WeChatApp

目录

列表循环

1. `wx:for="{{数组或对象}}" wx:for-item="循环项的名字" wx:for-index="循环项的索引"`
2. `wx:key="唯一的值"`
    1. 所绑定一个普通字符串的时候，那么这个字符串名称 肯定是 循环数组 中的 对象的唯一值。
    2. wx:key="*this" 表示一个普通数组， *this表示循环项
3. 数组嵌套循环，绑定的名称不要重名
4. 只有一层循环`wx:for-item="item" wx:for-index="index"`可以省略

```
//在js中绑定一个数组
Page({
    "list":[
        {
```

```
            id: 0,
            name: "A"
        },
        {
            id: 1,
            name: "B"
        },
        {
            id: 2,
            name:"C"
        }
    ]
})
```

```
<view>
    <view wx:for="{{list}}" wx:for-item="item" wx:for-index="index">
    索引:  {{index}}
    --
    值:    {{item.name}}
    </view>
</view>

// Run
引索: 0 --值: A
引索: 1 --值: B
引索: 2 --值: C
```

对象循环

1. `wx:for="{{对象}}" wx:for-item="对象的值" wx:for-index="对象的属性"`
2. 循环对象

   `wx:for-item="value" wx:for-index="key"`

Block 标签

仅仅起到占位的作用

条件渲染

1. `wx:if="{{true/false}}"`
2. if， else, if else
3. hidden
    1. 在标签上直接加
    2. hidden="{{condition}}"
4. 使用场景：
    1. 频繁切换标签，使用`wx:if`。 `wx:if`是直接把标签从页面结构内删除

2. 频繁切换标签，使用hidden。 hidden是直接通过添加样式不显示该标签。**(hidden 属性不要和样式display一起使用。)**

```
<!-- 单个的条件渲染 -->
<view wx:if="{{condition}}"> True </view>

<!-- if循环的条件渲染 -->
<view wx:if="{{length > 5}}"> 1 </view>
<view wx:elif="{{length > 2}}"> 2 </view>
<view wx:else> 3 </view>

<!-- hidden的使用 -->
<view hidden>隐藏文字</view>
<view hidden="{{condition}}">根据条件判断是否隐藏</view>
```

事件绑定

1. 关键字 bindinput 绑定input事件,其中"handleInput"是js中的处理事件操作的函数名。

```
<input type="text" bindinput="handleInput"/>
```

```
handleInput(e)({
    ....
})
```

2. 通过事件源对象来获取值： e.detail.value
3. 把输入框的值 赋值到data中：

```
this.setData({
    num:e.detail.value
})
```

4. 点击事件
    1. bindtap
    2. 通过自定义属性来传参 data-operation ="{{data}}"

```
//wxml
<button bindtap="handleTap" data-operation="{{1}}">+</button>
```

```
//js
handletap(e)({
```

```
        //获取自定义属性operation
        const operation = e.currentTarget.detaset.operation;
        this.setData({
            num: this.data.num + operation
        })
    })
```

样式rpx

公式 page = 750rpx

设计稿：

- 宽度 未知 page
- 存在一个元素宽度：100px /item px
- 要求做适配

```
//利用公式计算rpx的值
view{
    width:calc(750rpx * item / page);
}
```

样式导入

`@import "url"` 其中url为相对路径

image图片标签

swiper轮播图标签

> 1. swiper 有制定宽高：宽度100% 高度150px;
> 2. swiper 高度计算公式：height = 原图高度/原图宽度*swiper宽度
> 3. 属性值：参考官方文档。

navigator导航标签

> 块级元素（默认换行） 可以设定宽和高
>
> 属性：
>
> 1. url: 要跳转的页面路径：绝对路径/相对路径
> 2. target: 要跳转到当前小程序/其他的小程序
>
>   self: 默认值 自己的小程序页面 miniProgram: 其他的小程序界面
>
> 3. open-type
>
> `navigate`: 默认值 对应wx.navigateTo功能

```
wx.navigateTo:
  保留当前页面，跳转到应用内的某个页面。
  但是不能跳到 tabbar 页面。
  使用 wx.navigateBack 可以返回到原页面。
```

小程序中页面栈最多十层。

redirect: 默认值 对应wx.redirectTo功能

```
wx.redirectTo:
   关闭当前页面，跳转到应用内的某个页面。
   但是不允许跳转到 tabbar 页面。
   在小程序插件中使用时，只能在当前插件的页面中调用。
```

switchTab： 对应wx.switchTab 功能

```
wx.switchTab:
   跳转到 tabBar 页面，并关闭其他所有非 tabBar 页面
```

reLaunch： 对应wx.reLaunch 功能

```
wx.reLaunch:
   关闭所有页面，打开到应用内的某个页面。
```

navigateBack： 对应wx.navigateBack 功能

```
wx.navigateBack:
   关闭当前页面，返回上一页面或多级页面。
   可通过 getCurrentPages 获取当前的页面栈，
   决定需要返回几层。
```

exit： 对应wx.exit 功能

```
wx.exit:
   退出其他的小程序。
```

rich-text富文本标签

nodes 属性

1. 接收标签字符串
2. 接收对象数组

```
//1. 接收标签字符串
 <rich-text nodes="{{htmlSnip}}"></rich-text>
//2. 接收对象数组
<rich-text nodes="{{nodes}}"></rich-text>
```

```
//.js
//字符串文本
const htmlSnip =
<div class="div_class">
  <h1>Title</h1>
  <p class="p">
    Life is <i>like</i> a box of
    <b> chocolates</b>.
  </p>
</div>

Page({
  data: {
    htmlSnip;
    nodes: [{
      name: 'div',
      attrs: {
        class: 'div_class',
        style: 'line-height: 60px; color: #1AAD19;'
      },
      children: [{
        type: 'text',
        text: 'You never know what you\'re gonna get.'
      }]
    }]
  }
})
```

button标签

> 属性

1. size 控制按钮大小：
    - mini
    - default
2. type 控制按钮颜色：
    - default 灰色
    - primary 绿色
    - warn 红色
3. plain 控制按钮是否镂空

> Open-type开放能力

属性	功能
share	转发当前的小程序 到微信朋友中 不能发送到朋友圈
contact	直接打开客服对话功能 非企业没有权限
getPhonenumber	获取当前用户的手机号码信息
getUserInro	获取当前用户的个人信息
launchApp	在小程序当中直接打开App
openSetting	打开小程序内置的授权页面
feedback	打开小恒旭内置的意见反馈页面

**getUserInfo 获取当前用户的个人信息

1. 绑定一个事件 bindgetUserInfo
2. 在事件的回调函数中 通过参数来获取信息
3. 没有加密，无需解析

radio 单选框标签

- radio标签 必须要和父元素 radio-group一起使用
- value 选中的单选框的值
- 需要给radio-group 绑定 change 事件
- 需要在页面中显示 选中的值

```
<!-- wxml -->
<radio-group bindchange="handleChange">
    <radio color="yellow" value="female">女</radio>
    <radio color="yellow" value="male">男</radio>
</radio-group>

<view>您选中的值是:{{gender}}</view>
```

```
//js
Page({
    data:{
        gender="";
    },
    handleChange(e){
        //获取单选框的值
        let gender=e.detail.value;
        //把单选框的值赋值给gender
        this.serData({
            //gender:gender
            gender
```

```
        })
    }
})
```

checkbox 复选框标签

- 复选框的功能类似单选框的功能，只是复选框可以同时选中两个选项。

自定义组件

自定义组件也是有`js`,`json`,`wxml`,`wxss`组成的功能标签。

创建流程

- 在主菜单中创建一个文件夹，命名为components
- 在该文件加下创建子文件夹，在其中放置组件的wxml,wxss,js,json文档

调用流程

- 在要调用components的文件中，打开.json文件。在usingComponents{}中加入`key-value`.其中value是url。

```
usingComponents{
    "Tabs":"../../components/Tabs/Tabs"
}
```

- 在要调用components的文件中，打开.wxml,加入组件标签。

```
<Tab></Tab>
```

父组件向子组件传递数据

通过属性值传递数据

- 在父组件的`<components>`标签中加入想要传给子组件的属性名和属性值。

```
<!-- 父组件 wxml -->
<Tabs aaa="a123a"></Tabs>
```

- 在子组件的properties中加入接受数据的数据类型和默认值
    - type 要接受的数据的类型
    - value 默认值

```

```

```
//子组件的js文件
properties:{
    aaa:{
        //type要接受的数据的类型
        type:String,
        //value 默认值
        value:""
    }
}
```

- 在子组件的`wxml`文件中使用接受到的数据，使用方式同data中的数据。用`{{data}}`绑定

```
<!-- 子组件的wxml -->
<view>{{aaa}}</view>
```

等到父组件调用子组件的时候，再传标签过来，替代插槽的位置。

```
<!-- 子组件的wxml文件 -->
<view class="tabs_content">
    <slot></slot>
</view>
```

```
<!-- 父组件的wxml文件 -->
<Tabs>
    <block wx:if="{{tabs[0].isActive}}">0</block>
    <block wx:elif="{{tabs[1].isActive}}">1</block>
    <block wx:elif="{{tabs[2].isActive}}">2</block>
    <block wx:else>3</block>
</Tabs>
```

示例

通过父组件向子组件传递数据，实现界面导航栏设计

- 导航栏分为：
  - tabs_title 标题模块: title_items 4个字块
  - tabs_content 内容模块
- `title_items` **用数组列表渲染方式显示，数据内容存放在父组件中。通过传值在子块接收响应。**
- 对标题的标签栏进行`bindTap`的事件绑定。**被绑定的事件要写在components的methods中。**
- 通过判断获取列表中的状态值`isActive`，在class中加上对应标签，显示对应的样式
- data-index是获取的引索值

```
<!-- components/Tabs/Tabs.wxml -->
<view class="tabs">
  <view class="tabs_title">
    <view
    wx:for="{{tabs}}"
    wx:key="id"
    class="title_item {{item.isActive?'active':''}}"
    bindtap="handleItemTap"
    data-index="{{index}}"
    >
      {{item.name}}
    </view>
  </view>
    <view class="tabs_content">
    <slot></slot>
  </view>
</view>
```

- 获取每次点击标签的引索值
- 获取原数组，深拷贝一份。在对拷贝后的数组内容进行修改，不影响原来传入值的内容。即实现：通过判断获取到的引索值和当前的遍历的引索值是否一致来更改isActive的值。

```
// components/Tabs/Tabs.js
Component({
  /**
   * 组件的属性列表
   */
  properties: {
    tabs:{
      type:Array,
      value:[]
    }
  },

  /**
   * 组件的初始数据
   */
  data: {},

  /**
   * 组件的方法列表
   * 1.页面.js 文件中 存放事件回调函数的时候 存放在data同层级下
   * 2.组件.js 文件中 存放事件回调函数的时候 必须存放在methods中
   */
  methods: {
    handleItemTap(e){
      /*
      1 绑定点击事件 需要在methods中绑定
      2 获取被点击的索引
      3 获取原数组
```

```
        4 对数组循环
          1 给每一个循环项 选中属性 都改为false
          2 给当前的索引项 添加激活选中效果
        */
       //2 获取索引
       const{index}=e.currentTarget.dataset;
       //3 获取data中的原数组
       //结构 对 复杂类型进行结构的时候 复制了一份变量的引用
       //最严谨的做法 重新拷贝一份数组 在对数组备份进行处理
       //let tabs=JSON.parse(JSON.stringify(this.data.tabs));
       //不要直接修改this.data
       let {tabs} = this.data;
       //4 循环数组
       // [].forEach 遍历数组 遍历数组的时候 v 被修改了， 也会导致原数组被修改。
       tabs.forEach((v,i)=>i===index?v.isActive=true:v.isActive=false);
       this.setData({
         tabs
       })
     }
   }
})
```

- 从父组件通过属性向子组件传递值

```
<!--pages/wxproject/3/component.wxml-->
<Tabs tabs="{{tabs}}"></Tabs>
```

- 把tabs数组的值存放在父组件data中。

```
// pages/wxproject/3/component.js
Page({
  /**
   * 页面的初始数据
   */
  data: {
    tabs:[
      {
        id:0,
        name:"首页",
        isActive:true
      },
      {
        id:1,
        name:"原创",
        isActive:false
      },
      {
        id:2,
```

```
        name:"分类",
        isActive:false
      },
      {
        id:3,
        name:"关于",
        isActive:false
      }
    ]
  }
})
```

改示例存在一个*问题*，即当点*击*切*换*之后，父*组*件*isActive*所*显*示的*boolean值*没有被更改。因*为*它的判断和修改是在子*组*件中，通*过备*份完成的，并没有返回*给*父*组*件。因此引入后一个知*识*点，子*组*件向父*组*件从*传递*数据的功能。

子组件向父组件传递数据

- 触发父组件中的自定义事件 同时传递数据给 父组件 this.triggerEvent("父组件自定义事件的名称"，要传递的参数)
- 通过触发事件完成
- 父组件中绑定的属性为：**bind+自定义事件的名称**

```
// components/Tabs/Tabs.js 子组件js文件
Component({
  properties: {
    tabs:{
      type:Array,
      value:[]
    }
  },
  data: {},
  methods: {
    handleItemTap(e){
    //获取索引
     const{index}=e.currentTarget.dataset;
    // 触发父组件中的自定义事件 同时传递数据给 父组件
      this.triggerEvent("ItemChange",{index});
    }
  }
})
```

- 把原本子组件中所完成的判断功能通过父组件来完成。

```
// pages/wxproject/3/component.js 父组件js文件
Page({
  //自定义事件，接受子组件传递的数据
  handleItemChange(e){
      //接受传递过来的参数
```

```
        const{index} = e.detail;
        let {tabs}=this.data;
        tabs.forEach((v,i)=>i===index?v.isActive=true:v.isActive=false);
        this.setData({
          tabs
        })
    },
    data: {
      tabs:[
        {
          id:0,
          name:"首页",
          isActive:true
        },
        {
          id:1,
          name:"原创",
          isActive:false
        },
        {
          id:2,
          name:"分类",
          isActive:false
        },
        {
          id:3,
          name:"关于",
          isActive:false
        }
      ]
    }
  })
```

slot 插槽标签

放在子组件标签中，起到占位符的作用。

小程序生命周期

生命周期就是一系列由小程序自己定义触发时间的事件。

应用的生命周期 app.js

> 1. onLaunch应用第一次启动时，触发事件。
>
> - 获取用户的个人信息。在其他页面中使用这个信息。

```
onLaunch(){}
```

> 2. onShow应用被用户看到时，触发事件。

- 对应用的页面效果 重置

```
onShow(){}    //每次切换后台，再切回来会触发。
```

3. onHide应用隐藏时，触发事件。

- 暂停或清除计时器

```
onHide(){}
```

4. onError应用的代码发生了报错的时候，触发事件。

- 在应用发生代码报错的时候，收集用户的错误信息，通过异步请求，将错误信息发送到后台。
- err 形成存的是报错的内容。

```
onError(err){}
```

5. onPageNotFound应用第一次启动时，如果找不到第一个入口页面时，触发事件。

如:下方的代码中，url:'11/22/33'并不存在。但是第一次启动的时候已经进入了app.js,在发生跳转的时候报错。这种情况并不会触发onPageNotFound.

```
//appjs
App({
  onLaunch({
    wx.navigateTo({
      url:'11/22/33'
    });
  })
})
```

- 如果页面不存在了，通过js方式重新跳转页面，跳转到第二个首页。用wx.navigateTo()

```
onPageNotFound(){}
```

页面的生命周期

- data
- onLoad 发送异步请求，初始化一些功能
- onShow 页面显示

- `onReady` 页面首次渲染完毕
- `onHide` 页面隐藏（1. 应用隐藏；2. 页面切换）
- `onUnload` 页面卸载 （页面跳转中，涉及到关闭页面功能时会触发。e.g: navigate open-type="redirect"）
- `onPullDownRefresh` 监听用户下拉动作，做一些页面数据/效果初始化
- `onReachBottom` 页面上拉到底事件的处理函数；上拉加载下一页数据
- `onShareAppMessage` 转发操作
- `onPageScroll` 页面一滚动 就触发
- `onResize` 页面尺寸发生改变时触发：小程序发生横屏/竖屏切换时，触发。

  要允许页面可以实现横竖屏切换，需要在json中加`"pageOrientation":""auto`
- `onTabItemTap` 仅针对tabBar页面，点击自己的tabItem时触发事件