
黑马程序员™
www.itheima.com
传智播客旗下
高端IT教育品牌
小程序的框架与逻辑

目录 Contents

1. 页面导航

1.1 页面导航的两种方式

页面导航就是页面之间的跳转，小程序中页面之间的导航有**两种方式**：

① **声明式导航**：通过点击 navigator 组件实现页面跳转的方式；

② **编程式导航**：通过调用小程序的 API 接口实现跳转的方式；

1.2 声明式导航

1. 导航到非 tabBar 页面

非 tabBar 页面指的是没有被当作 tabBar 进行切换的页面。

示例代码如下：

```
<navigator url='/pages/info/info'>去info页面</navigator>
```

上述代码使用 url 属性指定要跳转到的页面路径；其中，页面路径应该以 / 开头，且路径必须提前在 app.json 的 pages 节点下声明，才能实现正常的跳转。

1.2 声明式导航

2. 导航到 tabBar 页面

tabBar 页面指的是被当作 tabBar 进行切换的页面。如果 navigator 组件单纯使用 url 属性，无法导航到 tabBar 页面，需要结合 **open-type 属性**进行导航。

示例代码如下：

```
<navigator url='/pages/home/home' open-type='switchTab'>
    导航到home页面
</navigator>
```

关于 navigator 组件的更多用法，请翻阅官方文档：

https://developers.weixin.qq.com/miniprogram/dev/component/navigator.html

1. 页面导航

1.2 声明式导航

3. 后退导航

如果要后退到上一页面或多级页面，需要把 open-type 设置为 navigateBack，同时使用 delta 属性指定后退的层数，示例代码如下：

```
<navigator open-type='navigateBack' delta='1'>
    返回上一页
</navigator>
```

关于 navigator 组件的更多用法，请翻阅官方文档：

https://developers.weixin.qq.com/miniprogram/dev/component/navigator.html

1. 页面导航

1.3 编程式导航

1. 导航到非 tabBar 页面

通过 wx.navigateTo(Object object) 方法，可以跳转到应用内的某个页面。但是不能跳到 tabbar 页面。其中 Object 参数对象的属性列表如下：

属性	类型	是否必填	说明
url	string	是	需要跳转的应用内非 tabBar 的页面的路径, 路径后可以带参数。参数与路径之间使用 ?分隔，参数键与参数值用 = 相连，不同参数用 & 分隔；如 'path?key=value&key2=value2'
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）

1.3 编程式导航

2. 导航到 tabBar 页面

通过 wx.switchTab(Object object) 方法，可以跳转到 tabBar 页面，并关闭其他所有非 tabBar 页面。其中 Object 参数对象的属性列表如下：

属性	类型	是否必填	说明
url	string	是	需要跳转的 tabBar 页面的路径（需在 app.json 的 tabBar 字段定义的页面），路径后不能带参数。
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）

1.3 编程式导航

3. 后退导航

通过 wx.navigateBack(Object object) 方法，关闭当前页面，返回上一页面或多级页面。其中 Object 参数对象的属性列表如下：

属性	类型	是否必填	说明
delta	number	是	返回的页面数，如果 delta 大于现有页面数，则返回到首页。
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）

1.4 导航传参

1. 声明式导航传参

navigator 组件的 url 属性用来指定导航到的页面路径，同时路径后面还可以携带参数，参数与路径之间使用？分隔，参数键与参数值用 = 相连，不同参数用 & 分隔。

代码示例如下：

```
<navigator url='/pages/logs/logs?name=zs&age=20'>去logs页面</navigator>
```

1.4 导航传参

2. 编程式导航传参

wx.navigateTo(Object object) 方法的 object 参数中，url 属性用来指定需要跳转的应用内非 tabBar 的页面的路径, 路径后可以带参数。参数与路径之间使用？分隔，参数键与参数值用 = 相连，不同参数用 & 分隔。

代码示例如下：

```
wx.navigateTo({
  url: '/pages/logs/logs?name=zs&age=20',
})
```

1.4 导航传参

3. 页面接收导航传递过来的参数

不论是声明式导航还是编程式导航，最终导航到的页面可以在 onLoad 生命周期函数中接收传递过来的参数。

代码示例如下：

```
/**
* 生命周期函数--监听页面加载
*/
onLoad: function(options) {
   console.log(options) // options 就是导航传递过来的参数
}
```

1.4 导航传参

4. 自定义编译模式快速传参

小程序每次修改代码并编译后，会默认从首页进入，但是在开发阶段，我们经常会针对特定的页面进行开发，为了方便编译后直接进入对应的页面，可以配置自定义编译模式，步骤如下：

① 单击工具栏上的“普通编译”下拉菜单；

② 单击下拉菜单中的“添加编译模式”选项；

③ 在弹出的“自定义编译条件窗口”，按需添加模式名称、启用页面、启动参数、进入场景等。

目录 Contents

2. 网络数据请求

2.1 配置服务器域名

每个微信小程序需要事先设置一个**通讯域名**，**小程序只可以跟指定的域名进行网络通信**。

服务器域名请在 「**小程序后台-开发-开发设置-服务器域名**」 中进行配置，配置时需要注意：

- 域名只支持 https (request、uploadFile、downloadFile) 和 wss (connectSocket) 协议
- 域名不能使用 IP 地址或 localhost
- 域名必须经过 ICP 备案
- 服务器域名一个月内可申请5次修改

2.2 跳过域名校验

在微信开发者工具中，可以临时开启 「开发环境不校验请求域名、TLS 版本及 HTTPS 证书」 选项，跳过服务器域名的校验。此时，在微信开发者工具中及手机开启调试模式时，不会进行服务器域名的校验。

注意：在服务器域名配置成功后，建议开发者**关闭此选项**进行开发，并在各平台下进行测试，以确认服务器域名配置正确。

2.3 发起网络数据请求

1. 发起 get 请求

调用 wx.request(Object object) 方法发起 get 请求，代码示例如下：

```
wx.request({// 请求的URL地址，必须基于 HTTPS 协议
    url: 'https://www.liulongbin.top:8082/api/get',
    // 发送到服务器的数据
    data: { name: 'zs', age: 20 },
    // 成功之后的回调函数
    success: function(result) {
        console.log(result)
    }
})
```

2.3 发起网络数据请求

2. 发起 post 请求

调用 wx.request(Object object) 方法发起 post 请求，代码示例如下：

```
wx.request({
    url: 'https://www.liulongbin.top:8082/api/post',
    method: 'POST', // 设置请求类型，如果不设置，默认发起 GET 请求
    data: { name: 'ls', gender: '男' }, // 发送到服务器的数据
    success: function(result) {
        console.log(result)
    }
})
```

2.3 发起网络数据请求

3. 小程序中没有跨域限制

在普通网站开发中，由于**浏览器的同源策略**限制，存在数据的跨域请求问题，从而衍生出了 **JSONP** 和 **CORS** 两种主流的跨域问题解决方案。但是，小程序的内部运行机制与网页不同，小程序中的代码并不运行在浏览器中，**因此小程序开发中，不存在数据的跨域请求限制问题**。

关于微信小程序更多的数据请求内容，请翻阅 wx.request() 的相关文档：

https://developers.weixin.qq.com/miniprogram/dev/api/network/request/wx.request.html

目录 Contents

3. 自定义组件

3.1 组件的创建与引用

1. 创建组件

① 在项目的根目录中，鼠标右键，创建 components -> test 文件夹

② 在新建的 components -> test 文件夹上，鼠标右键，点击“新建 Component”

③ 为新建的组件命名之后，会自动生成组件对应的 4 个文件，后缀名分别为 .js，.json， .wxml 和 .wxss

注意：尽量将不同的组件，存放到单独的文件夹中，从而保证清晰的目录结构

3. 自定义组件

3.1 组件的创建与引用

2. 引用组件

① 在需要引用组件的页面中，找到页面的 .json 配置文件，新增 usingComponents 节点

② 在 usingComponents 中，通过键值对的形式，注册组件；键为注册的组件名称，值为组件的相对引用路径

③ 在页面的 .wxml 文件中，把注册的组件名称，以标签形式在页面上使用，即可把组件展示到页面上

注意：注册组件名称时，建议把名称注册为短横线连接的形式，例如 vant-button 或 custom-button

3. 自定义组件

3.1 组件的创建与引用

3. 使用样式美化组件

组件对应 wxss 文件的样式，只对组件 wxml 内的节点生效。编写组件样式时，需要注意以下几点：

① 组件和引用组件的页面不能使用id选择器（#a）、属性选择器（[a]）和标签名选择器，请改用class选择器。

② 组件和引用组件的页面中使用后代选择器（.a .b）在一些极端情况下会有非预期的表现，如遇，请避免使用。

③ 子元素选择器（.a>.b）只能用于 view 组件与其子节点之间，用于其他组件可能导致非预期的情况。

④ 继承样式，如 font 、 color ，会从组件外继承到组件内。

⑤ 除继承样式外， app.wxss 中的样式、组件所在页面的样式对自定义组件无效。

3.2 组件的 data 与 methods

1. 使用 data 定义组件的私有数据

小程序组件中的 data，和小程序页面中的 data 用法一致，只不过：

- 页面的 data 定义在 Page() 函数中
- 组件的 data 定义在 Component() 函数中

在组件的 .js 文件中：

- 如果要访问 data 中的数据，直接使用 this.data.数据名称 即可
- 如果要为data 中的数据重新赋值，调用 this.setData({ 数据名称: 新值 }) 即可

在组件的 .wxml 文件中：

- 如果要渲染 data 中的数据，直接使用 {{ 数据名称 }} 即可

3.2 组件的 data 与 methods

2. 使用 methods 定义组件的事件处理函数

和页面不同，组件的事件处理函数，必须定义在 methods 节点中。示例代码如下：

```
Component({
    methods: {
        // 按钮的点击事件处理函数
        btnHandler: function() {}
    }
})
```

3.3 组件的 properties

1. properties 的作用

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

在小程序中，组件的 properties 和 data 的用法类似，它们都是可读可写的，只不过：

- data 更倾向于存储组件的私有数据
- properties 更倾向于存储外界传递到组件中的数据

3. 自定义组件

3.3 组件的 properties

2. 声明 properties 的两种方式

```
Component({
    properties: {
        // 完整的定义方式
        propA: {            // 属性名
            type: String, // 属性类型
            value: ''       // 属性默认值
        },
        propB: String       // 简化的定义方式
    }
})
```

注意：type 的可选值为 Number，String、Boolean、Object、Array、null(表示不限制类型)

3.3 组件的 properties

3. 为组件传递 properties 的值

可以使用数据绑定的形式，向子组件的属性传递动态数据，示例代码如下：

```
<!-- 引用组件的页面模板 -->
<view>
  <component-tag-name prop-a="{{dataFieldA}}" prop-b="{{dataFieldB}}">
  </component-tag-name>
</view>
```

在以上例子中，组件的属性 propA 和 propB 将收到页面传递的数据。页面可以通过 setData 来改变绑定的数据字段。

注意：在定义 properties 时，属性名采用驼峰写法（propertyName）；在 wxml 中，指定属性值时，则对应使用连字符写法（property-name=“attr value”），应用于数据绑定时，采用驼峰写法（attr="{{propertyName}}"）。

3.3 组件的 properties

4. 在组件内修改 properties 的值

小程序中，properties 的值是可读可写的，它的用法与 data 几乎一致，因此可以通过 setData 修改 properties 中任何属性的值，示例代码如下：

```
properties: {
  count: Number
},
methods: {
  add: function() {
    this.setData({ count: this.properties.count + 1 })
  }
}
```

3.4 数据监听器

1. 什么是数据监听器

数据监听器可以用于监听和响应任何属性和数据字段的变化，从而执行特定的操作。作用类似于 vue 中的 watch。

数据监听器从小程序基础库版本 2.6.1 开始支持。

数据监听器的基本语法格式如下：

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

3. 自定义组件

3.4 数据监听器

2. 监听子数据字段的变化

```
Component({
  observers: {
    'some.subfield': function (subfield) {
      // 使用 setData 设置 this.data.some.subfield 时触发
      // （除此以外，使用 setData 设置 this.data.some 也会触发）
    },
    'arr[12]': function (arr12) {
      // 使用 setData 设置 this.data.arr[12] 时触发
      // （除此以外，使用 setData 设置 this.data.arr 也会触发）
    }
  }
})
```

3.4 数据监听器

3. 使用通配符 ** 监听所有子数据字段的变化

```
Component({
  observers: {
    'some.field.**': function (field) {
      // 使用 setData 设置 this.data.some.field 本身或其下任何子数据字段时触发
      // （除此以外，使用 setData 设置 this.data.some 也会触发）
      field === this.data.some.field
    }
  }
})
```

3.5 组件的生命周期

1. 组件的主要生命周期

组件的生命周期，指的是组件自身的一些函数，这些函数在特殊的时间点或遇到一些特殊的框架事件时被自动触发。

其中，最重要的生命周期是 created, attached, detached ，包含一个组件实例生命流程的最主要时间点。

- 组件实例刚刚被创建好时， created 生命周期被触发。此时还不能调用 setData 。 通常情况下，这个生命周期只应该用于给组件 this 添加一些自定义属性字段。
- 在组件完全初始化完毕、进入页面节点树后， attached 生命周期被触发。此时， this.data 已被初始化完毕。这个生命周期很有用，绝大多数初始化工作可以在这个时机进行。
- 在组件离开页面节点树后， detached 生命周期被触发。退出一个页面时，如果组件还在页面节点树中，则 detached 会被触发。

3.5 组件的生命周期

2. 组件可用的全部生命周期函数

生命周期	参数	描述	最低版本
created	无	在组件实例刚刚被创建时执行	1.6.3
attached	无	在组件实例进入页面节点树时执行	1.6.3
ready	无	在组件在视图层布局完成后执行	1.6.3
moved	无	在组件实例被移动到节点树另一个位置时执行	1.6.3
detached	无	在组件实例被从页面节点树移除时执行	1.6.3
error	Object Error	每当组件方法抛出错误时执行	2.4.1

3.5 组件的生命周期

3. 定义生命周期函数

生命周期方法可以直接定义在 Component 构造器的第一级参数中。

自小程序基础库版本 2.2.3 起，组件的的生命周期也可以在 lifetimes 字段内进行声明（这是推荐的方式，其优先级最高）。

3.5 组件的生命周期

3. 定义生命周期函数

代码示例如下：

```
Component({
  lifetimes: {
    attached() {}, // 在组件实例进入页面节点树时执行
    detached() {}, // 在组件实例被从页面节点树移除时执行
  },
  // 以下是旧式的定义方式，可以保持对 <2.2.3 版本基础库的兼容
  attached() {}, // 在组件实例进入页面节点树时执行
  detached() {}, // 在组件实例被从页面节点树移除时执行
  // ...
})
```

3.5 组件的生命周期

4. 组件所在页面的生命周期

有一些特殊的生命周期，它们并非与组件有很强的关联，但有时组件需要获知，以便组件内部处理。这样的生命周期称为“组件所在页面的生命周期”，在 pageLifetimes 定义段中定义。其中可用的生命周期包括：

生命周期	参数	描述	最低版本
show	无	组件所在的页面被展示时执行	2.2.3
hide	无	组件所在的页面被隐藏时执行	2.2.3
resize	Object Size	组件所在的页面尺寸变化时执行	2.4.0

3.5 组件的生命周期

4. 组件所在页面的生命周期

代码示例如下：

```
Component({
  pageLifetimes: {
    show() { // 页面被展示
    },
    hide() { // 页面被隐藏
    },
    resize(size) { // 页面尺寸变化
    }
  }
})
```

3.6 组件插槽

1. 默认插槽

在组件的 wxml 中可以包含 slot 节点，用于承载组件使用者提供的 wxml 结构。

默认情况下，一个组件的 wxml 中只能有一个slot。需要使用多 slot 时，可以在组件 js 中声明启用。

注意：小程序中目前只有默认插槽和多个插槽，暂不支持作用域插槽。

3. 自定义组件

3.6 组件插槽

1. 默认插槽

```
<!-- 组件模板 -->
<view class="wrapper">
  <view>这里是组件的内部节点</view>
  <slot></slot>
</view>
```

```
<!-- 引用组件的页面模板 -->
<view>
  <component-tag-name>
    <!-- 这部分内容将被放置在组件 <slot> 的位置上 -->
    <view>这里是插入到组件slot中的内容</view>
  </component-tag-name>
</view>
```

3.6 组件插槽

2. 启用多个插槽

在组件中，需要使用多 slot 时，可以在组件 js 中声明启用。示例代码如下：

```
Component({
  options: {
    multipleSlots: true // 在组件定义时的选项中启用多slot支持
  },
  properties: { /* ... */ },
  methods: { /* ... */ }
})
```

3.6 组件插槽

3. 定义多个插槽

可以在组件的 wxml 中使用多个 slot 标签，以不同的 name 来区分不同的插槽。示例代码如下：

```
<!-- 组件模板 -->
<view class="wrapper">
  <slot name="before"></slot>
  <view>这里是组件的内部细节</view>
  <slot name="after"></slot>
</view>
```

3.6 组件插槽

4. 使用多个插槽

使用时，用 slot 属性来将节点插入到不同的 slot 中。示例代码如下：

```
<!-- 引用组件的页面模板 -->
<view>
  <component-tag-name>
    <!-- 这部分内容将被放置在组件 <slot name="before"> 的位置上 -->
    <view slot="before">这里是插入到组件slot name="before"中的内容</view>
    <!-- 这部分内容将被放置在组件 <slot name="after"> 的位置上 -->
    <view slot="after">这里是插入到组件slot name="after"中的内容</view>
  </component-tag-name>
</view>
```

3.7 组件间的通信

1. 组件之间的三种基本通信方式

① WXML 数据绑定：用于父组件向子组件的指定属性传递数据，仅能设置 JSON 兼容数据（自基础库版本 2.0.9 开始，还可以在数据中包含函数）。

② 事件：用于子组件向父组件传递数据，可以传递任意数据。

③ 如果以上两种方式不足以满足需要，父组件还可以通过 this.selectComponent 方法获取子组件实例对象，这样就可以直接访问组件的任意数据和方法。

3.7 组件间的通信

2. this.selectComponent(string)

在小程序的组件中，调用 this.selectComponent(string)，可以返回指定组件的实例对象。示例代码如下：

```
<!-- wxml -->
<component-a class="customA" id= "cA" ></component-a>

<!--父组件的 .js 文件中，可以调用 selectComponent 函数并指定 id 或 class 选择器，获取子组件对象-->
Page({
  onLoad(){
    // 切记下面参数不能传递标签选择器（component-a），不然返回的是 null
    var component = this.selectComponent('.customA'); // 也可以传递 id 选择器 #cA
    console.log(component);
  }
})
```

3.7 组件间的通信

3. 通过事件监听实现子向父传值

事件系统是组件间通信的主要方式之一。自定义组件可以触发任意的事件，引用组件的页面可以监听这些事件。

通过事件监听实现子组件向父组件传值的步骤：

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，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 */ }) ，将数据发送到父组件

④ 在父组件的 js 中，通过 e.detail 获取到子组件传递过来的数据

目录 Contents

4. WePY 框架的安装和使用

4.1 WePY 框架概述

1. 什么是 WePY

WePY 是腾讯官方出品的一个小程序快速开发框架，对原生小程序的开发模式进行了再次封装，更贴近于 MVVM 架构模式，并支持ES6/7的一些新特性，同时语法风格更接近于 Vue.js，使用 WePY 框架能够提高小程序的开发效率。

注意：WePY 只是小程序的快速开发框架之一，市面上还有诸如 mpvue 之类的小程序开发框架也比较流行。

4. WePY 框架的安装和使用

4.1 WePY 框架概述

2. 为什么要使用 WePY

WePY 相比于原生小程序开发，拥有众多的**开发特性**和**优化方案**，例如：

- 开发风格接近于 Vue.js，支持很多vue中的语法特性；
- 通过 polyfill 让小程序完美支持 Promise；
- 可以使用ES6等诸多高级语法特性，简化代码，提高开发效率；
- 对小程序本身的性能做出了进一步的优化；
- 支持第三方的 npm 资源；
- 支持多种插件处理和编译器；
- etc...

4. WePY 框架的安装和使用

4.1 WePY 框架概述

3. 安装 WePY 框架

WePY 的安装或更新都通过 npm 进行，全局安装或更新 WePY 命令行工具，可以在终端运行以下命令：

```
npm install wepy-cli -g
```

4.2 创建 WePY 项目

1. 初始化 WePY 项目

WePY 命令行工具安装完毕后，可以运行如下命令，初始化项目结构：

```
wepy init standard myproject
```

其中，"wepy init" 是固定写法，代表要初始化 wepy 项目；"standard" 代表模板类型为标准模板，可以运行 "wepy list" 命令查看所有可用的项目模板； "myproject" 为自定义的项目名称。

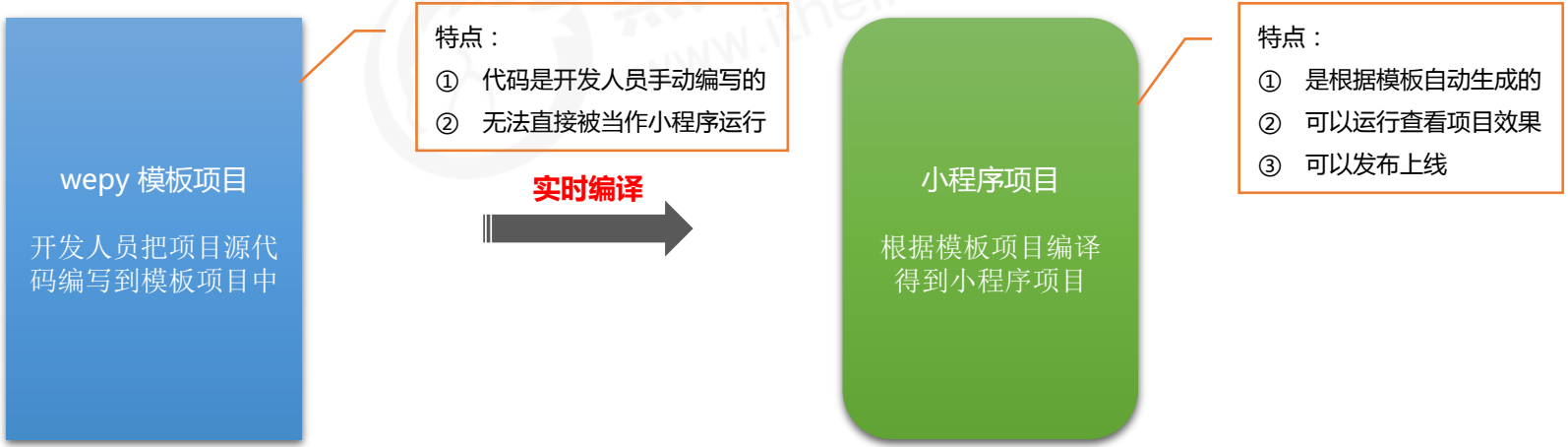
注意：创建项目的时候，要勾选 ESLint 选项！

4. WePY 框架的安装和使用

4.2 创建 WePY 项目

2. WePY 项目与小程序项目的关系

通过 wepy init 命令初始化的 wepy 项目，准确来说只是一个模板项目，不能直接当作小程序运行。需要运行相关的命令，**把模板项目编译为小程序项目**，才可以运行。



4.2 创建 WePY 项目

3. 实时编译 WePY 项目

使用 wepy init 命令初始化项目后，只是得到了一个模板项目，如果想开启实时编译，得到小程序项目，步骤如下：

① 运行 cd myproject 切换至 WePY 项目根目录

② 运行 npm install 安装 WePY 项目依赖项

③ 运行 wepy build --watch 开启实时编译

注意：wepy build --watch 命令，会循环监听 WePY 项目中源代码的变化，自动编译生成小程序项目，生成的小程序项目默认被存放于 dist 目录中。

4. WePY 框架的安装和使用

4.3 WePY 项目的目录结构

├── **dist** 小程序运行代码目录（**该目录由WePY的build指令自动编译生成，请不要直接修改该目录下的文件**）

├── node_modules

├── **src** 代码编写的目录 （**该目录为使用WePY后的开发目录**）

│ ├── components WePY组件目录（组件不属于完整页面，仅供完整页面或其他组件引用）

│ │ ├── com_a.wpy 可复用的WePY组件a

│ │ ├── com_b.wpy 可复用的WePY组件b

│ ├── pages WePY页面目录（属于完整页面）

│ │ ├── index.wpy index页面（经build后，会在dist目录下的pages目录生成index.js、index.json、index.wxml和index.wxss文件）

│ │ ├── other.wpy other页面（经build后，会在dist目录下的pages目录生成other.js、other.json、other.wxml和other.wxss文件）

│ └── app.wpy 小程序配置项（全局数据、样式、声明钩子等；经build后，会在dist目录下生成app.js、app.json和app.wxss文件）

└── package.json 项目的package配置

4.4 加载并配置 WePY 项目

1. 加载 WePY 项目到微信开发者工具

1.7.0 版本之后的 wepy-cli 工具生成的项目根目录下，包含 **project.config.json** 文件，记录了项目的基本配置信息，例如：项目的名称、appId、生成的小程序项目根路径等。

如果项目中存在 **project.config.json** 文件，使用 微信开发者工具 --> 导入项目，”项目目录”请选择 wepy 项目根目录，即可根据 project.config.json 文件中的配置，把 wepy 编译生成的小程序项目加载到微信开发者工具中。

4.4 加载并配置 WePY 项目

2. 解决 ESLint 报错的问题

ESLint 是好用的代码格式检查工具，能够帮助程序员养成良好的代码书写习惯。首次把 WePY 项目加载到微信开发者工具后，终端会报两个红色的错误信息，截图如下：

```
▶CLI报错:
C:\Users\liulongbin\Desktop\react-code\wepy_01\src\app.wpy
  15:1  error  More than 1 blank line not allowed  no-multiple-empty-lines

✖ 1 problem (1 error, 0 warnings)
▶CLI报错:
C:\Users\liulongbin\Desktop\react-code\wepy_01\src\pages\index.wpy
  91:1  error  More than 1 blank line not allowed  no-multiple-empty-lines

✖ 1 problem (1 error, 0 warnings)
```

这是因为 ESLint 规定，不允许在代码中出现连续多个空行，此时根据报错信息，找到对应的文件，删除多余的空行，重新编译 WePY 项目即可。

4.5 .wpy 文件的使用说明

1. .wpy 文件的组成部分

一个 .wpy 文件可分为三大部分，各自对应于一个标签：

① **脚本部分**，即 <script></script> 标签中的内容，又可分为两个部分：

- **逻辑部分**，除了 config 对象之外的部分，对应于原生的 .js 文件
- **配置部分**，即 config 对象，对应于原生的 .json 文件

② **结构部分**，即 <template></template> 模板部分，对应于原生的 .wxml 文件。

③ **样式部分**，即<style></style>样式部分，对应于原生的 .wxss 文件。

其中，**小程序入口文件 app.wpy 不需要 template**，所以编译时会被忽略。

4.5 .wpy 文件的使用说明

2. lang 和 src 属性

.wpy 文件中的 script、template、style 这三个标签都支持 lang 和 src 属性，lang 决定了其代码编译过程，src 决定是否外联代码，存在 src 属性且有效时，会忽略内联代码。

各标签对应的 lang 值如下表所示：

标签	lang 默认值	lang 支持值
style	css	css、less、scss、stylus、postcss
template	wxml	wxml、xml、pug(原jade)
script	babel	babel、TypeScript

4.5 .wpy 文件的使用说明

3. 代码高亮

文件后缀为.wpy，可共用 Vue 的高亮规则，但需要手动设置。如下是 VS Code 中实现代码高亮的相关设置步骤：

① 在 Code 里先安装 Vue 的语法高亮插件 **Vetur**。

② 打开任意 .wpy 文件。

③ 点击右下角的**选择语言模式**，默认为纯文本。

④ 在弹出的窗口中选择 **.wpy 的配置文件关联...**。

⑤ 在选择要与 .wpy 关联的语言模式 中**选择 Vue**。

⑥ 在 VS Code 编辑器设置中设置。// 文件-首选项-设置-settings.json 中，添加 "files.associations": { "*.wpy": "vue" }

注意：其它常见 IDE 或编辑器中设置代码高亮，可以参考：https://tencent.github.io/wepy/document.html#/?id=代码高亮

4.5 .wpy 文件的使用说明

4. 小程序入口 app.wpy

入口文件 app.wpy 中所声明的小程序实例继承自 **wepy.app** 类，包含一个 config 属性和其它全局属性、方法、事件。

在 build 编译期间：

- **config 属性**会被编译为小程序的 **app.json 全局配置文件**；
- **config 属性**之外的其它节点，会被编译为小程序的 **app.js 文件**；
- **style 标签**会被编译为小程序的 **app.wxss 全局样式**；

4.5 .wpy 文件的使用说明

5. 使用 app.wpy 全局配置小程序外观

在小程序的入口文件中找到 window 节点：app.wpy -> script标签 -> config -> window 即可全局配置小程序的外观，示例代码如下：

```
<script>
import wepy from 'wepy';
export default class extends wepy.app {
    config = { // 对应 app.json 文件
        "window": { // 全局配置小程序外观
            "backgroundTextStyle": "light",
            "navigationBarBackgroundColor": "#fff",
            "navigationBarTitleText": "WeChat",
            "navigationBarTextStyle": "black"
        }
    }
}
</script>
```

4.5 .wpy 文件的使用说明

6. 页面 .wpy 文件中 script 标签组成结构

页面文件 page.wpy 中所声明的页面实例继承自 **wepy.page** 类，该类的主要属性介绍如下：

属性	说明
config	页面配置对象，对应于原生的page.json文件，类似于app.wpy中的config
components	页面组件列表对象，声明页面所引入的组件列表
data	页面渲染数据对象，存放可用于页面模板绑定的渲染数据
methods	wxml事件处理函数对象，存放响应wxml中所捕获到的事件的函数，如bindtap、bindchange
events	WePY组件事件处理函数对象，存放响应组件之间通过$broadcast、$emit、$invoke所传递的事件的函数
其它	小程序页面生命周期函数，如onLoad、onReady等，以及其它自定义的方法与属性

目录 Contents

5. WePY 框架的开发规范

5.1 自定义默认首页

1. 创建 home 首页

在 src -> pages 目录下，创建 home.wpy 页面，并创建页面的基本代码结构，示例代码如下：

```
<template></template>

<script>
import wepy from 'wepy'
export default class Home extends wepy.page {
    config = {}
    methods = {}
}
</script>

<style></style>
```

5. WePY 框架的开发规范

5.1 自定义默认首页

2. 设置默认首页

如果要把刚创建的 home.wpy 设置为默认首页，需要打开 src -> app.wpy 入口文件，将 home.wpy 的页面路径，注册到 config -> pages 数组中，并调整为数组的第一项即可，示例代码如下：

```
<script>
import wepy from 'wepy'
import 'wepy-async-function'

export default class extends wepy.app {
  config = {
    pages: ['pages/home', 'pages/index']
  }
}
</script>
```

5. WePY 框架的开发规范

5.1 自定义默认首页

3. 创建 .wpy 页面的注意事项

在创建 .wpy 页面的时候，要注意以下事项：

① 每个页面的 script 标签中，必须导入 wepy 模块，并创建继承自 wepy.page 的页面类；否则会报错。

② 每个页面的路径，必须记录到 app.wpy 的 config -> pages 数组中。

③ 页面路径记录完毕之后，必须再回到页面文件中，摁下 Ctrl + S 快捷键重新编译生成页面，否则会报错。

5.2 为 WePY 页面绑定事件并传参

1. 使用 @ 绑定事件处理函数

原生小程序使用 bindtap、bindinput 等绑定事件处理函数，在 wepy 框架中，优化了事件绑定机制，支持类似于 Vue.js 的事件绑定语法，今后绑定事件可以采用如下方式：

- 原 bindtap="clickHandler" 替换为 @tap="clickHandler"
- 原 bindinput="inputHandler" 替换为 @input="inputHandler"

5.2 为 WePY 页面绑定事件并传参

2. 事件传参的优化写法

如果 @ 符号绑定的事件处理函数需要传参，可以采用优化的写法，示例如下：

- 原 bindtap="clickHandler" data-index={{index}} 替换为 @tap="click({{index}})"

5.2 为 WePY 页面绑定事件并传参

3. 定义事件处理函数的注意点

通过 @ 符号绑定的事件处理函数，必须定义到页面的 methods 节点下。

注意：对于 WePY 中的 methods 属性，因为与 Vue 中的使用习惯不一致，非常容易造成误解，这里需要特别强调一下：WePY 中的 methods 属性**只能声明**页面 wxml 标签的**事件处理函数，不能声明自定义方法**，自定义方法需要声明到和 methods 平级的节点位置，这与 Vue 中的用法是不一致的。

5.3 WePY 页面的数据绑定

1. 使用 data 定义私有数据

.wpy 页面中的私有数据，需要定义到 data 节点中，页面上可以使用双大括号语法 {{ }} 渲染 data 中的数据

示例代码如下：

```
<template>
    <view>{{msg}}</view>
</template>

data = {
    msg: 'hello wepy'
}
```

5.3 WePY 页面的数据绑定

2. 将文本框与 data 做双向数据绑定

实现步骤如下：

① 为 input 输入框绑定数据和事件处理函数，代码为：<input value="{{msg}}" @input="inputHandler" />

② 在 methods 中定义事件处理函数，函数名称为 inputHandler

③ 在事件处理函数中，通过事件参数 e.detail.value 获取到最新的文本框内容

④ 通过 this.msg = e.detail.value 为 data 中的 msg 重新赋值

5.3 WePY 页面的数据绑定

3. 使用 wxs 脚本

在 WePY 中使用 wxs 脚本的方式如下：

① 将 wxs 脚本定义为外联文件，并且后缀名为 .wxs

② 在 <script></script> 标签内，通过 import 导入相对路径的 wxs 模块

③ 在当前页面的 class 类中，通过 wxs = { } 注册刚才导入的 wxs 模块

注意：被注册的 wxs 模块，只能在当前页面的 template 中使用，不能在script中使用

5.4 WePY 页面中发起数据请求

1. 配置 promisify 启用 async 和 await

默认使用 wepy-cli 创建的项目，不支持使用 ES7 的 async 和 await 来简化 Promise API 的调用。需要手动开启此功能：打开 src -> app.wpy，找到 constructor() 构造函数，在构造函数中代码的最后一行，添加 this.use('promisify') 即可，示例代码如下：

```
constructor() {
  super()
  this.use('requestfix')
  this.use('promisify') // 添加此行代码，即可启用 async 和 await
}
```

5.4 WePY 页面中发起数据请求

2. 使用 wepy.request 发起 Get 请求

WePY 框架对原生小程序做了封装，之前通过 wx 调用的 API，都可以直接使用 wepy 进行调用，例如：wx.request() 是原生小程序的数据请求 API，现在可以直接通过 wepy.request() 发起网络数据请求。示例代码如下：

```
methods = {
  async getInfo() {
    const res = await wepy.request('https://www.liulongbin.top:8082/api/get')
    console.log(res)
  }
}
```

5.4 WePY 页面中发起数据请求

3. 使用 wepy.request 发起 Post 请求

通过 wepy.request() 方法发起 Post 请求的示例代码如下：

```
methods = {
  async postInfo() {
    const res = await wepy.request({
      url: 'https://www.liulongbin.top:8082/api/post',
      method: 'post',
      data: { name: 'zs', age: 20 }
    })
    console.log(res)
  }
}
```

5.4 WePY 页面中发起数据请求

4. 异步更新数据

在异步函数中更新数据的时候，页面检测不到数据的变化，**必须手动调用 this.$apply 方法。**作用是强制页面重新渲染，代码示例如下：

```
methods = {
  async getInfo() { // 被 async 修饰的函数叫做异步函数
    const res = await wepy.request('https://www.liulongbin.top:8081/api/get')
    this.getMsg = res.data
    this.$apply()
  }
}
```

传智播客旗下高端IT教育品牌