**微信小程序：**

优势：

无需安装、

用完即走、

连接人与服务（根据场景选择不同的服务）

**适合做成小程序的产品：**

1、简单用完即走

2、低频应用

3、性能不能有太高要求

例：滴滴、饿了么、猫眼

不适合出现在小程序的应用：

知乎、王者、直播

大型应用：

拆分来做成小程序

传统APP 与 微信小程序 对比：

体验差（先安装、先注册登录、后使用） 流程简单（无需安装、体验爽）

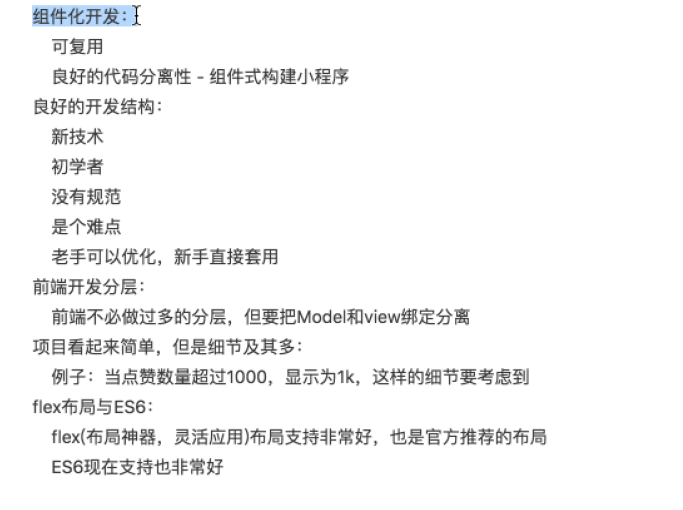
成本高（开发效率、跨平台、兼容性） 跨平台

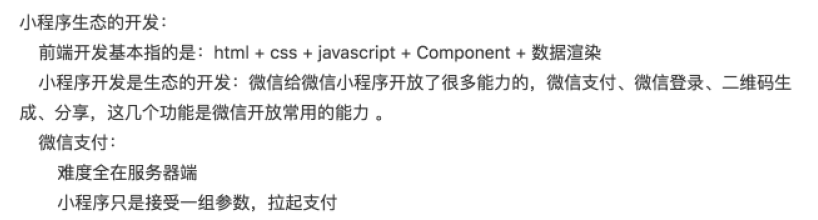
开发者的影响：

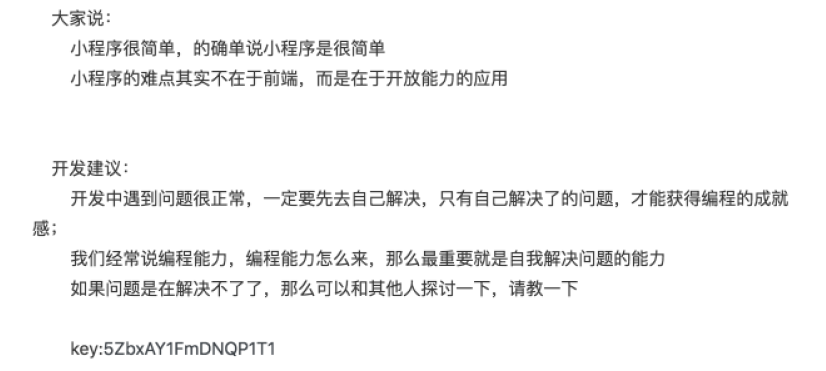
1、短期内将提升市场对js程序员的需求量

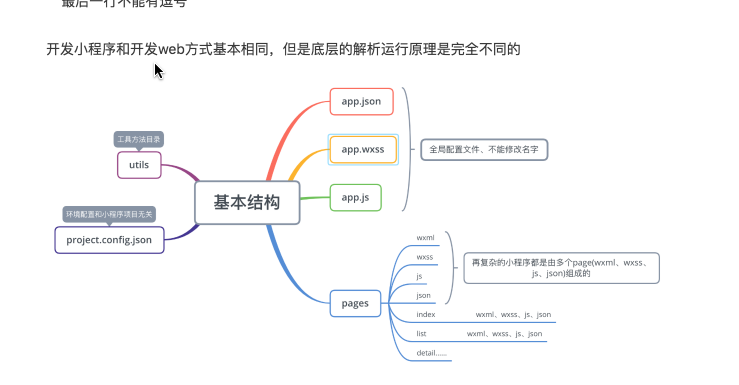
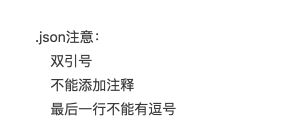
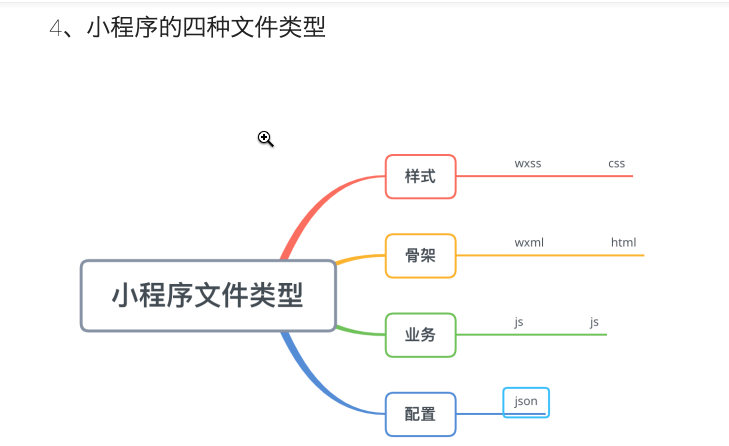
2、小程序是0基础开发者很好的入门平台

3、小程序不可以使用现在已经存在的js组件库（没有window对象，以DOM为组件库是用不了jq,等无法使用，，它是以数据驱动的不能操作DOM）









注册小程序：

1、打开微信公众平台管网，点击小程序中有一个查看详情

2、点击前往注册

3、

4、激活账号，用微信扫描二微码绑定

5、进入小程序页面，点击设置—》开发设置—》获取APPID （我的id：wxd05e679a16bd610c）

6、下载小程序的编辑器网址：

<https://developers.weixin.qq.com/miniprogram/dev/devtools/download.html?t=18112721>

7、



8、下载安装完成之后，点击



9、

编辑器的应用：

1. .json 后缀的 JSON 配置文件
2. .wxml 后缀的 WXML 模板文件
3. .wxss 后缀的 WXSS 样式文件
4. .js 后缀的 JS 脚本逻辑文件

#### 1、小程序配置 app.json

app.json 是当前小程序的全局配置，包括了小程序的所有页面路径、界面表现、网络超时时间、底部 tab 等。QuickStart 项目里边的 app.json 配置内容如下：

我们简单说一下这个配置各个项的含义:

1. pages字段 —— 用于描述当前小程序所有页面路径，这是为了让微信客户端知道当前你的小程序页面定义在哪个目录。
2. window字段 —— 定义小程序所有页面的顶部背景颜色，文字颜色定义等。

其他配置项细节可以参考文档 [小程序的配置 app.json](https://developers.weixin.qq.com/miniprogram/dev/framework/config.html) 。

https://shimo.im/docs/xHHDEFuelNc0d2iU/ 《微信小程序开发规范文档》

Thress.js可以做3D效果

小程序发起请求数据：

注：小程序接口协议只能用https，

不能用其它的协议，例：http

直接可以在视图的js里面用：

wx.request({

url: 'test.php', //仅为示例，并非真实的接口地址

data: {

x: '',

y: ''

},

header: {

'content-type': 'application/json' // 默认值

},

success (res) {

console.log(res.data)

}

})

#### data 参数说明

最终发送给服务器的数据是 String 类型，如果传入的 data 不是 String 类型，会被转换成 String 。转换规则如下：

对于 GET 方法的数据，会将数据转换成 query string

（encodeURIComponent(k)=encodeURIComponent(v)&encodeURIComponent(k)=encodeURIComponent(v)...）

* 对于 POST 方法且 header['content-type'] 为 application/json 的数据，会对数据进行 JSON 序列化
* 对于 POST 方法且 header['content-type'] 为 application/x-www-form-urlencoded 的数据，会将数据转换成 query string

（encodeURIComponent(k)=encodeURIComponent(v)&encodeURIComponent(k)=encodeURIComponent(v)...）

如果接口很多，可以封装下，在组件里面调用可以很方便：

第一步：把接口中相同的部分拿出来放在一个文件中，步骤如下：

1、在项目里面新建一个config.js文件：

例：

//全局配置

//"https://www.easy-mock.com/mock/test/" 指的是接口中相同的部分

//appKey: "fsdfdsd212122" 指的是其它相同的部分，也可以拿出来

const config = {

api\_base\_url: "https://www.easy-mock.com/mock/test/",

appKey: "fsdfdsd212122"

}

//单个导出

export default config

let func1 = function () {

}

// 如果有多个相同的配置，可以多个导出

export { config , func1}



第二步：把相同部分的文件导入一个http.js文件中，步骤如下：

1、在项目中新建一个utils的文件夹，在此文件夹里新建一个http.js文件

例：

//导入config.js文件里面一个

// import {config} from '../config.js'

//导入config.js文件里面两个或多个

import {config , func1} from '../config.js'

//根据返回的状态码code值，给予对应的提示 （此步可以自己添加）

const tips = {

1: '抱歉，出错了',

1005: 'appKey无效',

3000: "期刑不存在"

}

//ES6封装方法 类名封装

class HTTP{

request (params){ //params是形参，是每个请求接口要传入的对象

if(!params.method) {

params.method = "GET" // 如果没有在视图的js中传method 那就默认为GET方式

}

wx.request({

url: config.api\_base\_url + params.url, //相同部分和每个接口不同的部分拼接

method: params.method, //请求的方式

data: params.data, //请求的参数

header: {

"content-type": 'application/json', //默认的类型

"appkey" : config.appKey //config.js文件中其它相同部分

},

success: (res) => { //第一种返回的状态success

let code = res.statusCode.toString() //根据返回的code值来判断

//这个code=res.statusCode.toString()是根据后台返回的对象里找，不同接口返回的对象也不同，这个也是一个例子

if(code.startsWith('2')){ //判断code值为2开头的参数才是成功

params.success(res.data)

}else { //否则返回其它的参数为错误

let error\_code = res.data.error\_code

this.\_show\_error(error\_code)

}

},

fail: (err) => { //第二种返回的状态fail 只有在断网或者接口没有请求成功的情况下出来fail

this.\_show\_error(1)

}

})

}

//报错处理 —— 私有方法 （根据情况而定，给页面的提示）

\_show\_error(error\_code) {

if(!error\_code) error\_code = 1

wx.showToast({

title: tips[error\_code],

icon: "none",

duration: 2000

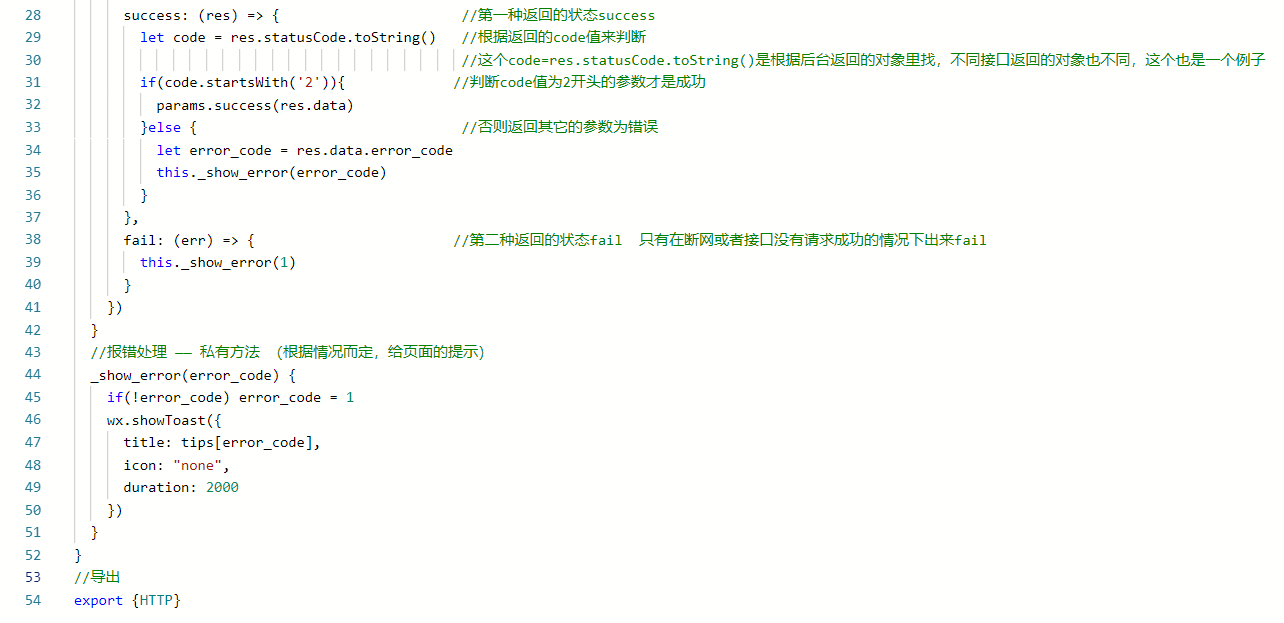
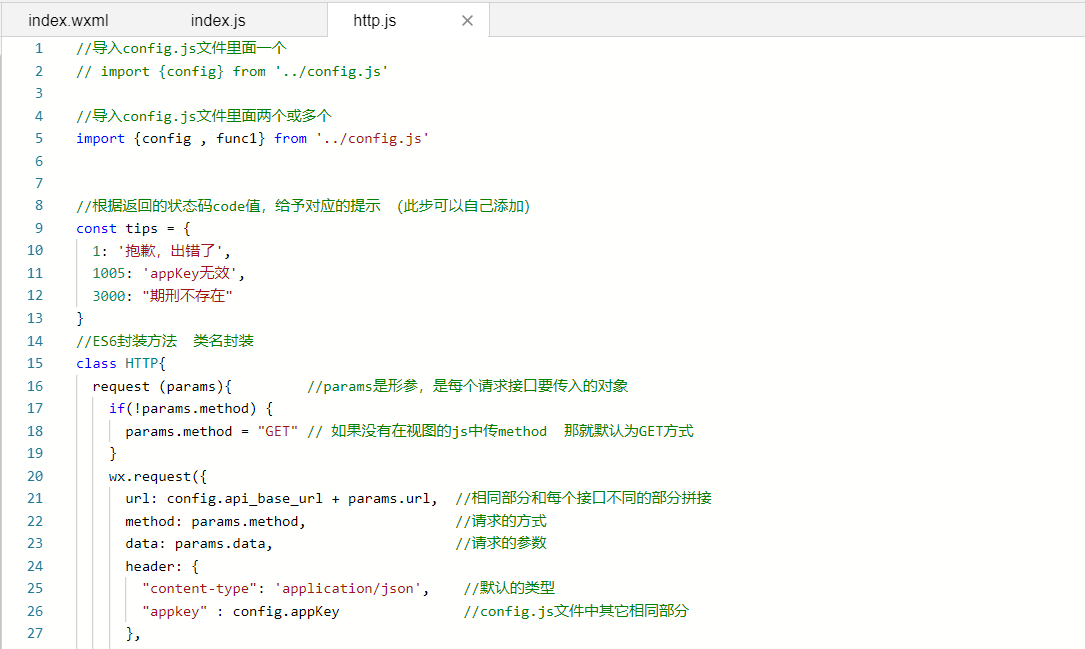
})

}

}

//导出

export {HTTP}



第四步：把封装好的wx请求，放在每个视图相对应请求具体接口的js文件中，需要新建models文件夹，在此文件夹里面新建每个相对应请求具体接口的js文件

【注：可跳过第三步操作，直接用第四步操作（MVC模式），把模块细化；（MVC：M指数据保存 V指用户界面 C指逻辑处理）】

第1步：在项目中新建models文件夹，在此文件夹中新建index.js

例：//把封装好的http.js文件引入

import { HTTP } from '../utils/http.js'

//通过extends继承父类（也可以实例化后调用） ES6用法

class IndexModel extends HTTP {

getDataList(sCallBack) {

//封装后的请求调用

this.request({

url:'list',

success: (res) => {

sCallBack(res)

}

})

}

}

//导出

export { IndexModel }



第2步：把models里面的index.js导入到每个视图里面的js逻辑层

例：

//index.js

//获取应用实例

import { IndexModel } from '../../models/index.js' //引入相应的请求接口models文件夹里面的index.js文件

let index = new IndexModel() //实例化

const app = getApp()

Page({

/\*\* \* 页面的初始数据 \*/

data: {

dataList: []

},

/\*\* \* 生命周期函数--监听页面加载 \*/

onLoad: function(){

//用回调函数的方法解决异步问题

index.getDataList((res) => { //调用获取到近回的数据res

console.log(res)

this.setData({

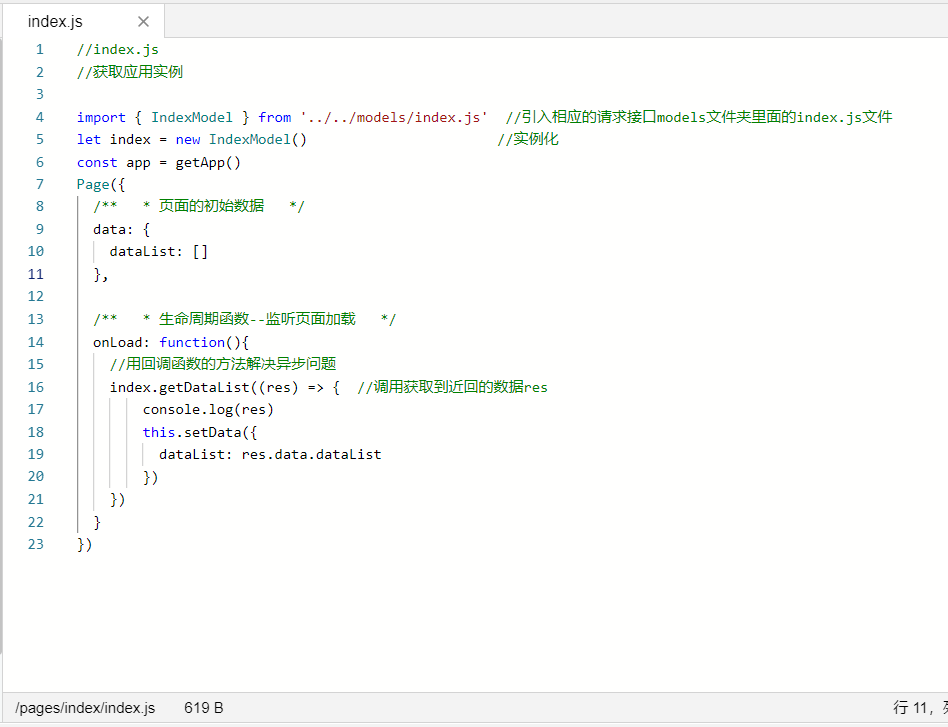
dataList: res.data.dataList

})

})

}

})



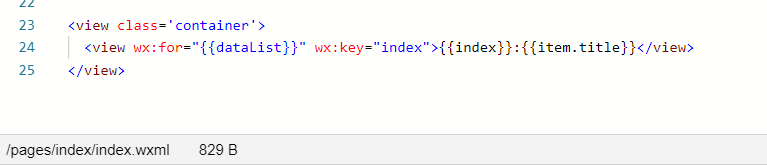
第3步：拿到的请求数据，进行视图的渲染

例：

<view class='container'>

<view wx:for="{{dataList}}" wx:key="index">{{index}}:{{item.title}}</view>

</view>



第三步的操作，按照规范不建议用这一步，可以跳过第三步，直接用第四步

第三步：把封装好的wx.request可以应用在视图中

例：在视图index文件夹中index.js文件引入

//index.js

//获取应用实例

import { HTTP } from '../../utils/http.js' //引入封装好的请求方式

let http = new HTTP() //实例化

const app = getApp()

Page({

/\*\*

\* 页面的初始数据

\*/

data: {

dataList: []

},

/\*\*

\* 生命周期函数--监听页面加载

\*/

onLoad: function (options) {

//请求调用

http.request({

url: "list",

success: (res) => {

console.log(res)

this.setData({

dataList: res.data.data.dataList

})

}

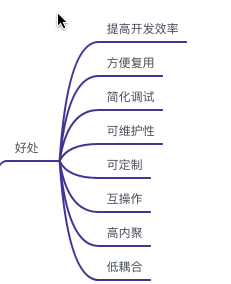
})

}

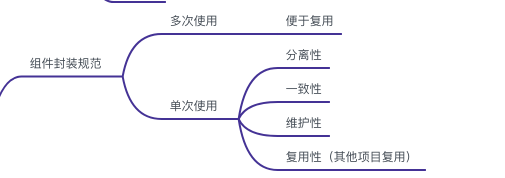
})

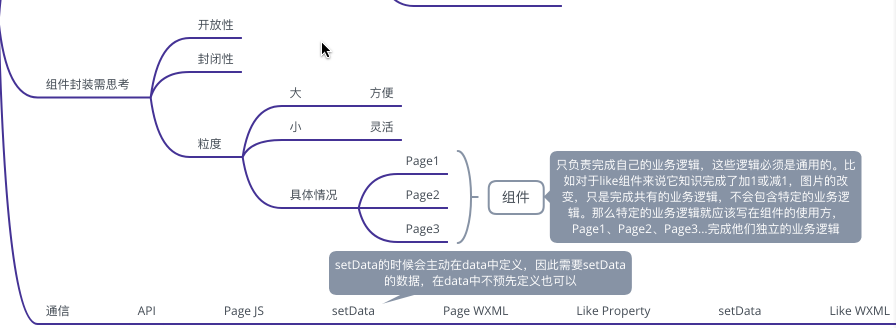


小程序组件：

  
高内聚：完整的组件，互不影响

低藕合：





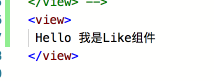
开放性：外来因素决定内部 比如：父组件（或后台数据）传递过来的数据来决定子组件里面的状态

封闭性：不需要外来因素来决定内部决态

粒度：指的是视图可以拆分多少个组件或者指的是这个视图由多少个组件组成的

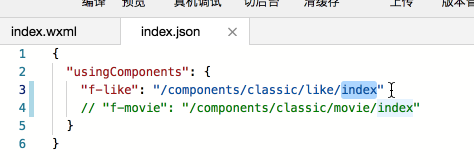
创建组件，并使用：

1、在components组件中写好内容：



2、在pages视图中引入相对应的组件

（1）在pages里面相应对json里面写入：



（2）在pages里面相对应的wxml里面写入：



组件通信：

<http://note.youdao.com/noteshare?id=def437cdfedc3167bf076787e56be195>

# Behavior：

<https://developers.weixin.qq.com/miniprogram/dev/framework/custom-component/behaviors.html>

<https://developers.weixin.qq.com/miniprogram/dev/api/BackgroundAudioManager.html>

.comment-container>f-tag:nth-child(1)>view {

background-color: #fffbdd;

}

通过自定义组件标签修改组件内部样式

f-tag是一个组件

https://developers.weixin.qq.com/miniprogram/dev/component/open-data.html

https://developers.weixin.qq.com/miniprogram/dev/component/button.html授权