

小程序基础

一、小程序导航 -- 声明式导航

001 - 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。

```
<navigator url="/pages/about/about">跳转到 about 页面</navigator>
```

- 注意事项
 - `url` 属性设置需要跳转的路径
 - 页面路径应该以 / 开头，
 - 路径必须提前在 `app.json` 的 `pages` 节点下声明

002 - 导航到 `tabBar` 页面

`navigator` 组件单纯使用 `url` 属性，无法导航到 `tabBar` 页面，必须需要结合 `open-type` 属性进行导航。

```
<navigator url="/pages/person/person" open-type="switchTab">跳转到 tabBar 页面</navigator>
```

003 - 后退导航

小程序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

```
<navigator open-type='navigateBack' delta='1'> 返回上一页 </navigator>
<navigator open-type='navigateBack' delta='2'> 返回上上一页 </navigator>
```

二、小程序导航 -- 编程式导航

001 - 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。

- 参数文档
 - [wx.navigateTo 详细文档](#)
- 代码案例

```
// 跳转到非导航页面
handle: function () {
  wx.navigateTo({
    url: '/pages/about/about',
    success: function () {
      console.log('Hello about')
    }
  })
},
```

002 - 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

- 参数文档
 - [wx.switchTab 详细文档](#)
- 案例代码

```
// 跳转到 tabBar 页面
tabBarHandle: function () {
  wx.switchTab({
    url: '/pages/person/person',
    success: function() {
      console.log('Hello Person')
    }
  })
},
```

003 - 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。

- 参数文档
 - [wx.navigateBack 详细文档](#)
- 案例代码

```
handle: function () {
  wx.navigateBack({
    delta: 1
  })
},
twoHandle: function () {
  wx.navigateBack({
    delta: 2
  })
},
```

三、小程序导航 -- 导航传参

001 - 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。

```
<navigator url="/pages/about/about?age=18&name=shuji">跳转到 about 页面</navigator>
```

002 - 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。

- 案例代码

```
// 跳转到 tabBar 页面
tabBarHandle: function () {
  wx.switchTab({
    url: '/pages/person/person?age=18&name=shuji',
    success: function() {
      console.log('Hello Person')
    }
  })
},
```

003 - 接受传递的参数

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。

```
onLoad: function (options) {
  // 打印传递出来的参数
  console.log(options)
},
```

004 - 导航栏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：
 - 单击工具栏上的“普通编译”下拉菜单
 - 单击下拉菜单中的“添加编译模式”选项
 - 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。



四、网络数据请求

001 - 小程序后台配置

- 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

注意：[网络配置详情](#)

002 - 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。

注意：[网络配置详情](#)

003 - 小程序发送 get 与 Post 请求

小程序发送请求使用 `wx.request()` 方法，

- Get 案例代码

```
getData: function () {
  wx.request({
    url: 'xxx',
    method: 'get',
    success: function (res) {
      console.log(res)
    }
  })
},
```

- Post 代码案例

```
postData: function () {
  wx.request({
    url: 'https://www.baidu.com/api/post',
    method: 'post',
    data: {
      name: 'shuji'
    },
    success: function (res) {
      console.log(res)
    }
  })
},
```

注意：method 如果不进行配置，默认参数是 get 请求方式

[wx.request 详细文档说明](#)

004 - 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。
- 注意：**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

五、小程序组件 -- 创建与引用

[组件创建详细文档](#)

001 - 组件的创建

- 在项目的根目录中，鼠标右键，创建 components 文件夹 --> Tabs
- 在新建的 components -> tabs文件夹上，鼠标右键，点击“新建 Component”
- 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：应当尽量将不同的组件，存放单独的文件夹中，从而保证清晰的目录结构

002 - 组件的引用

- 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
- 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**相对路径**
- 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

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

注册组件名称时，建议把组件名称使用中横线进行连接，例如 `vant-button` 或 `custom-button`

六、小程序组件 -- 组件的样式

[小程序组件样式详细文档](#)

- 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：
- 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。
- 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。
- 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。
- 继承样式，如 `font`、`color`，会从组件外继承到组件内。
- 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。

注意：以上语法不推荐死记硬背，建议使用 `class` 选择器

<https://3g.163.com/main>

tabs.wxml

```
<view class="tabs">
  <!--
  <view class="title">
    <view class="title-item active">首页</view>
    <view class="title-item">原创</view>
```

```

        <view class="title-item">要闻</view>
        <view class="title-item">推荐</view>
    </view>
-->
    <view
      wx:for="{{tabs}}"
      wx:key="id"
      class="title-item {{item.isActive?'active':''}}"
    >
      {{item.name}}
    </view>
  </view>
</view>

```

tabs.wxss

```

.tabs{}
.title{
  display: flex;
  padding: 10rpx 0;
}
.title-item{
  flex: 1;
  display: flex;
  justify-content: center;
  align-items: center;
}
.active{
  border-bottom: 5rpx solid red;
}
.tabs-content{}

```

tabs.js

```

// components/Tabs/Tabs.js
Component({
  /**
   * 组件的属性列表
   */
  properties: {},
  /**
   * 组件的初始数据
   */
  data: {
    tabs: [
      {
        id: 0, name: '首页', isActive: true,
      },
      {
        id: 1, name: '原创', isActive: false,
      },
      {
        id: 2, name: '要闻', isActive: false,
      },
    ],
  },
})

```

```

    },
    {
      id: 3, name: '推荐', isActive: false,
    },
  ],
},
},

/**
 * 组件的方法列表
 */
methods: {},
})

```

激活被选中的选项卡

在wxml当中,添加bindtap="handleItemTap"和 data-index="{{index}}"`

```

<view
  wx:for="{{tabs}}"
  wx:key="id"
  class="title-item {{item.isActive?'active':''}}"
  bindtap="handleItemTap"
  data-index="{{index}}"
>
  {{item.name}}
</view>

```

tabs.js

```

// components/Tabs.js
Component({
  /**
   * 里面存放的是 要从父组件中接收的数据
   */
  properties: {
  },
  /**
   * 组件的初始数据
   */
  data: {
    // tabs
    tabs: [
      {
        id: 0, name: '首页', isActive: true,
      },
      {
        id: 1, name: '原创', isActive: false,
      },
      {
        id: 2, name: '要闻', isActive: false,
      },
      {

```



```

        id: 3, name: '推荐', isActive: false,
      },
    ],
  },
  /*
  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 必须要存在在 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 给每一个循环性 选中属性 改为 false
        2 给 当前的索引的 项 添加激活选中效果就可以了!!!
      */
      // 2 获取索引
      const {index}=e.currentTarget.dataset;
      // 3 获取data中的数组
      // 解构 对 复杂类型进行结构的时候 复制了一份 变量的引用而已
      // 最严谨的做法 重新拷贝一份 数组, 再对这个数组的备份进行处理,
      // let tabs=JSON.parse(JSON.stringify(this.data.tabs));
      // 不要直接修改 this.data.数据
      let {tabs}=this.data;
      // let tabs=this.data;
      // 4 循环数组
      // [].forEach 遍历数组 遍历数组的时候 修改了 v , 也会导致源数组被修改
      tabs.forEach((v,i)=>i===index?v.isActive=true:v.isActive=false);
      this.setData({
        tabs:tabs
      })
    }
  }
})

```

七、小程序组件 -- 数据与方法

[组件详细的参数含义和使用](#)

001 - 使用 data 定义组件的私有数据

- 小程序组件中的 `data` 与小程序页面中的 `data` 用法一致，区别是：
 - 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

002 - 使用 methods 定义组件的事件处理函数

[组件间通信与事件 详细文档](#)

- 和页面不同，组件的事件处理函数，必须定义在 methods 节点中

```
methods: {  
  handle: function () {  
    console.log('组件的方法要定义在 methods 中')  
    this.setData({  
      num: this.data.num + 1  
    })  
    console.log(this.data.num)  
  }  
}
```

八、小程序组件 -- properties

001 - properties 简介

组件的对外属性，用来接收外界传递到组件中的数据。类似于 Vue 中的 props

- 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

002 - properties 语法结构

```
properties: {  
  a: {           // 属性名  
    type: String, // 属性的数据类型  
    value: ''     // 默认值  
  }  
}
```

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

003 - 向组件传递 properties 的值

使用数据绑定的形式，向子组件的属性传递动态数据

```
<second-com prop-price="{{ priceData }}"></second-com>
```

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}}")。

```
// 组件 com02.js
properties: {
  propPrice: {
    type: Number,
    value: 1
  }
},
```

```
<!-- 引用组件的页面 -->
<second-com prop-price="{{ priceData }}"></second-com>
```

```
<!-- 组件 com02.html -->
<view>{{ propPrice }}</view>
```

005 - 组件内修改 properties

properties 的值是可读可写的，可以通过 `setData` 修改 `properties` 中任何属性的值，

- 案例代码

```
methods: {
  handle: function () {
    this.setData({
      propPrice: this.properties.propPrice + 1
    })
    console.log(this.properties.propPrice)
  }
}
```

父组件向子组件传值

在父组件index.xml当中

```
<!-- <Tabs></Tabs> -->
1 父组件(页面) 向子组件 传递数据 通过 标签属性的方式来传递
1 在子组件上进行接收
2 把这个数据当成是data中的数据直接用即可
<tabs username="张山"></tabs>
```

在子组件当中，tabs.js

```

/**
 * 里面存放的是 要从父组件中接收的数据
 */
properties: {
  // 要接收的数据的名称
  username: {
    // type 要接收的数据的类型
    type: String,
    // value 默认值
    value: ""
  }
},

```

在子组件当中，Tabs.wxml直接使用

```

{{username}}
<view class="tabs">
  <view class="title">
    <!-- <view class="title-item active">首页</view>
    <view class="title-item">原创</view>
    <view class="title-item">要闻</view>
    <view class="title-item">推荐</view> -->
    <view wx:for="{{tabs}}" wx:key="id" class="title-item {{item.isActive?'active':''}}"
    bindtap="hanldeItemTap" data-index="{{index}}">
      {{item.name}}
    </view>
  </view>
</view>
</view>

```

我们把子组件当中的data数据放到父组件当中

把Tabs.js当中的数据放到父组件当中的index.js当中

```

Page({
  data: {
    tabs: [
      {
        id: 0, name: '首页', isActive: true,
      },
      {
        id: 1, name: '原创', isActive: false,
      },
      {
        id: 2, name: '要闻', isActive: false,
      },
      {
        id: 3, name: '推荐', isActive: false,
      },
    ],
  },
})

```

在index.wxml当中

```
<!-- <Tabs></Tabs> -->
<!-- <Tabs username="张山"></Tabs> -->

<Tabs tabs="{{tabs}}"></Tabs>
```

在子组件Tabs.js当中

```
properties: {
  // username: {
  //   // type 要接收的数据的类型
  //   type: String,
  //   // value 默认值
  //   value: '',
  // },
  tabs: {
    type: Array,
    value: [],
  }
},
```

子组件向父组件传值

在index.wxml

```
<!--
1 父组件(页面) 向子组件 传递数据 通过 标签属性的方式来传递
1 在子组件上进行接收
2 把这个数据当成是data中的数据直接用即可
2 子向父传递数据 通过事件的方式传递
1 在子组件的标签上加入一个 自定义事件
-->
<Tabs tabs="{{tabs}}" bind itemChange="handleItemChange" >
```

Tabs.js

```
5 点击事件触发的时候
    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)
    */
// 2 获取索引
methods: {
  hanldeItemTap(e){
    const {index}=e.currentTarget.dataset;
    // 5 触发父组件中的自定义事件 同时传递数据给
    this.triggerEvent("itemChange",{index});
  }
}
```

在index.js

```
// pages/demo17/demo18.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
      }
    ],
  },
  // 自定义事件 用来接收子组件传递的数据的
  handleItemClick(e) {
    // 接收传递过来的参数
    const { index } = e.detail;
    let { tabs } = this.data;
    tabs.forEach((v, i) => i === index ? v.isActive = true : v.isActive = false);
    this.setData({
      tabs
    })
  }
})
```

小程序插槽的使用

tabs.wxml

```
<view class="tabs">
  <view class="tabs_title">
    <!-- <view class="title_item active">首页</view>
    <view class="title_item">原创</view>
    <view class="title_item">分类</view>
    <view class="title_item">关于</view> -->

  </view>
```

```

wx:for="{{tabs}}"
wx:key="id"
class="title_item {{item.isActive?'active':''}}"
bindtap="hanldeItemTap"
data-index="{{index}}"
>
  {{item.name}}
</view>
</view>
<view class="tabs-content">
  <!-- 内容 -->
  <!--
    slot 标签 其实就是一个占位符 插槽
    等到 父组件调用 子组件的时候 再传递 标签过来 最终 这些被传递的标签
    就会替换 slot 插槽的位置
  -->
  <slot></slot>
</view>
</view>

```

九、小程序组件 -- 数据监听器

001 - 基本使用方法

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

[数据监听详细文档](#)

```

observers: {
  'propPrice, num': function (newPropPrice, newNum) {
    console.log(newPropPrice)
    console.log(newNum)
  }
},

```

002 - 监听子数据字段语法

- 案例代码

```
// 监控某个子数据的代码
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）
    }
  }
})
```

```
// 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
    }
  }
})
```

十、组件的生命周期

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。

- 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。

其他：[组件生命周期详解](#)

十一、小程序插槽的使用

001 - 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。
- 案例代码

```
// 组件模板
<view>
  <view>我是组件</view>
  <slot></slot>
</view>
```

```
// 引用组件的页面模板
<second-com>
  <view>你好，我是引用组件</view>
</second-com>
```

002 - 多个插槽

1. 在组件中，需要使用多 `slot` 时，可以在组件 `js` 中声明启用。
 - 案例代码

```
Component({
  options: {
    multipleSlots: true
  },
})
```

2. 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

```
// 引用组件的页面模板
<second-com prop-price="{{ priceData }}">
  <view slot="name">你好，这是 name 插槽 </view>
  <view slot="age">你好，这是 age 插槽</view>
</second-com>
```

3. 使用多个插槽

```
// 组件插槽
<view>
  <view>我是组件</view>
  <slot name="name"></slot>
  <slot name="age"></slot>
</view>
```

十二、组件间的通信

001 - 组件之间的三种基本通信方式

- `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

- 事件：用于子组件向父组件传递数据，可以传递任意数据。
- 父组件通过 `this.selectComponent` 方法获取子组件实例对象，便可以直接访问组件的任意数据和方法。

002 - `this.selectComponent` 使用

父组件的 `.js` 文件中，可以调用 `this.selectComponent(string)` 函数并指定 `id` 或 `class` 选择器，获取子组件对象调用，可以返回指定组件的实例对象

- 案例代码

```
// 使用组件的页面模板
<second-com class="second" id="second" prop-price="{{ priceData }}">
  <view slot="name">你好，这是 name 插槽 </view>
  <view slot="age">你好，这是 age 插槽</view>
</second-com>
```

```
// 使用组件的 .js 文件，使用方法触发
changeData: function () {
  // console.log(this.selectComponent('#second'))
  console.log(this.selectComponent('.second'))
},
```

- 注意事项
 - 不能传递标签选择器 (`component-a`)，不然返回的是 `null`

003 - 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。

- 实现步骤
 - 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
 - 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
 - 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
 - 在父组件的 `js` 中，通过 `e.detail` 获取到子组件传递过来的数据
- 案例代码

```
// 使用组件的页面模板自定义 myEvent 事件,接收 getCount 方法
<second-com bind:myEvent="getCount" class="second" id="second" prop-price="{{ priceData }}">
  <view slot="name">你好，这是 name 插槽 </view>
  <view slot="age">你好，这是 age 插槽</view>
</secondcom>
```

```
// 使用组件页面 js，生命 getCount 方法
getCount: function (e) {
  console.log(e.detail)
},
```

```
// 组件页面
this.triggerEvent('myEvent', {
  count: this.data.num
})
```

常用组件

image组件

```
<!--
image 图片标签
1 src 指定要加载的图片的路径
  图片存在默认宽度和高度 320 * 240      原图大小是 200 * 100
2 mode 决定 图片内容 如何 和 图片标签 宽高 做适配
  1 scaleToFill 默认值 不保持纵横比缩放图片，使图片的宽高完全拉伸至填满 image 元素
  2 aspectFit 保持宽高比 确保图片的长边 显示出来  页面轮播图 常用
  3 aspectFill 保持纵横比缩放图片，只保证图片的短边能完全显示出来。  少用
  4 widthFix 以前web的图片的 宽度指定了之后 高度 会自己按比例来调整  常用
  5 bottom。。 类似以前的background-position
3 小程序当中的图片 直接就支持 懒加载 lazy-load
  1 lazy-load 会自己判断 当 图片 出现在 视口 上下 三屏的高度 之内的时候 自己开始加载图片
-->
<image mode="bottom" lazy-load
src="https://tva2.sinaimg.cn/large/007DFXDhgy1g51jlzfb41j305k02s0sp.jpg" />
```

```
image{
  box-sizing: border-box;
  border: 1px solid red;
  width: 300px;
  height: 200px;
}
```

swiper组件

```
<!--
vw和vh是css3中的新的单位，是一种视窗单位，小程序中也同样适用
小程序中，窗口固定宽度为100vw，将窗口宽度平均分为100份，每一份是1vw。
小程序中，窗口固定高度为100vh，将窗口高度平均分为100份，每一份是1vh。
1 轮播图外层容器 swiper
2 每一个轮播项 swiper-item
3 swiper标签 存在默认样式
  1 width 100%
  2 height 150px      image 存在默认宽度和高度 320 * 240
  3 swiper 高度 无法实现由内容撑开
4 先找出来 原图的宽度和高度 等比例 给swiper 定 宽度和高度
  原图的宽度和高度 1125 * 352 px
  swiper 宽度 / swiper 高度 = 原图的宽度 / 原图的高度
  swiper 高度 = swiper 宽度 * 原图的高度 / 原图的宽度
  height: 100vw * 352 / 1125
```

```

5 autoplay 自动轮播
6 interval 修改轮播时间
7 circular 衔接轮播
8 indicator-dots 显示 指示器 分页器 索引器
9 indicator-color 指示器的未选择的颜色
10 indicator-active-color 选中的时候的指示器的颜色
-->
<swiper autoplay interval="1000" circular indicator-dots indicator-color="#0094ff" indicator-
active-color="#ff0094">
  <swiper-item> <image mode="widthFix"
src="//gw.alicdn.com/imgextra/i1/44/01CN013zKZP11CCByG5bAeF_!!44-0-luban.jpg" /> </swiper-item>
  <swiper-item> <image mode="widthFix"
src="//aecpm.alicdn.com/simba/img/TB1CWf9KpXXXXbuXpXXSutbFXXX.jpg_q50.jpg" /> </swiper-item>
  <swiper-item> <image mode="widthFix"
src="//gw.alicdn.com/imgextra/i2/37/01CN01syHZxs1C8zCFJj97b_!!37-0-luban.jpg" /> </swiper-item>
</swiper>

```

```

swiper {
  width: 100%;
  /* height: calc(100vw * 352 / 1125); */
  height: 31.28vw;
}
image {
  width: 100%;
}

```

navigator组件

```

<!--
  导航组件 navigator
  0 块级元素 默认会换行 可以直接加宽度和高度
  1 url 要跳转的页面路径 绝对路径 相对路径
  2 target 要跳转到当前的小程序 还是其他的小程序的页面
    self 默认值 自己 小程序的页面
    miniProgram 其他的小程序的页面
  3 open-type 跳转的方式
    1 navigate 默认值 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
    2 redirect 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。
    3 switchTab 跳转到 tabBar 页面，并关闭其他所有非 tabBar 页面
    4 reLaunch 关闭所有页面，打开到应用内的某个页面
-->
<navigator url="/pages/demo10/demo10"> 轮播图页面 </navigator>
<navigator url="/pages/index/index"> 直接跳转到 tabBar页面 </navigator>
<navigator open-type="redirect" url="/pages/demo10/demo10"> 轮播图页面 redirect </navigator>
<navigator open-type="switchTab" url="/pages/index/index"> switchTab直接跳转到 tabBar页面
</navigator>
<navigator open-type="reLaunch" url="/pages/index/index"> reLaunch 可以随便跳 </navigator>

```

rich-text组件

```

<!--
  rich-text 富文本标签
  1 nodes属性来实现
    1 接收标签字符串    2 接收对象数组
-->
<rich-text nodes="{html}"></rich-text>

```

js

```

Page({
  data: {
    // 1 标签字符串 最常用的
    // html: '<div><h1>hello</h1></div>'
    // 2 对象数组
    html: [
      {
        // 1 div标签 name属性来指定
        name: "div",
        // 2 标签上有哪些属性
        attrs: {
          // 标签上的属性 class style
          class: "my_div",
          style: "color:red;"
        },
        // 3 子节点 children 要接收的数据类型和 nodes第二种渲染方式的数据类型一致
        children: [
          {
            name: "p",
            attrs: {},
            // 放文本
            children: [
              {
                type: "text",
                text: "hello"
              }
            ]
          }
        ]
      }
    ]
  }
})

```

button组件

```

<!--
  1 外观的属性
    1 size 控制按钮的大小
      1) default 默认大小    2) mini 小尺寸
    2 type 用来控制按钮的颜色
      1) default 灰色    2) primary 绿色    3) warn 红色
    3 plain 按钮是否镂空，背景色透明

```

4 loading 文字前显示正在等待图标

```
-->
<button>默认按钮</button>
<button size="mini"> mini 默认按钮</button>
<button type="primary"> primary 按钮</button>
<button type="warn"> warn 按钮</button>
<button type="warn" plain> plain 按钮</button>
<button type="primary" loading> loading 按钮</button>
<!--
button 开发能力
open-type :
1 contact 直接打开 客服对话功能 需要在微信小程序的后台配置 只能够通过真机调试来打开
2 share 转发当前的小程序 到微信朋友中 不能把小程序 分享到 朋友圈
3 getPhoneNumber 获取当前用户的手机号码信息 结合一个事件来使用 不是企业的小程序账号 没有权限来获取用
户的手机号码
    1 绑定一个事件 bindgetphonenumber
    2 在事件的回调函数中 通过参数来获取信息
    3 获取到的信息 已经加密过了
        需要用户自己待见小程序的后台服务器，在后台服务器中进行解析 手机号码，返回到小程序中 就可以看到信息
了
4 getUserInfo 获取当前用户的个人信息
    1 使用方法 类似 获取用户的手机号码
    2 可以直接获取 不存在加密的字段
5 launchApp 在小程序当中 直接打开 app
    1 需要现在 app中 通过app的某个链接 打开 小程序
    2 在小程序 中 再通过 这个功能 重新打开 app
    3 找到 京东的app 和 京东的小程序
6 openSetting 打开小程序内置的 授权页面
    1 授权页面中 只会出现 用户曾经点击过的 权限
7 feedback 打开 小程序内置的 意见反馈页面
    1 只能够通过真机调试来打开
-->
<button open-type="contact">contact</button>
<button open-type="share">share</button>
<button open-type="getPhoneNumber" bindgetphonenumber="getPhoneNumber">getPhoneNumber</button>
<button open-type="getUserInfo" bindgetuserinfo="getUserInfo">getUserInfo</button>
<button open-type="launchApp">launchApp</button>
<button open-type="openSetting">openSetting</button>
<button open-type="feedback">feedback</button>
```

js

```
Page({
  // 获取用户的手机号码信息
  getPhoneNumber(e){
    console.log(e);
  },
  // 获取用户个人信息
  getUserInfo(e){
    console.log(e);
  }
})
```

icon组件

```
<!--
  小程序中的字体图标
  1 type 图标的类型
    success|success_no_circle|info|warn|waiting|cancel|download|search|clear
  2 size 大小
  3 color 图标的颜色
-->
<icon type="cancel" size="60" color="#0094ff"> </icon>
```

radio组件

```
<!--
  radio 单选框
  1 radio标签 必须要和 父元素 radio-group来使用  2 value 选中的单选框的值  3 需要给 radio-group 绑定 change事件  4 需要在页面中显示 选中的值
-->
<radio-group bindchange="handleChange">
  <radio color="red" value="male">男</radio>
  <radio color="red" value="female">女</radio>
</radio-group>
<view>您选中的是:{{gender}}</view>
```

js

```
handleChange(e){
  // 1 获取单选框中的值
  let gender=e.detail.value;
  // 2 把值 赋值给 data中的数据
  this.setData({
    // gender:gender
    gender
  })
}
})
```

checkbox组件

```
<view>
  <checkbox-group bindchange="handleItemChange">
    <checkbox value="{{item.value}}" wx:for="{{list}}" wx:key="id">
      {{item.name}}
    </checkbox>
  </checkbox-group>
  <view>
    选中的水果:{{checkedList}}
  </view>
</view>
```

js

```
<view>
  <checkbox-group bindchange="handleItemChange">
    <checkbox value="{{item.value}}" wx:for="{{list}}" wx:key="id">
      {{item.name}}
    </checkbox>
  </checkbox-group>
  <view>
    选中的水果:{{checkedList}}
  </view>
</view>
```