微信小程序基础Day02

今日目标:

- 能够使用 `WXML` 模板语法渲染页面结构
- 能够使用 `WXSS` 样式美化页面结构
- 能够使用 `app.json` 对小程序进行全局性配置
- 能够使用 `page.json` 对小程序页面进行个性化配置
- 能够知道如何发起网络数据请求

讲解目录:

- 1. `WXML` 模板语法
- 2. `WXSS` 模板样式
- 3.全局配置
- 4.页面配置
- 5.网络数据请求
- 6.案例 - 本地生活（首页）
- 7.总结

1. `WXML` 模板语法

1.1.数据绑定

小程序中, 数据绑定分为2个步骤:

① 在 data 中定义数据

② 在 `WXML` 中使用数据

> **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` 对象中即可

```
Page({
  /*
  * 页面的初始数据
  */
  data: {
    info: 'Hello World',
      // 如果是短横线的形式声明数据，需要使用双引号包裹
    "user-name": 'XX'
  },
  /*
  * 生命周期函数--监听页面加载
  */
  onLoad: function (options) {
  },
})
```

> **2. 在 `WXML` 中使用数据**

把data中的数据绑定到页面中渲染，使用 `Mustache` 语法（双大括号）将变量包起来即可。语法格式为

```
<!-- 插值表达式/大胡子语法 -->
<view>{{ info }}</view>
```

通过这两个步骤, 就可以将数据绑定到页面中进行展示.

下面对`Mustache`语法的应用场景进行介绍:

`Mustache`语法的主要应用场景如下:

- 绑定内容

  数据如下:

  ```
  Page({
      data: {
          info: 'init data'
      }
  })
  ```

  界面如下:

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

- 绑定属性

  数据如下:

  ```
  Page({
      data: {
          imgSrc: 'http://www.itheima.com/images/logo.png'
      }
  })
  ```

  页面如下:

  ```
  <!-- 动态绑定数据 -->
  <image src="{{ imgSrc }}" mode="widthFix"></image>
  ```

- 运算（三元运算、算术运算等）

  数据如下:

  ```
  Page({
      data: {
          randomNum: Math.random() * 10 // 生成 10 以内的随机数
      }
  })
  ```

  页面的结构如下:

  ```
  <view>
  {{ randomNum >= 5 ? '数字大于或等于5' : '数字小于5' }}
  </view>
  ```

注意: 在小程序中, 无论是标签的属性还是标签的内容, 都是使用`Mustache`语法进行数据绑定, 这和`Vue`有些差别, 在`Vue`中, 标签的属性是通过`v-bind`,标签的内容是通过`Mustache`语法.

1.2.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，反馈到逻辑层进行业务的处理

小程序中常用的事件如下

类型	绑定方式	事件描述
`tap`	`bindtap` 或 `bind:tap`	手指触摸后马上离开，类似于 HTML 中的 click 事件
`input`	`bindinput` 或 `bind:input`	文本框的输入事件
`change`	`bindchange` 或 `bind:change`	状态改变时触发

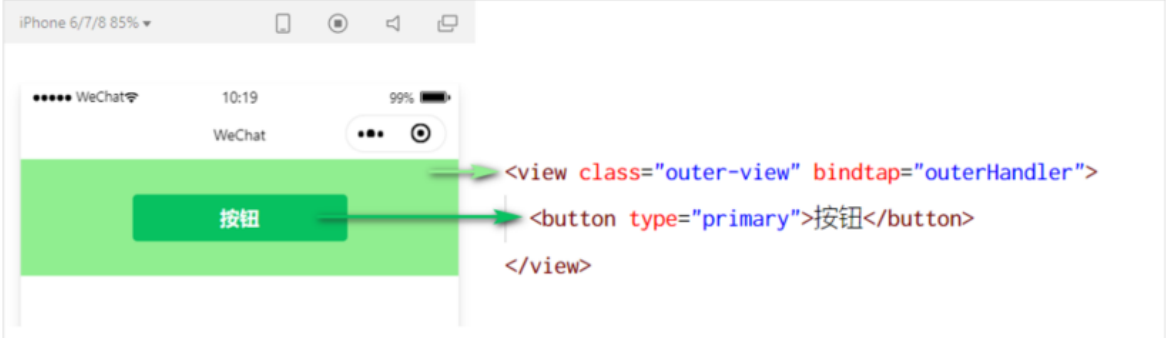
当事件回调触发的时候，会收到一个事件对象`event`，它的详细属性如下表所示：

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过的毫秒数
`target`	`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集合
`detail`	`Object`	额外的信息
`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

我们主要关注`target`和`detail`这两个属性

关于`target`属性, 它经常会和`currentTarget`容易混淆

- `target`是**触发该事件的源头组件**,
- `currentTarget`则是**当前事件所绑定的组件**。举例如下：



点击内部的按钮时，点击事件以**冒泡**的方式向外扩散，也会触发外层`view`的`tap`事件处理函数。

此时，对于外层的 view 来说：

`e.target` 指向的是触发事件的源头组件，因此，`e.target`是内部的按钮组件

`e.currentTarget` 指向的是当前正在触发事件的那个组件，因此，`e.currentTarget` 是当前的 view 组件

所以, 小程序的事件也是具备冒泡的哦.

1.2.1. `bindtap` 的使用

在小程序中，不存在 `HTML` 中的 `onclick` 鼠标点击事件，而是通过 `tap` 事件来响应用户的触摸行为。具体步骤如下:

① 通过`bindtap`，可以为组件绑定 tap 触摸事件

```
<!-- tap触摸事件 -->
<button type="primary" bindtap="onHandleTap">按钮</button>

<!-- +1按钮的事件 -->
<view class="countBox">{{ count }}</view>
<button bindtap="onAddHandle">点我试试</button>
```

② 在页面的 `.js` 文件中定义对应的事件处理函数，事件参数通过形参 `event`（一般简写成 `e`） 来接收, 注意,事件处理函数需要和`data`平级

```
Page({
 data: {

 },
 onHandleTap : function (e) {
    // 事件对象e
    console.log(e);
 },
 onAddHandle: function () {

 }
})
```

1.2.2. `js`中访问`data`中的数据

在`js`代码中, 如果想要访问`data中`的数据, 是通过`this.data.xxx`的方式进行访问

1.2.3. `js`中修改`data`中的数据

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

```
  onAddHandle: function () {
    this.setData({
      count: this.data.count + 1 // 在原来值基础上+1
    })
  }
```

1.2.4.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。

例如，下面的代码将不能正常工作:

```
<!-- 错误写法 ，小程序中会将bindtap对应的事件都当做事件名称 -->
<button bindtap="onAddHandle(1, 2)">点我试试</button>
```

因为小程序会把 `bindtap` 的属性值，统一当作事件名称来处理，相当于要调用一个名称为 `btnHandler(123)` 的事件处理函数。

我们可以为组件提供 `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：

```
<!--
  事件传参 data-*自定义属性传参，*代表参数名字
  info 会被解析为参数的数字
  数值2 会被解析为参数的值
-->
<button bindtap="onBtnString" data-info="2">事件传参-拼接字符串</button>
```

最终：

`info` 会作为名字存储在事件对象的 `target` 中的 `dataset` 中, 2 会被解析为值

在事件处理函数中，通过 `event.target.dataset.info` 即可获取到值，示例代码如下：

```
// 事件传参-拼接字符串
onBtnString (e) {
    this.setData({
        // this.data.count就是旧值
        count: this.data.count + e.target.dataset.info
    })
}
```

1.2.5. `bindinput` 的使用

在小程序中，通过 `input` 事件来响应文本框的输入事件，具体的使用步骤如下：

① 通过 `bindinput`，可以为文本框绑定输入事件：

```
<view class="iptBox">
  <!-- input事件 -->
  <input class="ipt1" bindinput="inputValue" focus></input>
</view>
```

② 在页面的 `.js` 文件中定义事件处理函数：

```
  inputValue (e) {
    console.log(e);
    console.log(e.detail.value);
  }
```

1.2.6.实现小程序中的双向数据绑定

实现步骤：

① 定义数据

② 渲染结构, 绑定 `data` 中的数据

③ 美化样式

④ 监听 input 事件

具体代码如下:

定义数据:

```
/*
* 页面的初始数据
*/
  data: {
    msg: '请输入...'
  }
```

渲染结构:将data中的`msg`绑定到输入框的`value`属性

```
<!-- input和data中的数据同步-双向数据绑定 -->
<view class="iptBox">
  <input value="{{ msg }}" bindinput="inputHandle" class="ipt2"
type="checkbox"></input>
</view>
```

美化样式:

```
.iptBox {
  width: 100%;
  height: 50px;
  display: flex;
  justify-content: center;
  align-items: center;
  margin-bottom: 10px;
  background-color: red;
}
input {
  width: 80%;
  padding: 0 20px;
  background-color: #fff;
  border: 1px solid #ccc;
  border-radius: 20px;
  color: #bbb;
}
```

监听 input 事件:

```
  // 实现input的数据和data数据同步
  inputHandle(e) {
    this.setData({
      msg: e.detail.value
    })
  }
```

1.3.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

也可以配合`wx:elif`和`wx:else`来添加`else`判断:

```
<!-- wx:if  wx:else-if wx:else 条件渲染 -->
<view wx:if="{{ type === 1 }}" class="text">男</view>
<view wx:elif="{{ type === 2 }}" class="text">女</view>
<view wx:else>保密</view>
```

1.3.1.结合 使用 `wx:if`

如果要一次性控制多个组件的展示与隐藏，可以使用一个 标签将多个组件包装起来，并在

标签上使用 `wx:if` 控制属性，使用block标签进行包裹，只是包裹性值，不会再页面渲染，类似于 `template` 标签示例如下：

```
<block wx:if="{{ true }}">
  <view>显示</view>
  <view>多个</view>
  <view>组件</view>
</block>
```

注意： 并不是一个组件，它只是一个包裹性质的容器，不会在页面中做任何渲染。

1.3.2.`hidden`

在小程序中，直接使用 `hidden="{{ condition }}"` 也能控制元素的显示与隐藏：

```
<!-- 使用 hidden="{{ condition }}" 也能控制元素的显示与隐藏 -->
<view hidden="{{ flag }}">条件位  true  时，隐藏元素</view>
```

1.3.3.`wx:if` 与 `hidden` 的对比

① 运行方式不同

- `wx:if` 以**动态创建**和**移除元素**的方式，控制元素的展示与隐藏
- `hidden` 以**切换样式**的方式（`display: none/block;`），控制元素的显示与隐藏

② 使用建议

- 频繁切换时，建议使用 `hidden`
- 控制条件复杂时，建议使用 `wx:if` 搭配 `wx:elif`、`wx:else` 进行展示与隐藏的切换

1.4.列表渲染

在小程序中，可以通过 `wx:for` 指定数组，进行循环渲染重复的组件结构，语法示例如下：

```
  data: {
    arr1: [
      '苹果',
      '华为',
      '小米'
    ]
  }

<!-- 循环渲染模板语法 -->
<view wx:for="{{ arr1 }}" class="text">
  索引是 {{ index }} 当前项是：  {{ item }}
</view>
```

默认情况下，当前循环项的索引用 `index` 表示；当前循环项用 `item` 表示。

我们也可以手动指定索引和当前项的变量名

- 使用 `wx:for-index` 可以指定当前循环项的索引的变量名
- 使用 `wx:for-item` 可以指定当前项的变量名

示例代码如下

```
<!-- 手动指定索引和当前项的变量名 -->
<view wx:for="{{ arr1 }}" class="text" wx:for-index="i" wx:for-item="item">
  索引是 {{ i }} 当前项是： {{ item }}
</view>
```

类似于 `Vue` 列表渲染中的 `**:key**`，小程序在实现列表渲染时，也建议为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

```
<!-- 手动指定索引和当前项的变量名 -->
<view wx:key="index" wx:for="{{ arr1 }}" class="text" wx:for-index="i"
wx:for-item="item">
  索引是 {{ i }} 当前项是： {{ item }}
</view>

<!-- wx:key  提高渲染效率 -->
<view wx:key="id" wx:for="{{ arr2 }}" class="text1">
  当前项是： {{ item.username }}
</view>
```

2. `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，类似于网页开发中的 `CSS`。

`WXSS` 具有 `CSS` 大部分特性，同时，`WXSS` 还对 `CSS` 进行了扩充以及修改，以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

2.1. `rpx`

> **1. 什么是`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。

> **2. `rpx` 的实现原理**

`rpx` 的实现原理非常简单：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` 把所有设备的屏幕，

在宽度上等分为 750 份，1份就是1`rpx`，所以：

- 在较小的设备上，1 `rpx` 所代表的宽度较小
- 在较大的设备上，1 `rpx` 所代表的宽度较大

小程序在不同设备上运行的时候，会自动把 `rpx` 的样式单位换算成对应的像素单位来渲染，从而实现屏幕适配。

> **3. `rpx` 与 `px` 之间的单位换算**

在 `iPhone6` 上，`CSS` 像素屏幕宽度为375 `px`，共有 750 个物理像素，等分为 `750rpx`。则：

750 `rpx` = 375 `px` = 750 物理像素

1 `rpx` = 0.5 `px` = 1物理像素

设备	`rpx` 换算 `px` (屏幕宽度/750)	`px` 换算 `rpx` (750/屏幕宽度)
`iPhone` 5	1 `rpx` = 0.42 `px`	`1px = 2.34rpx`
`iPhone6`	**1 `rpx` =** 0.5 `px`	`1px = 2rpx`
`iPhone6` Plus	1 `rpx` = 0.552 `px`	`1px = 1.81rpx`

官方建议：开发微信小程序时，设计师可以用 `iPhone6` 作为视觉稿的标准。

开发举例：在 `iPhone6` 上如果要绘制宽 `100px`，`高20px` 的盒子，换算成 `rpx` 单位，宽高分别为 `200rpx` 和 `40rpx`。

2.2.样式导入

使用 `WXSS` 提供的 `@import` 语法，可以导入外联的样式表。

`@import` 后跟需要导入的外联样式表的相对路径，用分号 ; 表示语句结束。示例如下：

```
/*
* 使用 @import
* 导入 字体图标样式表
* 注意使用绝对路径
* / 代表项目根路径
*/
@import "/icon/icon.wxss";

<!-- wx:if  wx:elif wx:else 条件渲染 -->
<!-- 使用外部引入的icon  字体图标 -->
<view wx:if="{{ type === 1 }}" class="text toutiao toutiao-wode">男</view>
<view wx:elif="{{ type === 2 }}" class="text toutiao toutiao-wode">女</view>
<view wx:else class="text toutiao toutiao-wode">保密</view>

```

2.3.全局样式和局部样式

> **1. 全局样式**

定义在 `app.wxss` 中的样式为全局样式，作用于每一个页面。

```css
/*
* 全局样式
*/
view {
  padding: 10rpx;
  margin: 10rpx;
  text-align: center;
  background-color: palevioletred;
  color: #fff;
  font-weight: 700;
}
```

> **2. 局部样式**

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。

注意：

① 当局部样式和全局样式冲突时，根据 `就近原则`，局部样式会覆盖全局样式

② 当局部样式的权重大于或等于全局样式的权重时，才会覆盖全局的样式

3.全局配置

3.1.全局配置文件及常用的配置项

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`

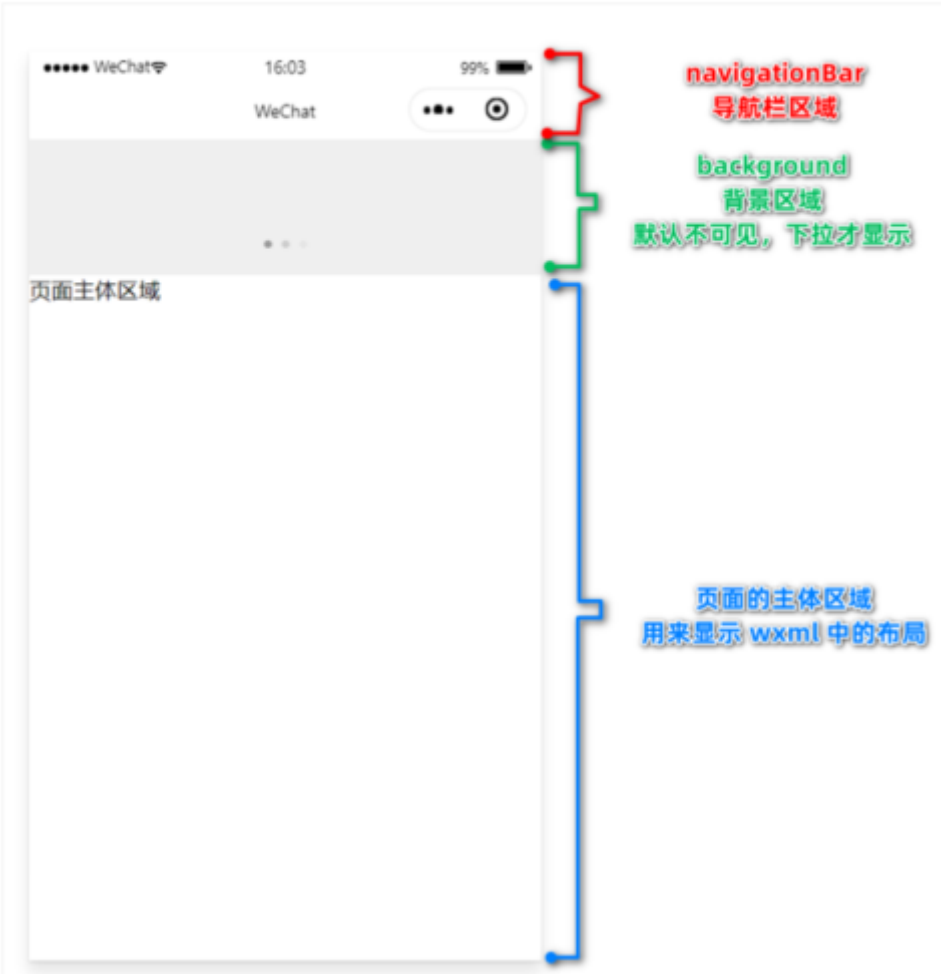
- 是否启用新版的组件样式

3.2.`window`

> **1. 小程序窗口的组成部分**

- 导航栏 - 顶部导航栏区域
- 背景区 - 默认不可见，下拉才显示
- 页面主体区 - 页面主体用来显示 `wxml` 中的布局

具体示意图如下：



2. 了解 window 节点常用的配置项

属性名	类型	默认值	说明
navigationBarTitleText	String	字符串	导航栏标题文字内容
navigationBarBackgroundColor	HexColor	#000000	导航栏背景颜色，如 #000000
navigationBarTextStyle	String	white	导航栏标题颜色，仅支持 black / white
backgroundColor	HexColor	#ffffff	窗口的背景色
backgroundTextStyle	String	dark	下拉 loading 的样式，仅支持 dark / light
enablePullDownRefresh	Boolean	false	是否全局开启下拉刷新
onReachBottomDistance	Number	50	页面上拉触底事件触发时距页面底部距离，单位为 px

```
"window":{
    // 下拉loading的样式，仅支持dark/light
  "backgroundTextStyle":"light",
    // 配置导航栏背景色,仅支持16进制的颜色
  "navigationBarBackgroundColor": "#fff",
    // 导航栏标题文本
  "navigationBarTitleText": "微信读书",
    // 导航栏标题颜色，仅支持 black/white
  "navigationBarTextStyle":"black",
    // 开启下拉刷新
  "enablePullDownRefresh": true,
```

```
      // 下拉窗口的背景色
    "backgroundColor": "#efefef",
      // 上拉触底的距离：默认50像素，单位省去,我们会在触发了上拉触底事件时获取下一页的数据
    "onReachBottomDistance": 50
  },
```

3. 设置导航栏的标题

设置步骤：`app.json` -> `window` -> `navigationBarTitleText`

需求：把导航栏上的标题，从默认的“`weChat`”修改为“黑马程序员”，效果如图所示：

4. 设置导航栏的背景色

设置步骤：`app.json` -> `window` -> `navigationBarBackgroundColor`

需求：把导航栏标题的背景色，从默认的 `#fff` 修改为 `#2b4b6b` ，效果如图所示：

5. 设置导航栏的标题颜色

设置步骤：`app.json` -> `window`-> `navigationBarTextStyle`

需求：把导航栏上的标题颜色，从默认的 `black` 修改为 `white` ，效果如图所示：

注意： `navigationBarTextStyle` 的可选值只有 `black` 和 `white` 两个可选值

6. 全局开启 下拉刷新 功能

下拉刷新是移动端的专有名词，指的是通过手指在屏幕上的下拉滑动操作，从而重新加载页面数据的行为。

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

注意：在 `app.json` 中启用下拉刷新功能，会作用于每个小程序页面！

7. 设置下拉刷新时窗口的背景色

当全局开启下拉刷新功能之后，默认的窗口背景为白色。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 `#efefef`。效果如下：

8. 设置下拉刷新时 loading 的样式

当全局开启下拉刷新功能之后，默认窗口的 loading 样式为白色，如果要更改 loading 样式的效果，

设置步骤为 `app.json` -> `window` -> 为 `backgroundTextStyle` 指定 dark 值。效果如下：

注意： `backgroundTextStyle` 的可选值只有 light 和 dark

> **9. 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

注意：默认距离为50 `px`，如果没有特殊需求，建议使用默认值即可。

3.3. `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：

- 底部 `tabBar`
- 顶部 `tabBar`

注意：

- `tabBar` 中只能配置最少 2 个、最多 5 个 `tab` 页签
- 当渲染顶部 `tabBar` 时，不显示 `icon`，只显示文本

1. `tabBar` 的 6 个组成部分

① `backgroundColor`：`tabBar` 的背景色

② `selectedIconPath`：选中时的图片路径

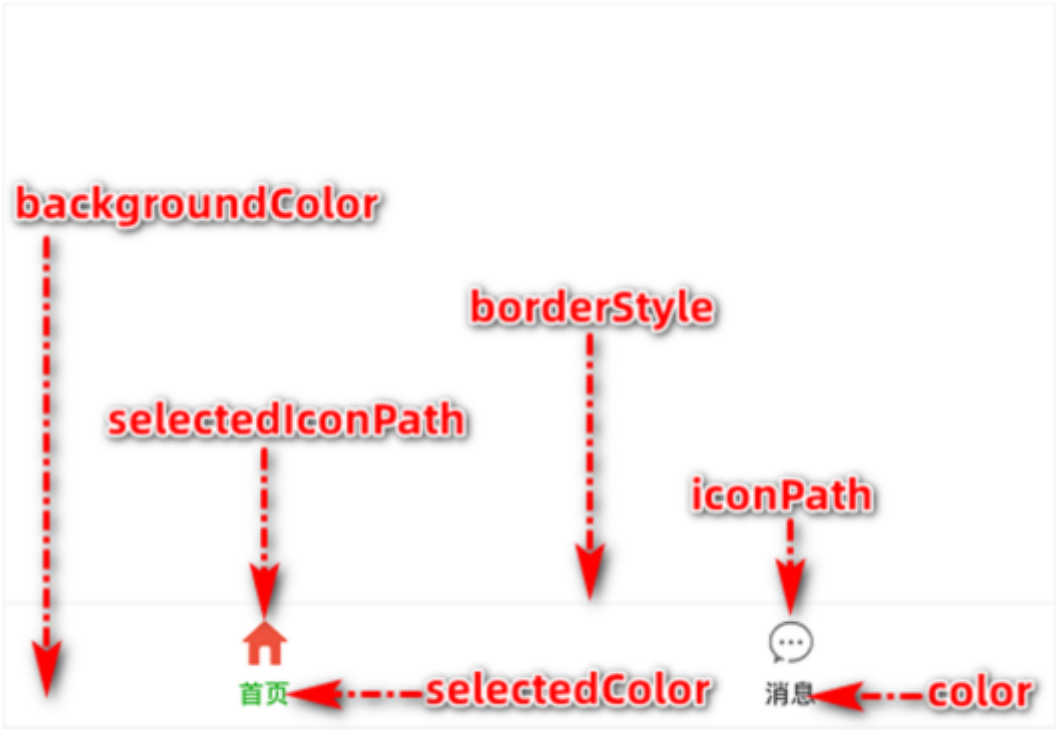
③ `borderStyle：tabBar` 上边框的颜色

④ `iconPath`：未选中时的图片路径

⑤ `selectedColor：tab` 上的文字选中时的颜色

⑥ `color：tab` 上文字的默认（未选中）颜色

具体示意图如下:



2. `tabBar` 节点的配置项

属性	类型	必填	描述
pagePath	String	是	页面路径，页面必须在 pages 中预先定义
text	String	是	tab 上显示的文字
iconPath	String	否	未选中时的图标路径；当 postion 为 top 时，不显示 icon
selectedIconPath	String	否	选中时的图标路径；当 postion 为 top 时，不显示 icon

4. 每个 tab 项的配置选项

属性	类型	必填	描述
pagePath	String	是	页面路径，页面必须在 pages 中预先定义
text	String	是	tab 上显示的文字
iconPath	String	否	未选中时的图标路径；当 postion 为 top 时，不显示 icon
selectedIconPath	String	否	选中时的图标路径；当 postion 为 top 时，不显示 icon

5. 案例: 配置 `tabBar`

- 根据资料中提供的小图标, 在小程序中实现如下效果:

- 实现步骤:
  - 拷贝图标资源
  - 新建 3 个对应的 `tab` 页面
  - 配置 `tabBar` 选项

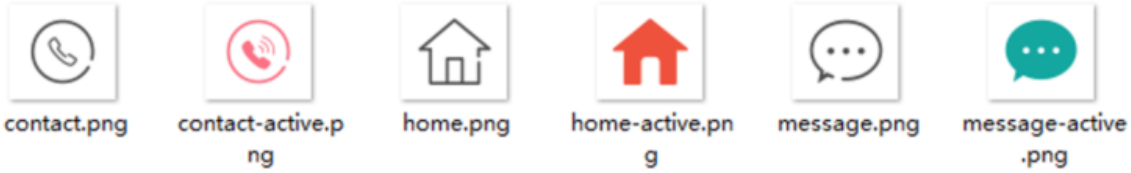
> **步骤1 - 拷贝图标资源**

① 把资料目录中的 images 文件夹，拷贝到小程序项目根目录中

② 将需要用到的小图标分为 3 组，每组两个，其中：

- 图片名称中包含 -active 的是选中之后的图标
- 图片名称中不包含 -active 的是默认图标

截图如下：



> **步骤2 - 新建 3 个对应的 tab 页面**

通过 `app.json` 文件的 `pages` 节点，快速新建 3 个对应的 `tab` 页面，示例代码如下：

```
{
    "pages": [
        "pages/home/home",
        "pages/message/message",
        "pages/contact/contact"
    ]
}
```

其中，`home` 是首页，`message` 是消息页面，`contact` 是联系我们页面。

> **步骤3 - 配置 `tabBar` 选项**

① 打开 `app.json` 配置文件，和 `pages`、`window` 平级，新增 `tabBar` 节点

② `tabBar` 节点中，新增`list`数组，这个数组中存放的，是每个 `tab` 项的配置对象

③ 在`list` 数组中，新增每一个 `tab` 项的配置对象。对象中包含的属性如下：

- `pagePath` 指定当前 tab 对应的页面路径【必填】
- `text` 指定当前 tab 上按钮的文字【必填】
- `iconPath` 指定当前 tab 未选中时候的图片路径【可选】
- `selectedIconPath` 指定当前 tab 被选中后高亮的图片路径【可选】

代码如下:

```
"tabBar": {
    "list": [
      {
          "pagePath": "pages/home/home",
          "text": "首页",
          "iconPath": "/images/tabs/home.png",
          "selectedIconPath": "/images/tabs/home-active.png"
      },
      {
          "pagePath": "pages/message/message",
          "text": "消息",
          "iconPath": "/images/tabs/message.png",
          "selectedIconPath": "/images/tabs/message-active.png"
      },
      {
          "pagePath": "pages/index/index",
          "text": "主页",
          "iconPath": "/images/icon_应用管理.png",
          "selectedIconPath": "/images/icon_应用管理.png"
      },
      {
          "pagePath": "pages/home/home",
          "text": "联系人",
          "iconPath": "/images/tabs/contact.png",
          "selectedIconPath": "/images/tabs/contact-active.png"
      }
    ]
  },
```

4.页面配置

> **1. 页面配置文件的作用**

小程序中，每个页面都有自己的 `.json` 配置文件，用来对当前页面的窗口外观、页面效果等进行配置。

> **2. 页面配置 和 全局配置 的关系**

小程序中，`app.json` 中的 window 节点，可以**全局配置**小程序中每个页面的窗口表现。

如果某些小程序页面**想要拥有特殊的窗口表现**，此时，页面级别的 `.json` 配置文件就可以实现这种需求。

注意：当页面配置与全局配置冲突时，根据**就近原则**，最终的效果以页面配置为准。

> **3. 页面配置中常用的配置项**

属性	类型	默认值	描述
navigationBarBackgroundColor	HexColor	#000000	当前页面导航栏背景颜色，如 #000000
navigationBarTextStyle	String	white	当前页面导航栏标题颜色，仅支持 black / white
navigationBarTitleText	String		当前页面导航栏标题文字内容
backgroundColor	HexColor	#ffffff	当前页面窗口的背景色
backgroundTextStyle	String	dark	当前页面下拉 loading 的样式，仅支持 dark / light
enablePullDownRefresh	Boolean	false	是否为当前页面开启下拉刷新的效果
onReachBottomDistance	Number	50	页面上拉触底事件触发时距页面底部距离，单位为 px

注意：

① 这些配置项, 在刚才学习的全局配置下的`window`节点我们已经都学习过了,无需额外记忆.

② 一般我们不需要每个页面都有下拉刷新的效果，因此，再全局配置文件中，不推荐全局配置下拉刷新，而是某个列表页面需要开启时，在页面配置文件中，单独开启下拉刷新

```
{
  "navigationBarBackgroundColor": "#f00",
  "navigationBarTitleText": "主页",
  "enablePullDownRefresh": true,
  "usingComponents": {}
}
```

5.网络数据请求

- 如何在小程序中发起网络请求?
  - 该请求不能称作ajax请求，但是可以称为网络请求
  - 准备工作：
    - 考虑安全性，微信官方要求域名使用`https协议`
    - 配置后台的request的合法域名
  - 具体代码
    - 如下
  - 小程序中的网络请求不存在跨域的问题

> **1. 小程序中网络数据请求的限制**

出于安全性方面的考虑，小程序官方对数据接口的请求做出了如下

两个限制：

① 只能请求 `HTTPS` 类型的接口

② 必须将接口的域名添加到信任列表中

> **2. 配置 `request` 合法域名**

需求描述：假设在自己的微信小程序中，希望请求 https://www.escook.cn/ 域名下的接口

配置步骤：登录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 域名不能使用 `IP` 地址或 `localhost`

③ 域名必须经过 `ICP` 备案

④ 服务器域名一个月内最多可申请 5 次修改

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 数据请求，示例代码如下：

```
// 发起GET请求
onTapGet () {
  wx.request({
    // 请求地址，必须是以https://开头
    // 必须是配置在request合法域名
    url: 'https://www.escook.cn/api/get',
    // 请求方式
    method: 'GET',
    // 请求参数
    data: {
        name: 'zs',
        age: 22
    },
    // 请求成功的回调
    success: (res) => {
        console.log(res)
    }
  })
```

4. 发起 POST 请求

调用微信小程序提供的 `wx.request()` 方法，可以发起 POST 数据请求，示例代码如下：

```
// 发起GET请求
onTapPost () {
  wx.request({
    // 请求地址，必须是以https://开头
    // 必须是配置在request合法域名
    url: 'https://www.escook.cn/api/post',
    // 请求方式
    method: 'POST',
    // 请求参数
    data: {
        name: 'lisi',
        age: 18
    },
    // 请求成功的回调
    success: (res) => {
        console.log(res)
    }
  })
```

5. 在页面刚加载时请求数据

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。此时需要在页面的 `onLoad` 事件中调用获取数据的函数，示例代码如下：

```
/**
 * 生命周期函数--监听页面加载--初始化页面的数据
 */
onLoad: function (options) {
  // 通过  this  关键字  调用上述两个方法
  // 可以自动发起请求，而不需要点击事件来触发
  this.onTapGet()
  this.onTapPost()
},
```

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。*

此时为了不耽误开发的进度，我们可以在微信开发者工具中，临时开启「**开发环境不校验合法域名、web-view(业务域名)、`TLS` 版本及 `HTTPS` 证书**」选项，跳过 `request` 合法域名的校验。

注意：

跳过 `request` 合法域名校验的选项，仅限在**开发与调试阶段**使用！

7. 关于 跨域 和 Ajax 的说明

跨域问题只存在于基于浏览器的 `web` 开发中。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

程序中不存在跨域的问题。

`Ajax` 技术的核心是依赖于浏览器中的`XMLHttpRequest` 这个对象，由于小程序的宿主环境是微信客户端，所

以小程序中不能叫做“发起 `Ajax` 请求”，而是叫做“发起网络数据请求”。

6.案例-本地生活

> **1. 首页效果以及实现步骤**

① 新建项目并梳理项目结构

② 配置导航栏效果

③ 配置 `tabBar` 效果

④ 实现轮播图效果

⑤ 实现九宫格效果

⑥ 实现图片布局

> **2. 接口地址**

① 获取轮播图数据列表的接口

- 【GET】https://www.escook.cn/slides

② 获取九宫格数据列表的接口

- 【GET】https://www.escook.cn/categories

> **3.完整代码**

`wxss` 样式

```
/* pages/home/home.wxss */
swiper {
  height: 350rpx;
}

swiper image {
  width: 100%;
  height: 100%;
}

.gird-list {
  display: flex;
  flex-wrap: wrap;
  border-left: 1rpx solid #efefef;
  border-top: 1rpx solid #efefef;
}
.gird-item {
  display: flex;
  box-sizing: border-box;
  flex-direction: column;
  align-items: center;
  justify-content: center;
  border-right: 1rpx solid #efefef;
  border-bottom: 1rpx solid #efefef;
  width: 33.33%;
  height: 200rpx;
```

```
}
.gird-item image {
  width: 60rpx;
  height: 60rpx;
}

.gird-item text {
font-size: 24rpx;
margin-top: 10rpx;
}
.img-box {
  display: flex;
  justify-content: space-around;
  padding: 20rpx 10rpx;
}
.img-box image {
  width: 45%;
  height: 256rpx;
}
```

`wxml`布局

```
<!--pages/home/home.wxml-->
<!-- 轮播图区域 -->
<swiper indicator-dots circular autoplay>
  <swiper-item wx:for="{{ swiperList}}" wx:key="id">
    <image src="{{ item.image }}"></image>
  </swiper-item>
</swiper>

<!-- 九宫格区域 -->
<view class="gird-list">
  <view class="gird-item" wx:for="{{girdList}}" wx:key="id">
    <image src="{{ item.icon }}"></image>
    <text>{{ item.name }}</text>
  </view>
</view>

 <!-- 图片区域 -->
<view class="img-box">
  <image src="/images/link-01.png"></image>
  <image src="/images/link-02.png"></image>
</view>
```

`js`逻辑

```
  /**
   * 页面的初始数据
   */
  data: {
    // 存放轮播图数据的列表
    swiperList: [],
    // 存放九宫格的数据
    girdList: []
  },

```

```
  /**
   * 生命周期函数--监听页面加载
   */
  onLoad: function (options) {
    // 初始化页面，调用方法
    this.getSwiperList()
    this.getGirdList()
  },
  // 获取轮播图数据的方法
  getSwiperList () {
    // 发起请求
    wx.request({
      url: 'https://www.escook.cn/slides',
      method: 'GET',
      success: (res) => {
        // console.log(res)
        // 修改data中的数据
        this.setData({
          swiperList: res.data
        })
      }
    })
  },

  // 获取九宫格数据的方法
  getGirdList () {
    wx.request({
      url: 'https://www.escook.cn/categories',
      method: 'GET',
      success: (res) => {
        // console.log(res)
        // 修改data中的数据
        this.setData({
          girdList: res.data
        })
      }
    })
  },
```

7.总结

① 能够使用 `WXML` 模板语法渲染页面结构

- `wx:if`、`wx:elif`、`wx:else`、`hidden`、`wx:for`、`wx:key`

② 能够使用 `WXSS` 样式美化页面结构

- `rpx` 尺寸单位、`@import` 样式导入、全局样式和局部样式

③ 能够使用 `app.json` 对小程序进行全局性配置

- `pages`、`window`、`tabBar`、`style`

④ 能够使用 `page.json` 对小程序页面进行个性化配置

- 对单个页面进行个性化配置、就近原则

⑤ 能够知道如何发起网络数据请求

- `wx.request()` 方法、 `onLoad()` 事件