WXML模板学习

1. 数据绑定

数据绑定的基本原则：

1. 在data中定义数、据
2. 在WXML中使用数据

1.1 在data中定义页面的数据

在页面对应的js文件中把数据定义到data对象当中。

JavaScript | 复制代码

```javascript
Page({
  data: {
    //字符串类型的数据
    info: 'init data',
    //数组类型的数据
    msgList: [{msg:'hello'},{msg:'world'}]
  }
})
```

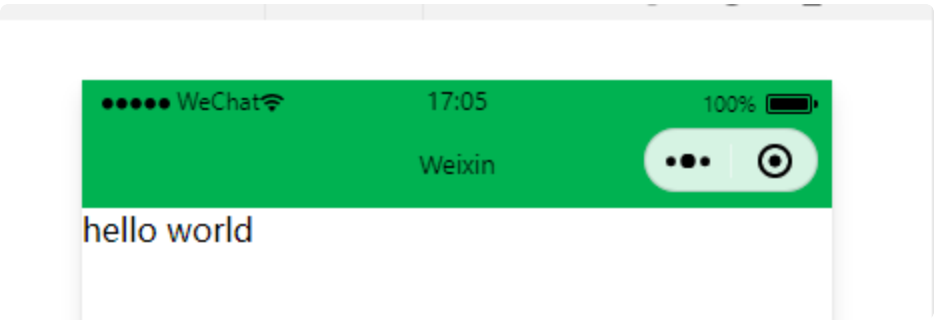
1.2 Mustache语法的格式

把data中的数据绑定到页面渲染当中，使用Mustache语言(双大括号)将变量包起来即可，语法格式为：

JavaScript | 复制代码

```javascript
<view>{{要绑定的数据名称}}</view>
```

```JavaScript
Page({
    //
    data: {
        info:'hello world'
    },
})

<view>{{info}}</view>
```



1.3 Mustache语法的应用场景

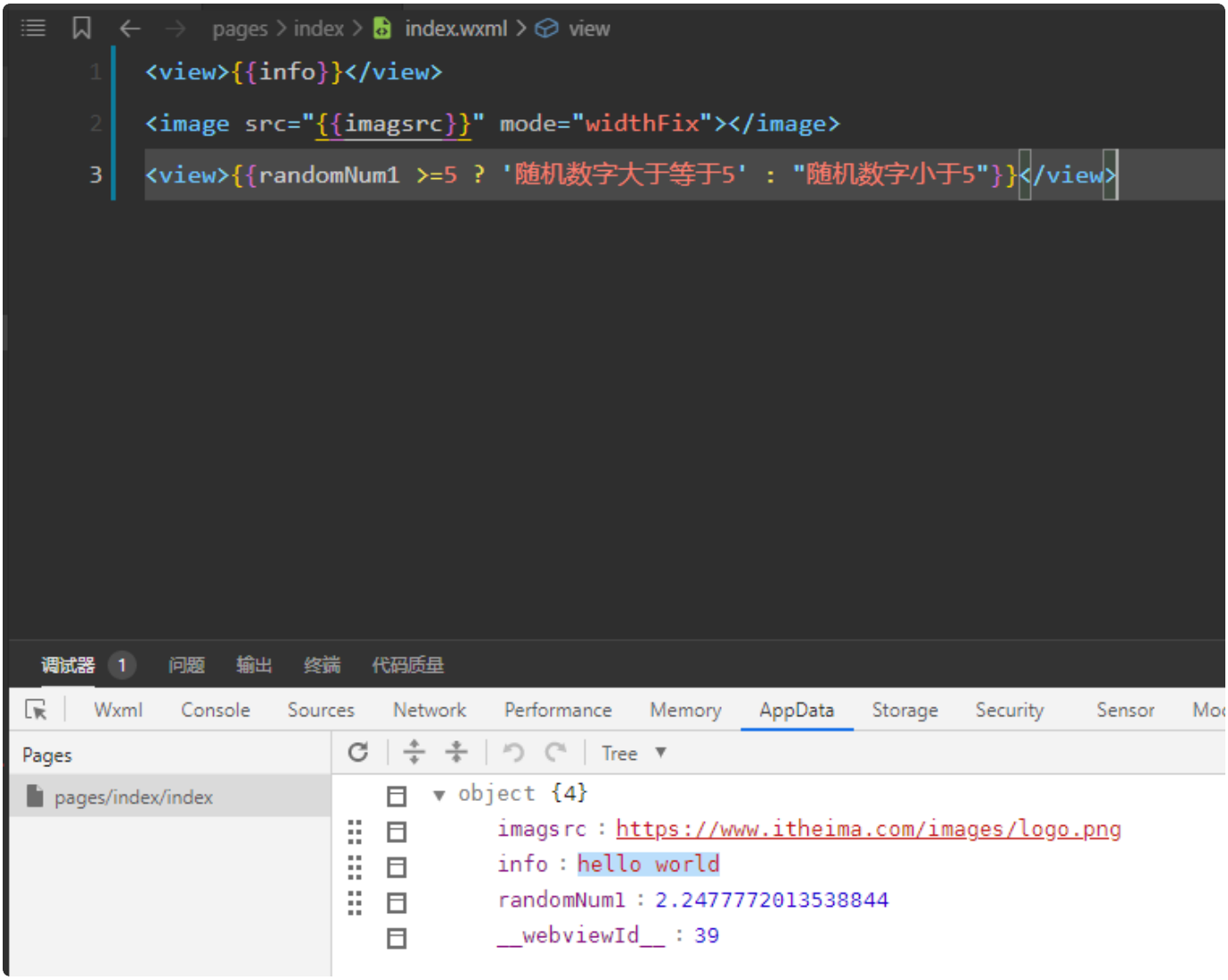
应用场景如下：

- 绑定内容
- 绑定属性
- 运算（三元运算 算数运算）

```JavaScript
Page({
    //
    data: {
        info:'hello world',
        imagsrc: 'https://www.itheima.com/images/logo.png'
    },
})

<view>{{info}}</view>
<image src="{{imagsrc}}" mode="widthFix"></image>>
```

JavaScript | 复制代码

```javascript
Page({
    //
    data: {
        info:'hello world',
        imagsrc: 'https://www.itheima.com/images/logo.png',
        randomNum1: Math.random()*10
    },
})

<view>{{info}}</view>
<image src="{{imagsrc}}" mode="widthFix"></image>
<view>{{randomNum1 >=5 ? '随机数字大于等于5' : "随机数字小于5"}}</view>
```



2. 事件绑定

事件是渲染层到逻辑层的通讯方式，通过事件可以将用户在渲染层产生的行为反馈到逻辑层来进行业务的处理。

2.1 小程序中常用的事件

类型	绑定方式	事件描述
tap	bindtap或者是bind:tap	手指触摸后马上离开，类似于HTML中的点击事件
input	bindinput或者是bind:input	文本框的输入事件
change	bindchange或者是bind:change	状态改变的时候触发

2.2 事件对象的属性列表

当事件回调触发的时候会收到一个事件对象event，它的详细属性如下表所示：

属性	类型	说明
type	String	事件类型
timeStamp	Integer	页面打开到触发事件所经历的毫秒数
targer	Object	触发事件的组件的一些属性值集合
currentTarget	Object	当前组件的一些属性值的集合
detail	Object	额外的信息
touches	Array	触摸事件，当前停留在屏幕上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

2.3 target和currentTarget的区别

target是触发该事件的源头组件，而currentTarget是当前事件所绑定的组件。

2.4 bindtap的语法格式

在小程序当中不存在HTML中的onclick鼠标点击事件，而是通过tap事件相应用户的触摸行为。

1. 通过bindtap可以为组件绑定tap触摸事件，语法如下：

```
<button type="primary" bindtap="btnTapHundler">按钮</button>
```

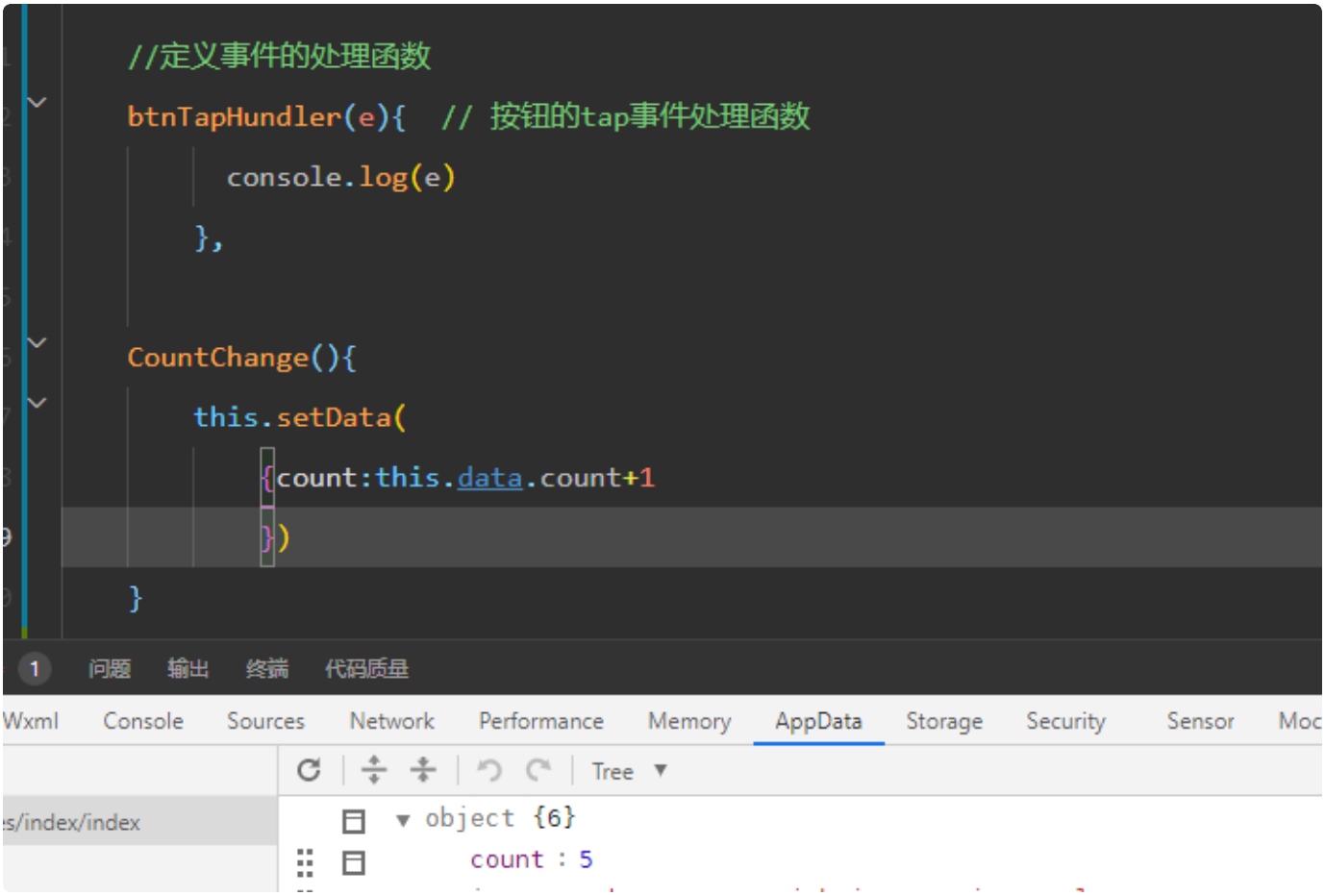
2. 在页面的.js文件中定义对应的事件处理函数，事件通过形参event(一般写成e)来接收：

```
Page({
  BtnTapHundler(e){  // 按钮的tap事件处理函数
    console.log(e)
  }
})
```

```
Component "pages/index/index" does not have a method "btnTapHundler" to handle event "tap".
{type: "tap", timeStamp: 18058, target: {…}, currentTarget: {…}, mark: {…}, …}
{type: "tap", timeStamp: 19530, target: {…}, currentTarget: {…}, mark: {…}, …}
```

2.5 在事件处理函数中为data中的数据赋值

通过调用this.setData(dataObject)方法可以为data中的数据重新赋值，示例如下：



2.6 事件传参

小程序中的事件传参比较特殊，不能在绑定事件的同时为事件处理函数传递参数。

错误传参方法 JavaScript 复制代码

```
<button type="primary" bindtap="btnTapHundler(123)">按钮</button>
```

可以为组件提供data-*自定义属性传参，其中*代表的是参数的名字，示例代码如下：

JavaScript 复制代码

```
<button type="primary" bindtap="btnTapHundler" data-info="{{2}}">事件传参
</button>
```

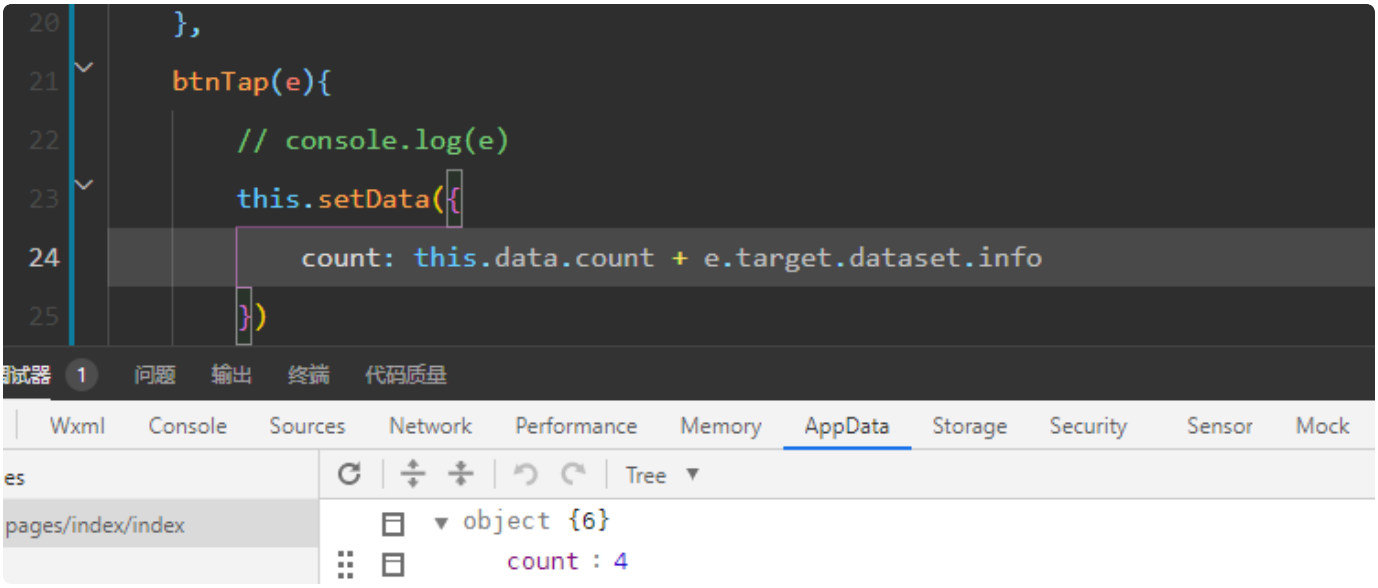
最终：

- info会被解析为参数的名字

- 数值2会被解析为参数的值

在事件处理函数当中通过event.target.dataset.参数名就可以获取到具体参数的值，代码如下：

```javascript
btnTapHundler(event){
  //dataset是一个对象包含了所有通过data-*传递过来的参数项
  console.log(event.target.dataset)
  //通过dataset可以访问到具体参数的值
  console.log(event.target.dataset.info)
}
```



2.7 bindinput的语法格式

在小程序当中通过input事件来响应文本框的输入事件，语法格式如下：

1. 通过bindinput可以为文本框绑定输入事件：

```javascript
<input bindinput="inputhandler"></input>
```

2. 在页面的.js文件中定义事件处理函数：

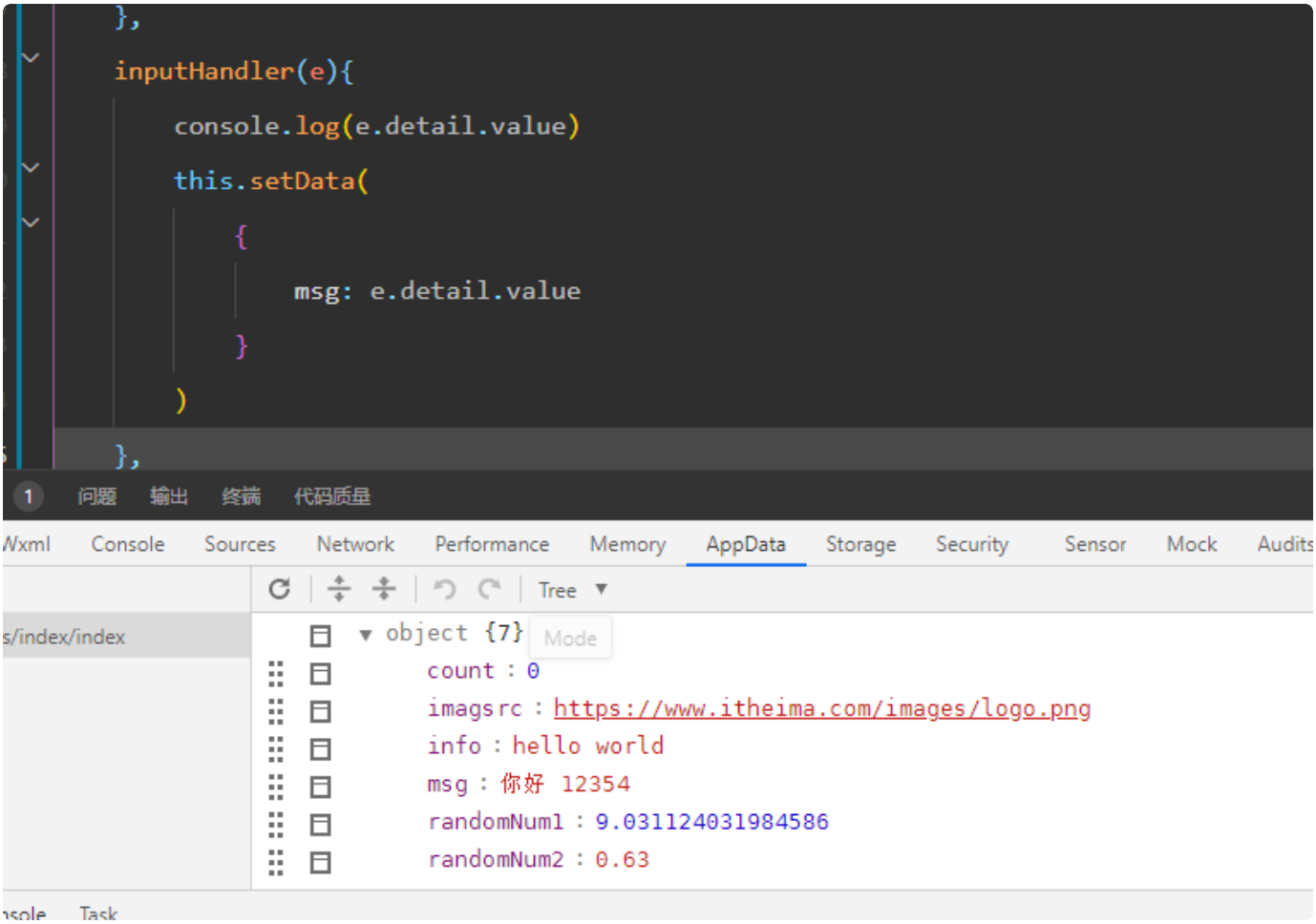
```JavaScript
inputHandler(e) {
  //e.detail.value是变化后文本框的最新的值
  console.log(e.detail.value)
}
```

2.8 实现文本框和data之间的数据同步

实现步骤：

1. 定义数据
2. 渲染结构
3. 美化样式
4. 绑定input事件处理函数

```JavaScript
input{
    border: 1px solid royalblue;
    margin: 5px;
    padding: 5px;
    border-radius: 3px;
}

    inputHandler(e){
        console.log(e.detail.value)
        this.setData(
            {
                msg: e.detail.value
            }
        )
    },
```



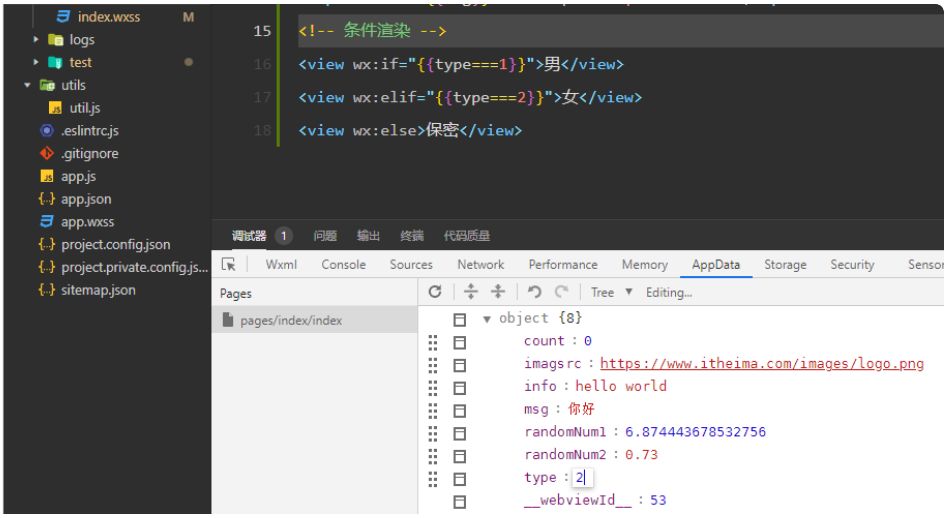
2.9 条件渲染

在小程序中使用wx::if="{{condition}}"来判断是否需要渲染该代码块：

JavaScript | 复制代码

```
<view wx:if="{{condition}}">True</view>
```

也可以使用wx:elif和wx:else来添加else判断。



2.9.1 结合<block>标签使用wx:if

如果要一次性控制多个组件的展示和隐藏，可以使用一个<block></block>标签将多个组件包装起来，并在<block>标签上使用wx:if控制属性。

<block>并不是一个组件，只是一个包裹性质的容器不会在页面上做出任何渲染。

JavaScript | 复制代码

```
<view wx:if="{{false}}">  //隐藏
    <view>view1</view>
    <view>view2</view>
    <view>view3</view>
</view>

<block wx:if="{{true}}">  //显示
    <view>view4</view>
    <view>view5</view>
    <view>view6</view>
</block>
```

2.9.2 hidden使用

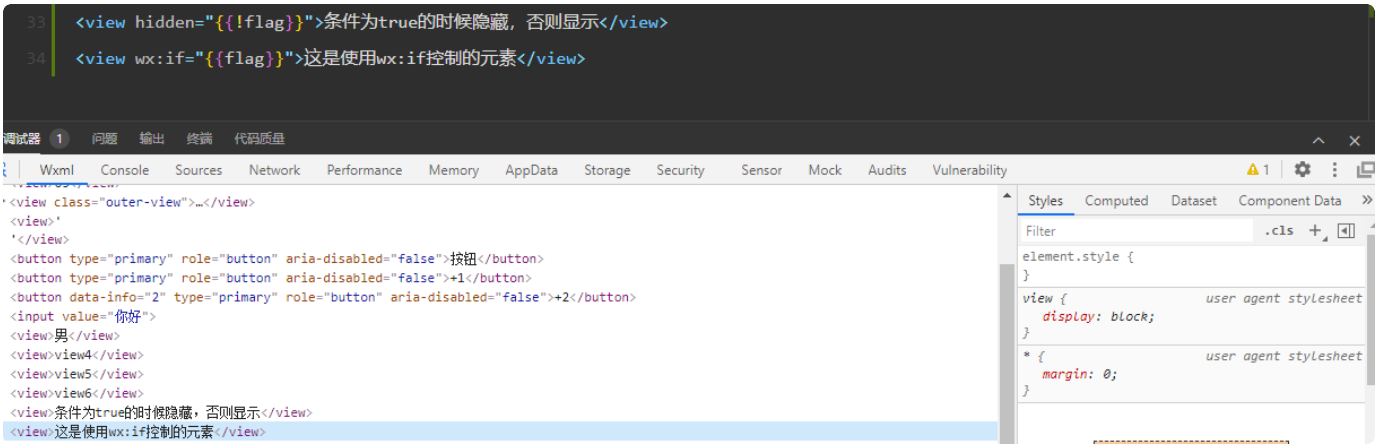
在小程序中直接使用hidden="{{condition}}"也可以控制元素的显示和隐藏。

```
</block>
<view hidden="{{flag}}">
条件为true表示隐藏，若为false则显示
</view>
```

2.9.3 二者的比较

1. 运行方式不同

- wx:if是以动态创建和移除元素的方式来控制元素的展示和隐藏
- hidden是以切换样式的方式(display：none/block)来控制元素的显示和隐藏



2. 使用建议

- 频繁切换的时候建议使用hidden
- 控制条件比较复杂的时候建议使用wx:if搭配wx:elif，wx:else来进行展示与隐藏的切换

2.9.4 列表渲染

通过wx:for可以根据指定的数组循环渲染重复的组件结构，语法示例如下：

```
JavaScript | 复制代码

<view wx:for="{{array}}">
  索引是:{{index}} 当前项是:{{item}}
</view>
```

默认情况下当前循环的索引使用index表示，当前循环使用item表示。

2.9.5 wx:key的使用

类似于Vue列表渲染中的key，小程序在实现列表渲染的时候也建议渲染出来的列表项指定唯一的key值，从而提高渲染的效率，示例代码如下：

```javascript
userliet:[
            {id:1,name:"小红"},
            {id:2,name:"小兰"},
            {id:3,name:"小黄"}
        ]
```

```
<view wx:for="{{userliet}}" wx:key="id">
  当前项是:{{item.name}}
</view>
```

调试器 1 问题 输出 终端 代码质量

Wxml Console Sources Network Performance Memory AppData

```
索引是:2 当前项是:小米
</view>
<view>--------------------------</view>
<view>
  当前项是:小红
</view>
<view>
  当前项是:小兰
</view>
<view>
  当前项是:小黄
</view>
</page>
```

3. WXSS模板样式

WXSS(weixin Style Sheets)是一套样式语言，用于美化WXML的组件样式，类似于网吧也开发中的CSS。

3.1 WXSS与CSS的关系

WXSS具有CSS的大部分特性，同时WXSS还对CSS进行了扩充以及修改来适应微信小程序的开发。

与CSS相比，WXSS扩展的特性有：

- rpx尺寸单位
- @import样式导入

3.1.1 rpx尺寸单位

rpx(responsive pixel)是微信小程序独有的，用来解决屏适配的尺寸单位。

rpx的实现原理是很简单，即鉴于不同设备屏幕的大小不同为了实现屏幕的自动适配，rpx吧所有设备的屏幕在宽度上等分为了750份，即当前屏幕的总宽度为750rpx.。

小程序在不同设备上运行的时候会自动把rpx的样式单位换算为对应的像素单位来进行渲染，从而实现屏幕适配。

3.1.2 样式导入

使用wxss的@import可以导入外联的样式表。

@import需要跟上外联样式表的相对路径，用;表示语句结束。

JavaScript | 复制代码

```
//common.wxss
.small-p{
  padding:5px;
}

//app.wxss
@import "common.wxss";
.middle-p{
  padding:5px;
}
```

JavaScript | 复制代码

```
.username{
    color: red;
}
```

当前项是:小红
当前项是:小兰
当前项是:小黄

3.2 全局样式和局部样式

全局样式定义在app.wxss中的样式会作用于每一个页面。

在页面的.wxss文件中定义的样式为局部样式，只作用于当前页面。

- 当局部样式和全局样式冲突的时候根据就近原则局部样式会覆盖全局样式
- 当局部样式的权重大于或者等于全局样式的时候才会覆盖全局的样式

4. 全局配置

小程序根目录下的app.json文件是小程序的全局配置文件，常用的配置项如下：

1. pages:记录当前小程序所有页面的存放路径
2. window：全局设置小程序窗口的外观
3. tabBar：设置小程序底部的tabBar效果
4. style：是否启用新版的组件样式

4.1 设置导航栏的标题

设置步骤：app.json->window->navigatorBarTitleText

4.2 设置导航栏的背景色

设置步骤：app.json->window->navigatonBarBackgroundColor

4.3 设置导航栏的标题颜色

设置步骤：app.json->window->navigatonBarTextStyle

4.4 全局开启下拉刷新功能

设置步骤：app.json->window->把enablePullDownRefresh的值设置为true.

在app.json中启用下拉刷新功能会作用于每一个小程序页面。

4.5 设置下拉刷新时窗口的背景颜色

当全局开启下拉刷新功能之后默认窗口背景为白色，如果自定义下拉刷新窗口背景色，设置步骤为：app.json–>window–>为backgroundColor指定16进制的颜色值。

4.6 设置下拉刷新时loading的样式

设置步骤为：app.json–>window为backgroundTextStyle指定dark值。

4.7 设置上拉触底的距离

上拉触底是移动端的专有名词，通过手指在屏幕上的上拉滑动操作从而加载更多数据的行为。

设置步骤：app.json–>window–>为onReachBottomDistance设置新的数值。默认情况下为50px。

4.8 tabBar

tabBar是移动端应用常见的页面效果，用于实现多页面的快速切换，小程序中通常将其分为：

1. 底部tabBar
2. 顶部tabBar

注意:

1. tabBar中只能配置最少2个,最多5个tab标签
2. 当渲染顶部tabBar时不显示icon,只显示文本.

tabBar的六个组成部分:

1. backgroundColor:tabBar的背景颜色
2. selectIconPath:选中时的图片路径
3. borderStyle:tabBar上边框的颜色
4. iconPath:未选中时的图片路径
5. selectedColor:tab上的文字选中时候的颜色
6. color:tab上文字的默认(未选中)颜色

```JavaScript
"tabBar": {
      "list": [
          {
              "pagePath": "pages/index/index",
              "text": "index"
          },
          {
                "pagePath": "pages/test/test",
                "text": "test"

          }
      ]
  },
```

5. 网络数据请求

5.1 小程序当中网络数据请求的限制

处于安全方面的考虑,小程序官方堆数据接口的请求做出了以下的两个限制:

1. 只可以请求HTTPS类型的接口
2. 必须将接口的域名添加到信任列表当中

5.2 配置request合法域名

需要描述:假设在自己的微信小程序当中希望请求https://www.escook.cn/域名下的接口

配置步骤:登录自己的微信小程序管理后台->开发->开发设置->服务器域名->修改request合法域名

注意事项:

1. 域名只支持https协议
2. 域名不可以使用IP地址或者是localhost

3. 域名必须经过ICP备案
4. 服务器与域名一个月内最多可以申请5次修改

5.3 发起get请求

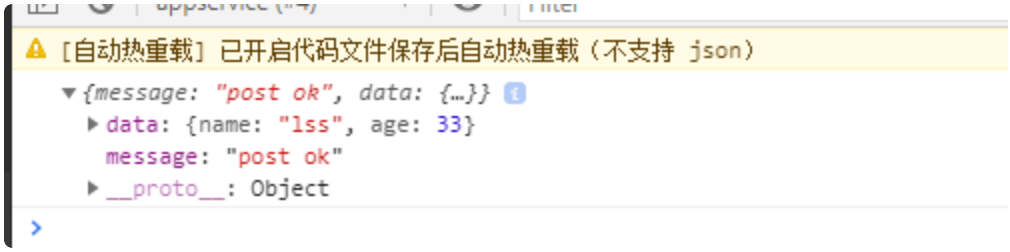
调用微信小程序提供的wx.request()方法可以发起GET请求.

JavaScript | 复制代码

```
    getInfo(){
        wx.request({
          url: 'https://www.escook.cn/api/get',
          method:'GET',
          data: {
              name:"zs",
              age:20
          },
          success: (res) =>{
              console.log(res.data)
          }
        })
    },
```

5.4 发起post请求

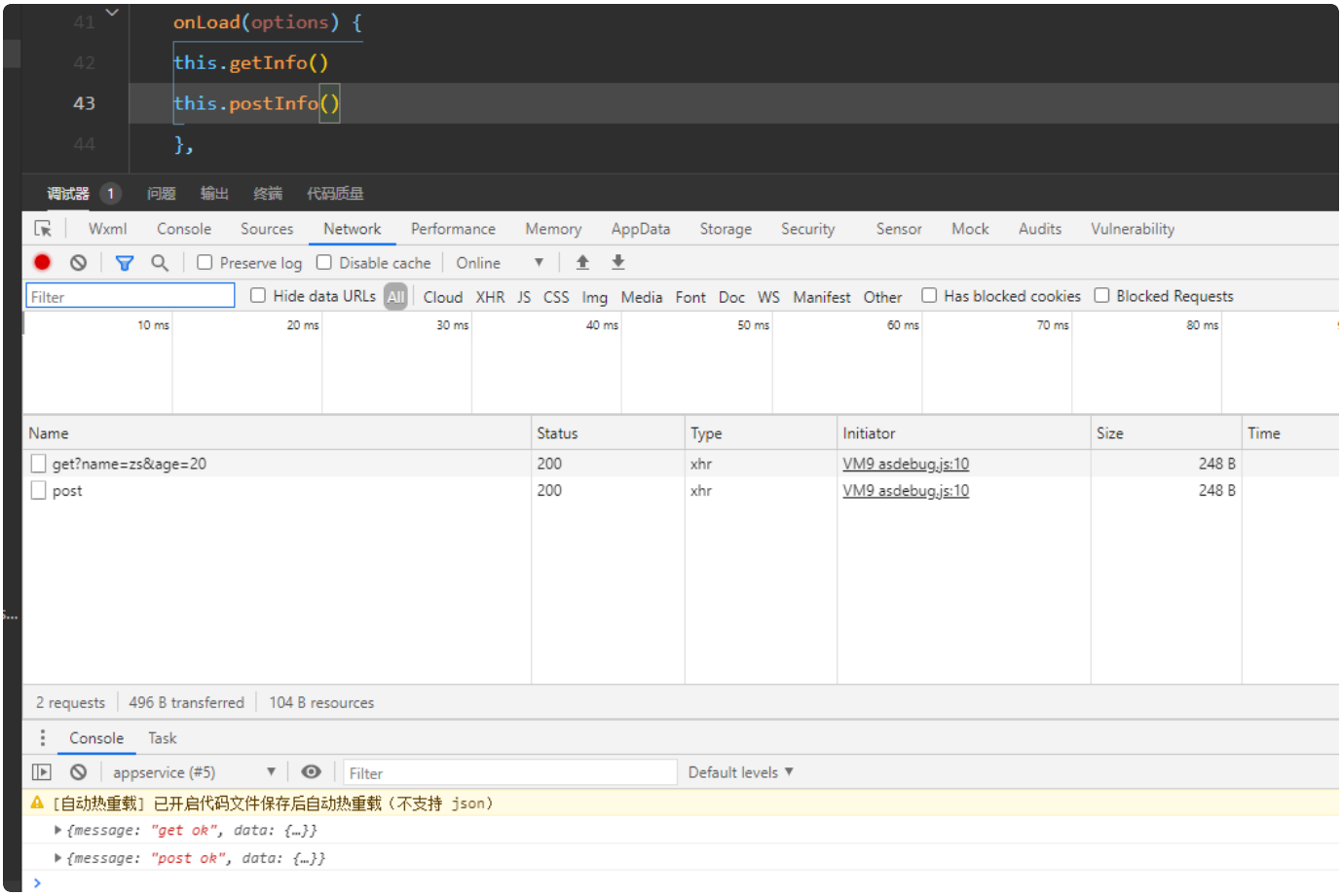
```javascript
    // 发起post请求
    postInfo(){
        wx.request({
          url: 'https://www.escook.cn/api/post',
          method:'POST',
          data: {
              name:"lss",
              age:33,
          },
        //   res接受服务器响应回来的结果
          success: (res) =>{
              console.log(res.data)
          }
        })
    },
```



5.5 在页面加载的时候请求数据

在很多情况下我们需要在页面刚加载的时候自动请求一些初始化的数据,此时需要子啊页面的onload事件中调用获取数据的函数,示例代码如下:

```javascript
onLoad:function(options){
  this.getSwiperList()
  this.getGridList()
},
```



5.6 跳过request合法域名校验

如果后端程序员仅仅提供了http协议的接口暂时没有提供https协议的接口,为了不耽误开发的进度可以在微信开发者工具中临时开启[开发环境不校验请求域名 TLS版本以及HTTPS证书]选项来跳过request合法域名的检验,

注意:跳过request合法域名的校验的选项仅限在开发和调试阶段使用.在项目上线的时候不可以勾选该项.

5.7 关于跨域和Ajax的说明

跨域问题只是存在于浏览器的web开发当中,由于小程序的宿主环境不是浏览器而是微信客户端,所以小程序不存在跨域的问题.

Ajax技术的核心是依赖于浏览器中XMLhttpRequest这个对象,由于小程序的宿主环境是微信客户端所以小程序不可以叫做"发起Ajax请求"而应该叫做"发起网络数据请求".

6. 页面导航

小程序中实现页面导航的两种方式：

1. 声明式导航
- 在页面上声明一个<navigator>导航组件
- 通过点击<navigator>组件实现页面跳转
2. 编程式导航
- 调用小程序的导航API来实现页面的跳转

6.1 声明式导航

6.1.1导航到tabBar页面

在使用<navigator>组件跳转到指定的tabBa页面的时候需要指定url属性和open-type属性，其中：

- url表示要跳转的页面地址吧，必须以/开头
- open-type表示跳转的方式必须为switchTab

示例代码如下：

JavaScript | 复制代码

```
<navigator url="/pages/message/message" open-type="switchTab">导航到消息页面</navigator>
```

6.1.2 导航到非tabBar页面

在使用<navigator>组件跳转到指定的非tabBa页面的时候需要指定url属性和open-type属性，其中：

- url表示要跳转的页面地址吧，必须以/开头
- open-type表示跳转的方式必须为navigate（也可以省略不写）

6.1.3 后退导航

如果需要后退到上一个页面或者是多级页面则需要指定open-type属性和delta属性，其中：

- open-type的值必须是navigateBack，表示要进行后退导航。
- delta的值必须是数字，表示要后退的层级

```JavaScript
<navigator open-type="navigateBack" delta="1">后退</navigator>
```

为了简便如果只是回退到上一个页面，那么可以省略delta属性，因为默认值就是1。

6.2编程式导航

6.2.1 导航到tabBar页面

调用wx.switchTab(Object objecrt)方法，可以跳转到tabBar页面，其中Object参数对象的属性列表如下：

属性	类型	是否必选	说明
url	string	是	需要跳转的tabBatr页面的路径，路径不可以带参数
success	function	否	接口调用成功的回调函数
fail	function	否	接口调用失败的回调函数
complete	function	否	接口调用结束的回调函数(调用成功、失败都会执行)

```JavaScript
<button bindtap="gotoMessage">跳转到message页面</button>

    gotoMessage(){
        wx.switchTab({
          url: '/pages/message/message',
        })
    },
```

6.2.2 跳转到非tabBar页面

调用wx.navigateTo(Object)方法可以跳转到非tabBar页面，其中Object对象的属性列表如下：

属性	类型	是否必选	说明
url	string	是	需要跳转的tabBatr页面的路径，路径不可以带参数
success	function	否	接口调用成功的回调函数
fail	function	否	接口调用失败的回调函数
complete	function	否	接口调用结束的回调函数(调用成功、失败都会执行)

JavaScript | 复制代码

```
<button bindtap="gotoInfo">info</button>

    gotoInfo(){
        wx.navigateTo({
          url: '/pages/info/info',
        })
    },
```

6.2.3 后退导航

调用wx.navigateBack(Object)方法可以返回上一页面或者多级页面，其中Object对象的参数对象如下：

属性	类型	是否必选	说明
delta	number	是	返回页面数，如果delta大于现有的页数就返回到首页(默认值为1)
success	function	否	接口调用成功的回调函数
fail	function	否	接口调用失败的回调函数
complete	function	否	接口调用结束的回调函数(调用成功、失败都会执行)

JavaScript | 复制代码

```
<button bindtap="goback">后退</button>

    goback(){
        wx.navigateBack()
    },
```

6.3 导航传参

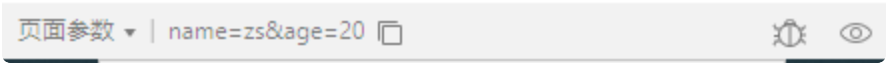
6.3.1 声明式导航传参

navigator组件的url属性用来指定要跳转的页面的路径，同时路径的后面还可以跟上参数：

- 参数与路径之间使用?分隔
- 参数键与参数值之间使用=连接
- 不同参数之间使用&分隔

JavaScript | 复制代码

```
<navigator url="/pages/info/info?name=zs&age=20">跳转到info页面</navigator>
```



6.3.2 编程式导航传参

调用wx.navigateTo(Object)方法跳转页面的时候，也可以携带参数，示例代码如下：

JavaScript | 复制代码

```
<button bindtap="gotoInfo2">跳转到info页面</button>

    gotoInfo2(){
        wx.navigateTo({
          url: '/pages/info/info?name=ls&gender=男',
        })
    },
```

6.3.3 在onload中接受导航参数

JavaScript | 复制代码

```
    onLoad(options) {
        console.log(options)
        this.setData({
            query:options
        })
    },
```

7. 页面事件

7.1 下拉刷新

启用下拉刷新有两种方式：

1. 全局开启下拉刷新

- 在app.json的window节点中将enablePullDownRefresh设置为true

2. 局部开启下拉刷新

- 在页面的.json页面中将enablePullDownRefresh设置为true

7.2 配置下拉刷新窗口的样式

在全局或页面的.json配置文件中通过backgroundColor和backgroundTextStyle来配置下拉刷新窗口的样式，其中：

- backgroundColor用来配置下拉刷新窗口的鳖精颜色，仅支持16进制的颜色值
- backgroundTextStyle用来配置下拉刷新loading的样式，仅支持dark和light

JavaScript | 复制代码

```
{
    "usingComponents": {},
    "enablePullDownRefresh": true,
    "backgroundColor": "#efefef",
    "backgroundTextStyle": "dark"
}
```

7.3 监听页面的下拉刷新事件

在页面的.js文件中通过onPullDownRefresh()函数即可监听当前页面的下拉刷新事件。

7.4 停止下拉刷新的效果

当处理完下拉刷新后，下拉刷新的loading效果会一直显示不会主动消失，所以需要手动停止loading效果，此时调用wx.stopPullDownRefresh()可以停止当前页面的下拉刷新。

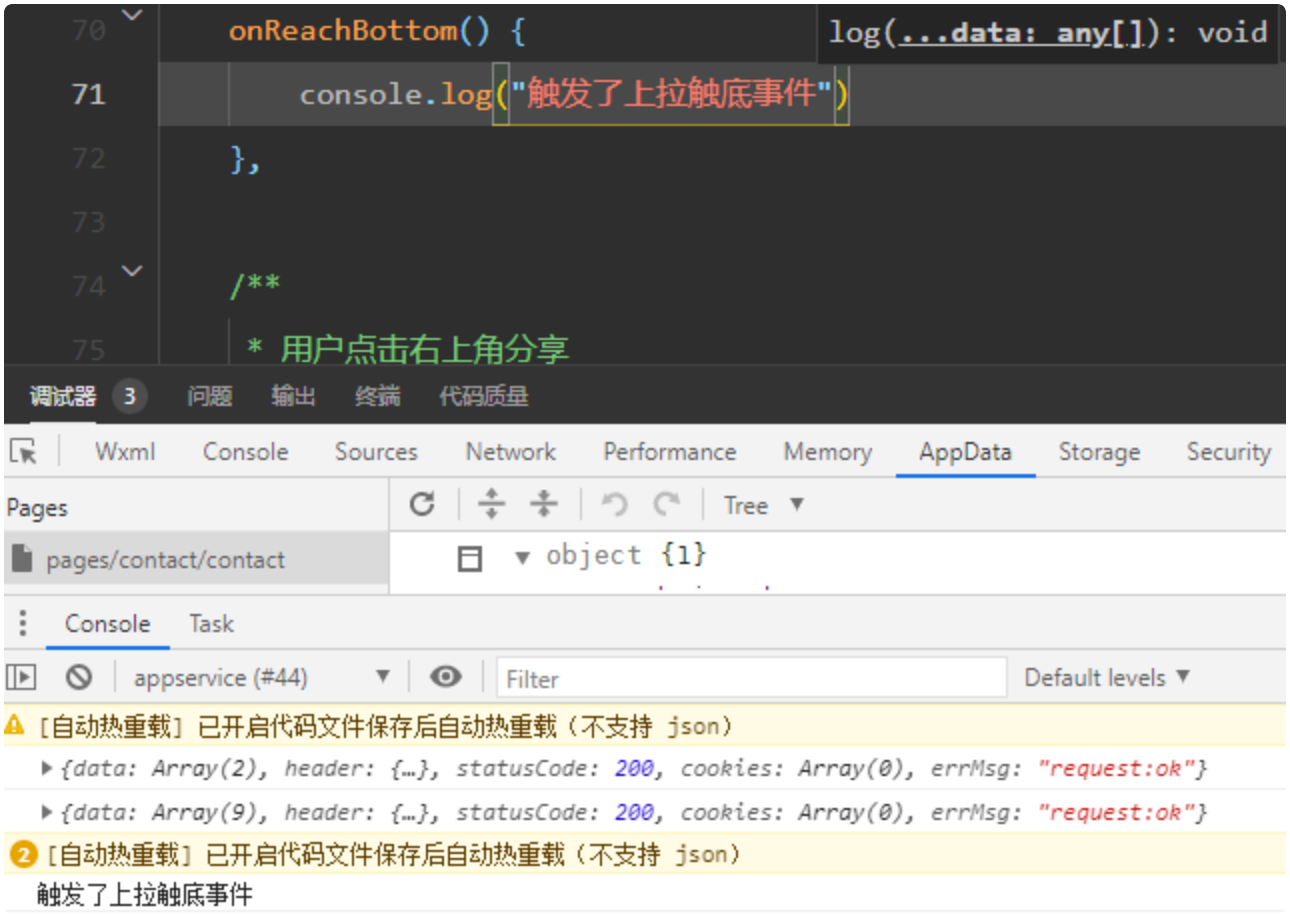
JavaScript | 复制代码

```
    onPullDownRefresh() {
        console.log("触发了下拉刷新")
        this.setData({
            count:0
        })
        wx.stopPullDownRefresh()
    },
```

7.5 上拉触底

上拉触底是移动端的专有名词，通过手指在屏幕上的上拉滑动操作来加载更多数据的行为。主要是为了实现分页的效果。

在页面的.js文件中通过onReachBottom()函数即可监听当前页面上的上拉触底事件。



7.5.1 配置上拉触底距离

上拉触底距离指的是触发上拉触底事件时候，滚动条距离页面底部的距离。

可以在全局或者页面的.json配置文件中通过onReachBottomDistance属性来配置上拉触底的距离。默认50px。

7.5.2 添加loading效果

```JavaScript
    // 需要展示loading效果
    wx.showLoading({
    title: '数据加载中',
    })

  complete:()=>{
              wx.hideLoading({})
            }
          })
```

7.5.3 对上拉触底进行节流处理

1. 在data中定义isloading节流阀
- false表示当前没有进行任何数据请求
- true表示当前进行数据请求
2. 在getColors()方法中修改isloading节流阀的值
- 在刚调用getColors()的时候将节流阀设置为true
- 在网络请求的complete回调函数中将节流重置为false
3. 在onReachVBottom中判断节流阀的值，从而对数据请求进行节流控制
- 如果节流阀的值为true表示阻止当前请求
- 如果当前节流阀的值为false表示发起数据请求

8. 生命周期

生命周期是指一个对象从创建–>运行–>销毁的整个阶段，强调的是一个时间段。

在小程序中生命周期分为了2类。分别是：

1. 应用生命周期
- 特指小程序从启动–>运行–>销毁的过程
2. 页面生命周期
- 特指小程序中每一个页面的加载–>渲染–>销毁的过程

生命周期函数：是小程序框架提供的内置函数，会伴随着生命周期自动按照次序执行

生命周期函数的作用：允许程序员在特定的时间执行某些特定的操作，例如页面刚加载的时候可以在onload生命周期函数中初始化页面的数据。

小程序的生命周期函数需要在app.js中进行声明，实例代码如下：

JavaScript | 复制代码

```
  onLaunch: function () {
    console.log("onlaunch")
  },

  /**
   * 当小程序启动，或从后台进入前台显示，会触发 onShow
   */
  onShow: function (options) {
    console.log("onshow")
  },

  /**
   * 当小程序从前台进入后台，会触发 onHide
   */
  onHide: function () {
    console.log("onhide")
  },
```

8.1 页面的生命周期函数

小程序的页面生命周期函数需要在页面的.js文件中进行声明。

JavaScript | 复制代码

```
Page({
  onLoad:function(options) {},//监听页面加载，一个页面只调用一次
  onShow:function(){},  //监听页面显示
  onReady:function(){},  //监听页面初次渲染完成，一个页面只调用一次
  onHide:function(){},  //监听页面隐藏
  onUnload:function(){}  //监听页面卸载，一个页面只调用一次
})
```

9. WXS脚本

WXS（weixin script）是小程序独有的一套脚本语言，结合WXML可以构建出页面的结构。

wxml无法调用在页面的.js文件中定义的函数，但是wxml可以调用wxs中定义的函数，因此小程序中wxs的典型应用场景就是“过滤器”。

9.1 WXS与Javascript的关系

虽然wxs的语法类似于Javascript，但是wxs和Javascript是完全两种不同的语言：

1. wxs有自己的数据类型

- number数值类型、string字符串类型、boolean布尔类型、object对象类型、function类型、
- array数组类型、date日期类型、regexp正则

2. wxs不支持类似于ES6及以上的语法形式

- 不支持：let、const、解构赋值、展开运算符、箭头函数、对象属性简写
- 支持：var定义变量、普通的function函数类似于ES5的语法

3. wxs遵循CommonJS规范

- module对象
- require()函数
- module.exports对象

9.2 WXS基础语法

9.2.1 内嵌wxs脚本

wxs代码可以编写在wxml文件中的<wxs>标签内的，就像Javascript代码可以编写在html文件中的<script>标签内的一样。

wxml文件内的每一个<wxs></wxs>标签必须提供module属性，用来指定当前wxs的模块名称方便在wxml中访问模块的成员。

```
<view>{{m1.toUpper(username)}}</view>
<wxs module="m1">
    module.exports.toUpper = function(str){   //module.exports 表示就可以共外界调用
        return str.toUpperCase()
    }
</wxs>
```

9.2.2 定义外联的wxs脚本

wxs代码还可以编写在.wxs为后缀的文件内，就像javascript代码可以编写在.js为后缀的文件中一样。

tool.wxs

```
function toLower(str) {
    return str.toLowerCase()
}

module.exports={
    toLower:toLower
}
```

在wxml中引入外联的wxs脚本的时候，必须以<wxs>标签添加module和src属性，其中;

- module用来指定模块的名称
- src用来指定要引入的脚本的路径，且必须是相对路径

```
<view>{{m2.toLower(country)}}</view>
<wxs src="../../utils/tools.wxs" module="m2"></wxs>
```

9.2.3 wxs的特点

wxs的典型应用场景就是"过滤器"，经常配合Mustache语法进行使用，但是在wxs中定义的函数不可以作为组件的事件回调函数。

错区示范 JavaScript | 复制代码

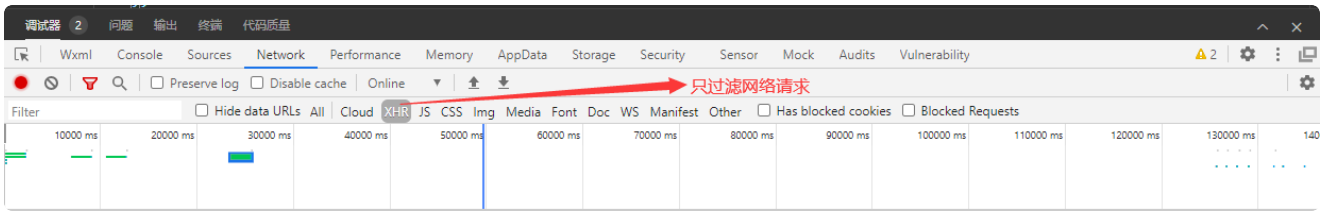
```
1    <button bindtap="m2.toLower">按钮</button>
```

wxs的运行环境和其他的Javascript代码是隔离的。

1. wxs不可以调用js中定义的函数
2. wxs不可以调用小程序提供的API

性能好

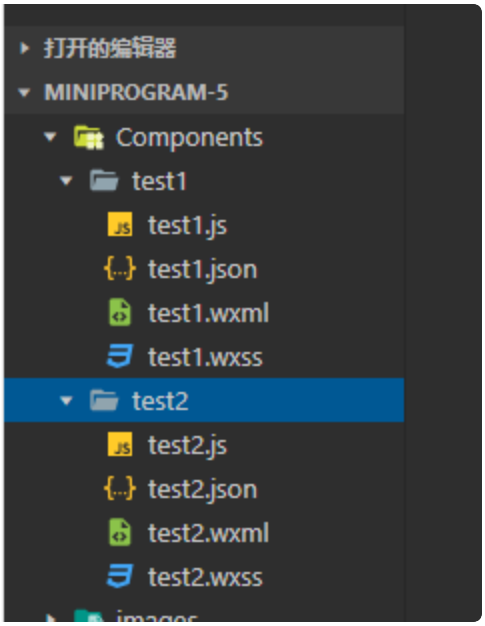
- 在ios设备上小程序内的wxs会比Javascript代码快2~20倍
- 在安卓设备上效率差异不大



10. 自定义组件

10.1创建组件

1. 在项目的根目录上鼠标右键，创建components–>test文件夹
2. 在新建的components–>test文件件上鼠标右键点击"新建components"
3. 键入组件的名称之后回车会自动生成组件对应的4个文件，后缀名是.js .json .wxml .wxss。



10.2 组件引用

在组件的引用方式可以分为：“局部引用‘、”全局引用“：

1. 局部引用：组件只能在当前被引用的页面内使用
2. 全局引用：组件可以在每一个小程序页面中使用

10.2.1 局部引用组件

在页面的.json配置文件中引用组件的方式叫做局部引用。

JavaScript | 复制代码

```
//info.json
"usingComponents": {
        "my-test1":"/Components/test1/test1"
    },

//info.wxml
<my-test1></my-test1>
```

Components/test1/test1.wxml

10.2.2 全局引用组件

在app.json全局配置文件中引用组件的方式叫做全局引用。

JavaScript | 复制代码

```
//app.json
"usingComponents": {
        "my-test1":"/Components/test1/test1"
    },

//页面.wxml
<my-test1></my-test1>
```

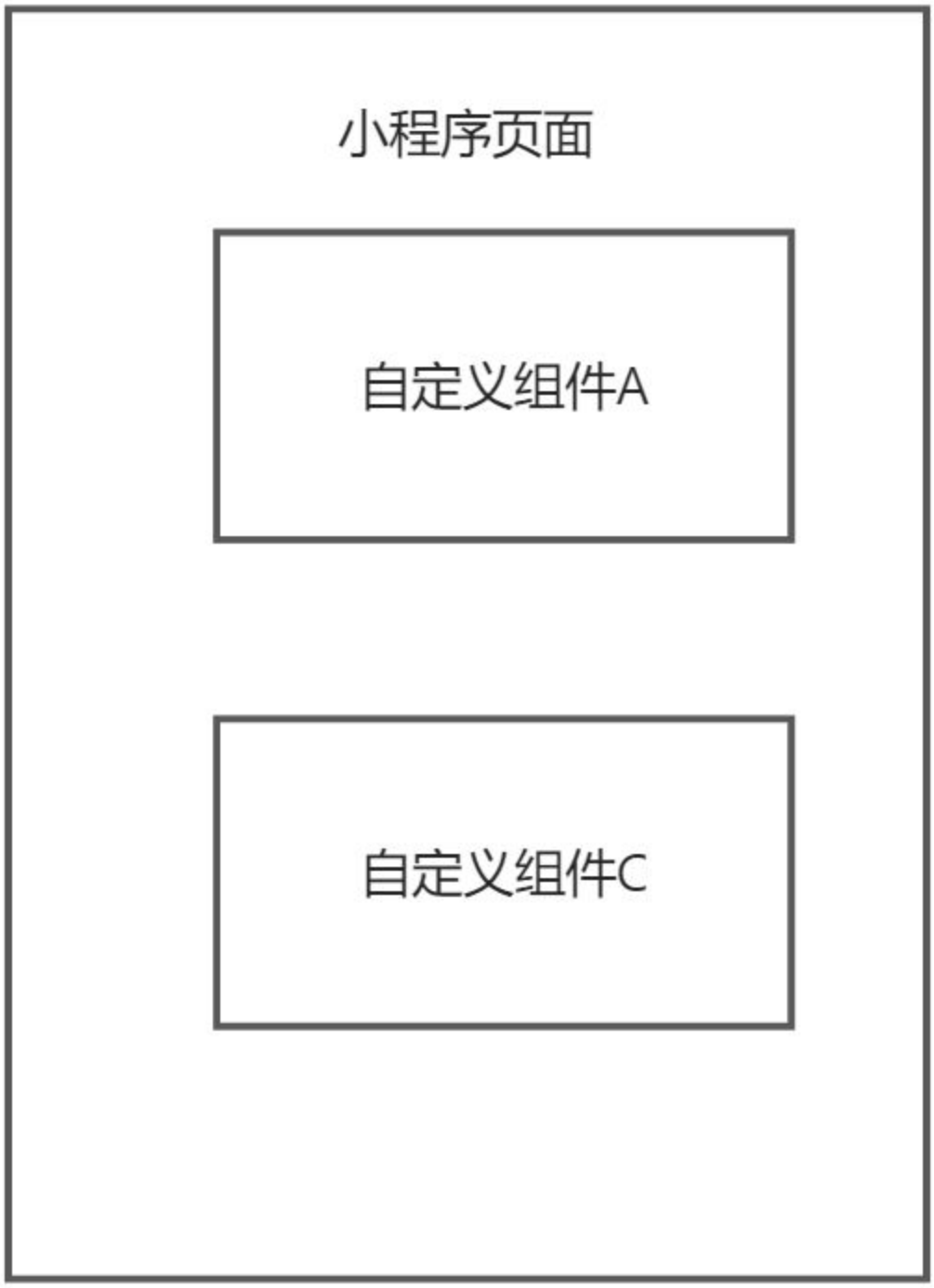
10.2.3 组件和页面的区别

从表面上看来组件和页面都是.js、.json、.wxml、.wxss这四个文件组成的，但是组件和页面的.js、.json文件有明显的的区别。

1. 组件的.json文件需要声明“component”:true属性
2. 组件的.js文件中调用的是Component()函数，页面的.js文件调用的是Page()函数
3. 组件的事件处理函数需要定义到methods节点中，而页面的处理函数只需要放到与data节点平级即可。

10.3 自定义组件的样式

默认情况下自定义组件的样式只对当前组件有效，不会影响到组件之外的UI结构。



- 组件A的样式不会影响到组件C的样式
- 组件A的样式不会影响到小程序页面的样式
- 小程序页面的样式不会影响到组件A 和C 的样式

10.3.1 组件样式隔离的注意点

- app.wxss中的全局样式对组件无效
- 只有class选择器会有样式隔离效果，id选择器，属性选择器、标签选择器都不受样式隔离的影响

建议：在组件和引用组件的页面中建议使用class选择器，不要使用id，属性，标签选择器

10.3.2 修改组件样式隔离选项

默认情况下自定义组件的样式隔离特性可以防止组件内外样式相互干扰的问题，但如果希望可以控制组件内部样式的样式那么可以通过styleisolation修改组件的样式隔离选项。

JavaScript | 复制代码

```
//在组件的.js文件中新增如下设置
Component({
  options:{
    styleIsolation: 'isolated'
  }
})

//或者是在组件的.json文件中新增如下配置
{
  styleIsolation: 'isolated'
}
```

可选值	默认值	描述
isolated	是	表示启用样式隔离，在自定义组件内外使用class指定的样式将不会相互影响
apply-shared	否	表示页面的wxss样式会影响到自定义组件，但是自定义组件的wxss不会影响到页面
shared	否	表示页面wxss样式会将影响到自定义组件，自定义组件wxss中指定的样式也会影响到页面和其他设置了apply-shared或者shared的自定义组件

10.4 自定义组件-数据，方法和属性

在小程序组件中用于组件模板渲染的私有数据需要定义到data节点中。

```JavaScript
Component({
data: {
        count: 0
    },
})
```

在小程序组件中事件处理函数和自定义方法都需要定义到methods节点中，示例代码如下：

```JavaScript
        // 点击事件处理函数
        addCount11(){
            this.setData({
                countss:this.data.countss+1
            })
            this._showCount()
        },
        _showCount(){
            wx.showToast({
              title: 'Count是'+this.data.countss,
              icon:'none'
            })
        }


    data: {
        countss: 0
    },

<view>count的值是: {{countss}}</view>
<button bindtap="addCount11">+1</button>

```

10.5 自定义组件–properties属性

在小程序组件中properties是组件的对外属性，用来接收外界传递到组件中的数据。

JavaScript | 复制代码

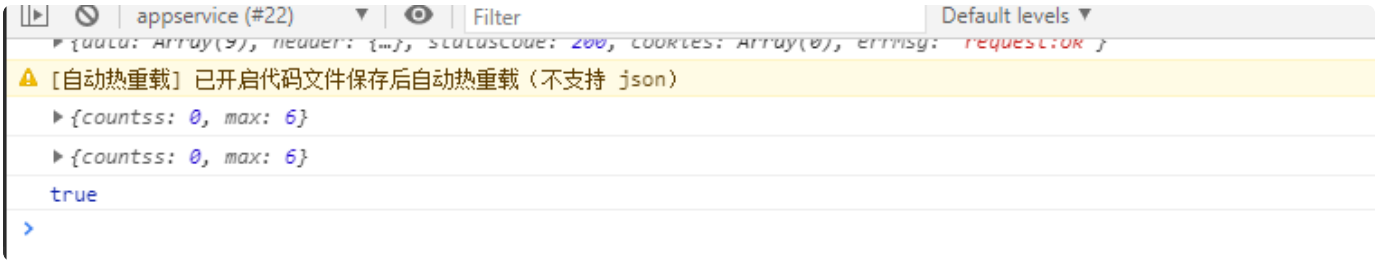
```
Component({
  //属性定义
  properties:{
  max:{   //完整定义属性的方式（如果需要指定属性默认值的时候建议使用该方式）
    type:Number,  //属性值的数据类型
    value; 10     //属性的默认值
  },
  max:Number         //简化定义属性的方式（当不需要指定属性的默认值可以使用该方式）
  }
})

<my-test1 max="10"></my-test1>
```

10.6 data和properties的区别

在小程序组件中properties属性和data数据的用法相同，都是可读可写的，但是

1. data更侧重于存储组件的私有数据
2. properties更侧重于存储外界传递到组件中的数据

```
JavaScript | 复制代码

1         showInfo(){
2             console.log(this.data)
3             console.log(this.properties)
4             console.log(this.data===this.properties)
5         }
```



10.7 使用setData修改properties的值

由于data数据和properties属性在本质上没有区别，因此properties属性值也可以渲染页面。或使用setData为properties中的属性重新赋值。

```
JavaScript | 复制代码

1         addCount11(){
2             if(this.data.countss>=this.properties.max) return
3             this.setData({
4                 countss:this.data.countss+1,
5                 max:this.properties.max+1
6             })
7             this._showCount()
8         },
```

10.8 数据监听器

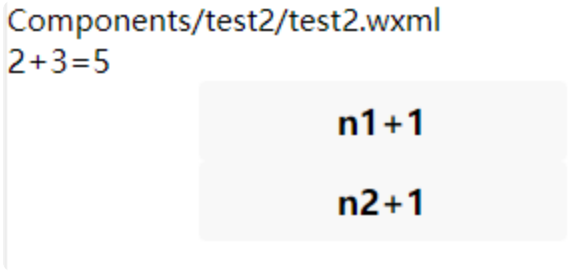
10.8.1 基本语法

数据监听器用于监听和相应人么和属性和数据字段的变化，从而执行特定的操作，类似于Vue中的watch监听器，在小程序组件中数据监听器的基本语法是：

```JavaScript
Component({
  observers:{
    '字段A','字段B':function(字段A的新值,字段B的新值) {
      // do something
  }
  }
})
```

10.8.2 基本用法

```JavaScript
methods: {
        addN1(){
            this.setData({
                n1:this.data.n1+1
            })
        },
        addN2(){
            this.setData({
                n2:this.data.n2+1
            })
        }
    },
    observers:{
        'n1,n2':function(newN1,newN2) {
            this.setData({
                sum:newN1+newN2
            })
        }
    }

<view>{{n1}}+{{n2}}={{sum}}</view>
<button bindtap="addN1">n1+1</button>
<button bindtap="addN2">n2+1</button>
```



10.8.3 监听数据对象属性的变化

数据监听器支持监听对象中单个或者多个属性的变化：

```javascript
Component({
  observers:{
    '对象.属性A,对象.属性B': function(属性A的新值,属性B的新值) {
      //以下三个情况会触发该监听器
      //为属性A赋值
      //为属性B赋值
      //为对象赋值
    }
  }
})
```

```JavaScript
    methods: {
        changeR(){
            this.setData({
                'rgb.r':this.data.rgb.r+5 >=255 ?255:this.data.rgb.r+5
            })
        },
        changeG(){
            this.setData({
                'rgb.g':this.data.rgb.g+5 >=255 ?255:this.data.rgb.g+5
            })
        },
        changeB(){
            this.setData({
                'rgb.b':this.data.rgb.b+5 >=255 ?255:this.data.rgb.b+5
            })
        }
    },
    observers:{
        'rgb.r,rgb.g,rgb.b':function(r,g,b) {
            this.setData({
                fullcolor:`${r},${g},${b}`
            })
        }
    }
```

如果某个对象需要监听的属性太多，为了方便使用可以使用通配符**来监听对象中所有属性的变化。

```JavaScript
    observers:{
        'rgb.**':function(obj) {
            this.setData({
                fullcolor:`${obj.r},${obj.g},${obj.b}`
            })
        }
    }
```

10.8.4 纯数据字段

纯数据字段是指不用于页面渲染的字段。

应用场景：某些data中的字段既不会展示在界面上也不会传递给其他的组件，仅仅在当前组件的内部使用，带有这种特性的data字段适合设置为纯数据字段。

在Component构造器的options节点中指定pureDataPattern为一个正则表达式，字段符合这个正则表达式的字段将成为纯数据字段，实例代码如下：

JavaScript | 复制代码

```
    options:{
        // 所有以下划线开头的变量都是纯数据字段
        pureDataPattern:/^_/
    },

    data: {
        _rgb:{
            r:0,
            g:0,
            b:0
        },
        fullcolor:'0,0,0'
    },
```

10.8.5 组件的生命周期

小程序组件的可用的生命周期函数如下：

生命周期函数	参数	描述说明
created	无	在组件实例刚刚被创建的时候执行
attached	无	在组件实例进入页面节点树的时候执行
ready	无	在组件视图层布局完成后执行
moved	无	在组件实例被移到节点树的另外一个位置时候执行
detached	无	在组件实例被从页面节点树移除的时候执行
error	Object Error	每当组件方法抛出错误的时候执行

1. 组件刚被创建好的时候created生命周期函数就会被触发
- 此时还不可以调用setData
- 通常这个生命周期函数中只应该给组件的this添加一些自定义的属性字段
2. 在组件完全初始化完毕，进入页面节点树后attached生命周期函数会被触发
- 此时this.data已经初始化完毕
- 这个生命周期函数很有用，大多数的初始化工作可以在这个时候进行（比如请求获取数据）
3. 在组件离开页面节点树的时候detached生命周期函数会被触发
- 退出一个页面时候，会触发页面内每个自定义组件的detached生命周期函数
- 此时适合做一些清理性质的工作

在小程序组件中，生命周期函数可以直接定义在Component构造器的第一级参数中，可以在lifetimes字段内进行声明（推荐的方式，优先级最高）。

```javascript
Component({
//推荐用法
    lifetimes:{
      attached() {},
      detached() {},

},
//旧方式的定义
  attached(){},
  detached(){},
})
```

10.8.6 组件所在页面的生命周期

在自定义组件中组件坐在的页面生命周期函数有下面三个：

生命周期函数	参数	描述
show	无	组件坐在的页面被展示的时候执行
hide	无	组件所在页面被隐藏的时候执行
resize	Object Size	组件所在页面尺寸变化时候执行

```javascript
    pageLifetimes:{
        show(){
            console.log('show')
        },
        hide(){
            console.log('hide')
        },
        resize(){
            console.log('resize')
        }
    }
})
```

```javascript
        _randomColor(){
            this.setData({
                _rgb:{
                    r:Math.floor(Math.random()*256),
                    g:Math.floor(Math.random()*256),
                    b:Math.floor(Math.random()*256)
                }
            })
        }
    pageLifetimes:{
        show(){
            console.log('show')
            this._randomColor()
        },
    }
```

10.8.7 插槽

在自定义组件的wxml中可以提供一个<slot>节点（插槽）用于承载组件使用者提供的wxml结构。

1. 单个插槽：在小程序中默认每一个组件只允许使用一个<slot>进行占位，这种叫做单个插槽。

```javascript
<view>
    <view>这是组件内部结构</view>
    <slot></slot>
</view>

<my-test4>
    <view>
        这是通过一个插槽填充的内容
    </view>
</my-test4>
```

2. 多个插槽：可以在组件的,js文件中进行先烈的方式来进行引用：

```JavaScript
    options:{
        multipleSlots:true
    },

<my-test4>
<view slot='before'>这是通过一个插槽填充的内容 before</view>
    <!-- <view>
        这是通过一个插槽填充的内容
    </view> -->
<view slot='after'>这是通过一个插槽填充的内容 after</view>
</my-test4>

<view>
    <slot name='before'></slot>
    <view>这是组件内部结构</view>
    <!-- <slot></slot> -->
    <slot name='after'></slot>
</view>
```

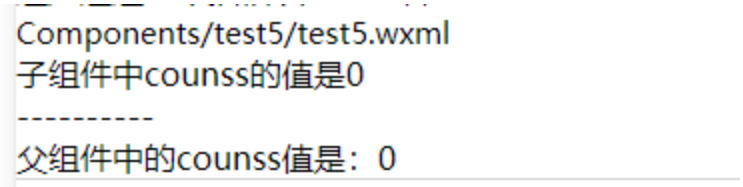
10.8.8 父子组件之间的通信

1. 父子组件之前通信的三种方式

- 属性绑定：用于父组件向子组件的指定属性设置数据，仅能设置JSON兼容的数据
- 事件绑定：用于子组件向父组件传递数据，可以传递任意数据
- 获取组件实例：父组件可以通过this.selectComponent()获取子组件的实例对象。这样就可以直接访问子组件的任意数据和方法。

属性绑定用于实现父向子传值，而且只可以传递普通类型的数据，无法将方法传递给子组件

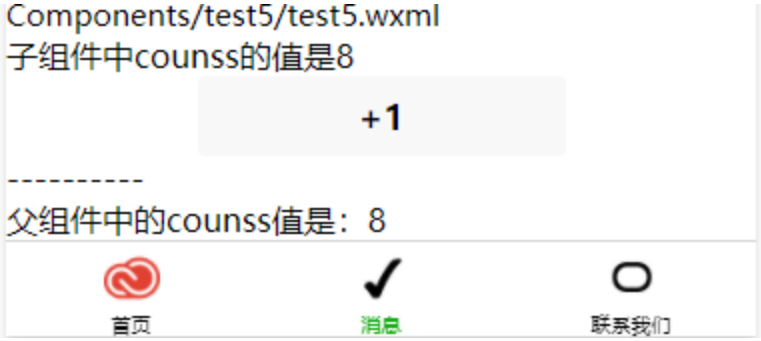
```javascript
//父组件的data节点
data:{
  count : 0
}

//父组件的wxml结构
<my-test5 counss="{{counss}}"></my-test5>
<view>----------</view>
<view>父组件中的counss值是：{{counss}}</view>

//test5.js
    properties: {
        counss:Number  //子组件属性绑定
    },
//test5.xml
<view>子组件中counss的值是{{counss}}</view>
```



事件绑定的步骤

1. 在父组件的js中定义一个函数，这个函数即将通过自定义事件的形式来传递给子组件
2. 在父组件的wxml中，通过自定义事件的形式将步骤一中定义的函数引用传递给的子组件
3. 在子组件的js中通过调用this.triggerEvent('自定义事件名称',{/*参数对象*/})来讲数据发送到父组件
4. 在父组件的js中通过e.detail获取子组件传递过来的数据

```
    SyncCount(e){
        console.log('synccount')
        console.log(e)
        console.log(e.detail.value)
        this.setData({
            counss:e.detail.value
        })
    },

<my-test5 counss="{{counss}}" bind:sync="SyncCount"></my-test5>
<view>----------</view>
<view>父组件中的counss值是: {{counss}}</view>

addCountss(){
            this.setData({
                counss:this.properties.counss+1
            })
            // 触发自定义事件 将数值同步给父组件
            this.triggerEvent('sync',{value:this.properties.counss})
        }
```



获取组件实例

可在父组件里调用this.selectComponent("id或者class选择器"),获取子组件的实例对象，从而直接访问子组件的任意数据和方法，调用的时候需要传入一个选择器，例如this.selectComponent(".my-component")。

```JavaScript
    getChild(){
        // 下面两种方式都可以 但是不可以提供标签选择器
        // const child = this.selectComponent('.customA')
        const child = this.selectComponent('#CA')
        console.log(child)
        child.setData({
            counss:child.properties.counss+1  //注意使用child而不是this
        })
         child.addCountss()
    },

      addCountss(){
            this.setData({
                counss:this.properties.counss+1
            })
            // 触发自定义事件 将数值同步给父组件
            this.triggerEvent('sync',{value:this.properties.counss})
        }
```

10.8.9 自定义组件–behaviors

behaviors是小程序中用于实现组件间代码共享的特性，类似于Vue.js中的“mixins”。每一个behaviors可以包含一组属性、数据、声明，生命周期函数和方法，组件引用它的时候。属性、方法、数据会被合并到组件中。每一个组件可以引用多个behavior，behavior也可以引用其他的behavior。

在组件中使用require()方法导入需要的behavior，挂载后即可访问behavior的数据和方法，示例代码如下：

```javascript
const myBehavior = require('../../behaviors/my-behavior')
Component({
    behaviors:[myBehavior],

<view>在behavior中定义的用户名: {{username}}</view>

module.exports  = Behavior({
    data:{
        username: 'zs'
    },
    properties:{},
    methods:{}

})
```

11. 使用npm包

小程序目前已经支持使用npm安装第三方包，从而提供小程序的开发效率，但是在小程序中都是用npm包有下面三个限制：

1. 不支持依赖于Node.js内置库的包
2. 不支持依赖于浏览器内置对象的包
3. 不支持依赖于C++插件的包

11.1 Vant Weapp

Vant Weapp是开源的一套小程序UI组件库，用于快速搭建小程序应用，使用的是MIT开源许可协议，对商业比较友好。

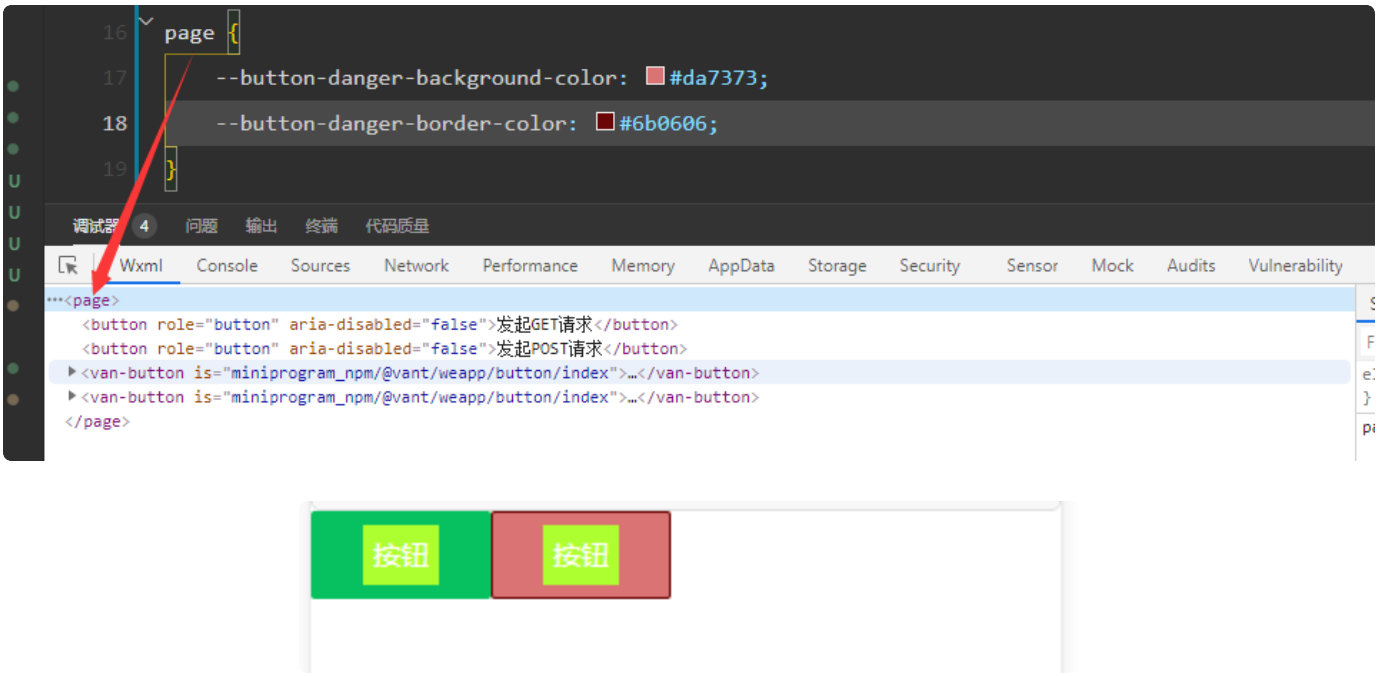
安装Vant Weapp

```
npm init -y

将 app.json 中的 "style": "v2" 去除，小程序的新版基础组件强行加上了许多样式，难以覆盖，不关闭将造成部分组件样式混乱。

打开微信开发者工具，点击 工具 -> 构建 npm，并勾选 使用 npm 模块 选项，构建完成后，即可引入组件。
```

```
  "usingComponents": {
    "van-button": "@vant/weapp/button/index"
  }

<van-button type="primary">按钮</van-button>
```

11.2 定制全局主题样式

在app.wxss中写入CSS变量就可以对全局有效。



定制主题

11.3 API Promose化

默认情况下小程序官方提供的异步API都是基于回调函数实现的，例如网络请求的API需要按照下面的方式来进行调用：

JavaScript | 复制代码

```
wx.request({
  method:'',
  url:'',
  data:{},
  success:()=>{},
  fail:()=>{},
  complete:()=>{}
})
```

缺点：容易造成回调地狱的问题导致可读性 维护性差。

> API Promise化指的是通过额外的配置将官方提供的基于回调函数的异步API升级为基于Promise的异步API，从而提高代码的可读性、维护性、避免回调地狱的问题。

11.3.1实现API Promise

在小程序中实现API Promise化主要依赖于miniprogram-api-promise这个第三方的npm包，安装和使用步骤如下：

JavaScript | 复制代码

```
npm install --save miniprogram-api-promise@1.0.4

需要删除node_miniprogram_npm文件夹重新构建一下
```

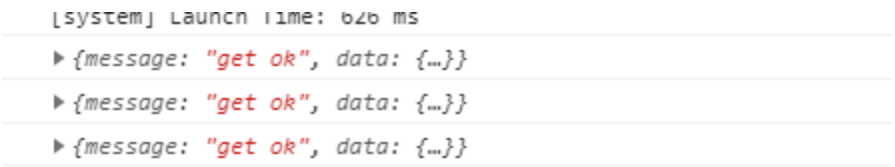
JavaScript | 复制代码

```
import{promisifyAll} from './miniprogram_npm/miniprogram-api-promise/src/promise.js'
const wxp = wx.p = {}
promisifyAll(wx,wxp)
```

11.3.2 调用Promise化之后的异步API

JavaScript | 复制代码

```
async getInfos(){
    // const res = await wx.p.request({
    const {data:res} = await wx.p.request({
        method:'GET',
        url:'https://www.escook.cn/api/get',
        data:{
            name: 'zs',
            age: 10
        }
    })
    console.log(res)
},
```



12. 全局数据共享

全局数据共享（状态管理）是为了解决组件之间数据共享的问题。在小程序中可以使用mobx-miniprogram配合mobx-miniprogram-bindings实现全局数据共享。其中：

1. mobx-miniprogram用来创建Store实例对象
2. mobx-miniprogram-bindings用来把Store中的数据共享或者方法绑定到组件或者页面里使用。

12.1 安装Mobx

JavaScript | 复制代码

```
npm install --save mobx-miniprogram@4.13.2 mobx-miniprogram-bindings@1.2.1
重新构建将node_modules下相应的包传入到miniprogram_npm文件夹中
```

12.2 使用Mobx

JavaScript | 复制代码

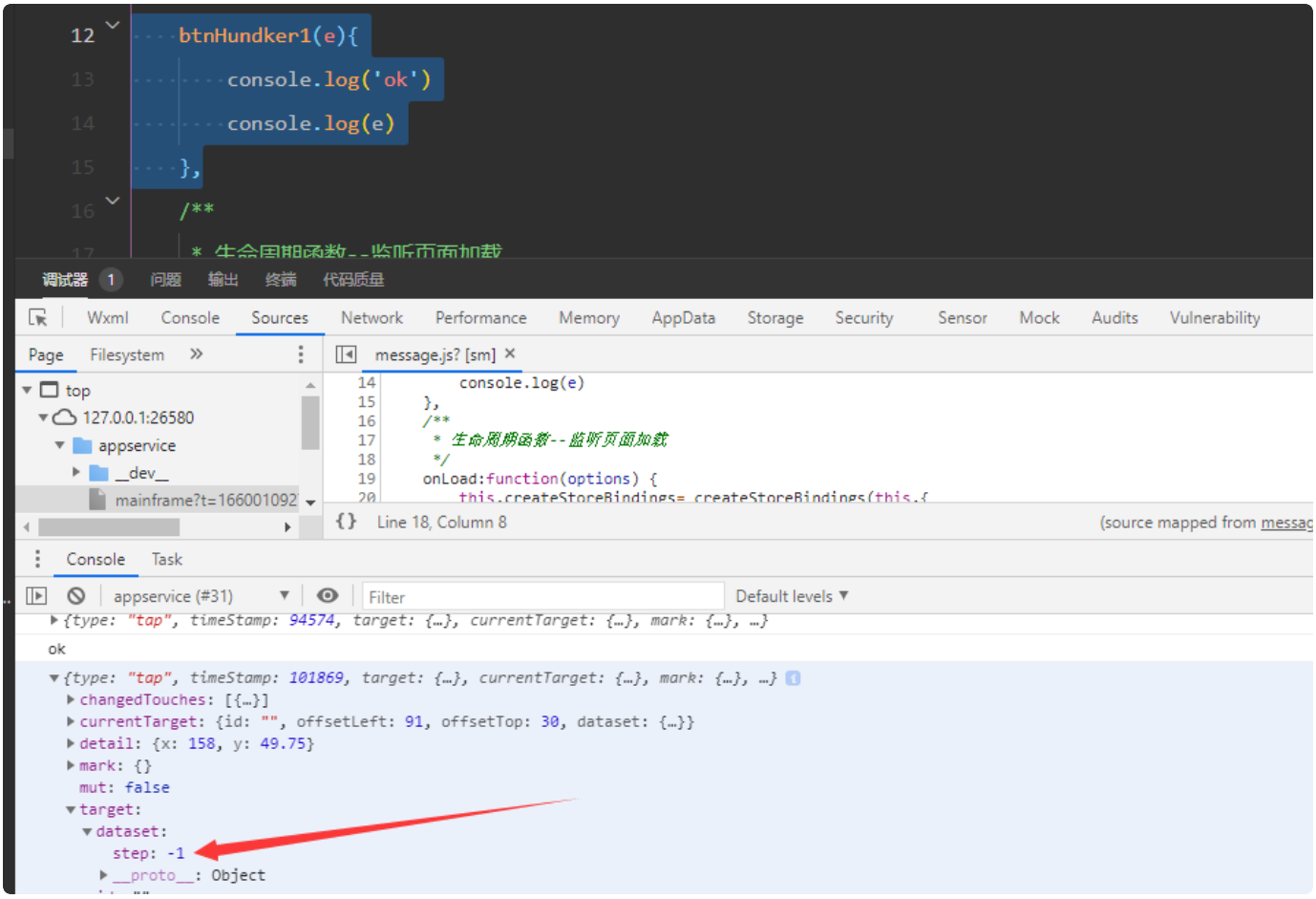
```
// 在该文件中创建Store实例对象
import {observable,action} from "mobx-miniprogram"

// 可挂载一些共享的数据
export const store =  observable({
    // 数据字段
    numA: 1,
    numB: 2,
    // 计算属性  get表示sum只是可读
    get sum(){
        return this.numA+this.numB
    },
    // actions方法用来修改store中的数据
    updateNum1:action(function(step){
        this.numA+=step
    }),
    updateNum2:action(function(step){
        this.numB+=step
    })
})
```

12.2.1 将store中的成员绑定到页面中

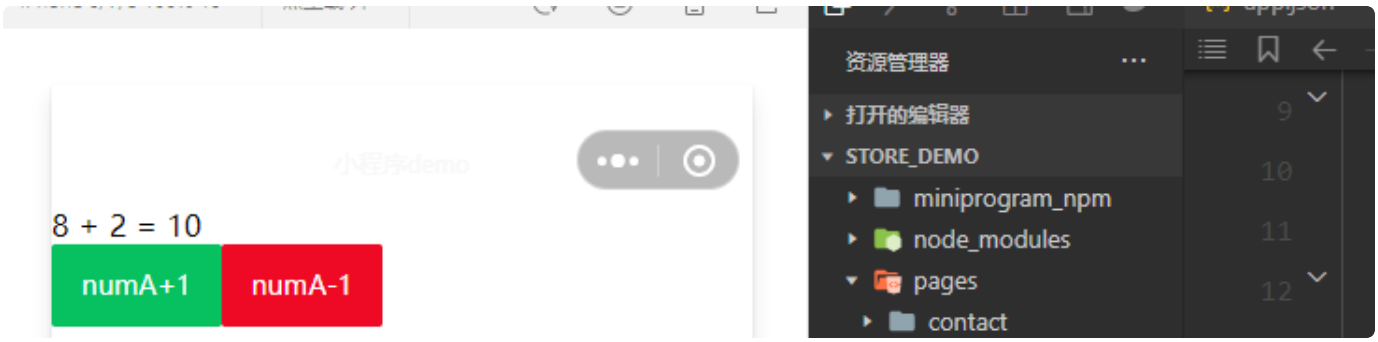
```javascript
import {createStoreBindings} from 'mobx-miniprogram-bindings'
import {} from '../../store/store.js'

    onLoad:function(options) {
        this.createStoreBindings= createStoreBindings(this,{
            store,
            fields:['numA','numB','sum'],
            actions:['updateNum1']
        })
    },

      onUnload:function() {
        this.storeBindings.destroyStoreBindings()
    },
```

12.2.2在页面上使用Store成员

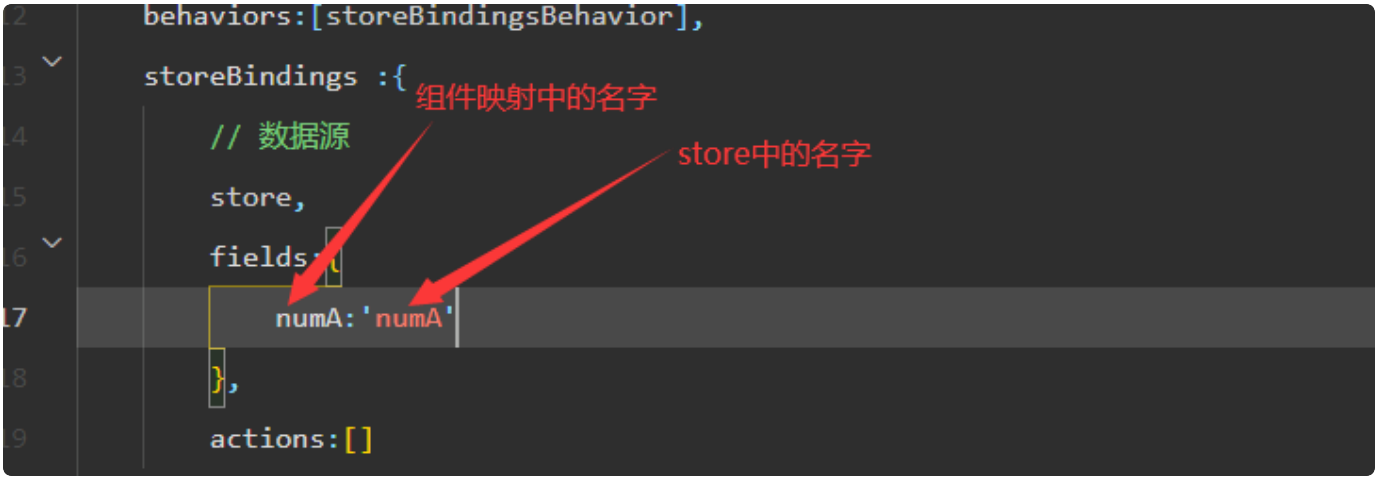
```javascript
<view>{{numA}} + {{numB}} = {{sum}}</view>
<van-button type="primary" bindtap="btnHundker1" data-step="
{{1}}">numA+1</van-button>
<van-button type="danger" bindtap="btnHundker1" data-step="{{-1}}">numA-
1</van-button>

    btnHundker1(e){
        console.log('ok')
        console.log(e)
    },
```




```javascript
btnHundker1(e){
    console.log('ok')
    console.log(e)
    this.updateNum1(e.target.dataset.step)
},
```



12.2.3 将Store成员绑定到组件中



JavaScript | 复制代码

```javascript
import {storeBindingsBehavior} from "mobx-miniprogram-bindings"
import {store} from "../../store/store"

behaviors:[storeBindingsBehavior],
storeBindings :{
        // 数据源
        store,
        // 映射属性
        fields:{
            numA:'numA', //第一个sumA是可以任意更改的
            numB:'numB',
            sum:'sum'
        },
        // 映射方法
    actions:{
    updateNum2:'updateNum2'
        }
    },
```

12.2.4 组件中使用Store成员

```javascript
    methods: {
        btnHundker2(e){
            console.log('1111')
            console.log(e)
            this.updateNum2(e.target.dataset.step)

        }

    }
```

13. 分包

分包指的是把一个完整的小程序项目按照需要划分为不同的子包，在构建的时候打包成不同的分包用户在使用的时候按照需要进行加载。

分包前小程序项目中所有页面和资源都被打包到一起导致整个项目体积过大，影响小程序首次启动的下载时间。

分包之后小程序项目会由一个主包+多个分包组成。

1. 主包：一般只包含项目的启动页面或者TabBar页面，以及所有的分包都需要用到一些公共资源。
2. 分包：只包含和当前分包有关的页面和私有资源。

13.1 分包的加载规则

1. 在小程序启动的时候默认会下载主包并包含主包内页面

- tabBar页面需要放到主包中

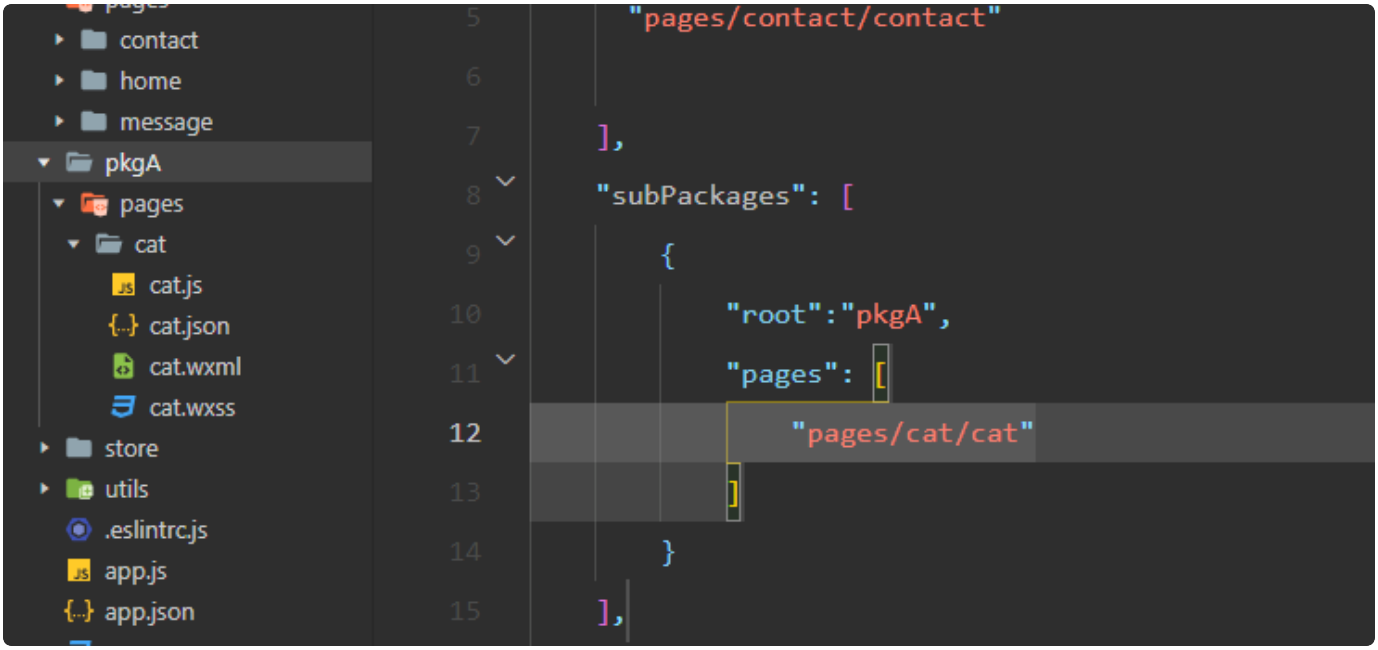
2. 当用户进入分包内某个页面的时候客户端会把对应的分包下载下来，下载完成再进行展示

- 非tabBar页面可以按照功能的不同划分为不同的分包之后进行按需下载。

分包体积的限制

1. 整个小程序所有分包大小不可以超过16M（主包+所有分包）
2. 单个分包/主包大小不能超过2M

13.2 使用分包



JavaScript | 复制代码

```
    "subPackages": [
        {
            "root":"pkgA",
            "pages": [
                "pages/cat/cat",
                "pages/dog/dog"
            ]
        },
        {
            "root": "pkgB",
            "pages": [
                "pages/apple/apple"
            ]
        }
    ],
```

13.2.1 查看分包的大小

13.2.2 打包原则

1. 小程序会按照subpackages的配置进行分包，subpackages之外的目录将被打包到主包中。
2. 主包也可以有自己的pages(即最外层的pages字段)
3. tabbar页面必须在主包内
4. 分包之间不可以相互嵌套

13.2.3 引用原则

1. 主包无法引用分包内的私有资源
2. 分包之间不能相互引用私有资源
3. 分包可以引用主包内的公共资源

13.3 独立分包

独立分包本质上也是分包，只不过可以独立于主包和其他分包而单独运行。

13.3.1 独立分包的应用场景

开发者按需将某些具有一定独立性页面配置到独立分包中，原因如下：

1. 当小程序从普通的分包页面启动的时候需要首先下载主包
2. 而独立分包不依赖于主包就可以运行，可以很大程度上提升分包页面启动的速度

> 一个小程序可以有多个独立分包

13.3.2 独立分包的引用原则

独立分包和普通分包以及主包之间是相互隔绝的，不可以相互引用彼此的资源。

1. 主包无法引用独立分包内的私有资源
2. 独立分包之间不可以相互引用私有资源
3. 独立分包和普通分包之间不可以相互引用私有资源
4. 独立分包不可以引用主包内的公共资源

13.4 分包的预下载

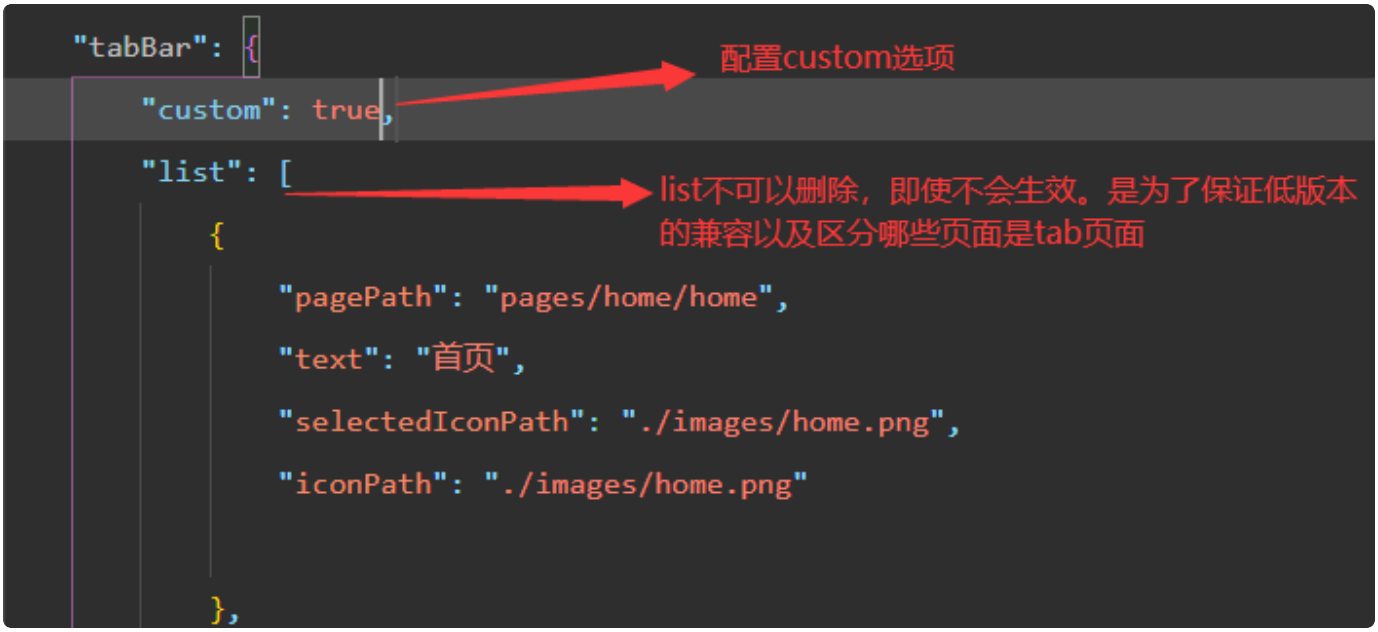
预下载的行为会在进入指定页面的时候触发，在app.json中使用preloadRule节点定义分包的预下载规则

```javascript
    "preloadRule": {
        "pages/contact/contact":{
            "packages": ["pkgA"],
            "network": "all"
        }
    },
```

分包预下载的限制

同一个分包内的页面享有共同的预下载大小的限制是2M。

13.5 自定义tabBar



14. uniapp

uni-app 是一个使用 Vue.js 开发所有前端应用的框架。开发者编写一套代码，可发布到 iOS、Android、H5、以及各种小程序（微信/支付宝/百度/头条/QQ/钉钉/淘宝）、快应用等多个平台。

目录结构

```
├─components            uni-app组件目录
│  └─comp-a.vue         可复用的a组件
├─pages                 业务页面文件存放的目录
│  ├─index
│  │  └─index.vue       index页面
│  └─list
│     └─list.vue        list页面
├─static                存放应用引用静态资源（如图片、视频等）的目录，注意：静态资源只能存放于此
├─main.js               Vue初始化入口文件
├─App.vue               应用配置，用来配置小程序的全局样式、生命周期函数等
├─manifest.json         配置应用名称、appid、logo、版本等打包信息
└─pages.json            配置页面路径、页面窗口样式、tabBar、navigationBar 等页面类信息
```