Contents

[Reactive Native 3](#_Toc445584286)

[Ionic 4](#_Toc445584287)

[Overview 4](#_Toc445584288)

[Install 4](#_Toc445584289)

[CSS COMPONENTs 5](#_Toc445584290)

[Header 5](#_Toc445584291)

[Content 6](#_Toc445584292)

[Footer 6](#_Toc445584293)

[Buttons 6](#_Toc445584294)

[List 7](#_Toc445584295)

[Cards 8](#_Toc445584296)

[Forms & Inputs 10](#_Toc445584297)

[Tabs 14](#_Toc445584298)

[Grid 15](#_Toc445584299)

[Colors 16](#_Toc445584300)

[Icons 16](#_Toc445584301)

[Padding 16](#_Toc445584302)

[PLATFORM CUSTOMIZATION 17](#_Toc445584303)

[Platform Classes 17](#_Toc445584304)

[Styling using AngularJS 17](#_Toc445584305)

[Using the Platform to Dynamically Style Elements 18](#_Toc445584306)

[JAVASCRIPT 19](#_Toc445584307)

[Action Sheet: $ionicActionSheet 19](#_Toc445584308)

[Backdrop: $ionicBackdrop 20](#_Toc445584309)

[Content: ionContent 20](#_Toc445584310)

[ion-checkbox 21](#_Toc445584311)

[on-hold 21](#_Toc445584312)

[ion-header-bar 21](#_Toc445584313)

[ion-list 21](#_Toc445584314)

[Loading: $ionicLoading 22](#_Toc445584315)

[$ionicModal 22](#_Toc445584316)

[导航（路由：AnguarUI Router） 23](#_Toc445584317)

[Platform: $ionicPlatform 25](#_Toc445584318)

[Popover: $ionicPopover 26](#_Toc445584319)

[Popup:$ionicPopup 27](#_Toc445584320)

[Scroll: ion-scroll 28](#_Toc445584321)

[Side Menus:ion-side-menus 29](#_Toc445584322)

[Slide Box: ion-slide-box 29](#_Toc445584323)

[Spinner:ion-spinner 29](#_Toc445584324)

[Tabs:ion-tabs 30](#_Toc445584325)

[Testing in a Browser 30](#_Toc445584326)

[PhoneGap 34](#_Toc445584327)

Hybrid apps are essentially small websites running in a browser shell in an app that have access to the native platform layer

# Reactive Native

Install android SDK

(<http://facebook.github.io/react-native/docs/android-setup.html#content>

Android SDK Build-tools version 23.0.1

Android 6.0 (API 23)

Android Support Repository)

Install npm > 4.0

>npm install –g react-native-cli

>react-native init AwesomeProject

(note: fails to connect to git://github.com

>git config –global url.<https://github.com/>.insteadOf [git@github.com](mailto:git@github.com)

>git config –global url.https://.insteadOf git://

Current progress requires several minutes)

AwesomeProject>react-native start –port 9988 //change the default port

# Ionic

## Overview

Ionic comes with very native-styled mobile UI elements and layouts that you’d get with a native SDK on iOS or Android but didn’t really exist before on the web

Since Ionic is an HTML5 framework, it needs a native wrapper like Cordova or PhoneGap in order to run as a native app

Ionic apps aren’t meant to be run in a mobile browser app like Chrome or Safari, but rather the low-level browser shell like iOS’s UIWebView or Android’s WebView, which are wrapped by tools like Cordova/PhoneGap.

Eager developers might also dig down into the native layer with custom Cordova plugins or native code

**Ionic targets iPhone and Android devices** (currently). We support **iOS 6+, and Android 4.0+** (though 2.3 should work).

Ionic is a bunch of UI elements made in HTML5 & CSS3 that covers a lot of the mobile interactions

The big advantage of Ionic is all the UI components are AngularJS Directives

• Built with Sass and optimized for AngularJS

• Beautifully designed

• Extends the HTML vocabulary

• **UI Components using Directives and Services**

• Proven for large-scale app development

• Ionicons (over 700 MIT licensed font-icons)

• Supported by Drifty and has a large community:

• Very active internal forum

## Install

//installation

**>npm install –g cordova ionic**

//project template

**Project Parent> ionic start myApp tabs**

(or ionic start myApp blank, ionic start myApp sidemenu)

//自定义样式

$ionic setup sass

运行以后，就会对scss/ionic.app.scss文件监控，有修改，会自动编译该文件

//build via android or ios

**myApp> ionic platform add android**

**myApp> ionic build android**

//test

**myApp> ionic emulate android // testing in simulator**

(or substitute android with ios to build for ios)

Note: 若emulate失败，可以将myApp\platforms\android\build\outputs\apk\android-debug.apk拷到mobile device安装测试

**myApp> ionic serve //testing in a browser**

//publish

myApp> cordova plugin rm cordova-plugin-console // disenable the debug console plugin

//generate our private key

>keytool -genkey -v -keystore **my-release-key.keystore** -alias alias\_name -keyalg RSA -keysize 2048 -validity 10000

//sign the unsigned APK

>jarsigner -verbose -sigalg SHA1withRSA -digestalg SHA1 -keystore **my-release-key.keystore** **HelloWorld-release-unsigned.apk** alias\_name

//optimize the APK

>zipalign -v 4 HelloWorld-release-unsigned.apk HelloWorld.apk

Yeoman: node必须是最新版

>npm install –g generator-ionic

(如果有错误，类似operaton not permitted, rename ‘…\npm-cache\...’, 要先清空

>npm cache clean)

>mkdir my-ionic-project && cd $\_

my-ionic-project>yo ionic

## CSS COMPONENTs

### Header

Headers are fixed regions at the top of a screen that can contain a title label, and left/right buttons for navigation or to carry out various actions.

<div class="bar **bar-header** bar-light">

<h1 class="title">bar-light</h1>

</div>

Headers come in a variety of default color options: bar-light, bar-stable, bar-positive, bar-calm, bar-balanced,bar-energized,bar-assertive,bar-royal,bar-dark

Sub Header

A secondary header bar can be placed below the original header bar

remember to include the has-subheader CSS class to your ion-content directive.

<div class="bar bar-header"><h1 class="title">Header</h1></div>

<div class="bar **bar-subheader**"><h2 class="title">Sub Header</h2></div>

### Content

<ion-content></ion-content> refer to Javascript

The content area in Ionic is the scrollable viewport of your app. While your headers and footers will be fixed to the top and bottom, respectively, the content area will fill the remaining available space.

### Footer

Footers are regions at the bottom of a screen that can contain various types of content.

Footers have the same color options as the headers

<div class="bar **bar-footer** bar-balanced">

<button class="button button-clear">Left</button>

<div class="title">Title</div>

<button class="button button-clear">Right</button>

</div>

### Buttons

By default a button has display: inline-block applied.

Like the Header, they come in the full spectrum of Ionic's default colors.

<button class="button button-light">button-light</button>

Block Buttons

Adding button-block to a button applies display: block display. A block button will however go 100% of its parent's width. In the example, the button's containing content element also has padding applied, so there is some breathing room between the edge of the device and the buttons.

<button class="button **button-block** button-positive">Block Button</button>

Full Width Block Buttons

Adding button-full to a button not only applies display: block, but also removes borders on the left and right, and any border-radius which may be applied. This style is useful when the button should stretch across the entire width of the display. Additionally, the button's parent element does not have padding applied.

<button class="button **button-full** button-positive">Full Width Block Button</button>

Different Sizes: Adding **button-large** to a button makes it larger, adding button-small makes it smaller.

Outlined Buttons: Use **button-outline** to apply an outline button style, which also has a transparent background.

Clear Buttons: Add **button-clear** to remove the border and make the text stand out.

Icon Buttons: Icons can easily be added to any button by using either the built in Ionicons, or any custom font-pack you choose.

<button class="button **icon-left ion-home**">Home</button>

<a class="button button-outline icon-right ion-navicon button-balanced">Reorder</a>

Buttons in Headers

When buttons are placed in headers or footers, they take the style of the bar by default

<div class="bar bar-header">

<button class="button icon ion-navicon"></button>

<h1 class="title">Header Buttons</h1>

<button class="button">Edit</button>

</div>

Clear Buttons in Headers

simply add the button-clear classname to remove the background button color and border.

<div class="bar bar-header">

<button class="button button-icon icon ion-navicon"></button>

<div class="h1 title">Header Buttons</div>

<button class="button button-clear button-positive">Edit</button>

</div>

Button Bar

Buttons can also be easily grouped together using the button-bar classname

<div class="button-bar">

<a class="button">First</a>

<a class="button">Second</a>

<a class="button">Third</a>

</div>

### List

List views support various interaction modes such as editing, swipe to edit, drag to reorder, and pull to refresh.

List Dividers

List items can also be dividers to organize and group the list items. By default list item dividers will have a different background color and font-weight, but this is easily customizable.

List Icons

Lists can have icons assigned either to the left and/or right side of each list item, uses an <a> element for each item, which allows the entire list item to be tappable. If the item is an <a> or <button> element, and no icon has been added to the right, then a small right arrow will automatically be added.

<div class="list">

<a class="item **item-icon-left item-icon-right item-divider**" href="#">

<i class="icon ion-chatbubble-working"></i>

Call Ma

<i class="icon ion-ios-telephone-outline"></i>

</a>

</div>

List Buttons

<div class="list">

<div class="item **item-button-right**">

Call Ma

<button class="button button-positive">

<i class="icon ion-ios-telephone"></i>

</button>

</div>

...

</div>

Item Avatars

Item avatars are essentially a showcase of an image larger than an icon, but smaller than a thumbnail.

<div class="list">

<a class="item **item-avatar**" href="#">

<img src="venkman.jpg">

<h2>Venkman</h2>

<p>Back off, man. I'm a scientist.</p>

</a>

...

</div>

Item Thumbnails

Item Thumbnails are essentially a showcase of an image larger than an icon, and will often span/define the entire height of the list item.

<div class="list">

<a class="item **item-thumbnail-left**" href="#">

<img src="cover.jpg">

<h2>Pretty Hate Machine</h2>

<p>Nine Inch Nails</p>

</a>

...

</div>

### Cards

Cards are usually placed on top of one another, but they can also be used like a "page" and swiped between, left and right.

Card Headers and Footers

<div class="**card**">

<div class="**item** **item-divider**">

I'm a Header in a Card!

</div>

<div class="**item** item-text-wrap">

This is a basic Card with some text.

</div>

<div class="**item item-divider**">

I'm a Footer in a Card!

</div>

</div>

Card Lists

<div class="**list card**">

<a href="#" class="**item item-icon-left**">

<i class="icon ion-home"></i>

Enter home address

</a>

<a href="#" class="item item-icon-left">

<i class="icon ion-ios-telephone"></i>

Enter phone number

</a>

</div>

Card Images

<div class="**list card**">

<div class="**item item-avatar**">

<img src="avatar.jpg">

<h2>Pretty Hate Machine</h2>

<p>Nine Inch Nails</p>

</div>

<div class="**item item-image**">

<img src="cover.jpg">

</div>

<a class="item item-icon-left assertive" href="#">

<i class="icon ion-music-note"></i>

Start listening

</a>

</div>

Card Showcase

a showcase of a card using several different items.

<div class="list card">

<div class="item item-avatar">

<img src="mcfly.jpg">

<h2>Marty McFly</h2>

<p>November 05, 1955</p>

</div>

<div class="**item item-body**">

<img class="**full-image**" src="delorean.jpg">

<p>

This is a "Facebook" styled Card. The header is created from a Thumbnail List item,

the content is from a card-body consisting of an image and paragraph text. The footer

consists of tabs, icons aligned left, within the card-footer.

</p>

<p>

<a href="#" class="subdued">1 Like</a>

<a href="#" class="subdued">5 Comments</a>

</p>

</div>

</div>

### Forms & Inputs

A list is also used to group related input elements. Both item-input and item is then used to designate each individual input field and it's associated label.

#### Text Input: Placeholder Labels

<div class="list">

<label class="item **item-input**">

<input type="text" placeholder="Name">

</label>

<label class="item item-input">

<textarea placeholder="Comments"></textarea>

</label>

</div>

#### Text Input: Inline Labels

Use input-label to place a label to the left of the input element

<div class="list">

<label class="item **item-input**">

<span class="**input-label**">Username</span>

<input type="text">

</label>

<label class="item item-input">

<span class="input-label">Password</span>

<input type="password">

</label>

</div>

#### Text Input: Stacked Labels

Stacked labels always places the label on top of the input

<div class="list">

<label class="item **item-input item-stacked-label**">

<span class="**input-label**">First Name</span>

<input type="text" placeholder="John">

</label>

<label class="item item-input item-stacked-label">

<span class="input-label">Last Name</span>

<input type="text" placeholder="Suhr">

</label>

<label class="item item-input item-stacked-label">

<span class="input-label">Email</span>

<input type="text" placeholder="john@suhr.com">

</label>

</div>

#### Text Input: Floating Labels

<div class="list">

<label class="item **item-input item-floating-label**">

<span class="**input-label**">First Name</span>

<input type="text" placeholder="First Name">

</label>

<label class="item item-input item-floating-label">

<span class="input-label">Last Name</span>

<input type="text" placeholder="Last Name">

</label>

<label class="item item-input item-floating-label">

<span class="input-label">Email</span>

<input type="text" placeholder="Email">

</label>

</div>

#### Inset Forms

By default each input item will fill 100% of the width of its parent element (the list). However, you can inset the list using either the list list-inset or card classnames.

<div class="**list list-inset**">

<label class="item item-input">

<input type="text" placeholder="First Name">

</label>

<label class="item item-input">

<input type="text" placeholder="Last Name">

</label>

</div>

#### Inset Inputs

Using list-inset will inset the entire list, whereas placing item-input-inset will inset an input into an individual list item. Placing a button inside the item

<div class="list">

<div class="**item item-input-inset**">

<label class="**item-input-wrapper**">

<input type="text" placeholder="Email">

</label>

<button class="button button-small">

Submit

</button>

</div>

</div>

#### Input Icons

Icons can be easily added to the left of an item-input input. Simply add an icon before the <input>. By default the icon will take the color of label text. However, you can also use add placeholder-icon to give it a placeholder color.

<div class="list list-inset">

<label class="item item-input">

<i class="**icon ion-search placeholder-icon**"></i>

<input type="text" placeholder="Search">

</label>

</div>

#### Header Inputs

Inputs can also be placed in headers, along with buttons to submit or cancel the form.

<div class="bar bar-header item-input-inset">

<label class="item-input-wrapper">

<i class="icon ion-ios-search placeholder-icon"></i>

<input type="search" placeholder="Search">

</label>

<button class="button button-clear">

Cancel

</button>

</div>

#### Toggle

A toggle technically is the same thing as an HTML checkbox input, except it looks different and is easier to use on a touch device. Ionic prefers to wrap the checkbox input with the <label> in order to make the entire toggle easy to tap or drag.

#### Checkbox

Checkboxes allow the user to select a number of items in a set of choices. A good use for a checkbox list would be a filter list to apply multiple filters to a search.

<**ion-list**>

<**ion-checkbox** ng-model="filter.blue">Red</ion-checkbox>

<ion-checkbox ng-model="filter.yellow">Yellow</ion-checkbox>

<ion-checkbox ng-model="filter.pink">Pink</ion-checkbox>

</ion-list>

#### Radio Button List

Radio buttons let the user select one option in a set of options, unlike a checkbox which allows for multiple selections.

<**ion-list**>

<**ion-radio** ng-model="choice" ng-value="'A'">Choose A</ion-radio>

<ion-radio ng-model="choice" ng-value="'B'">Choose B</ion-radio>

</ion-list>

#### Range

<div class="item **range**">

<i class="icon ion-volume-low"></i>

**<input type="range" name="volume">**

<i class="icon ion-volume-high"></i>

</div>

<div class="list">

<div class="item range range-positive">

<i class="icon ion-ios-sunny-outline"></i>

<input type="range" name="volume" min="0" max="100" value="33">

<i class="icon ion-ios-sunny"></i>

</div>

</div>

#### Select

Ionic's select is styled so its appearance is prettied up relative to the browser's default style. However, when the select elements is opened, the default behavior on how to select one of the options is still managed by the browser.

Each platform's user-interface will be different as the user is selecting an option. For example, on a desktop browser you'll see the traditional drop down interface, whereas Android often has a radio-button list popup, and iOS has a custom scroller covering the bottom half of the window.

<div class="list">

<label class="item item-input **item-select**">

<div class="input-label">

Lightsaber

</div>

<select>

<option>Blue</option>

<option selected>Green</option>

<option>Red</option>

</select>

</label>

/div>

### Tabs

Tabs are a horizontal region of buttons or links that allow for a consistent navigation experience between screens. It can contain any combination of text and icons, and is a popular method for enabling mobile navigation.

<div class="**tabs tabs-icon-top**">

<a class="**tab-item**">

<i class="icon ion-home"></i>

Home

</a>

<a class="tab-item">

<i class="icon ion-star"></i>

Favorites

</a>

<a class="tab-item">

<i class="icon ion-gear-a"></i>

Settings

</a>

</div>

tabs-icon-top can be changed to tabs-icon-left

### Grid

Ionic's grid system is different than most because of its use of the CSS Flexible Box Layout Module standard.

Simply add columns you want in a row, and they'll evenly take up the available space

By default every col added inside a row will automatically receive an equal amount of the available area

<div class="row">

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

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

</div>

Grid: Explicit Column Sizes

You can explicitly state the size of a column if for example you'd want specific columns to be larger than the others in the same row

An advantage with this grid system is that you only have to state the percentage for the column that needs it, and the others will still evenly divide up the available areas.

<div class="row">

<div class="col col-50">.col.col-50</div>

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

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

</div>

.col-10 10%, .col-33 33.3333%, .col-67 66.6666%

Grid: Offset Columns

Columns can also be offset from the left.

<div class="**row**">

<div class="**col col-33 col-offset-33**">.col</div>

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

</div>

Grid: Vertically Align Columns

In the demo, we've made the last column in each row to have the tallest content in order to demonstrate how the content of the others vertically align. The first row shows the default which is to take the same height as the tallest column in the same row.

<div class="row">

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

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

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

<div class="col">1<br>2<br>3<br>4</div>

</div>

<div class="row">

<div class="col col-top">.col</div>

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

<div class="col col-bottom">.col</div>

<div class="col">1<br>2<br>3<br>4</div>

</div>

<div class="row **row-center**">

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

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

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

<div class="col">1<br>2<br>3<br>4</div>

</div>

Responsive Grid

There may be cases where a row of columns will not fit nicely in the available area. The responsive classes can be used to turn each column in a row into its own row at certain breakpoints.

For example, if you want a row of columns to turn in to stacked rows when the viewport is pretty small, you would use the .responsive-sm class. The example to the right is a simulation of what it'd look like.

<div class="row **responsive-sm**">

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

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

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

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

</div>

Responsive Class Break when roughly

.responsive-sm Smaller than landscape phone

.responsive-md Smaller than portrait tablet

.responsive-lg Smaller than landscape tablet

### Colors

To customize the colors you can simply override those coming from the ionic.css CSS file. Additionally, since Ionic is built using Sass, for more power and flexibility you could also modify and extend the color variables within the \_variables.scss file.

### Icons

Ionic also comes with its own free and open-sourced icon font, Ionicons, with over 500 icons to choose from.

Simply add icon and the Ionicon classname for the icon to show, which can be easily looked up on the Ionicons homepage.

<i class="icon ion-star"></i>

### Padding

Many components in Ionic purposely have both padding and margin reset set to zero. In many instances apps will have components bleed to the edge of the screen, and by starting each component at zero developers can easily control padding and margins throughout the app.

padding Adds padding around every side.

padding-vertical Adds padding to the top and bottom.

padding-horizontal Adds padding to the left and right.

padding-top Adds padding to the top.

padding-right Adds padding to the right.

padding-bottom Adds padding to the bottom.

padding-left Adds padding to the left.

## PLATFORM CUSTOMIZATION

### Platform Classes

Ionic automagically adds classes to the <body> of your project based on the device you are using to view the project. This means if you are viewing your app on an iOS device, the <body> will have the platform-ios class applied. Some information about the different classes can be found below.

Browser platform-browser

Cordova platform-cordova

Webview platform-webview

iOS platform-ios

iPad platform-ipad

Android platform-android

Windows Phone platform-windowsphone

iOS 8 platform-ios8

iOS 8.4 platform-ios8\_4

Android 4 platform-android4

Android 4.4 platform-android4\_4

Using Platform Classes to Override Styling

You can use any of the above classes to override Ionic styling. For example, if you wanted to override the header title on Android to be uppercase, you could use the following code:

.platform-android .bar-header {

text-transform: uppercase;

}

### Styling using AngularJS

he ionic.Platform utility can be used in your JavaScript controller to set the platform for your app

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

$scope.platform = ionic.Platform.platform();

})

### Using the Platform to Dynamically Style Elements

<ion-tabs class="tabs-stable" ng-class="{'tabs-positive': platform == 'android', 'tabs-icon-top': platform != 'android'}">

<!-- ion-tab directives go here -->

</ion-tabs>

<!-- if the platform is android don't add an icon, all other devices get an icon -->

<ion-tab title="Home" ng-attr-icon="{{ platform != 'android' ? 'ion-home' : undefined}}" href="#/tab/home">

Dynamically Loading Templates

Sometimes showing or changing elements based on the platform isn’t enough. There may be times when you need to use two different structures for your project, and you don’t want to place the logic in the HTML. In these cases, you can use ionic.Platform to decide which template to load in a given state.

.state('tab', {

url: "/tab",

abstract: true,

controller: 'AppCtrl',

templateUrl: function() {

if (ionic.Platform.isAndroid()) {

return "templates/home-android.html";

}

return "templates/home.html";

}

})

You can create this directory at the top level of your project, alongside the www and platforms directory

merges/

ios/

index.html

css/

platform.css

js/

app.js

android/

index.html

css/

platform.css

js/

app.js

Cordova will copy the platform specific files to the www directory in the platforms directory when you run the app

## JAVASCRIPT

### Action Sheet: $ionicActionSheet

The Action Sheet is a slide-up pane that lets the user choose from a set of options.

Dangerous options are highlighted in red and made obvious.

$ionicActionSheet.show(options): Load and return a new action sheet.

A new isolated scope will be created for the

action sheet and the new element will be appended into the body.

angular.module('mySuperApp', ['ionic'])

.controller(function($scope, $ionicActionSheet, $timeout) {

// Triggered on a button click, or some other target

$scope.show = function() {

// Show the action sheet

var hideSheet = $ionicActionSheet.show({

buttons: [

{ text: '<b>Share</b> This' },

{ text: 'Move' }

],

destructiveText: 'Delete',

titleText: 'Modify your album',

cancelText: 'Cancel',

cancel: function() {

// add cancel code..

},

buttonClicked: function(index) {

return true;

}

});

// For example's sake, hide the sheet after two seconds

$timeout(function() {

hideSheet();

}, 2000);

};

});

### Backdrop: $ionicBackdrop

Shows and hides a backdrop over the UI. Appears behind popups, loading,

and other overlays.

Often, multiple UI components require a backdrop, but only one backdrop is

ever needed in the DOM at a time.

Therefore, each component that requires the backdrop to be shown calls

$ionicBackdrop.retain() when it wants the backdrop, then $ionicBackdrop.release()

when it is done with the backdrop.

For each time retain is called, the backdrop will be shown until release is called.

For example, if retain is called three times, the backdrop will be shown until release

is called three times.

Notes:

- The backdrop service will broadcast ‘backdrop.shown’ and ‘backdrop.hidden’ events from the root scope,

this is useful for alerting native components not in html.

function MyController($scope, $ionicBackdrop, $timeout, $rootScope) {

//Show a backdrop for one second

$scope.action = function() {

$ionicBackdrop.retain();

$timeout(function() {

$ionicBackdrop.release();

}, 1000);

};

// Execute action on backdrop disappearing

$scope.$on('backdrop.hidden', function() {

// Execute action

});

// Execute action on backdrop appearing

$scope.$on('backdrop.shown', function() {

// Execute action

});

}

### Content: ionContent

Delegate: $ionicScrollDelegate

The ionContent directive provides an easy to use content area that can be configured

to use Ionic’s custom Scroll View, or the built in overflow scrolling of the browser.

<ion-content

[delegate-handle=""] [direction=""] [locking=""] [padding=""] [scroll=""] [overflow-scroll=""]

[scrollbar-x=""] [scrollbar-y=""] [start-x=""] [start-y=""] [on-scroll=""] [on-scroll-complete=""]

[has-bouncing=""] [scroll-event-interval=""]> ...

</ion-content>

### ion-checkbox

<ion-checkbox ng-model="isChecked">

Checkbox Label

</ion-checkbox>

### on-hold

Touch stays at the same location for 500ms. Similar to long touch events available for AngularJS and jQuery.

<button on-hold="onHold()" class="button">Test</button>

### ion-header-bar

Adds a fixed header bar above some content.Can also be a subheader (lower down) if the ‘bar-subheader’ class is applied.See the header CSS docs.

<ion-header-bar align-title="left" class="bar-positive">

<div class="buttons">

<button class="button" ng-click="doSomething()">Left Button</button>

</div>

<h1 class="title">Title!</h1>

<div class="buttons">

<button class="button">Right Button</button>

</div>

</ion-header-bar>

<ion-content>

Some content!

</ion-content>

### ion-list

Delegate: $ionicListDelegate

Both the list, which contains items, and the list items themselves can be any HTML element. The containing element requires the list class and each list item requires the item class. However, using the ionList and ionItem directives make it easy to support various interaction modes such as swipe to edit, drag to reorder, and removing items.

<ion-list ng-controller="MyCtrl"

show-delete="shouldShowDelete"

show-reorder="shouldShowReorder"

can-swipe="listCanSwipe">

<ion-item ng-repeat="item in items" class="item-thumbnail-left">

<img ng-src="{{item.img}}">

<h2>{{item.title}}</h2>

<p>{{item.description}}</p>

<ion-option-button class="button-positive" ng-click="share(item)">

Share

</ion-option-button>

<ion-option-button class="button-assertive" ng-click="edit(item)">

Edit

</ion-option-button>

<ion-delete-button class="ion-minus-circled" ng-click="items.splice($index, 1)">

</ion-delete-button>

<ion-reorder-button class="ion-navicon" on-reorder="reorderItem(item, $fromIndex, $toIndex)">

</ion-reorder-button>

</ion-item>

</ion-list>

### Loading: $ionicLoading

An overlay that can be used to indicate activity while blocking user interaction.

angular.module('LoadingApp', ['ionic'])

.controller('LoadingCtrl', function($scope, $ionicLoading) {

$scope.show = function() {

$ionicLoading.show({

template: 'Loading...'

});

};

$scope.hide = function(){

$ionicLoading.hide();

};

});

### $ionicModal

The Modal is a content pane that can go over the user’s main view temporarily. Usually used for making a choice or editing an item.

A modal will broadcast ‘modal.shown’, ‘modal.hidden’, and ‘modal.removed’ events from its originating scope, passing in itself as an event argument. Both the modal.removed and modal.hidden events are called when the modal is removed.

<script id="my-modal.html" type="text/ng-template">

<ion-modal-view>

<ion-header-bar>

<h1 class="title">My Modal title</h1>

</ion-header-bar>

<ion-content>Hello! </ion-content>

</ion-modal-view>

</script>

angular.module('testApp', ['ionic'])

.controller('MyController', function($scope, $ionicModal) {

$ionicModal.fromTemplateUrl('my-modal.html', {

scope: $scope,

animation: 'slide-in-up'

}).then(function(modal) { $scope.modal = modal; });

$scope.openModal = function() { $scope.modal.show(); };

$scope.closeModal = function() { $scope.modal.hide(); };

//Cleanup the modal when we're done with it!

$scope.$on('$destroy', function() { $scope.modal.remove(); });

// Execute action on hide modal

$scope.$on('modal.hidden', function() { // Execute action });

// Execute action on remove modal

$scope.$on('modal.removed', function() { // Execute action });

});

### 导航（路由：AnguarUI Router）

Like Angular’s core $route service, URLs can be used to control the views. However, the AngularUI Router provides a more powerful state manager in that states are bound to named, nested, and parallel views, allowing more than one template to be rendered on the same page. Additionally, **each state is not required to be bound to a URL**, and data can be pushed to each state which allows much flexibility.

By default, views are cached to improve performance. When a view is navigated away from, its element is left in the DOM, and its scope is disconnected from the $watch cycle. When navigating to a view that is already cached, its scope is then reconnected, and the existing element that was left in the DOM becomes the active view.

By default, Ionic will cache a maximum of 10 views, Cached can be disabled, or the maximum number of cached views changed in $ionicConfigProvider

By default, when navigating back in the history, the “forward” views are removed from the cache. If you navigate forward to the same view again, it’ll create a new DOM element and controller instance. Basically, any forward views are reset each time

Directive: **ionNavView**

Ionic is able to keep track of their navigation history. By knowing their history, transitions between views correctly enter and exit using the platform’s transition style.

<ion-nav-view></ion-nav-view>

Directive: **ionView**

A container for view content and any navigational and header bar information. Views can be cached, which means controllers normally only load once, which may affect your controller logic. To know when a view has entered or left, events have been added that are emitted from the view’s scope. These events also contain data about the view, such as the title and whether the back button should show. Also contained is transition data, such as the transition type and direction that will be or was used.

Directive: **ionNavBar**

If we have an ionNavView directive, we can also create an <ion-nav-bar>, which will create a topbar that updates as the application state changes.

Note that the ion-nav-bar element will only work correctly if your content has an ionView around it.

Directive: **ionNavBackButton**

The back button will appear when the user is able to go back in the current navigation stack Additionally, the button is automatically set to $ionicGoBack() on click/tap. By default, the app will navigate back one view when the back button is clicked

Directive: **ionNavButtons**

Any buttons you declare will be positioned on the navbar’s corresponding side. Primary buttons generally map to the left side of the header, and secondary buttons are generally on the right side. However, their exact locations are platform-specific. For example, in iOS, the primary buttons are on the far left of the header, and secondary buttons are on the far right, with the header title centered between them. For Android, however, both groups of buttons are on the far right of the header, with the header title aligned left.

Note that ion-nav-buttons must be immediate descendants of the ion-view or ion-nav-bar element (basically, don’t wrap it in another div).

Directive: **ionNavTitle**

The nav title directive replaces an ionNavBar title text with custom HTML from within an ionView template. This gives each view the ability to specify its own custom title element, such as an image or any HTML, rather than being text-only. Alternatively, text-only titles can be updated using the

view-title ionView attribute.

Note that ion-nav-title must be an immediate descendant of the ion-view or

ion-nav-bar element (basically don’t wrap it in another div).

**$ionicNavBarDelegate**

Delegate for controlling the ionNavBar directive.

$ionicNavBarDelegate.align([direction]): The direction to the align the title text towards. Available: 'left', 'right', 'center'. Default: 'center'.

$ionicNavBarDelegate.showBackButton([show]): Set/get whether the ionNavBackButton is shown (if it exists and there is a previous view that can be navigated to).

$ionicNavBarDelegate.showBar(show): Set/get whether the ionNavBar is shown.

$ionicNavBarDelegate.title(title): Set the title for the ionNavBar.

**$ionicHistory**

$ionicHistory keeps track of views as the user navigates through an app. Similar to the way a browser behaves, an Ionic app is able to keep track of the previous view, the current view, and the forward view (if there is one). However, a typical web browser only keeps track of one history stack in a linear fashion.

Unlike a traditional browser environment, apps and webapps have parallel independent histories,

such as with tabs. Should a user navigate few pages deep on one tab, and then switch to a new

tab and back, the back button relates not to the previous tab, but to the previous pages

visited within that tab.

$ionicHistory facilitates this parallel history architecture.

$ionicHistory.viewHistory()

$ionicHistory.currentView()

$ionicHistory.currentHistoryId()

$ionicHistory.currentTitle([val])

$ionicHistory.backView()

$ionicHistory.backTitle()

$ionicHistory.forwardView()

$ionicHistory.currentStateName()

$ionicHistory.goBack([backCount])

$ionicHistory.clearHistory()

$ionicHistory.clearCache()

$ionicHistory.nextViewOptions()

Example:

<body ng-app="starter">

<!-- The nav bar that will be updated as we navigate -->

<ion-nav-bar class="bar-positive">

<ion-nav-back-button>

</ion-nav-back-button>

</ion-nav-bar>

<!-- where the initial view template will be rendered -->

<ion-nav-view>

<ion-view>

<ion-nav-buttons side="primary">

<button class="button" ng-click="doSomething()">primary</button>

</ion-nav-buttons>

<ion-nav-title><img src="logo.svg"></ion-nav-title>

<ion-content>

Hello!

</ion-content>

</ion-view>

</ion-nav-view>

</body>

### Platform: $ionicPlatform

An angular abstraction of ionic.Platform.

Used to detect the current platform, as well as do things like override the Android back button in PhoneGap/Cordova.

$ionicPlatform.onHardwareBackButton(callback)

$ionicPlatform.offHardwareBackButton(callback)

### Popover: $ionicPopover

The Popover is a view that floats above an app’s content. Popovers provide an easy way to present or gather information from the user and are commonly used in the following situations:

Show more info about the current view

Select a commonly used tool or configuration

Present a list of actions to perform inside one of your views

<p>

<button ng-click="openPopover($event)">Open Popover</button>

</p>

<script id="my-popover.html" type="text/ng-template">

<ion-popover-view>

<ion-header-bar>

<h1 class="title">My Popover Title</h1>

</ion-header-bar>

<ion-content>

Hello!

</ion-content>

</ion-popover-view>

</script>

angular.module('testApp', ['ionic'])

.controller('MyController', function($scope, $ionicPopover) {

// .fromTemplateUrl() method

$ionicPopover.fromTemplateUrl('my-popover.html', { scope: $scope}).

then(function(popover) { $scope.popover = popover; });

$scope.openPopover = function($event) { $scope.popover.show($event); };

$scope.closePopover = function() { $scope.popover.hide(); };

//Cleanup the popover when we're done with it!

$scope.$on('$destroy', function() { $scope.popover.remove(); });

// Execute action on hide popover

$scope.$on('popover.hidden', function() { // Execute action });

// Execute action on remove popover

$scope.$on('popover.removed', function() { // Execute action});

});

### Popup:$ionicPopup

The Ionic Popup service allows programmatically creating and showing popup windows that require the user to respond in order to continue.

The popup system has support for more flexible versions of the built in alert(), prompt(), and confirm() functions that users are used to, in addition to allowing popups with completely custom content and look

angular.module('mySuperApp', ['ionic'])

.controller('PopupCtrl',function($scope, $ionicPopup, $timeout) {

// Triggered on a button click, or some other target

$scope.showPopup = function() {

$scope.data = {};

// An elaborate, custom popup

var myPopup = $ionicPopup.show({

template: '<input type="password" ng-model="data.wifi">',

title: 'Enter Wi-Fi Password',

subTitle: 'Please use normal things',

scope: $scope,

buttons: [

{ text: 'Cancel' },

{

text: '<b>Save</b>',

type: 'button-positive',

onTap: function(e) {

if (!$scope.data.wifi) {

//don't allow the user to close unless he enters wifi password

e.preventDefault();

} else {

return $scope.data.wifi;

}

}

}

]

});

myPopup.then(function(res) {

console.log('Tapped!', res);

});

$timeout(function() {

myPopup.close(); //close the popup after 3 seconds for some reason

}, 3000);

};

// A confirm dialog

$scope.showConfirm = function() {

var confirmPopup = $ionicPopup.confirm({

title: 'Consume Ice Cream',

template: 'Are you sure you want to eat this ice cream?'

});

confirmPopup.then(function(res) {

if(res) {

console.log('You are sure');

} else {

console.log('You are not sure');

}

});

};

// An alert dialog

$scope.showAlert = function() {

var alertPopup = $ionicPopup.alert({

title: 'Don\'t eat that!',

template: 'It might taste good'

});

alertPopup.then(function(res) {

console.log('Thank you for not eating my delicious ice cream cone');

});

};

});

### Scroll: ion-scroll

Delegate: $ionicScrollDelegate

Creates a scrollable container for all content inside.

<ion-scroll zooming="true" direction="xy" style="width: 500px; height: 500px">

<div style="width: 5000px; height: 5000px; background: url('https://upload.wikimedia.org/wikipedia/commons/a/ad/Europe\_geological\_map-en.jpg') repeat"></div>

</ion-scroll>

Note that it’s important to set the height of the scroll box as well as the height of the inner content to enable scrolling. This makes it possible to have full control over scrollable areas.

### Side Menus:ion-side-menus

Delegate: $ionicSideMenuDelegate

A container element for side menu(s) and the main content. Allows the left and/or right side menu

to be toggled by dragging the main content area side to side.

By default, side menus are hidden underneath their side menu content and can be opened by swiping

the content left or right or by toggling a button to show the side menu.

<ion-side-menus>

<!-- Left menu -->

<ion-side-menu side="left">

</ion-side-menu>

<ion-side-menu-content>

<!-- Main content, usually <ion-nav-view> -->

</ion-side-menu-content>

<!-- Right menu -->

<ion-side-menu side="right">

</ion-side-menu>

</ion-side-menus>

### Slide Box: ion-slide-box

Delegate: $ionicSlideBoxDelegate

The Slide Box is a multi-page container where each page can be swiped or dragged between:

<ion-slide-box on-slide-changed="slideHasChanged($index)">

<ion-slide>

<div class="box blue"><h1>BLUE</h1></div>

</ion-slide>

<ion-slide>

<div class="box yellow"><h1>YELLOW</h1></div>

</ion-slide>

</ion-slide-box>

### Spinner:ion-spinner

Spinners enables you to give your users feedback that the app is processing/thinking/waiting/chillin’ out

By default, the ionRefresher feature uses this spinner, Ionic offers ten spinners out of the box, and by default, it will use the appropriate spinner

for the platform on which it’s running

<ion-spinner icon="spiral"></ion-spinner>

### Tabs:ion-tabs

Delegate: $ionicTabsDelegate

For iOS, tabs will appear at the bottom of the screen. For Android, tabs will be at the top of the screen, below the nav-bar.

<ion-tabs class="tabs-positive tabs-icon-top">

<ion-tab title="Home" icon-on="ion-ios-filing" icon-off="ion-ios-filing-outline">

<!-- Tab 1 content -->

</ion-tab>

<ion-tab title="About" icon-on="ion-ios-clock" icon-off="ion-ios-clock-outline">

<!-- Tab 2 content -->

</ion-tab>

<ion-tab title="Settings" icon-on="ion-ios-gear" icon-off="ion-ios-gear-outline">

<!-- Tab 3 content -->

</ion-tab>

</ion-tabs>

### Testing in a Browser

$ ionic serve [options]

this command starts LiveReload which is used to monitor changes in the file system

By default, LiveReload will watch for changes in your www/ directory, excluding www/lib/. To change this, you can specify a watchPatterns property in the ionic.project file located in your project root to watch (or not watch) for specific changes

The serve command can add some proxies to the http server. These proxies are useful if you are developing in the browser and you need to make calls to an external API. With this feature you can proxy request to the external api through the ionic http server preventing the No 'Access-Control-Allow-Origin' header is present on the requested resource error.

In the ionic.project file you can add a property with an array of proxies you want to add. The proxies are object with two properties:

path: string that will be matched against the beginning of the incoming request URL.

proxyUrl: a string with the url of where the proxied request should go.

{

"name": "appname",

"email": "",

"app\_id": "",

"proxies": [

{

"path": "/v1",

"proxyUrl": "https://api.instagram.com/v1"

}

]

}

Using the above configuration, you can now make requests to your local server at http://localhost:8100/v1 to have it proxy out requests to https://api.instagram.com/v1

angular.module('starter.controllers', [])

.constant('InstagramApiUrl', '')

// .constant('InstagramApiUrl','https://api.instagram.com')

//In production, make this the real URL

.controller('FeedCtrl', function($scope, $http, InstagramApiUrl) {

$scope.feed = null;

$http.get(InstagramApiUrl + '/v1/media/search?client\_id=1&lat=48&lng=2.294351').then(function(data) {

console.log('data ' , data)

$scope.feed = data;

})

})

Live Reload App During Development (beta)

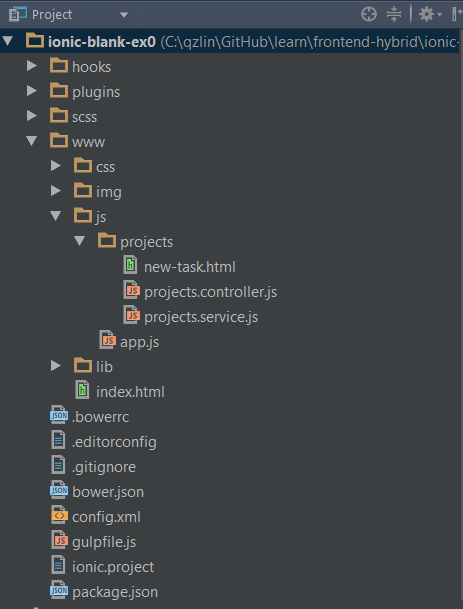
The run or emulate command will deploy the app to the specified platform devices/emulators. You can also run live reload on the specified platform device by adding the --livereload option. The live reload functionality is similar to ionic serve, but instead of developing and debugging an app using a standard browser, the compiled hybrid app itself is watching for any changes to its files and reloading the app when needed. This reduces the requirement to constantly rebuild the app for small changes. However, any changes to plugins will still require a full rebuild. For live reload to work, the dev machine and device must be on the same local network, and the device must support web sockets.Live Reload App During Development (beta)

Gathering information about your runtime

$ ionic info

### Structure

Ionic apps are built with Cordova. Cordova is a means of packaging html/css/js into apps that can run on mobile and desktop devices and provides a plugin architecture for accessing native functionality beyond the reach of JS run from a web browser. As such, Ionic apps have the Cordova file structure.



**platforms** directory contains your iOS and Android projects

**hooks** is for custom actions to be taken as your app moves through the Cordova development process. It may be useful for larger projects that require automated processes to run and source code modification but will normally be unused.

**merges** is to override files for specific platforms. Say you have merges/ios/someFile.js and www/someFile.js. When deploying to iOS, someFile will be replaced by the version in merges. It can be an effective way to create platform-specific modifications to your app, but other methods are often preferred

**plugins** are where Cordova stores the plugins that you add to your project. Plugins are added by the command: ionic plugin add {plugin}, Where {plugin} is the plugin’s ID or github URL.

**scss** is for your app’s SASS file. Using SASS is optional in Ionic, but Ionic itself is built with SCSS, and there are many default styles you can change to quickly and cleanly customize Ionic without adding a myriad of CSS overrides.

**www** is where your app is developed and where you’ll spend most of your time when building an Ionic app. By default, Ionic organizes your app in to a series of directories: css, img, js, lib, and templates.

**css** contains either your app’s specific CSS file, or your SCSS generated output file, should you use it, along with any other CSS files you wish to add. CSS is added to your project by a <link> tag in your index.html

**img** is pretty self explanatory; put your images there.

**js** gets more complicated. By default, our starter apps are broken up into app.js, which contains your Angular run and config methods. This is where you define your app’s routing and environmental variables, like what kind of tab style to use, or whether you should use iOS style headers on Android.

**controllers.js** contains your Angular controllers for the states that require them.

**services.js** is not always included in the starters, but it contains the custom Angular services your app may use, such as one that calls out to a 3rd party API to get information your app uses.

**directives.js** is not always included in the starters, but it contains custom Angular directives that your app may use. There’s no magic to these files and their names; you are free to modify these file names and structure as you see fit. JS files must be added to your project’s index.html, and Angular modules must be added to your Angular app definition.

**lib** is where Ionic and any other libraries you use can be placed. It follows the Bower formatting, and new libs can be added and updated using Bower. If you do not use Bower, you can modify the structure of this directory as you see fit or even remove it entirely, as long as you copy the Ionic files to a new location in your project.

**templates** is where your view files go. Your project does have a main index.html file in the WWW directory, but your app likely contains many template views that are added dynamically. *Unlike your CSS and JS files, TEMPLATE files do not need to be mentioned in your index.html file. You define them in your $state definition, and UI Router and Ionic will take care of the rest, including pre-loading the template file. These files are Angular templates and are parsed before being added to the DOM.*

Your Ionic app is, at its heart, an Angular app, so when it comes to figuring out how to accomplish user interaction and communicate data throughout your app and to external sources, many of the same rules apply. Angular has an MVVM architecture where your controller’s inject a $scope object. This object’s properties are available to your template, and your template will update automatically as you change $scope property values. With 2-way data binding, input fields can also be bound, and updates to the input are automatically available to your controller’s $scope object.

Controllers may be deactivated or destroyed as you navigate away from a given page, and one controller does not have direct access to previous view’s controller. **Data can be shared between pages in a number of ways, however. The most appropriate way is usually through Angular’s model implementation, services**. Services are singletons, and their local vars and state will persist as you navigate from page to page. Services can be injected into your app in the same way $scope is. This dependency injection is a fundamental concept to Angular and building large, complex apps. Note that services themselves can inject other services. This way you can cleanly abstract large, complicated data management processes. Ionic provides many services like $loading, $actionSheet, and various delegate services. Angular itself has several services. Possibly the most common service is $http for making ajax calls to remote services.

**Directives** are a way of extending HTML’s capabilities. When you see things like <ion-view> or <ion-tabs>, these custom HTML elements are directives. Creating custom directives can take a while to master, but used correctly, they can greatly reduce unnecessary complexity and duplicated code.

Hybrid Apps

• HTML 5 that acts like native

• Web wrapper in native layer

• Direct access to native APIs

• A single code base

• Familiar web development environment

A lot of components

Swipeable List Options

• Side menus

• Actionsheets

• Tabs

• Pull to Refresh

• Slidebox

• Infinite Scroll

• Popup

• Popover

• Loading Overlay

• Inputs

• Buttons

Cached Views

View elements left in DOM

• $scope disconnected from cache

• State maintained

• Scroll position maintained

• Life Cycle events

• Highly configurable

Collection-Repeat

Replacement for ng-repeat

• Scroll through thousands of items

• Only renders the viewable items

• Smooth scrolling

Ionic-Cli

• Testing in a browser

• Live Reload App During Development

• Emulating your app

• Running your app on device

• Building your app (with or without SDK)

• Icon and Splash Screen Image Generation

• Crosswalk for Android

ngCordova: CORDOVA WITH THE POWER OF ANGULARJS

ngCordova is a collection of 63+ AngularJS extensions on

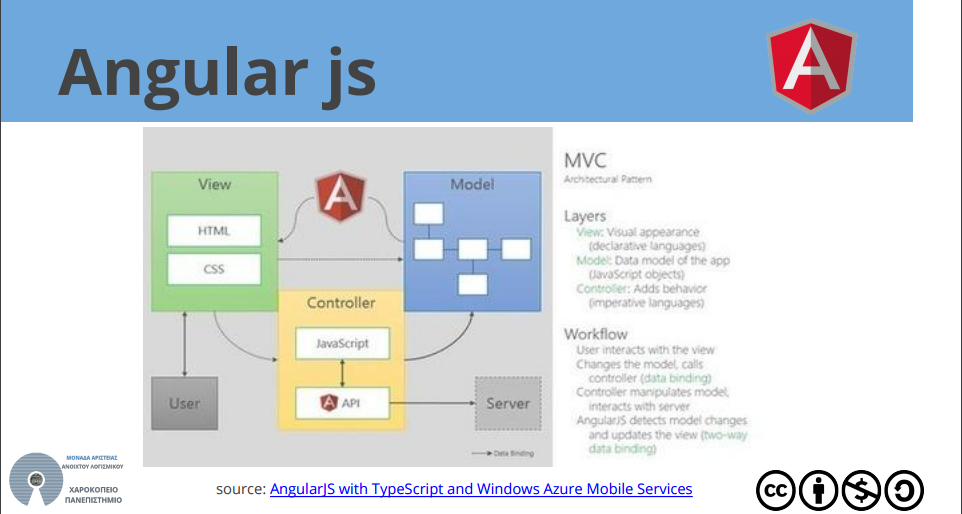
top of the Cordova API that make it easy to build, test, and

deploy Cordova mobile apps with AngularJS.

Apache Cordova

● Set of device APIs that allow a mobile app

developer to access native device function such as the camera or accelerometer from JavaScript.



# PhoneGap

Apache Cordova is the engine that powers Adobe PhoneGap™, similar to how WebKit powers Chrome or Safari. However, Adobe PhoneGap™ provides additional tools that tie into other Adobe services, including tools like the PhoneGap Developer App, the PhoneGap Desktop App, PhoneGap Build and PhoneGap Enterprise.

Install PhoneGap desktop app ->computer

Install PhoneGap Developer App ->mobile device

是在移动设备上调试 PhoneGap 应用的工具，不需要每次打包、安装就可以直接在手机上调试 PhoneGap 应用

PhoneGap Desktop starts a small web server to host your project and returns a server address you can then enter into the PhoneGap Developer app running on your mobile device.