QuickStart 项目生成了不同类型的文件:

.json 后缀的 JSON 配置文件

.wxml 后缀的 WXML 模板文件

.wxss 后缀的 WXSS 样式文件

.js 后缀的 JS 脚本逻辑文件

JSON配置

app.json 是当前小程序的全局配置

pages用于指定小程序由哪些页面组成，每一项都对应一个页面的 路径+文件名 信息。文件名不需要写文件后缀，框架会自动去寻找对于位置的 .json, .js, .wxml, .wxss 四个文件进行处理

"pages": [

"pages/index/index",

"pages/userConsole/userConsole",

]

* 数组的第一项代表小程序的初始页面（首页）
* 小程序中新增/减少页面，都需要对 pages 数组进行修改。
* 其实指向js文件，所以这个文件夹下没有js文件会报错

"window": {

"backgroundColor": "#ff4400",

"backgroundTextStyle": "light",

"navigationBarBackgroundColor": "#ff4400",

"navigationBarTitleText": "Demo",

"navigationBarTextStyle": "black",

"enablePullDownRefresh": true

}

* enablePullDownRefresh 默认false

是否全局开启下拉刷新

app.json不可添加任何注释

app.wxss 是整个小程序的公共样式表

project.config.json 配置开发工具的

page.json页面配置

app.js

App({

globalData: 'hello world',

onLaunch: function () {

},

onShow: function(){

},

onHide: function(){

}

})

app.js整个程序执行一次，注册函数

在任意一个page的js中(page()方法外)调用getApp()，返回App()里头那个对象

var app=getApp();

console.log(app. globalData)

生命周期回调函数

* onLoad(Object query)

加载时触发。一个页面只会调用一次

query 打开当前页面路径中的参数

* onShow()

显示/切入前台时触发。

* onReady()

初次渲染完成时触发。一个页面只会调用一次，代表页面已经准备妥当，可以和视图层进行交互。

先onLoad后onShow最后onReady

* onHide()

隐藏/切入后台时触发。 如 navigateTo 或底部 tab 切换到其他页面，小程序切入后台等。

* onUnload()

页面卸载时触发。如redirectTo或navigateBack到其他页面时。

index.js

里面有一个Page({

})

参数里有：

data Object 页面的初始数据

页面生命周期函数

* onLoad 监听页面加载
* onShow 监听页面显示
* onReady 监听页面初次渲染完成
* onHide 页面隐藏
* onUnload 监听页面卸载
* onPullDownRefresh Function 监听用户下拉动作

在app.js的window中设置enablePullDownRefresh: true

wx.startPullDownRefresh也可以触发

* onReachBottom Function 页面上拉触底事件的处理函数

在app.js的window中设置触发距离onReachBottomDistance: "15px";

每滑到距离页面底部15px时触发一次

* onShareAppMessage Function 用户点击右上角转发
* onPageScroll Function 页面滚动触发事件的处理函数
* onTabItemTap Function 当前是 tab 页时，点击 tab 时触发

其他 Any 开发者可以添加任意的函数或数据到参数中，在页面的函数中用 this 可以访问

WXML 充当的就是类似 HTML 的角色

页面样式表index.wxss中的样式规则会层叠覆盖 app.wxss 中的样式规则

index.json 是页面的配置文件：

当有页面的配置文件时index.json，配置项在该页面会覆盖 app.json 的 window 中相同的配置项

**模块化**

可以将一些公共的代码抽离成为一个单独的 js 文件，作为一个模块。通过 module.exports 或者 exports 才能对外暴露接口。

推荐采用 module.exports

// common.js

function sayHello(name) {

console.log('Hello ${name} !')

}

module.exports.sayHello = sayHello

​在需要使用这些模块的文件中，使用 require(path) 将公共代码引入

var common = require('common.js')

Page({

helloMINA: function() {

common.sayHello('MINA')

},

})

<text class="window">Hello</text>

**事件**

在组件中绑定一个事件处理函数。

如bindtap，点击该组件时会在该页面对应js中的Page中找到相应的事件处理函数。

<view id="tapTest" bindtap="tapName"> Click me! </view>

在相应的Page定义中写上相应的事件处理函数，参数是event。

Page({

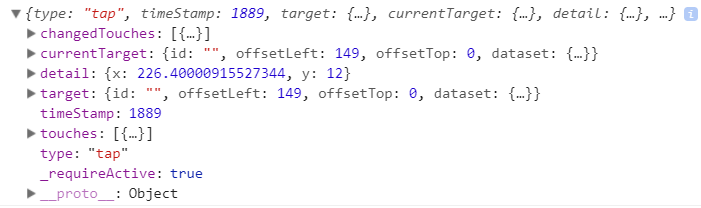
tapName: function(event) {

console.log(event)

}

})

可以看到log出来的信息大致如下：



事件对象

组件触发事件时，逻辑层绑定该事件的处理函数会收到一个事件对象。就是传入的那个参数event

* type String 事件类型
* timeStamp Integer 事件生成时的时间戳(页面打开到触发事件所经过的毫秒数)
* target Object 触发事件的源组件的一些属性值集合
* currentTarget Object 当前组件的一些属性值集合

CustomEvent 自定义事件对象属性列表（继承 BaseEvent）：

* detail Object 额外的信息

TouchEvent 触摸事件对象属性列表（继承 BaseEvent）：

* touches Array 触摸事件，当前停留在屏幕中的触摸点信息的数组
* changedTouches Array 触摸事件，当前变化的触摸点信息的数组

特殊事件： <canvas/> 中的触摸事件不可冒泡，所以没有 currentTarget。

target和currentTarget

* id String 事件源组件的id
* tagName String 当前组件的类型
* dataset Object 事件源组件上由data-开头的自定义属性组成的集合

<view id="middle" catchtap="handleTap2">

middle view

<view id="inner" bindtap="handleTap3">

inner view

</view>

</view>

点击 inner view 时，handleTap3 收到的事件对象 target 和 currentTarget 都是 inner，而 handleTap2 收到的事件对象 target 就是 inner，currentTarget 就是 middle。因为这个事件不是因为点击了middleView，而是冒泡上来的

dataset

在组件中可以定义数据，这些数据将会通过事件传递给 SERVICE。 书写方式： 以data-开头，多个单词由连字符-链接，不能有大写(大写会自动转成小写)如data-element-type，最终在 event.currentTarget.dataset 中会将连字符转成驼峰elementType。和事件一起使用

<view data-alpha-beta="1" data-alphaBeta="2" bindtap="bindViewTap"> DataSet Test

</view>

Page({

bindViewTap:function(event){

event.currentTarget.dataset.alphaBeta === 1 // - 会转为驼峰写法

event.currentTarget.dataset.alphabeta === 2 // 大写会转为小写

}

})

touches

touches 是一个数组，每个元素为一个 Touch 对象（canvas 触摸事件中携带的 touches 是 CanvasTouch 数组）。 表示当前停留在屏幕上的触摸点

Touch 对象

* identifier触摸点的标识符
* pageX, pageY距离文档左上角的距离，文档的左上角为原点 ，横向为X轴，纵向为Y轴
* clientX, clientY距离页面可显示区域（屏幕除去导航条）左上角距离，横向为X轴，纵向为Y轴

detail

点击事件的detail 带有的 x, y 同 pageX, pageY 代表距离文档左上角的距离。

事件分为冒泡事件和非冒泡事件：

* 冒泡事件：当一个组件上的事件被触发后，该事件会向父节点传递。
* 非冒泡事件：当一个组件上的事件被触发后，该事件不会向父节点传递。

冒泡事件

类型 触发条件

touchstart 手指触摸动作开始

touchmove 手指触摸后移动

touchcancel 手指触摸动作被打断，如来电提醒，弹窗

touchend 手指触摸动作结束

tap 手指触摸后马上离开

longpress 手指触摸后，超过350ms再离开，如果指定了事件回调函数并触发了这个事件，tap事件将不被触发

longtap 手指触摸后，超过350ms再离开（推荐使用longpress事件代替）

transitionend 会在 WXSS transition 或 wx.createAnimation 动画结束后触发

animationstart 会在一个 WXSS animation 动画开始时触发

animationiteration 会在一个 WXSS animation 一次迭代结束时触发

animationend 会在一个 WXSS animation 动画完成时触发

事件绑定的写法同组件的属性，以 key、value 的形式。

key 以bind或catch开头，然后跟上事件的类型，如bindtap、catchtouchstart

value: Page 中定义同名的函数

* bind事件绑定不会阻止冒泡事件向上冒泡，catch事件绑定可以阻止冒泡事件向上冒泡。

如在下边这个例子中，点击 inner view 会先后调用handleTap3和handleTap2(因为tap事件会冒泡到 middle view，而 middle view 阻止了 tap 事件冒泡，不再向父节点传递)，点击 middle view 会触发handleTap2，点击 outer view 会触发handleTap1。

<view id="outer" bindtap="handleTap1">

outer view

<view id="middle" catchtap="handleTap2">

middle view

<view id="inner" bindtap="handleTap3">

inner view

</view>

</view>

</view>

Page({

data: {

color: "red"

},

click: function(event){

this.setData({

color: 'blue'

})

}

})

page中data改变时，直接改变data='window'可以改，但是wxml中绑定的变量有可能没改。所以在这些函数的后边，一般是点击事件的末尾，要加上

this.setData({

color: 'aaabcd'

})

这样wxml的数据也同步更新

微信ml html

view div

text p

button button

image img

navigator a

切换

1.<navigator url="../page2/page2">点击跳转</navigation>

page2不用加后缀

2.<button bindtap="click1">点击跳转</button>

js里边

click1: function(){

wx.navigateTo({

url: '../index/index'

})

}

在微信里window不能用，所以alert不能用因为是window.alert

微信里边是wx

设置**缓存**

* wx.setStorageSync('key','value')：同步

wx.setStorageSync('key',{

game: "跳一跳",

gold: "666"

})

同步设置第二个参数也可以是对象

* wx.setStorage({key:'key',data:'value'})：异步

当第一个参数值相同的时候，后面的会覆盖前面的

对象属性只能是key和value

获取缓存

* wx.getStorageSync('key')：同步 直接返回获取到的值
* wx.getStorage({key:'key',callback})：异步 通过回调函数返回获取的值

wx.setStorage({

key: "跳一跳",

data: "666"

})

var a = wx.getStorage({

key:'跳一跳',

success: function (res) {

console.log(res.data);

console.log(res);

}

});

清除缓存

清除缓存的方法有两种，clearStorage和removeStorage，每一种都有同步和异步之分：

// 从本地缓存中异步移除指定 key

wx.removeStorage({

key: 'key',

success: function(res) {

console.log(res.data)

}

})

// 同步移除指定 key

try {

wx.removeStorageSync('key')

} catch (e) {

// Do something when catch error

}

//清理本地数据缓存

wx.clearStorage()

// 同步清理本地数据缓存

try {

wx.clearStorageSync()

} catch(e) {

// Do something when catch error

}



条件渲染

<view wx:if="{{flag}}">

flag==true时显示

</view>

<view wx:elif="{{flag==3}}">

flag==3时显示

</view>

<view wx:else>

其他情况时显示

</view>

列表渲染wx:for

wx:for 控制属性绑定一个数组，使用数组中各项的数据重复渲染该组件。

* 使用 wx:for-item、wx:for-index可以指定数组当前元素、下标的变量名

<block wx:for="{{[1, 2, 3]}}">

<view> {{index}}: </view>

<view> {{item}} </view>

</block>

* wx:key

如果列表中项目的位置会动态改变或者有新的项目添加到列表中，并且希望列表中的项目保持自己的特征和状态（如 <input/> 中的输入内容，<switch/> 的选中状态），需要使用 wx:key 来指定列表中项目的唯一的标识符。

wx:key 的值以两种形式提供

* 字符串，代表在 for 循环的 array 中 item 的某个 property，该 property 的值需要是列表中唯一的字符串或数字，且不能动态改变。
* 保留关键字 \*this 代表在 for 循环中的 item 本身，这种表示需要 item 本身是一个唯一的字符串或者数字

当数据改变触发渲染层重新渲染的时候，会校正带有 key 的组件，框架会确保他们被重新排序，而不是重新创建，以确保使组件保持自身的状态，并且提高列表渲染时的效率。

如不提供 wx:key，会报一个 warning， 如果明确知道该列表是静态，可以选择忽略。

* 当 wx:for 的值为字符串时，会将字符串解析成字符串数组

<view wx:for="array">

{{item}}

</view>

等同于

<view wx:for="{{['a','r','r','a','y']}}">

{{item}}

</view>

注意： 花括号和引号之间如果有空格，将最终被解析成为字符串

组件

switch 

"tabBar": {

"list":[

{

"pagePath": "pages/index/index",

"text": "hello"

},

{

"pagePath": "pages/page1/page1",

"text": "world"

}

]

}

tabBar示例 底部菜单栏

模板

<template name="msgItem">

<view>

<text> {{index}}: {{msg}} </text>

<text> Time: {{time}} </text>

</view>

</template>

使用

is 属性声明需要的使用的模板，然后将模板所需要的 data 传入，如：

<template is="msgItem" data="{{...item}}"/>

Page({

data: {

item: {

index: 0,

msg: 'this is a template',

time: '2016-09-15'

}

}

})

<import src="../../templates/courseList.wxml"/>可以把所有模板写在一个.wxml然后import

尺寸单位

rpx（responsive pixel）: 可以根据屏幕宽度进行自适应。规定屏幕宽为750rpx

样式导入

使用@import语句可以导入外联样式表，@import后跟需要导入的外联样式表的相对路径，用;表示语句结束。

@import "common.wxss";

.middle-p {

padding:15px;

}

display: flex;

justify-content: space-between/\* space-around, center…\*/

wx的flex和css的基本相同

在开发者工具中预览效果

<icon type="success" size="40" color='black'/>

success, success\_no\_circle, info, warn, waiting, cancel, download, search, clear

size单位px