Ionic

移动端常见的3种开发模式

- **WebApp**

  网页版的app，通过前端技术(css/html/js)来进行编码，而生成的可以运行在手机浏览器中的网页；

  可以提供类似nativeApp的应用体验

- **NativeApp**

  原生的app，通过对应的Android/iOS所对应的开发语言(oc/swift/java/kotlin),进行编程，

  生成的可以安装在手机操作系统的app

- HybridApp

  结合着前端技术和基础的原生开发技巧，生成可以安装在手机操作系统的app

Android 开发的基本流程

> http://www.androiddevtools.cn/
>
> https://developer.android.google.cn/

- **搭建环境**

  ① 安装jdk（提供java语言的编译环境）

  java development kit java 的开发工具箱

  ② 安装adt（提供android开发编辑器，sdk等工具

  android developer tools Android 的开发者工具

- **介绍 eclipse**

  ① 启动eclipse

  找到 eclipse.exe

  ② 以管理员身份运行

  选中要打开的软件，同时按下shift和鼠标右键，以管理员身份运行

  eclipse在使用时技巧：

  ① 恢复默认视窗

  window->reset perspective

  ② 调整文本字号大小

  window->preferences->General->Appereance->colors and fonts ->basic->Text Font

  双击进行编辑

  ③ 重启adb服务

sdk\platform-tools

在该文件中，选择在此处打开命令行窗口

adb.exe kill-server

adb.exe start-server

④ 如何开启logcat?

window->show view ->other ->logcat

- **完成一个Android项目的创建和编译**
  - **创建**

① eclipse点击左上角的File

file->new->Android Application Project

② 指定名字

ApplicationName demo01

PackageName com.tedu.demo01

注意：package的名字一旦指定之后，不要随便修改

③ 配置图标

点击browse按钮，在文件选择窗口中，选择对应的图片设置为应用程序未来启动时的图标

  - **运行**

选中资源管理器中的工程名字,demo01

按下鼠标右键

run As --> AndroidApplication

代码提示：alt + /

- **Android 开发的基本流程**

编写视图->编写xml(res/layout/***.xml)

业务逻辑->编写java(src/**/**.java)

知识点1：**android如何给按钮绑定单击事件？**

```
findViewById(R.id.btn1).setOnClickListener(new View.OnClickListener())
```

知识点2：**怎么显示一个通知？**

```
Toast.makeText(
   getApplicationContext(),
   "123",
   Toast.LENGTH_LONG
).show()
```

编写Android应用程序写完之后，怎么找到最终所生成的apk包?

步骤1：工程的bin目录,**.apk

步骤2：鼠标右键点击，properties查看Location

标准模板项目介绍：

src 存储java文件（处理业务逻辑）

assets 存储不希望在打包时被编译的文件，主要是在混编中存储前端的资源比如html/css/js

bin 有一个安装包

res 放图片、布局文件

AndroidManifest.xml 清单配置文件(版本号、权限)

混合编程基本步骤

- 方案 1：

  ① 编写前端代码，完成前端的功能

  ② 创建一个Android项目

  ③ 将前端的代码拷贝到Android项目中的assets文件夹中

  ④ 调用Android中的WebView原生控件，加载assets目录下的html

  ```
  // 创建一个WebView类型的对象
      WebView wv = new WebView(getApplicationContext());
  // 载入前端的网页
      wv.loadUrl("file:///android_asset/test01.html");
  // 将配置好的wv设置为内容视图
      setContentView(wv)
  ```

- 方案 2：

  ①编写前端代码

  ②创建android

  ③将前端代码部署在服务器

  http://localhost/framework/WebApp/day01/test03.html

  ④调用WebView载入服务器端前端文件

  ⑤添加网络权限

  步骤1：找到当前工程根目录下的 AndroidManifest.xml

  步骤2：双击打开文件，选中 permissions 标签页

  步骤3：点击add按钮，在弹窗中选择uses permission，从右侧的下拉菜单中选中了网络权限

android.permission.INTERNET

- 对比：

  方案1加载网页速度比较快

  方案2升级app更方便（代码全在服务器端）

> 注意事项：
>
> ① 允许执行js
>
> wv.getSettings().setJavaScriptEnabled(true)
>
> ② 在Android4.4.2以下的版本，不支持在混合编程中出现es6\c3等高级特性
>
> ③ 如何在混编中调试？
>
> wv.setWebChromeClient(new WebChromeClient())

Ionic

技术栈：

- Vue:

  VueJS + VueRouter + Vuex + axios

  移动端: mintui / weex

- Angular:

  core + router + HttpClientModule

  移动端: ionic

在Ionic模板项目中有一个www的文件夹，

这个文件夹所存储的是src目录中页面类编译完之后的文件，是可以直接部署在服务器端！

概述

> ionic = angular + icon + ionicModule + cordova

- **what/when?**

  用来构建移动端的应用程序的开源框架（提供了收费版本：有跟方便的编辑工具和涉及工具）

- **why?**

  ① 基于Angular

  ② 使用前端技术来编写跨平台的应用程序

  ③ 基于cordova(phoneGap)

④ 支持typescript

⑤ 有强大的命令行支持

- **how?**

  npm install -g ionic

  ionic start myApp tabs

  cd myApp

  ionic serve/npm start

- **Ionic 启动流程**

  ① 启动根模块

  src/app/app.module.ts

  ② 在bootstrap时指定要启动IonicApp

  ③ IonicModule指定根组件时MyApp(src/app/app.component.ts)

  ④ 指定了rootPage为HomePage

页面类的创建和调用

- ① 创建

  ionic g page demo01

- ② 声明

```
// - app.module.ts
import {Demo01Page} from '../pages/demo01/demo01'
@NgModule({
  delcarations:[Demo01Page],
  entryComponents:[Demo01Page]
})
```

- ③ 调用

```
// - app.component.ts
import {Demo01Page} from '../pages/demo01/demo01';
rootPage:any = Demo01Page
```

Ionic中常见组件类的用法

- **button**

  ion-button

color 颜色 primary/secondary/danger/light/dark

theme/variables.scss 自定义颜色

outline 边框

clear 只有文本

small 小

large 大

block 占据父容器的宽度

图标：

```
<button ion-button icon-left>
   <ion-icon name=""></ion-icon>
   clickMe
</button>
```

- **列表**

  普通列表

```
<ion-list>
  <ion-item></ion-item>
</ion-list>
```

  分组

```
<ion-item-group>
  <ion-item-divider></ion-item-divider>
  <ion-item></ion-item>
</ion-item-group>
```

  icon列表

```
<ion-list>
  <ion-item>
    <ion-icon item-start/end name=""></ion-icon>
  </ion-item>
</ion-list>
```

  avatar列表

```
<ion-list>
  <ion-item>
    <ion-avatar item-start/end >
      <img src=""/>
    </ion-avatar>
  </ion-item>
</ion-list>
```

thumbnail 列表

```
<ion-list>
  <ion-item>
    <ion-thumbnail item-start/end >
      <img src=""/>
    </ion-thumbnail>
  </ion-item>
</ion-list>
```

侧滑动列表项

```
<ion-list>
  <ion-item-sliding>
    <ion-item></ion-item>
    // 默认是在右边
    <ion-item-options side="left">
        <button></button>
    </ion-item-options>
  </ion-item-sliding>
</ion-list>
```

列表进阶

- **下拉刷新**

  ① 调用ion-refresher

  在ionContent的第一个子元素

  ② 绑定对应的事件

  `<ion-refresher (ionRefresh)="doRefresh($event)"></ion-refresher>`

  ③ 在事件处理函数中完成数据的异步操作并结束刷新动作

```
doRefresh(refresher){
  //async 异步的数据处理
  //通过ionRefresher结束掉正在刷新的动作
  refresher.complete()
}
```

- **上拉加载更多**

  ① 调用ion-infinite-scroll

  ionContent最后一个子元素的位置

```
<ion-infinite-scroll>
  <ion-infinite-scroll-content>
  </ion-infinite-scroll-content>
</ion-infinite-scroll>
```

  ② 绑定事件 ionInfinite

```
<ion-infinite-scroll (ionInfinite)="loadMore($event)">
  <ion-infinite-scroll-content
      loadingSpinner="bubbles"
      loadingText=""
  >
  </ion-infinite-scroll-content>
</ion-infinite-scroll>
```

  ③ 在事件处理函数中完成数据的异步操作并结束加载更多的操作

```
loadMore(infiniteScroll){
  //数据操作完成之后,结束加载更多的操作
  infiniteScroll.complete()
}
```

> 注意事项：
>
> 处理网络请求的基本步骤：
>
> ① 在根模块中指定依赖于HttpClientModule
>
> app.module.ts
>
> import {HttpClientModule} from '@angular/common/http
>
> @NgModule({
>
> imports: [HttpClientModule]
>
> })
>
> ② 根模块中任何一个组件，都可以使用HttpClient
>
> demo08-lianxi.ts

> import {HttpClient} from '@angular/common/http'
>
> constructor(private http:httpClient){}
>
> // rxjs 基于观察者、订阅者模式的一种异步处理方案
>
> this.http.get/post().subscribe(()=>{})

窗口

AlertController（警告窗、确认窗、输入提示）

LoadingController(加载中..)

ToastController(显示一个通知)

ActionSheetController(上拉菜单)

ModalController（自定义模态窗）

- **AlertController**

  ① import {AlertController} from 'ionic-angular'

  ② 实例化

  constructor(private alertCtrl:AlertController){}

  ③ 创建窗口，显示窗口

```
var myAlert = this.alertCtrl.create({
  title:'标题内容',
  inputs:[
    {
      type:'text/password/number...',
      placeholder:'',
      value:'',
      name:''
      ...
    }
  ],
  buttons:[
    { text:'确认', handler:()=>{} },
    { text:'取消', handler:()=>{} }
  ]
});
myAlert.present()
// 注意事项：在AlertController创建的输入提示窗，如何获取输入框的数据?
  handler:(data)=>{
  data[0] //第0个输入框的值
}
```

LoadingController

① 引入

import {LoadingController} from 'ionic-angular'

② 实例化

constructor(private loadingCtrl:LoadingController){}

③ 创建和显示

```
var MyLoading = this.loadingCtrl.create({
  content:'',
  duration:1000
});
MyLoading.present();
//手工关闭
MyLoading.dismiss();
```

ToastController

> 显示一个通知（给用户操作完成之后一个结果的提示）

① import

② 实例化

③ 创建、显示

```
create({
  content:'',
  position:'top/middle/bottom',
  showCloseButton:true,
  closeButtonText:''
})
```

ActionSheetController

> 上拉菜单（让用户从多个选项中做选择）

① 引入

② 实例化

③ 创建、显示

```
create({
  title:'',
  buttons:[
    {text:'', handler:()=>{}, role:'cancel/destructive'}
  ]
})
```

ModalController

> 用来显示一个自定义的窗口的（配置更灵活）
>
> 区别：在创建时是将一个组件类作为参数，而不再是直接写一个对象

关闭模态窗

```
import {ViewController} from 'ionic-angular'
constructor(private viewCtrl:ViewController){}
this.viewCtrl.dismiss();
```

进阶知识

模态窗在关闭时，进行数据(参数)的发送和接收

发送:

this.viewCtrl.dismiss({result:'成功'})

接收：

this.myModal.present();

this.myModal.onDidDismiss((data)=>{
    //data.result
})

> 注意事项：
>
> 窗口在创建完毕之后，属性已经配置过了，但是依然可以通过一些方法继续对属性做修改，
>
> 比如： myToast.setMessage('login failed')

Card

> 目的：为了更有效的组织信息给用户呈现

```
<ion-card>
  <ion-card-header></ion-card-header>
  <ion-card-content>
    <ion-card-title></ion-card-title>
  </ion-card-content>
</ion-card>
```

FAB

> Floating Action Button 悬浮式操作按钮
>
> 将一个按钮固定在屏幕的指定一个位置，也支持在点击时 显示更多的隐藏按钮

```
// left right top middle bottom
<ion-fab right bottom>
  <button ion-fab></button>
  <ion-fab-list side='top/bottom/left/right'>
    <button ion-fab></button>
    <button ion-fab></button>
  </ion-fab-list>
</ion-fab>
```

Grid

> 栅格的定位：实现自定义布局

- 栅格的基本用法:

```
<ion-grid>
  <ion-row>
    <ion-col></ion-col>
    <ion-col></ion-col>
  </ion-row>
  <ion-row></ion-row>
</ion-grid>
```

- 进阶用法:

  ① 指定列宽

  `<ion-col col-*></ion-col>`

  ② 设置一行所有的列纵向对齐

  `<ion-row align-items-start/center/end></ion-row>`

③ **设置某一列的纵向对齐**

`<ion-col align-self-start/center/end></ion-col>`

④ **设置某一个行列的水平对齐**

`<ion-row justify-content-center/start/end></ion-row>`

⑤ **移动列**

向右推：

`<ion-col push-*></ion-col>`

向左拉：

`<ion-col pull-*></ion-col>`

⑥ **设置偏移量**

`<ion-col offset-*></ion-col>`

ionSlides

```
<ion-slides autoplay loop pager paginatinType direction>
   <ion-slide></ion-slide>
</ion-slides>
```

forms

> 表单的使用都是通过ion-list和ion-item来进行管理的

常见的表单元素

不同的 label 配合着 input 来使用:

① **placeholder labels**

```
<ion-item>
  <ion-input placeholder=""></ion-input>
</ion-item>
```

② **fixed labels**

```
<ion-item>
  <ion-label fixed></ion-label>
  <ion-input placeholder=""></ion-input>
</ion-item>
```

③ stacked labels

```
<ion-item>
  <ion-label stacked></ion-label>
  <ion-input placeholder=""></ion-input>
</ion-item>
```

④ floating labels

```
<ion-item>
  <ion-label floating></ion-label>
  <ion-input placeholder=""></ion-input>
</ion-item>
```

⑤ inline labels

```
<ion-item>
  <ion-label></ion-label>
  <ion-input placeholder=""></ion-input>
</ion-item>
```

- 复选框

```
<ion-checkbox></ion-checkbox>
```

- 单选框

```
<ion-list radio-group>
  <ion-item>
    <ion-radio></ion-radio>
  </ion-item>
  <ion-item>
    <ion-radio></ion-radio>
  </ion-item>
</ion-list>
```

- 滑动开关

```
<ion-toggle></ion-toggle>
```

- 滑动组件

```
<ion-range min=0 max=100 step=10 pin=true>
  <ion-label range-left></ion-label>
  <ion-label range-right></ion-label>
</ion-range>
```

- 下拉菜单

```
// color='blue'
<ion-select [(ngModel)]="color">
  <ion-option value='red'>红色</ion-option>
  <ion-option value='blue'>蓝色</ion-option>
</ion-select>
```

导航

跳转

js:

① 引入NavController，目的地的页面类

import {NavController} from 'ionic-angular'

import {DestinationPage} from '**'

② 实例化NavController

constructor(private navCtrl:NavController){}

③ 定义一个跳转的方法

jump(){

this.navCtrl.push(DestinationPage)

}

属性:

① 引入跳转的目的地的页面类

import {DestinationPage} from '**'

② 在类中定义一个变量保存目的地页面类

desPage = DestinationPage

③ 指定navPush

`<any [navPush]="desPage"></any>`

传参

回顾:

- **Angular**

  发送:

this.myRouter.navigateByUrl('/detail/2')

接收：

/detail-->/detail/:id

import {ActivatedRoute} from '@angular/core'

constructor(private aRoute:ActivatedRoute){}

this.aRoute.params.subscribe(()=>{})

- **Vue**

  发送:

  this.$router.push('/detail/2')

  接收：

  /detail --> /detail/:id

  this.$route.params.id

- **ReactNative**

  发送

  this.props.navigation.navigate('detail',2)

  接收:

  this.props.navigation.state.params

Ionic

① 明确发送方和接收方

send--->rcv

② 发送

方案1：

`this.navCtrl.push(Demo21RcvPage,{name:'zhangsan'})`

方案2：

`<any [navPush]="demo21Rcv" [navParams]="{name:'zhagnsan'}"></any>`

③ 接收

import {NavParams} from 'ionic-angular'

constructor(private navParams:NavParams)

this.navParams.data.name

this.navParams.get('name')

页面守卫

> 在需要控制能够访问的页面中，指定一个生命周期的钩子函数

```
ionViewCanEnter(){
  //return true//允许访问
  //return false //禁止访问
}
```

ionViewDidLoad　　// 在页面第一次加载时

ionViewWillUnload　　// 在页面真正被销毁的时候

ionViewWillEnter　　// 每一次页面被访问时 都会执行的一个方法

lifecycle

> 任何一个组件都有从创建到调用到销毁的过程，一个组件的生命周期
>
> 生命周期的钩子函数，就是一些框架定义好的一些方法，方法在对应的生命周期的阶段就会被主动调用

- 初始化

  ngOnInit beforeCreate/created componentWillMount/DidMount

  ionViewDidLoad/ionViewWillEnter

- 数据变化

  ngOnChanges beforeUpdate/updated

  componentWillUpdate/DidUpdate

- 销毁

  ngOnDestroy

  beforeDestroy/destroied

  componentWillUnmount

  ionViewWillUnload

tabs

```
<ion-tabs selectedIndex='2'>
  // root是用来指定当前的tab显示是哪个页面
  <ion-tab root='tab1' tabTitle="首页" tabIcon=""></ion-tab>
  <ion-tab></ion-tab>
  <ion-tab></ion-tab>
  <ion-tab></ion-tab>
  <ion-tab></ion-tab>
</ion-tabs>
// 显示第三个tab，点击显示demo19-send
// 图标是star 标题是收藏
```

ionScroll

```
// reactNative ScrollView
// 实现一个支持滚动的容器
<ion-scroll scrollX='true' scrollY='true'></ion-scroll>

//注意事项：ionScroll需要指定高度才能显示内部的东西，否则是不显示

//实例:实现一个支持横向滚动的容器
<ion-scroll style='height:200px;white-space=nowrap'></ion-scrcoll>
```