## 页面配置

【page】.json

可以配置和全局的window一致的属性，注意不需要加window，如果有相同的属性，会把app.json中的覆盖掉。tabBar不能在页面中去配置 。

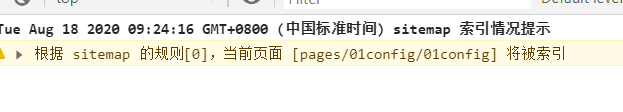
disableScroll:不允许页面滚动。默认为false

## Sitemap配置

爬虫。

默认规则，是允许微信收录的。同时所有页面也能够被抓取到的。

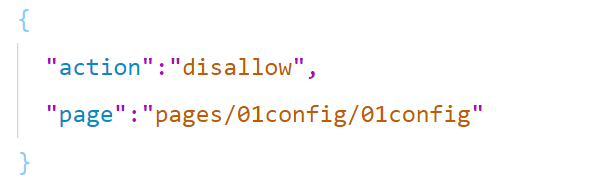
<https://developers.weixin.qq.com/miniprogram/dev/reference/configuration/sitemap.html>



索引到、收录到、查询到。

### 属性就一个rules：array

* action:是否收录或禁止收录
* Page:action对应的页面起作用。\* 代表所有页面



* params:array
* maching

Params:[‘a’,’b’]

Maching:’exact’

Pages/a/b?name=ujiuye&age=18

Pages/a/b

Pages/a/b?a=1&b=2 命中

Pages/a/b?a=1&b=2&c=3 不会



## 关于使用其它的编译方式





## 场景值

在开发当中，我们可以根据用户进入小程序不同的方式，做具体的业务

KFC为例

1：进入店铺后，通过扫描二维的方式进入点餐程序

直接进入点餐页面 ，你都已经来了，没有优惠券

1011

2：公众号，通过公众号进入的kfc

你是我的潜在用户，显示或跳转到优惠券页面。

1035

如何获取场景值 。

在刚初始化小程序的时候去获取的。

### App.js中获取

<https://developers.weixin.qq.com/miniprogram/dev/framework/app-service/scene.html>

onShow或onLaunch获取 。通过api去获取（wx.getLaunchOptionsSync()） 。

## 逻辑层

Js模块 。

app.js。[page].js 。Utils下的js文件 。

没有dom和bom。

语法的东西都支持：声明变量、声明函数、数组、对象、分支、循环、函数调用、

时间、数组、RegExp、String、Number（ 小程序中内嵌了解析ECmascript的环境 ）

视图层绑定事件函数把用户的反馈到逻辑层。

小程序还提供了框架的api。

App() Page() wx.getLaunchOptionsSync()

多页面应用 。路由进行管理、某个api获取当前程序中有哪些页面被打开了。

我们可以使用模块化。Es6的模块化和commonjs的模块化

### App.js

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

作用：注册小程序的唯一实例 。我们可以注册小程序的应用生命周期。

定义一些公共函数和一些公共数据 。

App({

//初始化小程序，只执行一次。

// 如果你的小程序 必须让用户登录 才能够使用服务，就可以在这里进行写业务 。

onLaunch(){

console.log('onLaunch')

},

onShow(){

console.log('小程序的显示，可以执行多次')

},

onHide(){

console.log('小程序的隐藏，可以执行多次')

},

onPageNotFound(){

console.log('页面不存在')

//通过api的形式跳转到其他活动页或首页。

},

onError(){

}

})

**注意：**

可以定义生命周期

可以定义公共数据。



可以在访问到app实例的地方使用公共数据但是不要调用生命周期函数。

### [page].js

作用：

Page( {} )

注册对应的页面实例对象,页面的生命周期，页面的私有数据，定义页面中的事件函数，或者定义当前页面中的方法，

我们也可以一些普通的js代码 。同时也可以在任意page.js中获取小程序实例。同时也可以调用小程序的任意api。

// bbb()

Page({ //Page构造函数

data:{

name:'zbj',

job:'背媳妇儿'

},

fn(){

console.log( this,'我是fn函数' )

console.log( this.data ) //在逻辑层中直接使用data里数据，必须通过this.data.xxx

},

onLoad(){

console.log('页面加载，只执行一次')

this.fn()

},

onShow(){

console.log('显示了')

},

onHide(){

console.log('隐藏了')

},

onReady(){

console.log('准备好了,只执行一次')

},

//下拉事件发生以后

onPullDownRefresh(){

console.log('用户下拉了，开发者可以做刷新功能 ')

},

//上滑，距离底部多少的时候，会触发此事件 50 onReachBottomDistach 全局或局部都可以配置

onReachBottom(){

console.log('滑到了底部，可以请求下一页')

}

})

**总结：**

打开页面没有特殊情况 一定是在onLoad当中去请求ajax

有时候需要在onShow中请求ajax。

### getApp()

获取全局的app实例对象

在其它任意js页面中可以获取

let appObj = getApp()

console.log( appObj,'在index中获取到app实例' )

console.log( appObj.person.username )

### 生命周期

<https://developers.weixin.qq.com/miniprogram/dev/framework/app-service/page-life-cycle.html>

总结顺序问题

onReachBottom和onPullDownRefresh

App.js和【page】.js它的一些顺序

onShow 和 onHide的顺序。

显示时：先执行app.js中的onShow再执行对应page中的onSHow

显示时：先执行page中的onShow再执行app.js中的onShow

## 视图层

视图层由wxml和wxss组成 。

Wxml又由基础组件构成 。后期我们也可以自定义组件（了解）。

### Wxml语法 结构代码

数据的绑定

指的是页面的私有数据，数据是定义在page.js中的data属性上。然后可以在视图层去使用。

同时可以写一些简单的js表达式 。同时不支持函数调用，支持事件调用 。

使用mustache {{}}语法

<!-- 1. 基本的数据渲染 -->

<view>{{num}}</view>

<view>{{str}}</view>

<!-- 2. 数据绑定到属性上 -->

<view id="{{id}}">我是view元素</view>

<!-- 3. 关于布尔值的属性绑定,因为布尔值是关键字，所以必须用双大号括起来 -->

<!-- <checkbox checked="true"></checkbox>

<checkbox checked="false"></checkbox> -->

<checkbox checked="{{true}}"></checkbox>

<checkbox checked="{{isChecked}}"></checkbox>

<!--

4. 路径运算

访问数组和访问对象

-->

<view>访问数组和对象</view>

<view>{{arr[1]}}</view>

<view>{{person.name}} {{person.job}}</view>

**表达式的说明：**

<!--

表达式

算术、拼接、三目、条件判断 不支持函数自执行

-->

<view>{{ 1+1 }}</view>

<view>{{ 'a' + str }}</view>

<view>{{ 5 > 10 ? 'ok' : 'no ok' }}</view>

<!-- <view>{{fn()}}</view>

<view>{{Math.random()}}</view> -->

列表渲染

wx:for

wx:for-item

wx:for-index

wx:key

小程序循环的当前项的变量为item，当前下标为index。

二维数组肯定乃至循环嵌套，注意：可以显示的声明index和item为其它变量不要冲突 。

数组和对象、字符都可以遍历，遍历字符串没有意义 。

* 基本遍历

<!--

1. 基本遍历

-->

<view wx:for="{{nums}}">

当前值：{{item}} --- 当前下标：{{index}}

</view>

* 遍历对象

<!--

2. 遍历对象

wx:for-index 指定当前下标的变量

wx:for-item 指定当前值的变量

-->

<view wx:for="{{person}}">

当前值为:{{item}} --- {{index}}

</view>

<view wx:for="{{person}}" wx:for-item="value" wx:for-index="key">

当前值为:{{value}} --- {{key}}

</view>

* 在遍历中如果想使用data中的item或index的注意事项

<!--

3.在遍历中如果想使用data中的item或index的注意事项

-->

<view wx:for="{{nums}}" wx:for-item="val">

当前值：{{item}} --- 当前下标：{{index}}

</view>

<view>-------------------------------------</view>

* 遍历数组套对象

必须掌握

<!--

4.遍历数组套对象

-->

<view wx:for="{{products}}">

商品名为：{{item.name}} 商品价格：{{item.price}}

</view>

<view>-------------------------------------</view>

* block

<!--

5. block -> vue:template

代码的搬运工，不会生产代码 。

-->

<block wx:for="{{products}}">

<view>{{item.name}}</view>

<view>{{item.price}}</view>

</block>

* 遍历的注意事项

<!--

6. 遍历的注意事项

关于字符串的问题: 要遍历数组的变量的双大括号外部的前后不要加空格

-->

<view>-------------------------------------</view>

<view wx:for="{{ nums }}">

{{item}}

</view>

* 循环嵌套

<!--

7. 循环嵌套

-->

<view wx:for="{{product}}" class="father" wx:for-item="val1">

<view wx:for="{{val1}}" class="son" wx:for-item="val2">

{{val2.name}} --- {{val2.price}}

</view>

</view>

* Wx:key

<!--

wx:key:

1： 普通的数据 [11,22,33] ['web','java','python']

保留字|关键字： \*this

2: [ {goodsid,id},{},{} ]

id

-->

<view wx:for="{{nums}}" wx:key="\*this">

{{item}}

</view>

<view wx:for="{{products}}" wx:key="id">

{{item.name}}

</view>

条件渲染

wx:if

wx:elif

wx:else

If Vs hidden

* 本的条件渲染

<!--

1. 基本的条件渲染，条件渲染是一个整体 ，中间不能其它组件

-->

<view wx:if="{{age >= 18}}">成年了</view>

<view wx:else>呵呵，不要惹熊孩子</view>

* 多分支

<!--

2. 多分支

-->

<view wx:if="{{ score > 80 && score <= 99 }}">还可以</view>

<view wx:elif="{{ score > 60 && score <= 80 }}">不错哦</view>

<view wx:else>你要上天啊</view>

* if vs hidden

<!--

3. if vs hidden

-->

<!--

hidden

-->

<view hidden="{{ index != 0 }}">待支付</view>

<view hidden="{{index != 1}}">待评论</view>

<view hidden="{{index != 2}}">待收货</view>

<!--

if

-->

<view wx:if="{{carts.length > 0}}">

<view wx:for="{{carts}}">

要渲染购物车的数据

</view>

</view>

<view wx:else>

空空如也~

</view>

模板与引用

了解：

复用视图层：特指wxml的代码的复用 。

* 定义

<template name=””>代码块</template>

* 使用

<template is=”模板名称” />

<template is=”模板名称”></template>

* 模板中使用数据

1.3 模板中使用数据

<template data="{{key1:value1,key2:value2}}">

page.js

data:{

person:{ name,age }

}

<template data="{{ ...person }}">

* 引用

Import

include

C import B，B import A，在C中可以使用B定义的template，在B中可以使用A定义的template，但是C不能使用A定义的

### Wxss样式

rpx

Import

全局样式和局部样式

支持css3 和 flex

选择器不需要使用太复杂的

<https://developers.weixin.qq.com/miniprogram/dev/framework/view/wxss.html>

尺寸单位

rpx

rpx（responsive pixel）: 可以根据屏幕宽度进行自适应。规定屏幕宽为750rpx。如在 iPhone6 上，屏幕宽度为375px，共有750个物理像素，则750rpx = 375px = 750物理像素，1rpx = 0.5px = 1物理像素。

建议： 开发微信小程序时设计师可以用 iPhone6 作为视觉稿的标准。 也就是750像素 。

样式导入

解决公共样式的问题

@import “路径”; 后面加分号

@import "../../public/public.wxss";

全局样式与局部样式

app.wxss 其它页面不需要手动引入

### 事件系统

用户需要向逻辑反馈只能通过事件进行反馈。

事件也是视图层到逻辑层的基本通讯 方式

事件只能通过组件属性的形式进行绑定

<tagName onClick>

Js : document.getElementById().onclick

元素.addEventListener ( ‘事件名’，handle )

事件对象（只能通过事件对象传参） data-

1. 事件绑定不要使用箭头函数

访问不到this

<!--

1. 事件的基本绑定

bind+事件名=“函数名” 不能加括号

-->

<view bindtap="handle1">函数声明形式</view>

<view bindtap="handle2">对象的函数简写形式</view>

<view bindtap="handle3">箭头函数简写形式</view>

handle1:function(){

console.log('handle1',this)

},

handle2(){

console.log('handle2',this)

},

handle3:()=>{

console.log('handle3',this)

},

1. 冒泡与非冒泡

<!--

3: 事件类型：

冒泡与非冒泡

对于 有冒泡的事件

bind+冒泡的事件 ： 会形成冒泡

catch+冒泡的事件：会阻止冒泡

-->

<view class="outer" bindtap="outerHandle">

<view class="middle" bindtap="middleHandle">

<view class="inner" bindtap="innerrHandle"></view>

</view>

</view>

1. 什么时候用箭头

handle4(){

// this.data.nums.forEach(()=>{

// console.log(this.data.name)

// })

let \_this = this

this.data.nums.forEach(function(){

console.log(\_this.data)

})

},

1. 事件对象与传参

<!--

4.事件对象

是事件中最复杂的。

currentTarget.dataset

detail:

表单的数据，就在detail去接收。

-->

<view>-----------------------------</view>

<view bindtap="handle5">事件对象</view>

<view id="father" bindtap="handle5">

father

<view id="son">son</view>

</view>

<!--

5:关于data-\*-\* 传参

总结：事件绑定在哪个组件上，那么开发想主动传参 就把data-的形式写在此组件上

然后通过 事件对象.currentTarget.dataset 接收

data-name

data-user-name => userName

data-bg-color => bgColor

-->

<view>-----------------------------</view>

<view id="father" data-name="zbj" bindtap="handle6">

传参 father

<view id="son">son</view>

</view>