# 使用UI

## 1、colorUI

/app.wxss添加：

@import "colorui/main.wxss";

@import "colorui/icon.wxss";

@import "colorui/animation.wxss";

从colorUI目录下复制colorui目录

# 二、小程序组件

## 1、组件创建

---------------

1. 组件必须文件index.js、index.wxss、index.json、index.wxml
2. 在 /组件/index.json中声明，这里是组件

{"component": true,}

1. 在 /组件/index.js中添加：

Component({

properties: {

// 这里定义了innerText属性，属性值可以在组件使用时指定

innerText: {

type: String,

value: "default value",

}

},

data: {

// 这里是一些组件内部数据

someData: {}

},

methods: {

// 这里是一个自定义方法

customMethod () {}

}

})

**# 不是在page({})方法中添加**

1. 在 /组件/index.wxml中添加

<view>我是组件</view>

## 2、组件使用

------------------------

1、在/page/index/index.json中添加

"usingComponents": {

//组件位置

"navbottom": "/pages/Component/nav-bottom/index"

}

1. 在/page/index/index.wxml中添加

<navbottom inner-text="Some text"></navbottom>

## 3、父子组件

### 子组件触发父页面的方法

1. **创建插件<login></login>**
2. **插件.js**

//name 这里指明这个触发事件（父类）的名称

//param1 携带的data数据

//param2

this.triggerEvent("name", param1, param2) // 定义一个事件

1. **father.wxml（父类）**

这里在父页面调用子组件的标签上添加

// name 是childen中triggerEvent中绑定的事件名

//dataFromParent 子组件携带的参数

// inHidpage 是要触发的函数(父类)

// login是插件名称,**必须在当前插件上才能触发**

<login bind:name='isHidpage'></login>

1. **father.js**

定义isHidpage方法

### 父页面向子组件传值

1. **父类.json**

**// 注册组件**

usingComponents:{

"loading": "/pages/components/loading/loading"

}

1. **父类.wxml**

// cu-custom是自定义组件

// gotoUrl 是自定义变量名，{{goUrl}}对应js文件中data的数据

<cu-custom gotoUrl="{{goUrl}}"></cu-custom>

1. **父类.js**

data: {

   goUrl: "123", //定义的值，this.data.goUrl取值

}

1. **子类.js**

**//子类中可以使用 this.data.**gotoUrl**获取值**

Component({

properties: { // 定义接收父类值

gotoUrl: { // 变量名称

type: String, // 变量类型 可以填写数值[Boolean, String]

default: ''

},

})

1. **子类.wxml**

正常使用

### 父类触发子类方法

1. **父类.wxml**

**// 要触发的插件添加id**

<loading id="loading"></loading>

**2、父类.js**

a = this.selectComponent("#loading")

1. loadProgress()

**3、子类.js**

methods:{

loadProgress() {} //要触发的方法

}

## 4、slot插槽

**（1）封装组件**

<!--

slot插槽（占位），用于承载组件引用时提供的子节点

name用于标识slot

-->

<slot name=”custom1”></slot>

1. **在调用组件的位置填写**

<slotName> // 组件

<view slot="custom1">我将填写到name=custom1的组件位置</view>

</slotName>

## 5、组件生命周期

created (){...}组件实例化，但节点树还未导入，**因此这时不能用setData**

attached (){...}节点树完成，可以用**setData**渲染节点，但无法操作节点

ready (){...}组件布局完成，这时可以获取节点信息，也可以操作节点

moved (){...}组件实例被移动到树的另一个位置

detached (){...}组件实例从节点树中移除

# 三、app对象

1. 在app.js中定义要使用的变量
2. 在使用的时候

var app = getApp(); //必要条件

app.appData.user\_info = {"username"："jane"}; //赋值

user\_info = app.appData.user\_info; //取值

# 四、使用全局样式全局样式表

/page/组件/index.js

Component({

options: {

addGlobalClass: true //这里是设置项

},

data: {}

})

# 五、loyout布局

实现对页面的拆分，根据条件加载不同的页面

1. 制作组件，将不同的页面变成组件添加进inde.wxml页面中
2. 确定首页

/page/index/index.wxml

1. 组册组件，参考组件使用
2. 在 index.wxml 中添加

// 用于确定什么时候显示

<index wx:if="{{PageCur == "index"}}"></index>

//绑定函数

// bindtap绑定一个函数NavChange方法

// data-cur是携带的值，用于确定加载哪个组件

<view bindtap="NavChange" data-cur="index">...</view>

1. index.js

Page({

data: {

//用于确定加载哪个插件

PageCur: 'index'

},

//事件处理函数

NavChange(e) {

//设置

this.setData({

PageCur: e.currentTarget.dataset.cur

})

})}

# 六、持久缓存

## 1、插入

wx.setStorage({ //异步

key:"key",

data:"value"})

wx.setStorageSync('key', 'value') //同步

## 2、读取

wx.getStorage({

key: 'key',

success (res) { // 可不添加

console.log(res.data)

}})

var value = wx.getStorageSync('key')

//同步

## 3、删除

wx.removeStorage //异步

wx.removeStorageSync //同步

## 4、清空

wx.clearStorage //异步

wx.clearStorageSync //同步

## 5、获取缓存信息

wx.getStorageInfo

wx.getStorageInfoSync

以Sync结尾都是同步方法。同步方法和异步方法的区别是：

* 同步方法会堵塞当前任务，直到同步方法处理返回。
* 异步方法不会塞当前任务。
* 一般选用异步方法

## 6、保存信息 ex

//保存信息

wx.setStorage({

key: 'key1',

data: 'data1',

success: function(res){

console.log('异步保存成功')

}})

-----------------------------------------------------------------------------------

//保存信息

wx.setStorageSync('key2', 'data2')

# 七、功能标签

bindtap 绑定点击事件

----------------------

bindtap=”NavChange” //绑定NavChange方法

data-cur=”xx” //此参数传递给bindtap绑定的方法，NavChange()通过参数接收

-----------------------

open-type="getUserInfo" //新版微信接口修改，与bindgetuserinfo配合使用

bindgetuserinfo="onGetUserInfo" //调用onGetUserInfo函数

bindinput="textareaAInput" // 绑定输入事件

# 八、微信函数简记

wx.getSetting() //获取设置设置项，包含认证状态

wx.getUserInfo() //用户信息，头像，名称，

wx.login() //这个主要是获取登录的code用于换取信息

# view传值

wx:for-item=”{{index}}” //将wx:for-item的值放到index中

data-变量名=”{{index}}” // 将index值,通过data传递到js中

# 微信page生命周期

app调用本身放心需要使用this

onLoad （） {....}

# 部署

1. 部署后端到服务器

# 功能简记

## 1、登录流程code

主要方法使用<button>绑定事件

wx.getUserinfo使用api获取用户信息（需经用户授权）

open-date用于显示用户信息

1）小程序内通过wx.login接口获得code  
2）将code传入后台，后台对微信服务器发起一个https请求换取openid、session\_key  
3）后台生成一个自身的3rd\_session（以此为key值保持openid和session\_key），返回给前端。PS:微信方的openid和session\_key并没有发回给前端小程序   
4）小程序拿到3rd\_session之后保持在本地  
5）小程序请求登录区内接口，通过wx.checkSession检查登录态，如果失效重新走上述登录流程，否则待上3rd\_session到后台进行登录验证

### 登录/注册流程

***小程序/app.js***

1. wx.login({....}) //获取code
2. wx.request({...}) //将code发送到后台服务器

***开发者服务器/index.py***

（3）获取code，发送code到腾讯服务器

urllib、curl实现发送请求

（4）腾讯服务器返回session\_key与openid

（5）通过openid到数据库中查询查询用户信息

**已经注册过**

--> 生成3rd\_session，发送到客户端

使用hashlib库md5方法，生成3rd\_session，将3rd\_session为key，openid为value，保存到session或redis中

**没有组册**

--> 发送没有注册标识到客户端

***小程序/app.js***

保存3rd\_session

### 授权

1. <button>绑定点击事件

<button open-type="getUserInfo" bindgetuserinfo="onGetUserInfo">

1. 方法 onGetUserInfo( e ) {....} 的 e 中有用户基本信息。

### 验证用户是否授权

当用户已经授权wx.getSetting字段下的authsetting不为空

wx.getSetting({

success:data=>{

if(data.autSetting[‘scope.userInfo’]){ //不授权为空

//已经授权

}

}

})

### 授权过期

通道wx.checkSession或3rd\_session是否存在

### v-buttion

图片buttion

## 2、请求验证

设置请求头

getRequestHeader () {

return {

‘content-type’:’application/x-www-form-urlencoded’,

‘Authorization’: this.getCache(“token”) //服务器返回的token

}

}

## 3、获取页面ID

onload：(e)=>{

this.setData({

id=e.id

}

## 4、跳转

**常用wx.redirecTo方法**



**tabBar页需要在app.json中注册**



## 5、下拉刷新功能

1. wxml给页眉（tab）设置style

// 这是设置，下拉tab固定在页眉

style="position: fixed;width:100%;z-index:1000;top:0rpx"

1. json添加

"enablePullDownRefresh": true

1. js添加触发事件

onPullDownRefresh: function () { },

## 获取form表单的值

1. 填写表单点击调用方法

<form bindsubmit="back\_houtai">

1. 标签添加 name

<input **name="name"**></input> // 标示key

<view **name="type"**>{{**picker**}}</view>// 标示key

1. js

back\_houtai (event) {

console.log(event）

}

## 获取view\text等的值

通过设置data-变量名传值

1. **添加data属性**

<view data-name=”{{ss}}” bindtap=”fun\_c”></view>

1. **js获取**

fun\_c (event) {

console.log(event) // event中包含name的数据

}

## setData

setData会更新页面中的数据

* 设置第N个数据

this.setData({

“data\_str[1].name”: “myname”

})

* 设置第N个数据，但需要变量

let a = “data\_str[“ + num + “].name” // 设置key

this.setData({

[a]: “myname”

})

## 导入/导出

**/utils/login.js**

class Login() {}

module.exports = {

name: Login //name 导出的属性名,Login类

}

**/app.js**

const util = require('./utils/login.js') //添加绝对路径

let a = new util.name() **// 调用**

**=========================================================**

**/data/login.js**

const detail = {}

export default detail;

**/page/index.js**

import detail from '../data/login.js';

## 获取input数据

<view bindinput="getSearch"></view>

getSearch(e) {

let searchText = e.detail.value; // 搜索框中内容

}

## navigator传值

传值:

navigator的属性url后拼接?id(参数名字)=要传递的值

<navigator url=”/pages/index/a?courseUuid=11”></navigator>

取值：options 是包含url地址中参数的对象，可以直接 点 获取。

onload:function(option){

let a = options.courseUuid

}

## swiper滑动块的坑

**必须是这种结构，如果不是这种结构，很可能图片不能加载**

<swiper>

<block wx:for="{{image}}" wx:key>

**<navigator url="">**

<swiper-item>

<image src=""/>

</swiper-item>

</navigator>

</block>

</swiper>