```
/*
*@ 视图层：WXML语法、WXSS样式、事件系统、WXS脚本语法、微信小程序组件（基础和视图容器组件）
*@ 张斌
*@ time 2021/12/21
**/
```

一、知识点回顾

```
    1.微信开发概念
        微信开发平台(服务端)
            提供微信支付 微信登录等接口
        微信公众平台（前端）
            订阅号  服务号  企业微信（企业号）  微信小程序（微信小游戏）
    2.微信小程序：
        2017  微信之父张小龙   简化版的app应用
        优势和劣势  2条
    3.开发前期准备：
        账号注册  工具下载安装  项目创建 目录介绍
    4.配置介绍：
        全局配置app.json
            默认首页 页面路由  窗口表现  tabar  debug等
        页面配置.json
            窗口表现window
        优先级：页面配置 > 全局配置
    5.场景值：
        记录用户进行小程序的方式，比如通过扫码、连接跳转进入等
    6.小程序逻辑层：
        架构：逻辑层  视图层   vc
        逻辑层：
            注册应用
                app.js文件  app({})  应用级别生命周期函数 监听函数  自定义函数  自定义变量
                获取实例：getApp()
            注册页面
                .js文件   page({})   页面初始化数据  页面级别生命周期函数  自定义函数
    7.  模块化
            ES6  commonjs
```

二、作业

```
1.切换选项卡
2.五星好评

数据绑定 列表渲染 事件绑定
```

三、微信小程序视图层

框架的视图层由 WXML 与 WXSS 编写，由组件来进行展示。将逻辑层的数据反应成视图，同时将视图层的事件发送给逻辑层。
●WXML(WeiXin Markup language) 用于描述页面的结构。
●WXS(WeiXin Script) 是小程序的一套脚本语言，结合 WXML，可以构建出页面的结构。
●WXSS(WeiXin Style Sheet) 用于描述页面的样式。
●组件(Component)是视图的基本组成单元。

3.1、WXML语法(重点)

wxml语法包括：数据绑定 列表渲染 条件渲染 模板 引用

1.wxml概述及作用

WXML（WeiXin Markup Language）是框架设计的一套标签语言，结合基础组件、事件系统，可以构建出页面的结构,功能类似于html；wxml语法包含数据绑定、列表渲染、条件渲染、模板和引用

2.数据绑定

WXML 中的动态数据均来自对应 Page 的 data。通过{{}}语法在对应的.wxml页面解析输出即可
页面第一次渲染需要的业务数据，当页面加载的时候，data会以json字符串的形式将数据从逻辑层传递到视图层

使用语法:

```
<!-- ====================数据绑定====================== -->
<!-- {{}} -->

<!-- 1.组件内容绑定 -->
<!-- <view>hello</view> -->
<!-- <view>{{str}}</view> -->

<!-- 2.属性  id  注意引号和花括号之间不要有空格-->
<!-- <view id="1001"></view> -->
<!-- <view id="{{id}}"></view> -->


<!-- 3.属性 class -->
<!-- <view class="box">hello</view> -->
<!-- <view class="{{className}}">hello</view> -->


<!-- 4.属性  style -->
<!-- <view style="width: 100%;height: 200px;background-color:
green;">hello</view> -->
<!-- <view style="width: 100%;height: {{height}}px;background-color:
green;">hello</view> -->


<!-- 5.属性  值是bool类型  关键字 -->
<!--错误实例 <view hidden="true">hello</view> -->
```

```
<!-- <view hidden="{{true}}">hello</view>
<view hidden="{{false}}">world</view> -->


<!-- 6.运算  ++ -- * 三元运算  数据路径运算（array  object）   -->
<!-- <view>{{1+2}}+2</view>
<view>{{'hello'+'world'}}+你好</view> -->

<!-- <view>{{5>13 ? '大于' : '小于'}}</view> -->
<!-- <view hidden="{{5>12 ? false : true}}">hello</view> -->

<!-- <view>{{[1,2,3,4]}}</view> -->
<!-- <view>{{array[2]}}</view>
<view>{{user.name}}</view>
<view>英雄的姓名是：{{userInfo[0].name}}</view> -->
```

使用注意：

```
1.注意引号和花括号之间不要有空格
2.如果属性的值是bool类型，  注意bool类型一定嵌套在{{}}
3.在组件上不管是属性的值还是内容只要是变量一定嵌套在{{}}
```

3.列表渲染wx:for

在组件上使用 `wx:for` 控制属性绑定一个数组，即可使用数组中各项的数据重复渲染该组件。
默认数组的当前项的下标变量名默认为 `index`，数组当前项的变量名默认为 `item`

使用语法：

```
wx:for  默认下标名:index  默认值名:item   修改下标名:wx:for-index   修改item值名：
wx:for-item

<!-- wx:for  默认下标名:index  默认值名:item   修改下标名:wx:for-index   修改item值
名：wx:for-item -->
<!-- 1.循环数组 -->
<!-- <view wx:for="{{array}}" wx:for-index='i' wx:for-item='m'>
    下标是：{{i}}---值是:{{m}}
</view> -->

<!-- 2.循环对象 -->
<!-- <view wx:for="{{user}}">
    下标是：{{index}}---值是:{{item}}
</view> -->


<!-- 3.循环对象数组 -->
<!-- <view wx:for="{{userInfo}}">
    下标是：{{index}}---值是:{{item.name}}
</view> -->

<!-- 4.循环字符串 将字符串切割成数组 循环数组 -->
<!-- <view wx:for="12345">
    下标是：{{index}}---值是:{{item}}
```

```
</view> -->

<!-- 5.循环注意事项  引号和花括号之间不要空格-->
<view wx:for="{{[1,2,3]}}">
    下标是：{{index}}---值是:{{item}}
</view>
=======================================
<!-- 数组和字符串运算  切割成数组  循环数组 -->
<view wx:for="{{[1,2,3]}} ">
    下标是：{{index}}---值是:{{item}}
</view>
=======================================
<view wx:for="{{[1,2,3]+' '}}">
    下标是：{{index}}---值是:{{item}}
</view
```

使用注意：

```
1. 引号和花括号之间不能有空
```

wx:key：

作用：

```
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。
```

定义key：

```
1.如果列表是静态的可以不用定义key;
2.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。(数据的id)
3.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。
```

```
<view
    wx:for="{{array}}"
    wx:for-index='i'
    wx:for-item='m'
    wx:key='*this'
>
    下标是：{{i}}---值是:{{m}}
</view>


<!-- 3.循环对象数组 -->
<!--错误实例 <view wx:for="{{userInfo}}" wx:key="{{item.id}}"> -->
<!-- <view wx:for="{{userInfo}}" wx:key="id">
    下标是：{{index}}---值是:{{item.name}}
</view> -->
```

使用注意：

1.key的添加不需要{{}}嵌套

4.条件渲染

在小程序开发过程中，我们也会经常碰见分支操作，这里我们称为条件渲染，而在小程序中实现条件渲染，我们使用wx:if

使用语法:

```
wx:if   wx:elif   wx:else

<!-- wx:if   wx:elif   wx:else -->
<!-- <view wx:if="{{score >90}}">优秀</view>
<view wx:elif="{{score >70}}">良好</view>
<view wx:else>一般</view> -->
```

if VS hidden:

`wx:if` 也是**惰性的**，如果在初始渲染条件为 `false`，框架什么也不做，在条件第一次变成真的时候才开始局部渲染。

相比之下，`hidden` 就简单的多，组件始终会被渲染，只是简单的控制显示与隐藏。

一般来说，`wx:if` 有更高的切换消耗而 `hidden` 有更高的初始渲染消耗。因此，如果需要频繁切换的情景下，用 `hidden` 更好，如果在运行时条件不大可能改变则 `wx:if` 较好。

```
<!-- if  hidden(vue:v-if VS  v-show) -->
<!-- <view wx:if="{{true}}">hello</view>
<view wx:if="{{false}}">world</view>

=====================================
<view hidden="{{true}}">hello</view>
<view hidden="{{false}}">world</view> -->
```

block使用:

```
<!-- block使用 不会在页面输出显示 主要放置循环（wx:for）和判断(wx:if)  hidden不能使用block-->

<!-- <block wx:for="{{array}}">
    下标是{{index}}--值是{{item}}
</block> -->

<view class="info">
    <block wx:if="buffer">
        <!-- 用户登录 -->
        <image src="../../images/0.jpg"></image>
        <text>鲁班</text>
    </block>
    <!-- 用户没有登录提示登录 -->
    <button wx:else>请登录</button>
</view>
```

5.模板

WXML提供模板（template），可以在模板中定义代码片段，然后在不同的地方调用。可以提高代码的复用率

定义模板:

```
<template name='模板的名称'>
    业务组件.....
</template>
```

使用模板:

```
<template is='模板的名称' />
```

模板传参:

```
data=""    <template is='模板的名称' data="param1,"param2...." />
```

使用语法:

```
<!--代码复用  template  name  is   data  -->


<!-- 1.模板基本使用 -->
<!-- 1.1定义模板 name-->
<!-- <template name="demo1">
    <view>用户的姓名是：李四</view>
    <view>用户的年龄是：20</view>
</template> -->
<!-- 1.2使用模板 -->
<!-- <template is="demo1" /> -->


<!-- 2.模板传参  data -->
<!-- 2.1定义模板 -->
<!-- <template name="demo2">
    <view>用户的姓名是：{{user.name}}</view>
    <view>用户的年龄是：{{user.age}}</view>
</template> -->

<!-- 2.2使用模板 -->
<!-- <template is='demo2' data="{{user}}"></template> -->


<!-- 3.模板传参  data 展开运算符-->
<!-- 3.1定义模板 -->
<!-- <template name="demo3">
    <view>用户的姓名是：{{name}}</view>
    <view>用户的年龄是：{{age}}</view>
</template> -->
```

```
<!-- 3.2使用模板 -->
<!-- <template is='demo3' data="{{...user}}"></template> -->


<!-- 4.模板传参  data 展开运算符-->
<!-- 4.1定义模板 -->
<!-- <template name="demo4">
    <view>用户的姓名是：{{name}}</view>
    <view>用户的年龄是：{{age}}</view>
</template> -->

<!-- 4.2使用模板 -->
<!-- <template
    wx:for="{{userInfo}}"
    wx:key="id"
    is='demo4'
    data="{{...item}}"

></template> -->
```

使用注意：

```
1.模板本身是没有数据的，需要使用的页面进行传递（data）
2.模板本身没有样式，需要使用的页面进行设置
```

6.引用

```
WXML 提供两种文件引用方式import和include。
```

1.import
import引入目标文件中定义好的template。

```
<!-- 1. import :引入目标文件(.wxml文件)中定义好的template。-->

<!-- 1.1引入demo5.wxml文件中的模板 -->
<!-- <import src="../../common/temp/demo5.wxml" /> -->
<!-- 1.2使用demo5模板 -->
<!-- <template is='demo5' data="{{...user}}" /> -->
```

import作用域：
import 有作用域的概念，即只会 import 目标文件中定义的 template，而不会 import 目标文件 import的template。

```
<!-- 2.import 作用域 -->
<!-- 即只会 import 目标文件中定义的 template，而不会 import 目标文件 import的template。
-->
<!-- 2.1引入demo5.wxml文件 -->
<!-- <import src="../../common/temp/demo6.wxml" /> -->
<!-- Template `demo6` not found -->
<!-- <template is='demo6' /> -->
```

2.include

include 可以将目标文件除了 <template/> 和wxs模块外的整个代码引入，相当于是拷贝到 include 位置。

```
<include src="文件路径" />
<!-- 3.1引入demo5.wxml文件中的代码片段 -->
<include src="../../common/temp/demo5.wxml" />
```