小程序笔记

第一批小程序是2017年1月份上线的。

小程序账号注册及开发工具下载

1、先注册账号，打开微信服务平台，登录/注册账号

2、在文档 -> 工具 -> 下载 ->稳定版的开发工具

小程序基础知识必备：

1. flex布局(弹性盒子模型)：display :‘flex’

2、flex属性：

flex-direction：row (默认值)，主轴为水平方向，起点在左端

row-reverse 主轴为水平方向，起点在右端

column 主轴为垂直方向，起点在上沿

column-reverse 主轴为垂直方向，起点在下沿

flex的布局是可以进行嵌套的，align-items 可以实现居中

当布局使用display：flex时，子元素全部变成块级元素（不包括后代元素）

移动端相关知识：

物理像素：屏幕分辨率，设备控制的最小单元，就是手机最真实的像素，真实存在的像素

设备独立像素&css像素： 虚拟像素

dpr = 物理像素/设备独立像素（设备独立像素通常是iPhone6 的375\*667）

移动端的适配：

1、VW/VH

2、rem适配

3、viewport

小程序特点：

1、没有dom

2、组件化开发，组件：具备特定的功能的代码集合

3、体积小，单个压缩包不超过2M，否则无法上线

小程序设配方案：

1、小程序设配单位：rpx

2、规定屏幕为750rpx

3、iPhone6以下，1rpx = 1物理像素 = 0.5css像素/px

小程序的页面是以设备独立像素进行安排，但是编写程序是按照750来算，所以，页面的实际宽度（以iPhone6[ 375\*667 ]算），页面的长宽\*2 = rpx { 2rpx = 1px/css }

Rpx的计算 ：微信定制的750 / 实际手机机型的实际宽高 = rpx单位

小程序开发工具及使用：

1、打开按账号的开发工具

2、可以新建项目、导入项目

3、重命名、绑定ID（ID在官网，文档，左侧开发，开发设置，AppID）

4、开发模式（小程序）

5、后端服务（不使用云服务）

6、语言（js）

初始化项目文件介绍：

项目配置文件

project.config.json -> appid:"开发者ID"，以后拿到别人的项目修改此ID即可进行开发

-> projectname:'项目名'，以后要是更改项目名，即可通过这里修改项目称

sitemap -> rules里面的数组，里面的对象形式

必选项page，页面曝光率

app的文件：应用的文件（最外层的文件）：

app.js文件中App({ (注册app，只能注册一次，官网有接口，框架 -> 框架接口 -> App) })

app.json文件，所有的json文件都是相关配置文件（官网有接口，框架 -> 框架接口 -> App）

"pages":[

"pages/index/index",

"pages/logs/logs"

], 代表的是页面的路径，页面跳转等等相关配置，根据页面路径排位顺序，选择起始显示位置

"window":{

"backgroundTextStyle":"light", 背景文本风格

"navigationBarBackgroundColor": "#fff", 背景颜色

"navigationBarTitleText": "Weixin", 全局标题

"navigationBarTextStyle":"black" 全局字体颜色（只有黑白两种）

}, 代表的是全局的页面样式

page文件夹就是专门存放页面的，里面默认存放两个页面，每个页面的所有状态都在当前的文件夹内进行修改，而且必须要在app.json当中引入

从0到1创建项目:

Project.config.js、sitmap.json是页面初始化自带的文件,不能删除

可以根据编辑器报错一步一步创建文件:

1、创建App.json

2、然后报错为app.json内部必须为对象

3、然后在app.json的对象里，创建一个pages的数组，用于跳转页面

4、创建app.js,用于注册整个小程序，里面内容为：App({ }),

5、创建app.wxss，用于写整个最外层的样式

（初始页面三巨头）

6、创建pages总文件夹，里面放的是所有页面的文件夹

7、然后在pages里右键新建文件夹，再在当前文件夹右击新建page（自动生成4个页面，也可以手动新建），自动创建四个文件

页面结构：wxml:(后续补充)

<view></view>:相当于原生的div

开发中出现的问题：

问题一：内容使用flex布局，所以页面高度是用内容撑开，在最外层的view加上 height：100%，没有用。

所以不是view的问题，查看源代码，发现在最外层都会有一个page的包裹器，包裹器只有那么大，就算100%，也就那么大。

解决方案：在全局app.css当中，添加page：{ height：100% } 即可，然后通过继承即可实现所有页面的100%布局

小程序语法：

1、react :单向数据流 model --> view

2、vue :1、 单向数据流

2、双向数据绑定（v-model）

3、修改状态数据：this.xxx = value

3、小程序： 数据绑定 ：单向数据流 model --> view

数据流动和vue差不多，从JS流向wxml,引用用 {{ }}

修改状态数据 ：this.setData()

this指的是页面的实例，page实例

同步的修改

事件绑定 ：bind(事件名) 【冒泡事件】= “回调函数” 事件名在官网有

catch（事件名）【非冒泡事件】= “回调函数” 回调函数在js里，和生命周期函数平级

扩展：事件流分类：1、标准事件流 => 捕获（从外向内） 执行目标阶段 冒泡（从内向外）

2、IE事件流 => 没有捕获阶段

获取用户信息：

1、首次获取（未授权的情况下）

<button open-type = 'getUserInfo'/>，添加此标签即可出现授权弹窗

注意点：用户一旦授权以后，就不在弹出授权窗口

获取用户数据信息：bindgetuserinfo的回调中获取

然后用vue的方法，this.setdata，向data写入 再向 页面写入数据



2、再次使用（授权以后）

API：wx.getUserInfo()

注意点：必须在授权以后使用，否则会获取失败

获取用户信息：在其对应的回调中



注意点：小程序环境当中，全局对象是wx,浏览器才是window

小程序路由：【在api当中找路由相关，即可找到相关用法】

1、在页面的事件中添加属性，【catchtap="toLogs"】

2、在js当中添加跳转路径 【

toLogs(){

添加跳转方式，以及路径

}

】

小程序跳转方式：

1、wx.switchTab（{ 回调中添加url路径，指定的跳转页面 }） 跳转到 tabBar 页面，并关闭其他所有非 tabBar 页面

2、wx.relauch（{ 回调中添加url路径，指定的跳转页面 }） 关闭所有页面，打开到应用内的某个页面

3、wx.redirectTo（{ 回调中添加url路径，指定的跳转页面 }） 关闭当前页面，跳转到应用内的某个页面。但是不允许跳转到 tabbar 页面

4、wx.navigateTo（{ 回调中添加url路径，指定的跳转页面 }） 保留当前页面，跳转到应用内的某个页面。但是不能跳到 tabbar 页面。使用 wx.navigateBack 可以返回到原页面。小程序中页面栈最多十层

5、wx.navigateBack （{ 回调中添加url路径，指定的跳转页面 }） 关闭当前页面，返回上一页面或多级页面。可通过 getCurrentPages 获取当前的页面栈，决定需要返回几层

小程序的生命周期

1、onLoad 生命周期函数--监听页面加载 执行一次

2、onReady 生命周期函数--监听页面初次渲染完成 执行多次

3、onShow 生命周期函数--监听页面显示 执行一次

4、onHide 生命周期函数--监听页面隐藏

5、onUnload 生命周期函数--监听页面卸载

wx:if 和 wx:else 是做判断的，进行二选一的操作，判断的依据是wx.if后面的属性值的true，或者false

<image wx:if = "{{userInfo.avatarUrl}}" class = "avatarImg" src="{{userInfo.avatarUrl}}"></image>

<button wx:else open-type="getUserInfo" bindgetuserinfo = "hanldeGetUserInfo">获取用户信息</button>

硅谷音乐的案例笔记：

0 => 1的创建小程序的项目

1、新建文件夹

2、打开微信小程序的开发工具，进行配置（导入项目文件夹、添加开发者的ID、不使用云服务）

3、删除原生的页面

4、先是注册全局相关的

app.json【先是一个{}，里面有配置一“pages”，用于页面的跳转】

app.js【注册整个小程序的应用，App回车，里面装的是小程序的五个生命周期函数】

app.wxss【全局小程序的样式文件，第一个样式就是page{height：100%}，用于将页面全屏化】

1. 在page文件夹里新建页面文件夹等等

服务器的启动方式：在服务器根目录 在cmd中输入 npm start / npm run start

服务器在本地启动，然后根据启动的地址进行在浏览器填入，则进入的是服务器界面

小程序开发全部流程（全方面细致版）：

wxml的第一个包含器是view ==原来的HTML中的包含器 body

轮播图：在小程序的开发文档当中有具体的介绍，外边一个swiper标签，里面套着单张轮播图页，里面添加图片

轮播图的小圆点在官网文档当中有直接添加在swiper中，indicator-color （未选中的颜色） indicator-active-color（选中的颜色）

轮播图相关在官网都有，仔细观看

iconfont的使用：1、首先下载图标，必须先登录=>将喜欢的图标镜像添加至购物车 => 下载到本地 => 解压 => 将字体图标（全局）引入到自己的项目中

2、全局引入的自己的项目步骤：在app.wxss文件中，@importent "文件路径";

3、引入会报错，原因是编译器只会识别wxss后缀文件，所以把用css结尾的文件统统后缀改为wxss即可

4、在引入的地方使用class类后进行添加iconfont的编码

scroll-view：可滚动视图区域，就是相当于轮播图一样的效果，具体使用教程在小程序官网=>组件 => 视图容器中

图片标签必须是双标签形式，否则读不出来

display-flex横向布局没有用，因为在官方文档上明确说过，在wxml上开启横向布局enable-flex，横向布局才有用

单行文本溢出...代替：write-space ：nowrap(强制不换行)、overflow : hidden(块级元素溢出隐藏)、display：block（行内转块）、text-overflow:ellipsis(溢出转...)

多行文本溢出...代替：display：block（行内转块）、overflow : hidden(块级元素溢出隐藏)、text-overflow:ellipsis(溢出转...)、display : wibkit-box(给浏览器定义内核)、-webkit-line-clamp :2(设置行数)、-webkit-box-orient:vertical(规定子元素对齐方向)

.navScroll .scrollItem text{ (固定模板，多行文本强制不换行，并用...代替省略文本)

display:block;

font-size:26rpx;

overflow:hidden;

text-overflow:ellipsis;

display:-webkit-box;

-webkit-line-clamp:2;

-webkit-box-orient:vertical

}

前后端交互的方法（逻辑、接口的使用）：拿到接口文档注意的点有：发请求的方法（get或者post，默认是get）请求的地址、请求的参数等

测试接口（在浏览器打开相应的url,接上接口进行测试路径，再加上参数进行测试）

小程序当中进行前后端数据请求：在小程序的官方文档上有，开发 => API => 网络 => 发起请求 => wx.pequest ( Object object )

小程序自动帮我们处理了跨域的问题，后面直接加上后台服务器地址即可

发请求用的是JS，在JS当中操作，而且发请求在生命周期里发（onLoad，onReady）

交互方式在页面的生命周期里进行，因为交互的页面，而不是整个应用

完整的请求方式 wx.ruquest({

url:'http://localhost:3000/banner', 小程序不会自动补齐路径请求的接口

data:{type:2}, 请求的参数值

success:(res)=>{ 请求成功的回调

console.log（“请求成功”）；

console.log (res)

},

fail:(err)=>{ 请求失败的回调

console.log("请求失败")；

console.log（err）

}

})

封装函数请求的步骤：

1、封装函数请求的原因：为了代码量的减少，封装组件，进行后期调用

2、新建一个JS文件（所有的交互都是在JS文件当中进行的）

3、export default 进行函数接口的暴露，外边调用时使用

组件JS中：export default(url,data = {},method = "GET")=>{

return new Promise((resolve,reject)=>{

wx.request({

url:config.host + url,

data,

method,

success:(res)=>{

resolve(res.data）

},

fail:(err)=>{

reject(err)

}

})

})

}

4、在主JS文件中使用 import request from "../../utils/request" 引入封装的方法，并 且自定义变量接收请求的数据，将接收的数据返回到上面的data（自己定义的，一般在 最开始位置，生命周期函数之前）数据当中，供给页面使用

data: {

bannerList:[],

},

let bannerListData = await request("/banner");

接口注意点：1、请求的协议必须是https//的形式

2、一个接口最多配置20个域名（网址、服务器）

3、最大并发量是10个（同时发送的请求时10个）

4、开发前 在小程序管理后台页面 配置好域名（企业级开发会提前配置好域名）

开发注意点：1、开发过程中，请求得的服务器不满足小程序的要求，可以在小程序的开发工具设置“不校验合法域名”

当页面数据太多时，请求代码太多，所以提出封装代码的思想，将请求进行封装以便后期调用

封装思想：

函数：

1、功能点明确

2、函数体保留相同的代码，或者静态的代码

3、将动态的数据作为参数声明

4、使用者根据使用的需求，动态传入实参，进而去使用

组件：

1、功能点明确

2、组件内部保留相同的代码，或者静态的代码

3、将动态的数据抽取作为组件的props属性由外部导入

4、使用者需要根据需求动态传入数据，以标签属性的形式传入

5、良好的组件可以规定props属性的必要性和数据类型

如：props：{

num：{

type:Number， 数据类型

isRequired：true 必须传入

default：0， 默认值

}

}

封装组件以及调用的具体步骤：

1、新建一个文件夹，用来装定义的组件

2、在文件夹中新建JS文件，就是具体的组件

3、定义函数，镜像向外暴露

export default(url,data,method)=>{ 定义具体暴露的组件（动态的数据，外部向内传递的实参，可以帮助用户设定默认值）

export default(url,data = { },method = ‘GET’)=>{

wx.request({

url, 地址

data, 数据

method, 函数

success:(res)=>{ 成功的回调

console.log("请求成功");

console.log (res);

},

fail:(err)=>{ 失败的回调

console.log("请求失败")

console.log(err)

}

})

}

4、在需要引入的页面（JS）进行引入（import erquest from "request"）

5、在需要调用的地方直接写暴露的名字即可 ( 传入相应的实参 )

request("http://localhost:3000/banner",{type:2},"GET")

async await是属于promise的相关内容，解决同步异步的相关问题，后期添加笔记

resolve（promise的相关内容，请求成功时数据）

reject （promise的相关内容，请求失败时数据）

列表的渲染：

1、语法：wx:for ="{{ 从后台data传入数据的数组名字 }}"

进行数据前后台交互，首先从后台拿到轮播图数据，进行循环遍历的展示在页面上(补充点：在页面的循环当中，item的名字可以进行自定义的，在标签的名称后面，添加wx:for-item="自定义的名字"，这样做的目的只是简单的将原来的item进行简单的语义化操作)

1、页面的轮播图列表只需要写一个即可

2、页面数据向data获取（查看页面的data数据：在编辑工具的APPdata中有data数据）

3、初始化一个空数组，这个数据就是轮播图的数据

4、语义化，将生命周期函数的let名称修改掉，改为bannerListData,这个标签本来就是从后台获取数据的

5、更新bannerList的状态数据

使用setdata进行写入数据

this.setData({

bannerList:bannerListData.banners

})

1）data: { 定义一个空数组，进行向页面传递数据（vue知识点，data向页面挂载点传递数据）

bannerList:[],

},

2）onLoad: async function (options) { 生命周期函数

let bannerListData = await request("/banner",{type:2}) 根据路径请求数据，根据参数返回设备类型

this.setData({

bannerList:bannerListData.banners 将后台请求获取的数据进行赋值等操作，传递到页面

})

},

6、将获取的数据进行循环遍历到页面上（小程序官方文档上有，开发->框架 -> wxml语法参考 -> 列表渲染）

大致用法：在swiper中用wx:for={{ }}遍历从data返回的数据（本次小程序写的是bannerList）

在单项的图片区域，书写后台拿到的图片名称即可实现遍历图片进行展示

这是最简单数据请求方式交互逻辑（1、在页面的js文件的生命周期函数当中，发送请求方式，进行前台向后台请求数据。

2、将请求的数据进行传递，通过this.setData({ })设置数据，将请求的数据进行传递。

3、设置data对象，将传递的数据进行调用。

4、在页面进行循环遍历即可实现后台向前台传递参数进行轮播图效果）

封装组件：（封装样式相同的的代码，进行复用）

1、在根目录下创建conponents文件夹

2、里面新建单个组件的文件夹(名字尽量语义化，以便理解)

3、右键新建conponents(名字尽量语义化，以便理解)

4、（新建组件）拿到相应的样式，放入相应的文件中，页面的wxss 放入 组件的wxss , 页面的wxml放入组件的wxml

5、（注册组件）在外部JSON文件中，注册刚刚新建的新建组件，在usingComponents的对象中祖册组件，使用“key(页面使用的名称)”：“value（组件的路径）”

6、（使用组件）在页面的，使用注册组件的key值，进行复用

注意点：scroll-view的弊端，默认是纵向布局，flex布局横向化，但是高度还是scroll-view的高度，所以在以后的布局中，将高度进行固定

7、引用的数据进行动态化

8、通过组件的JS进行传值，组件的js文件当中，三个数对象代表不同的功能，properties{ 组件间的传值，通过外部传值到组件内部，内部的值官网有详细介绍，根据不同的调用，进行不同的传值，然后通过外部的标签传值，向内部传值，达到数据的动态化 }

组件JS的 properties: {

title: { // 属性名

type: String,

value: '测试默认值'

},

nav: { // 属性名

type: String,

value: '默认值'

},

},

组件JS向组件的页面输送信息，组件的数据通过{{ }}传递到组件的页面

<text class="title">{{title}}</text>

<text class="nav">{{nav}}</text>

组件的页面安插到总的页面上，直接调用注册组件的名字即可，同时通过总的页面的标签进行传值

封装组件以及动态传值的具体流程：

首先在根目录下新建文件夹components -> 新建文件夹新建components,命名为语义化的名字方便调用，这个文件夹则是一整个组件 -> （新建组件）将原始页面的公共样式添加到当前的样式当中 -> （注册组件）在外部JSON文件中，注册刚刚新建的新建组件，在usingComponents的对象中祖册组件，使用“key(页面使用的名称)”：“value（组件的路径）” -> （使用组件）在页面的，使用注册组件的key值，进行复用 -> 通过组件的js文件实现数据动态化propertise -> 组件JS向组件的页面输送信息，组件的数据通过{{ }}传递到组件的页面 ->组件的页面传递给总的页面（数据传递流程：总的页面 -> 组件的页面 -> 组件的JS。当外部页面没有传递数据的时候，js可以自带默认值逐层返回给页面）

底部tabbar的创建（就是底部导航栏）

在小程序官网有相关的介绍 =>文档 =>框架 => 小程序配置 =>全局配置 => tabbar

"tabBar": { 就是相当于路由的导航栏

"list": [ list相当于就是按钮项目，一般2-5个，多了就丑，list是数组形式，里面是对象，一个对象相当于夜歌按钮的组合

{

"pagePath": "pages/index/index", pagePath就是点击跳转到的页面路径（页面路径必须在“page”中有注册的）

"text": "主页", 导航显示的文本信息

"iconPath": "/static/images/tabs/tab-home.png", 添加的是未焦点时的状态，这里用的是iconfont

"selectedIconPath": "/static/images/tabs/tab-home-current.png" 添加的是焦点状态，用的也是iconfont

},

{

"pagePath": "pages/video/video",

"text": "视频",

"iconPath": "/static/images/tabs/select.png",

"selectedIconPath": "/static/images/tabs/selected.png"

},

{

"pagePath": "pages/personal/personal",

"text": "个人中心",

"iconPath": "/static/images/tabs/tab-my.png",

"selectedIconPath": "/static/images/tabs/tab-my-current.png"

}

],

"color":"#333", 在list有默认的字体颜色，通过官网可知，list平级有color，selectdeColor,就是覆盖掉原来默认自带的颜色，一个是未焦点的颜色，一个是焦点状态的颜色

"selectedColor":"#d43c33"

}

滑动逻辑：

小程序当中，对某个页面进行点击滑动等等操作时，需要给当前事件的外层的盒子套上三件套：bindtouchstart="handleTouchStart"，bindtouchmove="handleTouchMove"，bindtouchend="handleTouchEnd"

手指三件套，在wxml的事件案件对象上添加三角套属性，在逻辑的js中添加三个属性带来的触发事件，生命周期里添加触发事件的影响，然后通过{{ }}向页面发送数据

登录流程：

1、收集表单项内容

2、前端验证（1、利用正则验证用户输入是否合法，是否符合当前项目的规定。2、前端验证不通过，提示用户重新输入，不需要发请求。3、前端验证通过，发请求，进行后端验证）

3、后端验证（1、在前端验证通过后，将收集的表单项数据发请求交给后端。2、服务器验证当前用户的信息是否合法。3、验证不管通过或者不通过，都会返回数据给前端）

收集用户数据具体流程：

1、在pages文件夹中添加，login文件夹，初始化page文件

2、在wxml、wxss中书写骨架，书写样式

3、在js文件夹中添加 data 向页面添加空数据，用于收集页面传回来的数据

4、在wxml的相应标签添加属性，bindinput 属性，用于向后台data传数据，然后在js的相应的生命周期函数中添加相应的，变量名

扩展：bindtap,是跳转的意思，target 和 currentTarget 当前的目标，currentTarget中有，ID属性，用于标志触发事件，可以根据ID的不同，实现，同一个绑定事件，进行data的key的名，进行传值，从而达到同一个绑定事件，通过ID的不同，向data的不同空数组进行传值

5、给触发的标签绑定的事件，例：bindinput = “handleInput”，handleInput在JS中操作逻辑，打印的事件对象中有currentTarget有id名，微信的开发工具当中有此类型，通过不同的标签绑定不同的ID属性，然后ID的属性会传递给触发事件的currentTarget.id

6、通过let拿到currentTarget.id,然后this.setdata向data写入数据，注意：this.setdata的key名用[ ]包裹可以区分属性

wxml： <input bindinput="handleInput" id="phone" data-type="phone" type="text" placeholder="请输入手机号码"/>

js: handleInput(event){

let type = event.currentTarget.id;

this.setData({

[type]:event.detail.value

})

},

登录具体流程：

async login(){//前端验证

let {phone,password} = this.data;

if(!phone || !password){

wx.showToast({

title: '您输入的手机号或者密码不正确',

icon: 'none'

})

return

}else{//后端验证

let result = await request("/login/cellphone",{phone,password}) 后端验证必须向后台发送请求，然后判断状态码进行判断

console.log(result);

if(result.code === 200){ 根据状态码进行判断

wx.showToast({ 提示框（官方文档=> 开发 => API => 界面 => 交互）

title: '登陆成功',

icon: 'none',

})

wx.switchTab({ 操作逻辑是，登陆成功后跳转到 个人中心 页，wx.navgetTo不能跳转tabbar页面所以用wx.switchTab,进行跳转

url: '../personal/personal',

})

}else if(result.code === 400){

wx.showToast({

title: '您输入号码不正确',

icon: 'none'

})

}else if(result.code === 502){

wx.showToast({

title: '您输入密码不正确',

icon: 'none'

})

}else{

wx.showToast({

title: '登陆失败',

icon: 'none'

})

}

}

},