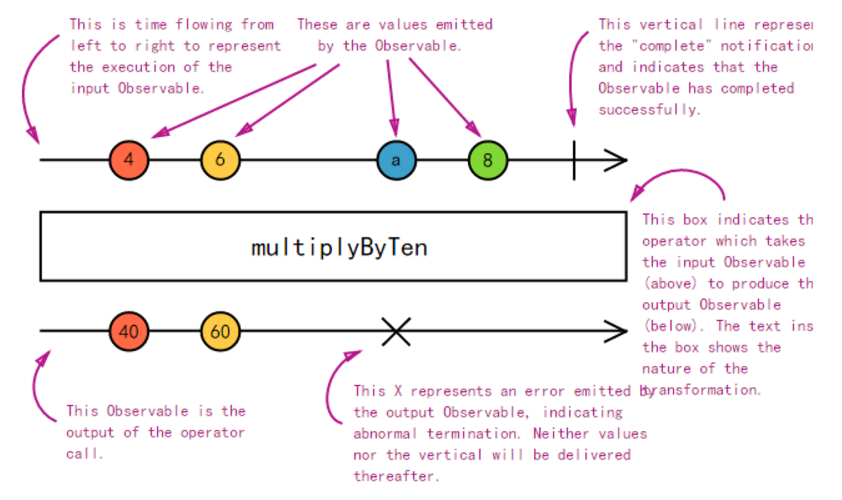
基于当前可观察对象创建一个新的可观察对象的函数

var observable1 = Rx.Observable.intervable(1000)//1000毫秒

var observable2 = Rx.Observable.create(...)

var merged = Rx.Observable.merge(observable1, observable2);

许多的操作符是和时间线密不可分的，在实际中他们可能需要对值进行延迟、采样、节流、或者抖动的发射。



Controlling the flow 控制流

// typing "hello world"

var input = Rx.Observable.fromEvent(document.querySelector('input'), 'keypress');

// Filter out target values less than 3 characters long

input.filter(event => event.target.value.length > 2).subscribe(value => console.log(value)); // "hel"

// Delay the events

input.delay(200).subscribe(value => console.log(value)); // "h" -200ms-> "e" -200ms-> "l" ...

// Only let through an event every 200 ms

input.throttleTime(200).subscribe(value => console.log(value)); // "h" -200ms-> "w"

// Let through latest event after 200 ms

input.debounceTime(200).subscribe(value => console.log(value)); // "o" -200ms-> "d"

// Stop the stream of events after 3 events

input.take(3).subscribe(value => console.log(value)); // "hel"

// Passes through events until other observable triggers an event

var stopStream = Rx.Observable.fromEvent(document.querySelector('button'), 'click');

input.takeUntil(stopStream).subscribe(value => console.log(value)); // "hello" (click)

producing values 生产值

// typing "hello world"

var input = Rx.Observable.fromEvent(document.querySelector('input'), 'keypress');

// Pass on a new value

input.map(event => event.target.value).subscribe(value => console.log(value)); // "h"

// Pass on a new value by plucking it

input.pluck('target', 'value').subscribe(value => console.log(value)); // "h"

// Pass the two previous values

input.pluck('target', 'value').pairwise().subscribe(value => console.log(value)); // ["h", "e"]

// Only pass unique values through

input.pluck('target', 'value').distinct().subscribe(value => console.log(value)); // "helo wrd"

// Do not pass repeating values through

input.pluck('target', 'value').distinctUntilChanged().subscribe(value => console.log(value)); // "helo world"

操作符见

http://reactivex.io/rxjs/manual/overview.html#creation-operators

Creation Operators:

ajax,bindCallback,bindNodeCallback,create,defer,empty,from,fromEvent,fromEventPattern,fromPromise,generate,interval,never,of,repeat,repeatWhen,range,throw,timer

Transformation Operators:

buffer, bufferCount, bufferTime, bufferToggle, bufferWhen, concatMap, concatMapTo, exhaustMap, expand, groupBy, map, mapTo, mergeMap, mergeMapTo, mergeScan, pairwise, partition, pluck, scan, switchMap, switchMapTo, window, windowCount, windowTime, , indowToggle, windowWhen

Filtering Operators:

debounce, debounceTime, distinct, distinctKey, distinctUntilChanged, distinctUntilKeyChanged, elementAt, filter, first, gnoreElements, audit, auditTime, last, sample, sampleTime, single, skip, skipUntil, skipWhile, take, takeLast, takeUntil, , akeWhile, throttle, throttleTime

Combination Operators:

combineAll, combineLatest, concat, concatAll, exhaust, forkJoin, merge, mergeAll, race, startWith, switch, withLatestFrom, zip, zipAll

Multicasting Operators:

cache, multicast, publish, publishBehavior, publishLast, publishReplay, share

Error Handling Operators:

catch,try, retryWhen

Utility Operators:

do, delay, delayWhen, dematerialize, finally, let, materialize, observeOn, subscribeOn, timeInterval, timestamp, timeout, imeoutWith, toArray, toPromise

Conditional and Boolean Operators:

defaultIfEmpty, every, find, findIndex, isEmpty

Mathematical and Aggregate :

count, max, min, reduce

## Scheduler调度者

调度者使得你可以确定可观察对象在什么执行上下文中给观察者发送通知

var observable = Rx.Observable.create(function (proxyObserver) {

proxyObserver.next(1);

proxyObserver.next(2);

proxyObserver.next(3);

proxyObserver.complete();

})

**.observeOn(Rx.Scheduler.async)**; //注意：不同于以往同步，这里是异步执行

var finalObserver = {

next: x => console.log('got value ' + x),

error: err => console.error('something wrong occurred: ' + err),

complete: () => console.log('done'),

};

console.log('just before subscribe');

observable.subscribe(finalObserver);

console.log('just after subscribe');

# Redux

适用场景：多交互、多数据源

组件角度: 某个组件的状态，需要共享; 某个状态需要在任何地方都可以拿到; 一个组件需要改变全局状态; 一个组件需要改变另一个组件的状态

Redux 的设计思想:

（1）Web 应用是一个状态机，视图与状态是一一对应的。

（2）所有的状态，保存在一个对象里面。

Core tenets:

* State is a single immutable data structure
* Actions describe state changes
* Pure functions called reducers take the previous state and the next action to compute the new state
* State accessed with the Store, an observable of state and an observer of actions

## 主要概念

Reducer

(state, action) => state 的纯函数。

state 的形式取决于你，可以是基本类型、数组、对象、甚至是 Immutable.js 生成的数据结构。惟一的要点是当 state 变化时需要返回全新的对象，而不是修改传入的参数。

const reducer = function (state, action) {

// ...

return new\_state;

};

对于大型应用,把 Reducer 函数拆分。不同的函数负责处理不同属性，最终把它们合并成一个大的 Reducer 即可

import { combineReducers } from 'redux';

const chatReducer = combineReducers({

chatLog,

statusMessage,

userName

})

Store

保存数据的地方，你可以把它看成一个容器。整个应用只能有一个 Store

import { createStore } from 'redux';

const store = createStore(reducer, window.STATE\_FROM\_SERVER);

window.STATE\_FROM\_SERVER就是整个应用的状态初始值。注意，如果提供了这个参数，它会覆盖 Reducer 函数的默认初始值

State

Store对象包含所有数据。如果想得到某个时点的数据，就要对 Store 生成快照。这种时点的数据集合，就叫做 State。

Redux 规定， 一个 State 对应一个 View。只要 State 相同，View 就相同

const state = store.getState();

Action

View -> Action -> State

const action = {

type: 'ADD\_TODO',

payload: 'Learn Redux'

};

每当store.dispatch发送过来一个新的 Action，就会自动调用 Reducer，得到新的 State。

store.dispatch(action)是 View 发出 Action 的唯一方法。

监听函数

一旦 State 发生变化，就自动执行这个函数

let unsubscribe = store.subscribe(() =>

console.log(store.getState())

);

unsubscribe();

工作流：

1. 用户发出 Action

store.dispatch(action);

1. Store 自动调用 Reducer

let nextState = todoApp(previousState, action);

1. State 一旦有变化，Store 就会调用监听函数。

store.subscribe(listener);

listener可以通过store.getState()得到当前状态

function listerner() {

let newState = store.getState();

component.setState(newState); //重新渲染View

}

范例：

import { createStore } from 'redux';

function counter(state = 0, action) {

switch (action.type) {

case 'INCREMENT':

return state + 1;

case 'DECREMENT':

return state - 1;

default:

return state;

}

}

let store = createStore(counter);

store.subscribe(() => console.log(store.getState()) );

store.dispatch({ type: 'INCREMENT' }); // 1

store.dispatch({ type: 'INCREMENT' }); // 2

store.dispatch({ type: 'DECREMENT' }); // 1