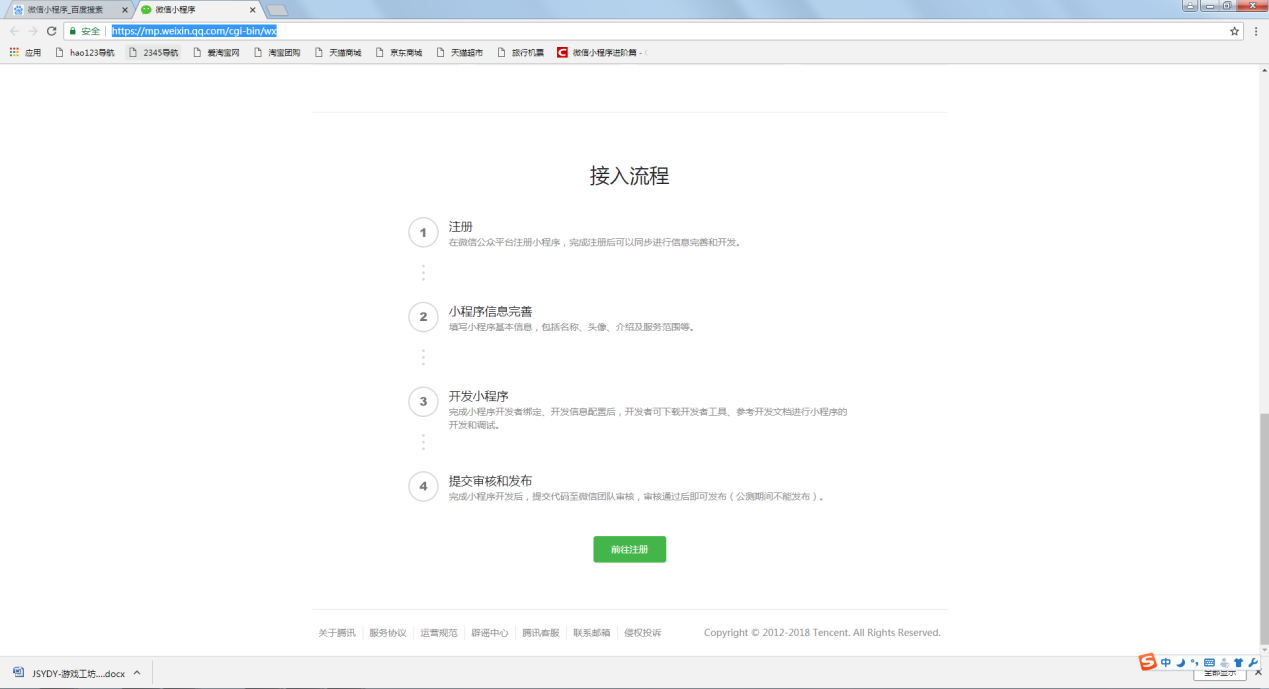
# 单元1微信web开发者工具

## 任务1.1 注册微信小程序账号

1.进入微信公众平台小程序模块：https://mp.weixin.qq.com/cgi-bin/wx。

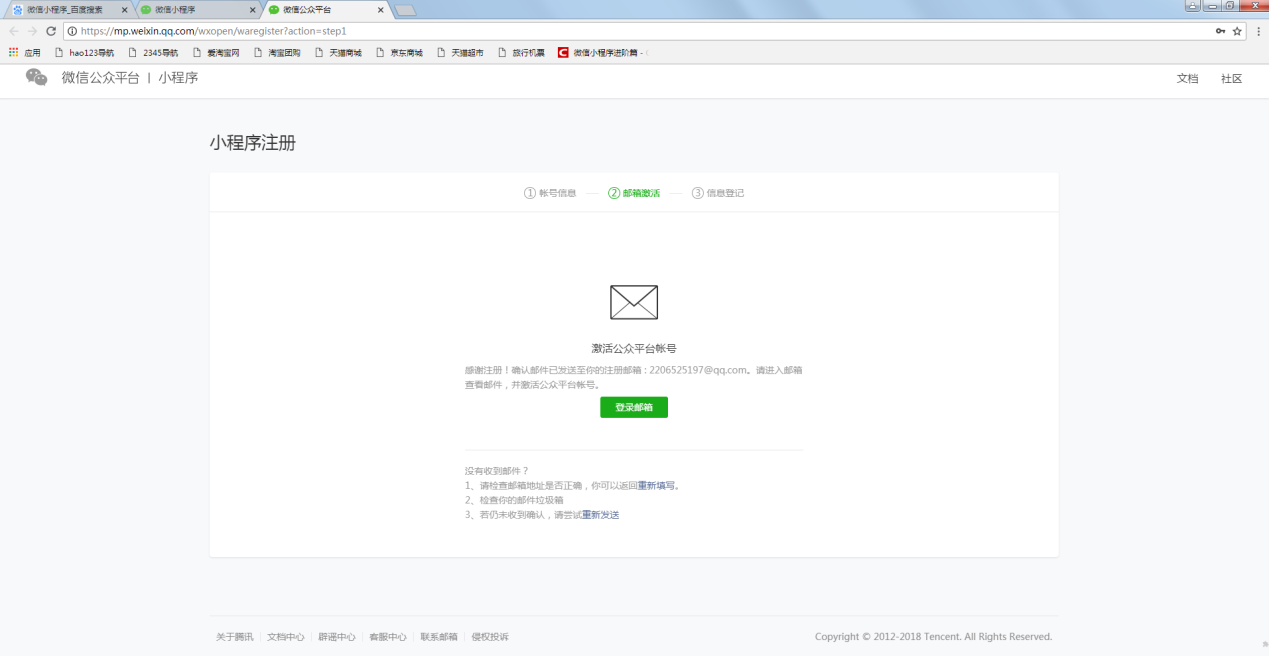
在页面最下方点击前往注册按钮。



2.根据页面要求，填写完注册信息。



3.根据页面提示，前往注册邮箱进行激活公众平台操作。

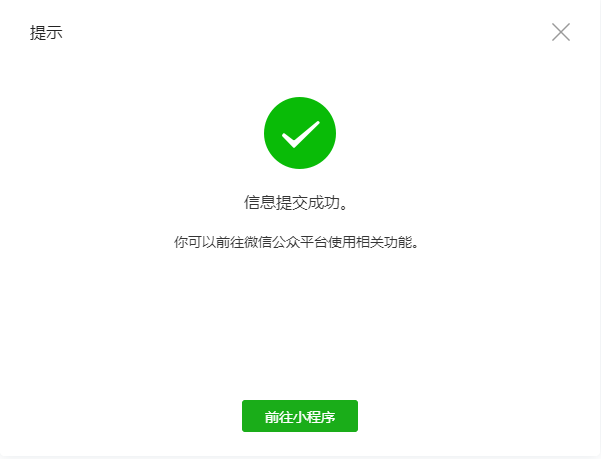


4.激活邮箱以后，进入信息登记页面。选择合适的主体类型，进行相关信息的登记。后续操作以“个人”为主体类型进行演示。



点击页面最下方的继续按钮。

5.根据提示，完成账号的申请。点击前往小程序按钮进入小程序配置页面。



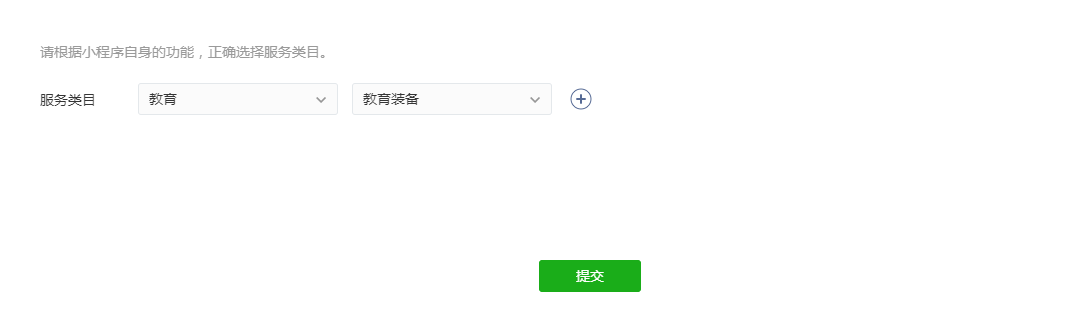
6.点击填写按钮进入填写小程序信息页面。



7.根据页面提示，完成信息的填写。



**注意：因为本项目虽然为小游戏，但是是以小程序开发方式实现开发，所以这里不能选择游戏类目，我们这以教育类目为例。**



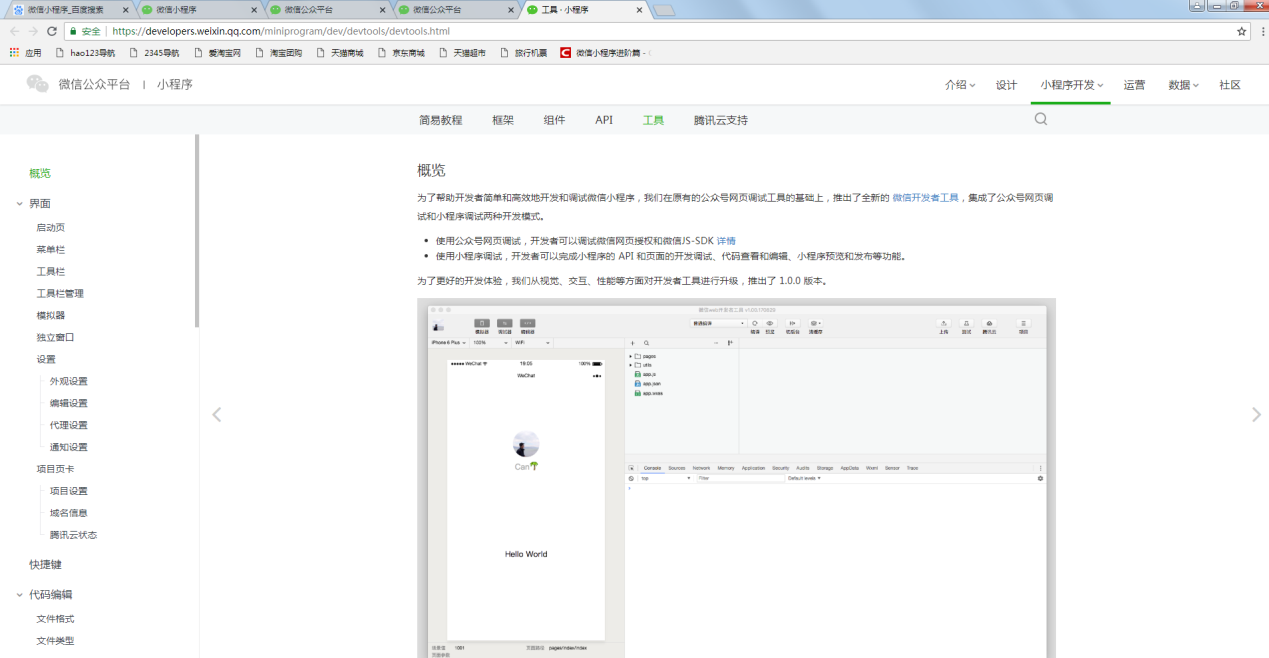
8.点击提交，就完成了小程序信息的录入。

## 任务1.2 开发工具

1.下载开发工具.



进入页面，继续点击微信开发者工具按钮

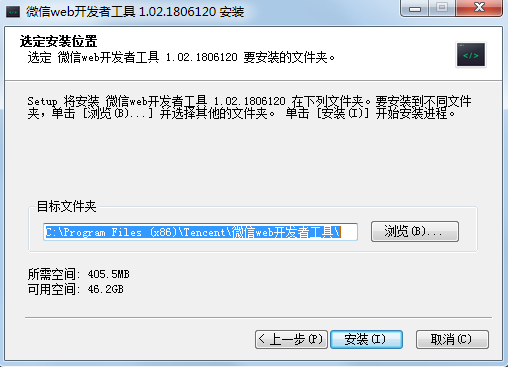


选择适合你机型的版本下载工具。



2.安装开发者工具。

点击运行刚下载的exe文件，根据提示一步步执行，最后点击安装按钮。



耐心等待以后，就提示安装完成了。

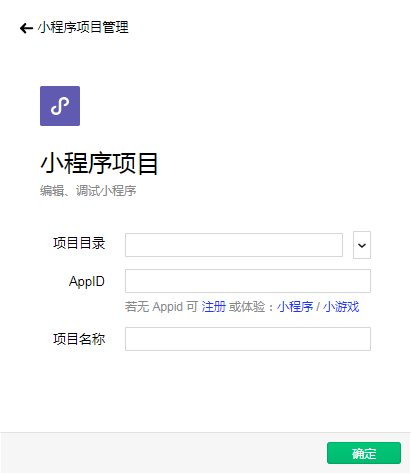


## 任务1.3新建小程序

1.选择小程序项目。



2.根据提示填写项目信息。



**注意：AppID可以直接选择小程序/小游戏选项，但不能进行小程序发布的操作。**

查看AppID，进入开发者公众平台，点击如下图所示按钮。



下图所示就是我们需要用的AppID。

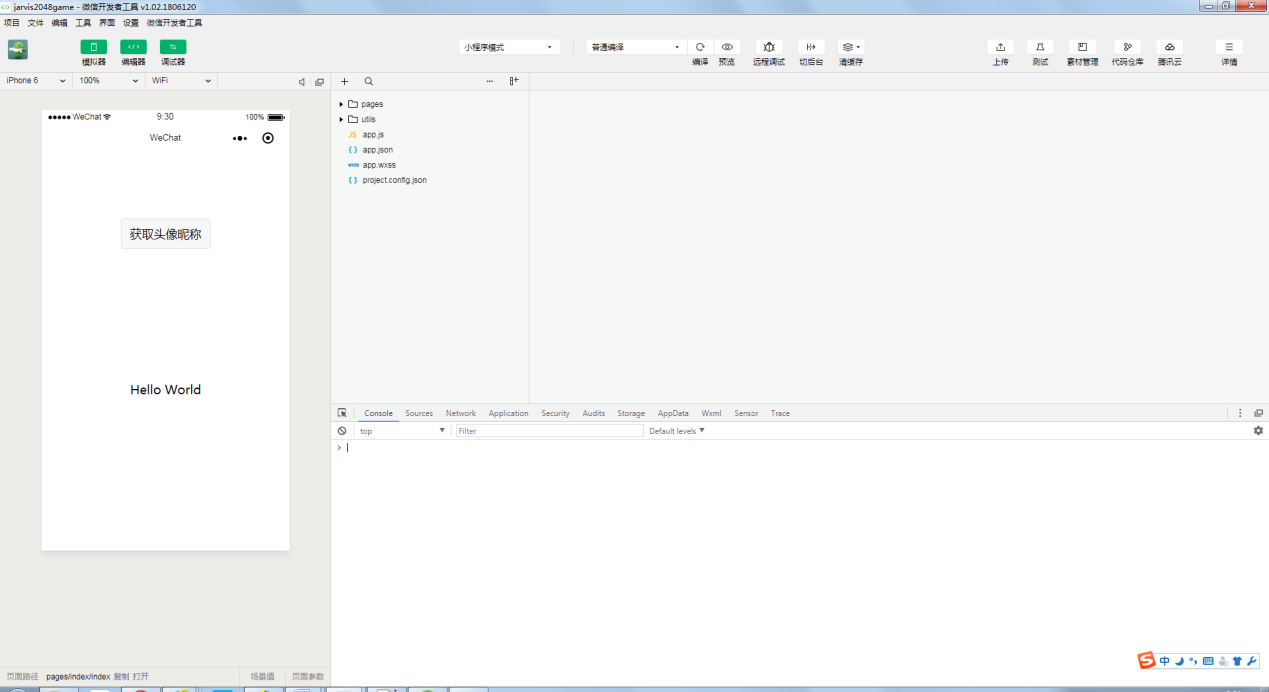


3.当填写完所有信息之后，会出现下图所示样子。



**注意：勾选并且选择简历普通快速启动模板。**

4.点击确认按钮，新建的第一个项目就完成了。



# 单元22048小游戏

## 任务2.1 启动页设计

1. 整理初始代码，删除部分文件及代码

1.1 打开app.js文件，删除自带代码，只保留以下代码。

//app.js

App({

})

1.2 删除logs文件包。

1.3 打开app.json文件，删除"pages/logs/logs"，保留以下代码。

{

"pages":[

"pages/index/index"

],

"window":{

"backgroundTextStyle":"light",

"navigationBarBackgroundColor": "#fff",

"navigationBarTitleText": "WeChat",

"navigationBarTextStyle":"black"

}

}

1.4 删除index.wxml和index.wxss中所有代码。

1.5 打开index.js文件，删除自带代码，只保留以下代码。

//index.js

//获取应用实例

const app = getApp()

Page({

})

1.6 打开app.wxss文件，修改代码。

/\*\*app.wxss\*\*/

.container {

height:100%;

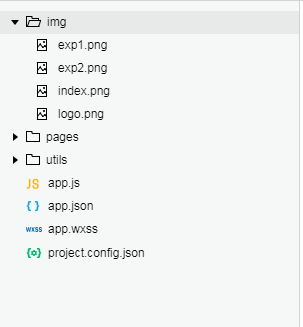
display:flex;

flex-direction:column;

align-items:center;

}

1.7 将附件中的img文件夹拷贝进项目，放在与pages同级目录下。



2. 启动页的视图层渲染

2.1 修改pages/index/index.wxml文件。

<!--index.wxml-->

<view class="container">

<text class='textStytle'>每次可以选择上下左右其中一个方向去滑动，每滑动一次，所有的数字方块都会往滑动的方向靠拢外，系统也会在空白的地方乱数出现一个数字方块，相同数字的方块在靠拢、相撞时会相加。不断的叠加最终拼凑出2048这个数字就算成功。</text>

<button wx:if="{{!hasUserInfo}}" open-type="getUserInfo" bindgetuserinfo="getUserInfo">开始游戏</button>

<block wx:else>

<view>

<image class='logo' src='../../img/logo.png'></image>

<view class='load'>

<text class='{{current == 0 ? "sct": ""}}'></text>

<text class='{{current == 1 ? "sct": ""}}'></text>

<text class='{{current == 2 ? "sct": ""}}'></text>

<text class='{{current == 3 ? "sct": ""}}'></text>

</view>

</view>

</block>

</view>

2.2 修改pages/index/index.wxss文件。

/\*\*index.wxss\*\*/

.container {

background-color: #faf8ee;

height: 100vh;

width: 100vw;

}

.logo {

width: 100px;

height: 100px;

border-radius: 20px;

}

.load {

text-align: center;

margin-top: 10px;

}

.load text {

display: inline-block;

width: 6px;

height: 6px;

border: 1px solid #ccc;

border-radius: 4px;

margin: 0 5px;

}

.load text.sct {

background-color: #ccc;

}

.textStytle{

padding: 10vh;

word-break: break-all;

font-size:28rpx;

}

3. 启动页逻辑层开发

3.1 定义数据变量。

打开index.js文件，在page函数下定义data对象，添加如下代码：

data: {

current: 0, //控制小圆点运动

hasUserInfo: false //判断是否已经登陆

}

3.2 定义等待加载动画方法。

在页面最上方定义全局变量，添加如下代码：

var timer;

在page下定义一个定时器，代码如下：

load: function () {

console.log("load")

var n = 1;

timer = setInterval(() => {

this.setData({

current: this.data.current + 1

});

if (this.data.current > 3)

this.setData({

current: 0

});

n++;

}, 400);

}

3.3 定义微信授权获取个人信息方法。

getInfo:function () {

wx.getUserInfo({

success: ({ userInfo }) => {

//获取当前用户的信息

console.log(userInfo);

}

});

}

3.4 页面加载监听。

在onLoad方法中添加获取微信个人信息，判断是否已用户授权。添加如下代码：

onLoad:function(){

// 获取用户信息

wx.getSetting({

success: res => {

if (res.authSetting['scope.userInfo']) {

// 已经授权，可以直接调用 getUserInfo 获取头像昵称，不会弹框

this.setData({

hasUserInfo:true

})

this.load();

this.getInfo();

}

}

})

}

3.5 页面按钮监听。

为页面上的开始游戏按钮添加单击事件。

getUserInfo:function(){

this.setData({

hasUserInfo: true

})

this.load();

this.getInfo();

}

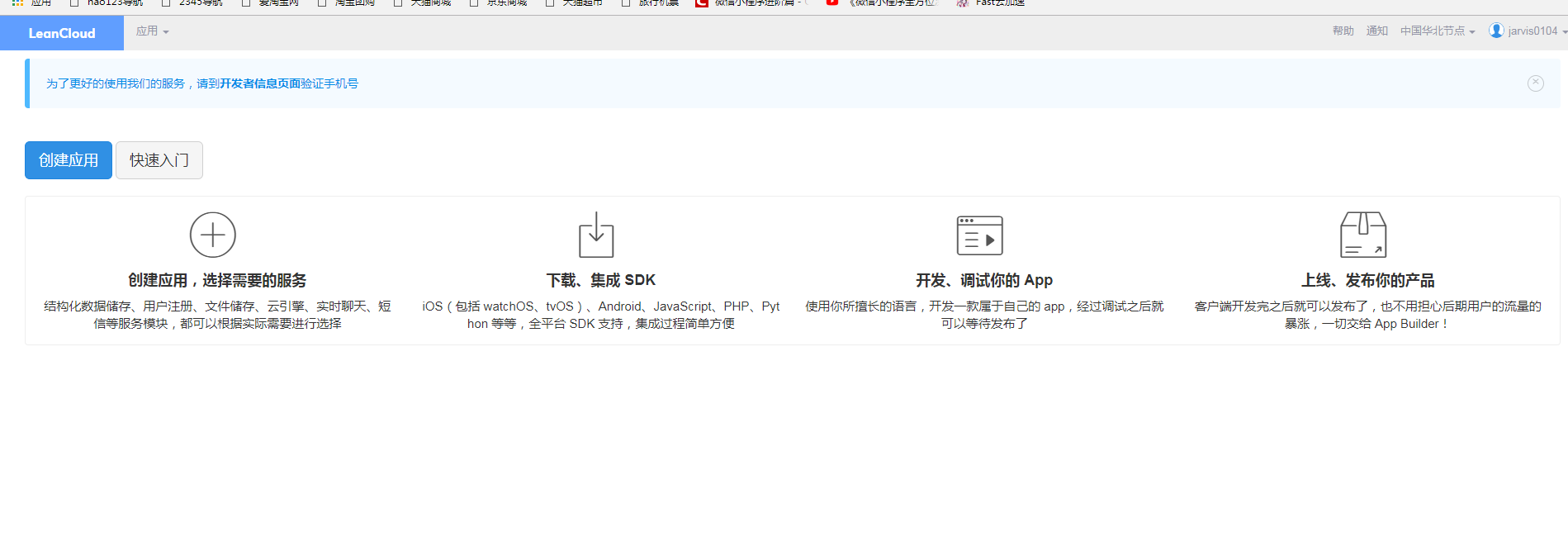
## 任务2.2 leancloud的使用

1. 注册leancloud账号

访问<https://leancloud.cn/dashboard/login.html#/signin>注册页面，根据提示注册账号。

2. 创建应用

根据提示创建应用。

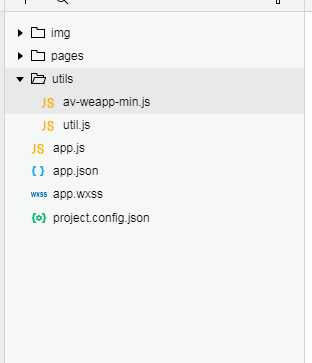


3. 下载存储SDK

前往 <https://releases.leanapp.cn/#/leancloud/javascript-sdk/releases，下载最新版本的 av-weapp-min.js>。

4. 引用SDK

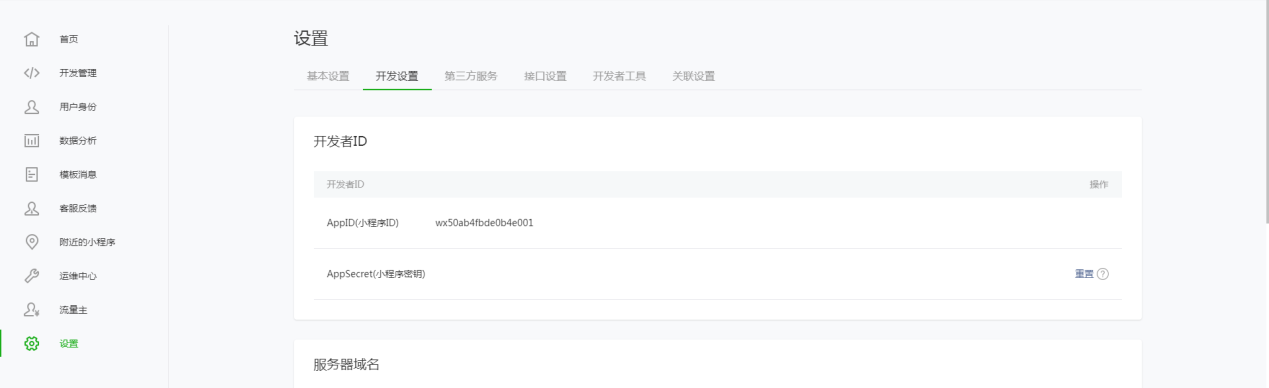
将下载的js文件拷贝至utils目录下。



5. 配置SDK环境

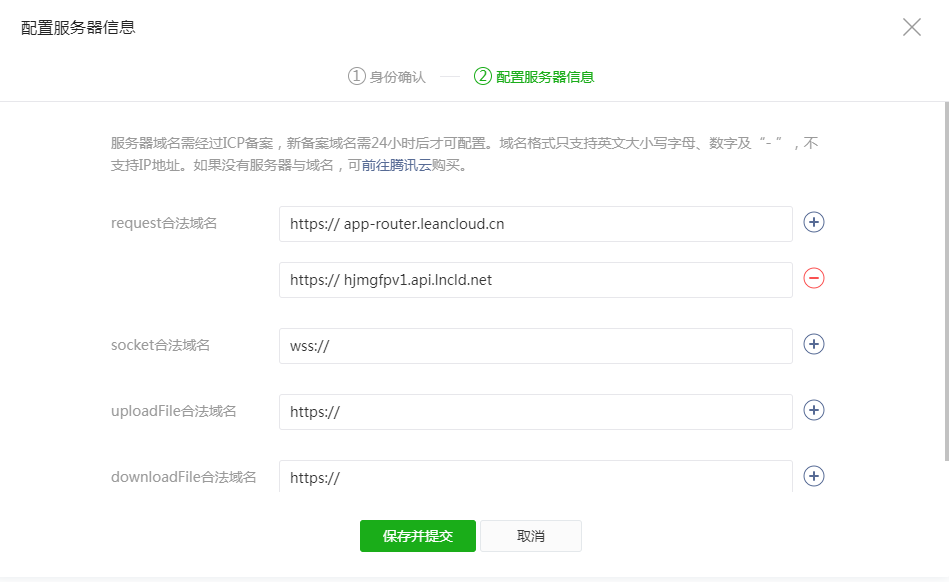
5.1 获取微信信息。

在微信公众平台获取AppID（小程序ID）和AppSecret（小程序密钥）。



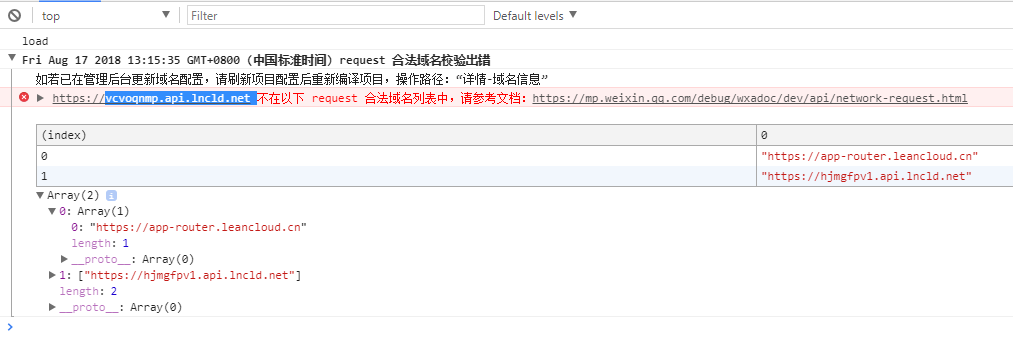
5.2 配置微信小程序有效服务器域名。

将app-router.leancloud.cn和hjmgfpv1.api.lncld.net配置进微信小程序服务器有效域名中，方法如下图：





**注意：此处域名配置可能会根据项目而有变化，在后续实际操作中，如遇下图所示错误提示，将图中域名按照上述方法添加即可。**



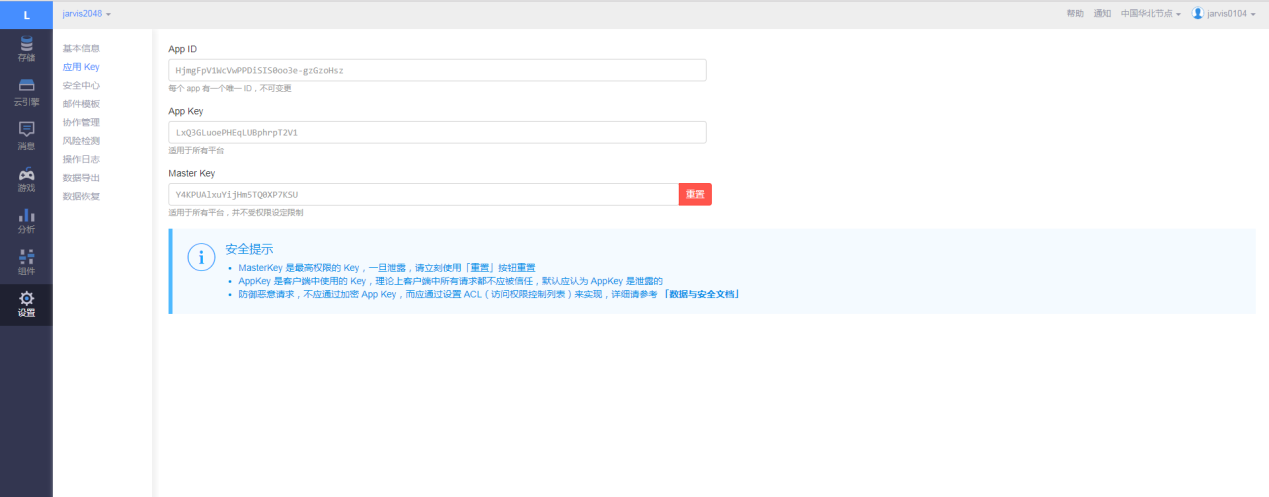
5.3 配置微信信息

进入leancloud控制台，点击组件中的社交，配置微信id和key。



5.4 获取leancloud基本参数。

进入leancloud自控制台，点击设置中的应用key，如图所示：



5.5 配置参数

打开app.js文件，注意替换自己项目的key和id，添加如下代码：

const AV = require('./utils/av-weapp-min');

AV.init({

appId: 'HjmgFpV1WcVwPPDiSIS0oo3e-gzGzoHsz',

appKey: 'LxQ3GLuoePHEqLUBphrpT2V1',

});

## 任务2.3 用户信息的存储

1. 引用av-weapp-min.js文件

打开index.js文件，在最上方添加引用代码：

const AV = require('../../utils/av-weapp-min');

2. 微信登陆

使用leancloud微信登陆方法登陆，替换getInfo方法，代码如下：

getInfo:function () {

AV.User.loginWithWeapp().then(function (user) {

console.log("登陆成功");

}), function (error) {

// 异常处理

console.log(error);

clearInterval(timer);

this.setData({

hasUserInfo: false

})

}

}

3. 判断是否是第一次登陆

在getInfo方法下console.log("登陆成功");下添加如下代码：

const users = AV.User.current();

if (users.attributes.url){

console.log("非第一次登陆");

clearInterval(timer);

}else{

console.log("第一次登陆");

}

4. 第一次登陆

该账号第一次登陆，需要将微信用户信息添加进数据库。在在getInfo方法下console.log("第一次登陆");下添加如下代码：

wx.getUserInfo({

success: ({ u }) => {

// 更新当前用户的信息

console.log(u);

users.set('score', 2);

users.set('count', 0);

users.set('name', u.userInfo.nickName);

users.set('url', u.userInfo.avatarUrl);

users.save();

clearInterval(timer);

}

});

5. 定义全局变量存储用户信息

5.1 打开app.js文件，在App下添加变量，代码如下：

globalData: {

userInfo: null

}

5.2 保存用户信息.

非第一次登陆的时候，将数据库信息保存到本地，在getInfo方法下console.log("非第一次登陆");下添加如下代码：

app.globalData.userInfo = users.attributes;

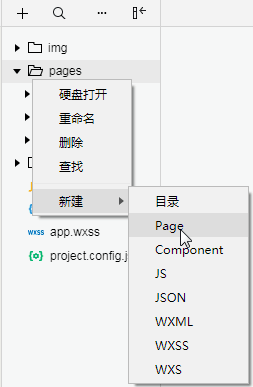
在第一次登陆的时候，将用户信息保存到本地，在getInfo方法下users.save();下添加如下代码：

app.globalData.userInfo = users.attributes;

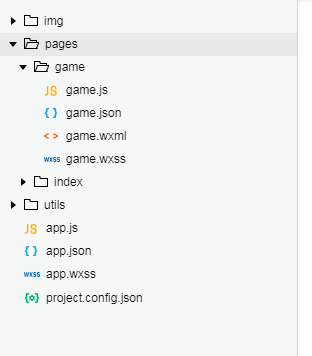
## 任务2.4游戏页设计

1. 新建页面

新建pages/game目录，在该目录下新建game页面。



最后目录结构如下图所示：



2. 页面跳转

打开index.js文件，在getInfo方法中，在clearInterval(timer);之后进行页面跳转，此处有两个地方需要添加：

wx.redirectTo({

url: '../game/game'

})

3. 设置页面不可滑动

打开game.json文件，添加如下代码：

{

"disableScroll": true

}

4. 游戏页的视图层渲染

4.1 项目公共设置。

修改app.json文件，替换window字段下代码：

"window":{

"backgroundTextStyle":"light",

"navigationBarBackgroundColor": "#faf8ee",

"navigationBarTitleText": "2048",

"navigationBarTextStyle":"black"

}

4.2 项目公共样式。

修改app.wxss文件，替换container字段下代码：

.container {

height:100%;

display:flex;

flex-direction:column;

align-items:center;

justify-content:space-between;

padding:200rpx0;

box-sizing:border-box;

}

4.3 游戏页面的渲染。

修改game.wxml文件，替换如下代码：

<viewclass='container'>

<!-- loading -->

<loadinghidden="{{hidden}}">

加载中...

</loading>

<viewclass='head'>

<viewclass='lside'>

<textclass='title'>2048</text>

<textclass='play' bindtap='gameStart'>{{start}}</text>

</view>

<viewclass='rside'>

<text>score <textclass='score'>{{score}}</text></text>

</view>

</view>

<viewclass="game" bindtouchstart="touchStart" bindtouchmove="touchMove" bindtouchend="touchEnd" disable-scroll="true">

<!--矩阵 -->

<viewwx:for="{{num}}" wx:for-item="row" wx:key="row.key" >

<viewclass='cell' wx:for="{{row}}" wx:for-item="cell" wx:key="cell.key">

<viewclass='cell-con cell-con-{{cell}}'>

{{cell}}

</view>

</view>

</view>

<!-- game over -->

<viewclass='game-over' hidden='{{!over}}'>

<textclass='nowScore'>历史最高分：{{bestScore}}</text>

<textclass='nowScore'>本次成绩：{{score}}</text>

<textclass='pro'>{{endMsg}}</text>

</view>

</view>

<viewclass='bottom'>

<viewclass='bottom\_left'>

<text>best score: {{bestScore}}</text>

<text>最高连击: {{bestCount}}</text>

</view>

<textclass='order'bindtap='record\_click'>排行榜</text>

</view>

</view>

4.4 为游戏页面添加样式。

修改game.wxss文件，添加如下代码：

.container {

color:#776e65;

background-color:#faf8ee;

height:100vh;

position:relative;

}

.head {

text-align:center;

position:absolute;

top:5vh;

width:93vw;

justify-content:space-around;

display:flex;

flex-direction:row;

}

.lside {

width:130px;

height:100px;

display:flex;

flex-direction:column;

}

.title {

font-size:40px;

font-weight:bold;

z-index:2;

}

.play {

font-size:20px;

background-color:#bbada0;

color:#fff;

border-radius:5px;

width:130px;

text-align:center;

height:40px;

line-height:40px;

font-weight:bold;

z-index:2;

position:relative

}

.rside {

background-color:#bbada0;

width:100px;

height:100px;

padding-top:10px;

border-radius:5px;

}

.rside text {

display:block;

text-align:center;

font-size:30px;

font-weight:bold;

}

.score {

color:#fff;

}

/\* =========================== game ==================== \*/

.game {

margin-top:12vh;

position:absolute;

padding-top:5vw;

padding-left:1vw;

background:#bbada0;

border-radius:6px;

width:93vw;

height:93vw;

box-sizing:border-box;

touch-action:none;

}

.cell {

width:19vw;

height:19vw;

margin-left:3vw;

margin-bottom:3vw;

float:left;

border-radius:5px;

box-sizing:border-box;

background:rgba(238, 228, 218, 0.35);

overflow:hidden;

}

.cell-con {

width:19vw;

height:19vw;

line-height:19vw;

position:absolute;

font-size:35px;

font-weight:bold;

text-align:center;

border-radius:5px;

}

.cell-con-2 { background:#eee4da; }

.cell-con-4 { background:#ede0c8; }

.cell-con-8 {

color:#f9f6f2;

background:#f2b179;

}

.cell-con-16 {

color:#f9f6f2;

background:#f59563;

}

.cell-con-32 {

color:#f9f6f2;

background:#f67c5f;

}

.cell-con-64 {

color:#f9f6f2;

background:#f65e3b;

}

.cell-con-128 {

color:#f9f6f2;

background:#edcf72;

font-size:30px;

}

.cell-con-256 {

color:#f9f6f2;

font-size:30px;

background:#edcc61;

}

.cell-con-512 {

color:#f9f6f2;

font-size:30px;

background:#edc850;

}

.cell-con-1024 {

color:#f9f6f2;

font-size:25px;

background:#edc53f;

}

.cell-con-2048 {

color:#f9f6f2;

font-size:25px;

background:#edc22e;

}

.game-over {

position:absolute;

z-index:3;

background-color:rgba(255, 255, 255, 0.8);

width:100%;

height:100%;

top:0;

left:0;

text-align:center;

padding-top:20vw;

box-sizing:border-box;

}

.game-over text {

display:block;

}

.nowScore {

font-size:20px;

margin-bottom:10px;

}

.pro {

font-weight:bold;

font-size:30px;

}

.bottom {

bottom:1vh;

position:absolute;

display:flex;

flex-direction:row;

justify-content:space-between;

width:93vw;

height:130rpx;

}

.bottom\_left {

display:flex;

flex-direction:column;

text-align:left;

margin-top:auto;

margin-bottom:auto;

}

.order{

font-size:20px;

background-color:#bbada0;

color:#fff;

border-radius:5px;

width:130px;

text-align:center;

height:40px;

line-height:40px;

margin-top:auto;

margin-bottom:auto;

}

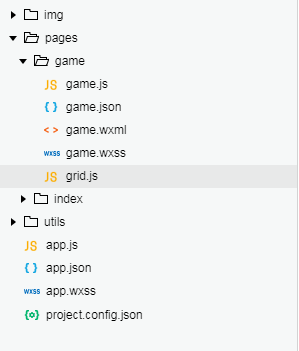
# 单元3游戏页逻辑层开发

## 任务3.1 游戏页表格的实现

1. grid.js表格工具类

1.1 新建js文件。

打开pages/game目录，新建grid.js文件，作为滑动表格的操作工具类。



1.2 定义初始化方法。

在grid.js文件下添如下代码，传入size参数，表示构建的表格行列数量。

function Board(size) {

this.size = size;

}

Board.prototype = {

};

module.exports = Board;

1.3 形成空矩阵。

在prototype在添加init方法，代码如下：

init() { // 形成一个空矩阵

var grid = [];

for(var i = 0; i < this.size; i++) {

grid[i] = [];

for(var j = 0; j < this.size; j++) {

grid[i].push("");

}

}

return grid;

}

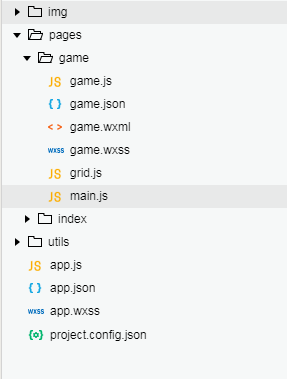
在Board方法下调用init方法，并接收返回值。添加代码如下：

this.grid = this.init();

2. main.js游戏操作类

2.1 新建js文件。

打开pages/game目录，新建main.js文件，作为游戏操作工具类。



2.2 定义初始化方法。

在main.js文件下添如下代码，传入size参数，表示构建的表格行列数量。

function Main(size) {

this.size = size;

}

Main.prototype = {

};

module.exports = Main;

2.3 调用grid.js文件。

在main.js文件下添加grid.js文件的引用，在页面最上方添加如下代码：

var Board = require("./grid.js");

在prototype在添加init方法，代码如下：

init() {

this.board = newBoard(this.size);

this.bproto = this.board.\_\_proto\_\_;

}

在Main方法下调用init方法，添加如下代码：

this.init();

3. 页面实现

3.1 调用main.js和grid.js文件。

打开game.js文件，在页面最上方添加如下代码：

var Board = require("./grid.js");

var Main = require("./main.js");

3.2 添加变量。

在data字段下添加如下变量：

data: {

hidden: false,

start: "开始游戏",

num: [],

score: 0, //此局分数

bestScore: 0, // 最高分

bestCount:0, //最高连击

maxCount: 0, //此局连击

endMsg: '',

over: false, // 游戏是否结束,

newRecord:false

}

3.3 初始化历史数据。

获取历史数据，在onReady方法下添加如下代码：

if (!wx.getStorageSync("bestCount"))

wx.setStorageSync('bestCount', 2);

if (!wx.getStorageSync("bestScore"))

wx.setStorageSync('bestScore', 0);

3.4 定义开始方法

定义gameStart开始方法，初始化Main，设置页面相关参数。

gameStart: function () { // 游戏开始

var main = new Main(4);

this.setData({

main: main,

});

this.data.main.\_\_proto\_\_ = main.\_\_proto\_\_;

this.setData({

hidden:true,

over: false,

score: 0,

bestCount: wx.getStorageSync('bestCount'),

bestScore: wx.getStorageSync('bestScore'),

num: this.data.main.board.grid,

newRecord:false

});

}

3.5 调用gameStart方法。

在onReady方法下调用gameStart方法，添加如下代码：

this.gameStart();

## 任务3.2 随机生成数据

1. 记录为空的格子

打开grid.js文件，在prototype下添加如下代码：

usefulCell() { // 记录为空的格子

var cells = [];

for(var i = 0; i <this.size; i++)

for(var j = 0; j <this.size; j++) {

if(this.grid[i][j] == "") { // 若可用则记录坐标

cells.push({

x: i,

y: j

});

}

}

return cells;

}

2. 随机选择可填充格子

打开grid.js文件，在prototype下添加如下代码：

selectCell() { // 从可填充格子中随机选一个

var cells = this.usefulCell();

if(cells.length) {

return cells[Math.floor(Math.random()\*cells.length)];

}

}

3. 判断是否还有可用格子

打开grid.js文件，在prototype下添加如下代码：

cellEmpty() { // 可用格子是否为空，为空返回true

return !this.usefulCell().length;

}

4. 添加初始数据

4.1 定义随机生成数据变量。

打开main.js文件，在Main方法下this.init();上面添加如下代码：

this.startData = 2; // 初始填充2个数据

4.2 填充随机数据。

打开main.js文件，在prototype下添加如下代码：

setDataRandom() { // 随机填充

for(var i = 0; i <this.startData; i++) {

this.addRandomData();

}

},

addRandomData() { //填充数据

if(!this.board.cellEmpty()) {

var value = Math.random() <0.9 ? 2 :4;

var cell = this.board.selectCell();

cell.val = value;

this.update(cell);

}

},

update(cell) { // 更新数据

this.board.grid[cell.x][cell.y] = cell.val;

}

4.3 调用添加数据方法。

在init方法下添加如下代码：

this.setDataRandom(); // 随机填充

this.startData = 1;

## 任务3.3滑动合成算法

1. 页面加载完毕随机用数字2或4填充两个单元格。

2. 判断用户滑动方向。

使用touchStart事件函数获取起始位置touchStartX、touchStartY，touchMove获取终点位置touchEndX、touchEndY。

3. 根据滑动方向（假设向右滑动）移动表格以及相同项合并。

⚫ 将2048的棋盘生成4\*4的二维数组list，为空的空格用0表示。

// 比如棋盘数据如下

var grid = [

[2, 2, 0, 0],

[0, 0, 0, 0],

[0, 8, 4, 0],

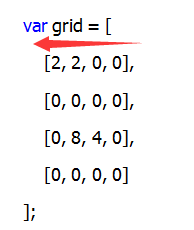
[0, 0, 0, 0]

];

⚫ 根据滑动方向生成4\*4二维数组。

**解释：**

① 向右滑动，那就是需要把左边的相同项合并到右边。如果顺序执行，操作就得从最后一位开始计算，倒退着完成合并。所以我们这边第一步先将数组自右往左重新排列生成一个数组，方便我们顺序执行。这样，我们只需要顺序把相同两项合并到前一项，将后一项置0就可以了。



图例：生成新list的方向。

var list = [

[0, 0, 2, 2], // 注意是0022不是2200，因为向右滑动所以从右边push入数组

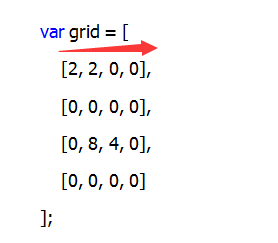
[0, 0, 0, 0],

[0, 4, 8, 0],

[0, 0, 0, 0]

];

② 向左滑动



图例：生成新list的方向

var list = [

[2, 2, 0, 0], //因为向左滑动所以从左边push入数组

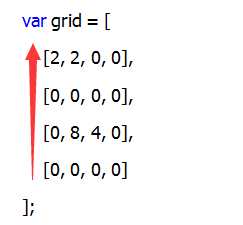
[0, 0, 0, 0],

[0, 8, 4, 0],

[0, 0, 0, 0]

];

③ 向下滑动，那就是需要把上边的相同项合并到下边。所以我们这边第一步先将数组自下往上重新排列生成一个数组，方便我们按顺序执行。



图例：生成新list的方向。

var list = [

[0, 0, 0, 2], //因为向下滑动所以从下边push入数组

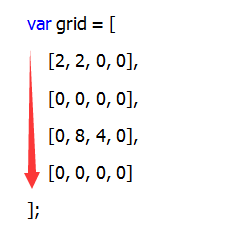
[0, 8, 0, 2],

[0, 4, 0, 0],

[0, 0, 0, 0]

];

④ 向上滑动



图例：生成新list的方向。

var list = [

[2, 0, 0, 0], //因为向下滑动所以从上边push入数组

[2, 0, 8, 0],

[0, 0, 4, 0],

[0, 0, 0, 0]

];

⚫ 将list的每一个小数组中的数字提到前面，0放到末尾。

**解释：**合并相同项的时候，无视中间为0的情况，将有数字的项按顺序挪到数组前面，0放到数组末尾。

list2 = [

[2, 2, 0, 0],

[0, 0, 0, 0],

[4, 8, 0, 0],

[0, 0, 0, 0]

];

⚫ 将相同值的单元格加起来，并将后面的一个单元格值变为0。

list2 = [

[4, 0, 0, 0],

[0, 0, 0, 0],

[4, 8, 0, 0],

[0, 0, 0, 0]

];

⚫ 将合并之后的list再做一次数字靠边操作（将list的每一个小数组中的数字提到前面，0放到末尾）。

**解释：**这样就可以把所有的数据都在一个方向上，避免出现合并以后，当中有空格的情况

⚫ 将list2回退为list并渲染数据到棋盘视图。

**解释：**完成以上所有操作是在list以倒叙形式情况下进行的。所以需要将倒叙的list在以正序的方法排列出来。

list = [

[0, 0, 0, 4],

[0, 0, 0, 0],

[0, 0, 8, 4],

[0, 0, 0, 0]

];

一次操作就完成了。

4. 重复步骤1。

5. 判断游戏是否结束。

⚫ 判断标准：4\*4单元格填满且任意一个单元格上下左右没有相同值的单元格。

6. 根据游戏结果给出相应提示。

## 任务3.4 游戏实现

1. 获取滑动方向

打开game.js文件，通过滑动事件监听touchStart，touchMove添加如下代码：

// 触摸

touchStartX: 0,

touchStartY: 0,

touchEndX: 0,

touchEndY: 0,

touchStart: function(ev) { // 触摸开始坐标

var touch = ev.touches[0];

this.touchStartX = touch.clientX;

this.touchStartY = touch.clientY;

},

touchMove: function(ev) { // 触摸最后移动时的坐标

var touch = ev.touches[0];

this.touchEndX = touch.clientX;

this.touchEndY = touch.clientY;

},

touchEnd: function () {

if (this.data.over){

return;

}

var disX = this.touchStartX - this.touchEndX;

var absdisX = Math.abs(disX);

var disY = this.touchStartY - this.touchEndY;

var absdisY = Math.abs(disY);

if (Math.max(absdisX, absdisY) >10) { // 确定是否在滑动

this.setData({

start: "重新开始",

});

var direction = absdisX > absdisY ? (disX<0 ? 1 : 3) : (disY <0 ? 2 : 0); // 确定移动方向

console.log("direction:" + direction)

}

}

2. 定义临时变量

打开main.js文件，在Main方法下this.init();之上添加如下代码：

this.score=0;

this.isChange=false;

this.isCombine = false;

3. 根据滑动方向生成数组列表

打开main.js文件，在prototype下添加代码，传入参数为滑动方向：

formList(dir) { // 根据滑动方向生成list的四个数组

var list = [[], [], [], []];

for(var i = 0; i <this.size; i++)

for(var j = 0; j <this.size; j++) {

switch(dir) {

case0:

list[i].push(this.board.grid[j][i]);

break;

case1:

list[i].push(this.board.grid[i][this.size-1-j]);

break;

case2:

list[i].push(this.board.grid[this.size-1-j][i]);

break;

case3:

list[i].push(this.board.grid[i][j]);

break;

}

}

return list;

}

4. 数字统一靠前

将数字统一放最前面，方便以后计算。根据条件判断出数据是否有移动，传入参数为数组，添加如下代码：

changeItem(item) { // 将 ['', 2, '', 2] 改为 [2, 2, '', '']

var cnt = 0;

for(var i = 0; i < item.length; i++){

if(item[i] != ''){

if (cnt != i) {

//判断是否数据有变化

this.isChange = true;

}

item[cnt++] = item[i];

}

}

for(var j = cnt; j < item.length; j++) {

item[j] = "";

}

return item;

}

5. 合并相同项

将整理好的数据进行相同项合并，记录下分值，根据条件判断出是否有合并操作，传入参数为数组列表，添加如下代码：

combine(list) { // 滑动时相同的合并

for(var i = 0; i < list.length; i++) // 数字靠边

list[i] = this.changeItem(list[i]);

for(var i = 0; i <this.size; i++) {

for(var j = 1; j <this.size; j++) {

if(list[i][j-1] == list[i][j] && list[i][j]!="") {

list[i][j-1] += list[i][j];

list[i][j] = "";

this.score = this.score + list[i][j - 1];

this.isCombine=true;

}

}

}

for (var i = 0; i < list.length; i++) // 再次数字靠边

list[i] = this.changeItem(list[i]);

return list;

}

6. 完整操作

根据滑动方向，完成游戏的一轮操作。将合并好的数据反向还原成原始顺序数据，传入参数为移动方向，添加如下代码：

move(dir) {

// 0:上, 1:右, 2:下, 3:左

var curList = this.formList(dir);

var list = this.combine(curList);

var result = [[],[],[],[]];

for(var i = 0; i <this.size; i++)

for(var j = 0; j <this.size; j++) {

switch (dir) {

case0:

result[i][j] = list[j][i];

break;

case1:

result[i][j] = list[i][this.size-1-j];

break;

case2:

result[i][j] = list[j][this.size-1-i];

break;

case3:

result[i][j] = list[i][j];

break;

}

}

this.board.grid = result;

if (this.isCombine){

this.setDataRandom();

this.isCombine = false;

this.isChange=false;

}else{

if (this.isChange) {

this.setDataRandom();

this.isChange = false;

}

}

return result;

}

7. 获取总分

打开main.js文件，在prototype下添加代码，获取分数：

getScore(){

returnthis.score;

}

8. 获取移动后的数组列表

打开game.js文件，在touchEnd方法下console.log("direction:" + direction)后面添加如下代码：

var data = this.data.main.move(direction);

9. 游戏页面视图变化

根据获得到的数组列表，更新游戏页面。循环遍历数组列表，获取最大数。如超过本地保存最大数，则将数据保存到缓存中。根据获得到的总分做同样的判断，存入缓存中。记录下是否有新纪录。传入参数为数组列表，添加如下代码：

updateView(data) {

var max = 0;

for(var i = 0; i <4; i++)

for(var j = 0; j <4; j++)

if(data[i][j] != ""&& data[i][j] > max)

max = data[i][j];

if (max >this.data.bestCount){

wx.setStorageSync('bestCount', max);

this.setData({

bestCount: max,

newRecord:true

});

}

if (this.data.main.getScore() >this.data.bestScore){

wx.setStorageSync('bestScore', this.data.main.getScore());

this.setData({

bestScore: this.data.main.getScore(),

newRecord: true

});

}

this.setData({

num: data,

score: this.data.main.getScore(),

maxCount:max

});

}

在touchEnd方法下var data = this.data.main.move(direction);后面添加调用方法代码：

this.updateView(data);

10. 结束游戏

10.1 打开main.js文件，在prototype下添加代码，进行游戏是否结束判断：

isOver() { // 游戏是否结束，结束条件：可用格子为空且所有格子上下左右值不等

this.board.\_\_proto\_\_ = this.bproto;

if (!this.board.cellEmpty()) {

returnfalse;

} else {

for (var i = 0; i <this.size; i++) // 左右不等

for (var j = 1; j <this.size; j++) {

if (this.board.grid[i][j] == this.board.grid[i][j - 1])

returnfalse;

}

for (var j = 0; j <this.size; j++) // 上下不等

for (var i = 1; i <this.size; i++) {

if (this.board.grid[i][j] == this.board.grid[i - 1][j])

returnfalse;

}

}

returntrue;

}

10.2 打开game.js文件，添加如下代码，进行游戏结束时的页面更新：

gameOver: function() { // 游戏结束

this.setData({

over: true

});

if (this.data.newRecord &&this.data.maxCount >= 2048){

this.setData({

endMsg: '恭喜达到2048,创造新纪录！'

});

}elseif (this.data.maxCount >= 2048) {

this.setData({

endMsg: '恭喜达到2048！'

});

} elseif (this.data.newRecord) {

this.setData({

endMsg: '创造新纪录！'

});

}else{

this.setData({

endMsg: '游戏结束！'

});

}

}

10.3 判断游戏是否结束。

替换touchEnd方法，判断游戏是否结束：

touchEnd: function () {

var disX = this.touchStartX - this.touchEndX;

var absdisX = Math.abs(disX);

var disY = this.touchStartY - this.touchEndY;

var absdisY = Math.abs(disY);

if (this.data.main.isOver()) { // 游戏是否结束

this.gameOver();

} else {

if (Math.max(absdisX, absdisY) >10) { // 确定是否在滑动

this.setData({

start: "重新开始",

});

var direction = absdisX > absdisY ? (disX<0 ? 1 : 3) : (disY <0 ? 2 : 0); // 确定移动方向

var data = this.data.main.move(direction);

this.updateView(data);

}

}

}

11. 记录存储

将游戏得分存储到数据库，添加如下代码：

saveRecord:function(){

var local\_count = getApp().globalData.userInfo.count;

var local\_score = getApp().globalData.userInfo.score;

const users = AV.User.current();

if (local\_count <this.data.maxCount){

users.set('count', this.data.maxCount);

users.save();

}

if (local\_score <this.data.score) {

users.set('score', this.data.score);

users.save();

}

getApp().globalData.userInfo = users.attributes;

// 声明类型

var Record = AV.Object.extend('Record');

// 新建对象

var record = new Record();

record.set('username', getApp().globalData.userInfo.username);

record.set('count', this.data.maxCount);

record.set('score', this.data.score);

record.save().then(function (todo) {

console.log("success record");

}, function (error) {

console.error(error);

});

}

在gameOver方法最下面添加方法调用：

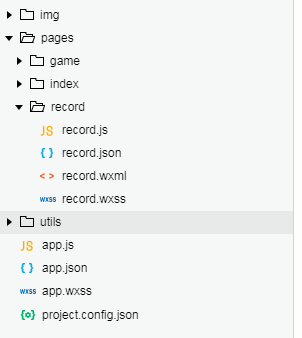
this.saveRecord();

# 单元4 排行榜

## 任务4.1 排行榜

1. 新建页面

新建pages/record目录，在该目录下新建record页面，用来展示排行榜。



2. 页面跳转

打开game.js文件，在Page下添加跳转页面方法，添加如下代码：

record\_click:function(){

wx.navigateTo({

url: '../record/record'

})

}

3. 视图层渲染

打开record.wxml文件，添加如下代码：

<viewclass="swiper-tab">

<viewclass="tab-list-left {{currentTab==0 ? 'on' : ''}}" data-current="0" bindtap="swichNav">个人排行</view>

<viewclass="tab-list-right {{currentTab==1 ? 'on' : ''}}" data-current="1" bindtap="swichNav">全部排行</view>

</view>

<scroll-viewscroll-yclass='scroll'>

<blockwx:for="{{list}}">

<viewclass="content">

<textclass='order' hidden='{{isshow}}'>{{index+1}}</text>

<imageclass="logo" mode="widthFix" src='{{item.url}}'></image>

<viewclass='right'>

<textclass='name'>{{item.name}}</text>

<textclass='record'>最高连击：{{item.count}}</text>

<textclass='record'>最高分数：{{item.score}}</text>

<textclass='record' hidden='{{isshowTime}}'>{{item.time}}</text>

</view>

</view>

</block>

</scroll-view>

4. 为视图层添加样式

打开record.wxss文件，添加如下代码：

.swiper-tab {

text-align:center;

line-height:10vh;

height:10vh;

margin-top:auto

}

.tab-list-left {

font-size:30rpx;

display:inline-block;

width:25%;

border:1pxsolid#bbada0;

line-height:60rpx;

color:#bbada0;

}

.tab-list-right {

line-height:60rpx;

font-size:30rpx;

display:inline-block;

width:25%;

border:1pxsolid#bbada0;

color:#bbada0;

}

.on {

color:#fff;

background-color:#bbada0;

}

.scroll{

height:90vh

}

.content{

display:flex;

margin:20rpx;

padding-bottom:20rpx;

justify-content:flex-start ;

border-bottom:1pxsolid#ccc;

align-items:center;

}

.content .right{

display:flex;

flex-direction:column ;

margin-left:20rpx;

}

.logo {

width:100rpx;

height:100rpx;

border-radius:20rpx;

}

.order{

color:#a0a0bb;

padding-right:20rpx;

}

.name{

font-size:20px;

font-weight:bold;

}

.record{

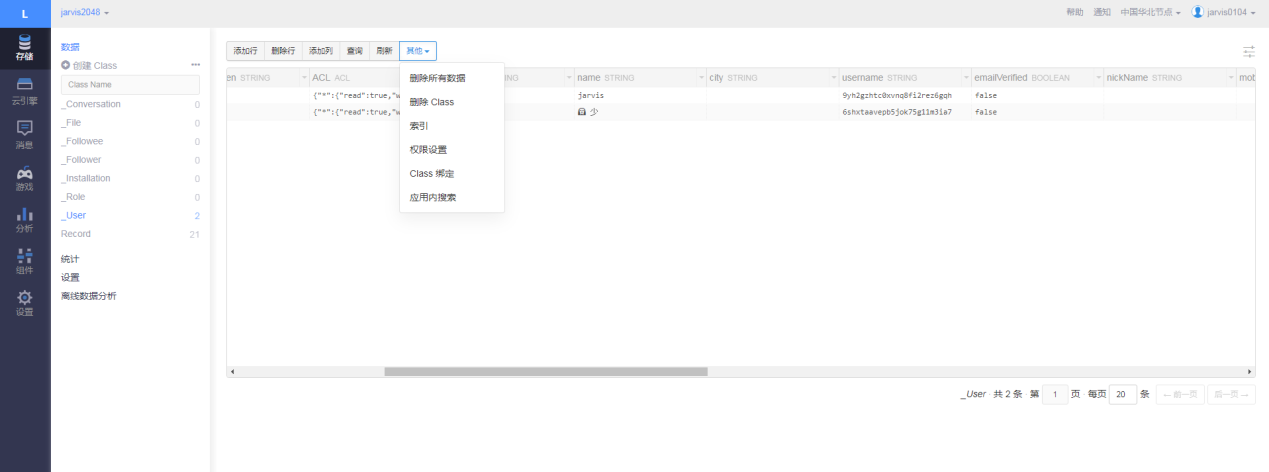
font-size:15px;

}

5. 逻辑层开发

5.1 修改权限.

打开leancloud控制台，修改­\_User表权限，如图所示：



选择所有用户选项。



5.2 定义参数。

打开record.js文件，在Page下添加如下代码：

data: {

currentTab: 0,

list: [],

isshow:true,

isshowTime:false

}

5.3 定义时间类型转化方法，传入参数为时间戳，添加如下代码：

formatDateTime: function (inputTime){

var date = new Date(inputTime);

var y = date.getFullYear();

var m = date.getMonth() + 1;

m = m <10 ? ('0' + m) : m;

var d = date.getDate();

d = d <10 ? ('0' + d) : d;

var h = date.getHours();

h = h <10 ? ('0' + h) : h;

var minute = date.getMinutes();

var second = date.getSeconds();

minute = minute <10 ? ('0' + minute) : minute;

second = second <10 ? ('0' + second) : second;

var time = y + '-' + m + '-' + d + ' ' + h + ':' + minute + ':' + second;

return time;

}

5.4 查询个人排行数据。

根据用户name查询Record表，根据updatedAt排序，将数据按照页面要求存入list中，添加如下代码：

loadPersonOrder:function(){

var that = this;

var query = new AV.Query('Record');

query.equalTo('username', getApp().globalData.userInfo.username);

query.descending('updatedAt');

query.limit(50);// 最多返回50条结果

query.find().then(function (results) {

var lists = [];

for (var i = 0; i < results.length; i++) {

var res = results[i].attributes;

var outTime = that.formatDateTime(results[i].updatedAt);

lists.push({

name: getApp().globalData.userInfo.name,

score: res.score,

count: res.count,

url: getApp().globalData.userInfo.url,

time: outTime

});

}

that.setData({

list: lists

})

}, function (error) { });

}

5.5 查询全部排行。

查询\_Ueser表，根据score排序，将数据按照页面要求存入list中，添加如下代码：

loadAllOrder:function(){

var that = this;

var query = new AV.Query('\_User');

query.addDescending('score');

query.limit(50);// 最多返回 10 条结果

query.find().then(function (results) {

var lists = [];

for (var i = 0; i < results.length; i++) {

var res = results[i].attributes;

console.log(res);

var outTime = that.formatDateTime(results[i].updatedAt);

lists.push({

name: res.name,

score: res.score,

count: res.count,

url: res.url,

time: outTime

});

}

that.setData({

list: lists

})

}, function (error) { });

}

5.6 请求数据。

在onLoad方法下添加请求个人数据方法，添加如下代码：

this.loadPersonOrder();

5.7 为按钮添加单击事件。

获取按钮上传过来的值，判断是点击了个人排行还是全部排行，根据情况调用不同的查询方法，添加代码如下：

swichNav: function(e) {

if (e.target.dataset.current==1){

this.loadAllOrder();

this.setData({

isshow: false,

isshowTime:true

})

}else{

this.loadPersonOrder();

this.setData({

isshow: true,

isshowTime: false

})

}

this.setData({

currentTab: e.target.dataset.current

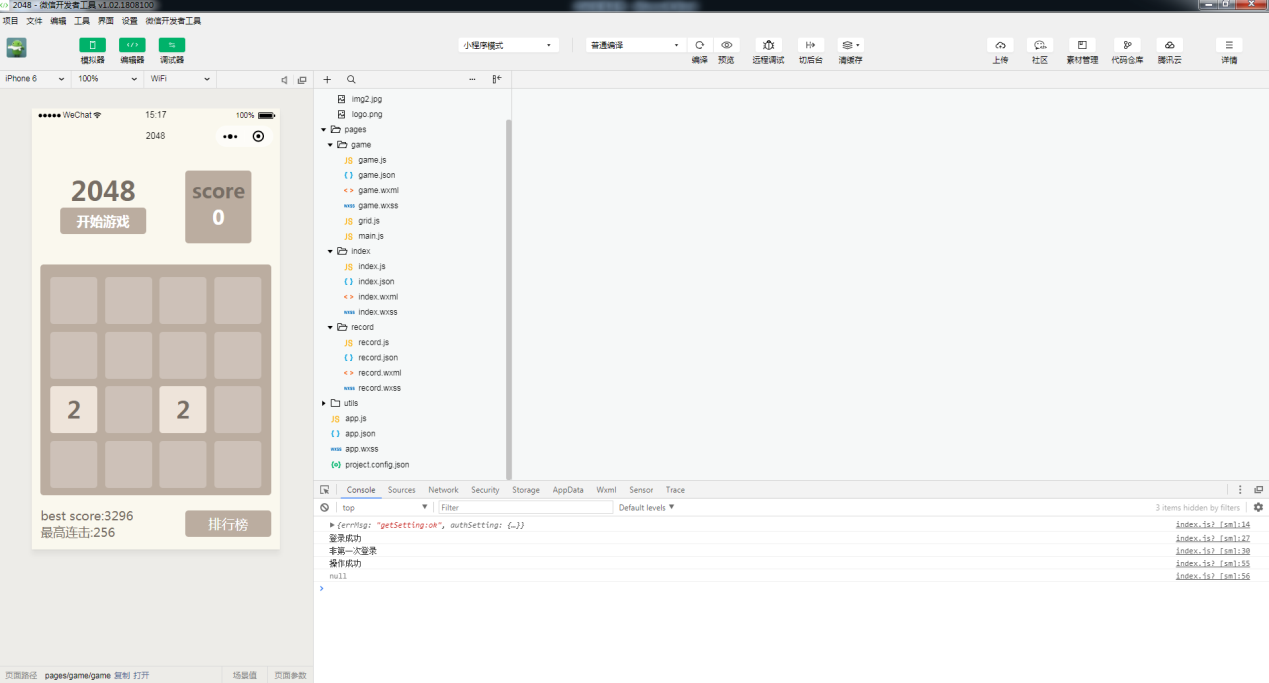
})

}

# 单元5 小程序发布

## 任务5.1 程序发布

1. 打开微信开发者工具，点击上传按钮。



根据提示完成上传操作。

2. 打开微信小程序后台。选择开发管理，就能到提交的开发版本。



3. 点击提交审核，根据提示填写功能页面配置。



4. 最后点击提交审核，就完成了本次版本提交，耐心等待微信审核结果就行。

