第3章 Chapter 3

小程序组件

小程序提供了一系列基础组件供开发者使用，通过组合这些基础组件，开发者可以迅速完成视图层的页面布局，逻辑层的数据处理及视图层和逻辑层之间的交互。根据微信官方文档提供的资料（https://developers.weixin.qq.com/miniprogram/dev/component/），小程序组件大致可分为视图容器组件、基础内容组件、表单组件、导航组件、媒体组件、地图组件、画布组件和其他组件。

本章首先简要介绍小程序组件的概念和分类，然后再依次介绍上述组件。

本章学习目标

- 了解小程序组件的概念及分类。
- 了解视图容器组件，熟练掌握 view、scroll-view 和 swiper 的用法。
- 了解基础内容组件，掌握 icon、progress 和 text 的用法。
- 了解表单组件，熟练掌握 label、button 和 radio 的用法。
- 了解媒体组件，掌握 audio、image 和 text 的用法。
- 了解导航、地图、画布和其他组件。

3.1 组件的概念和分类

1 组件的概念

在上一章视图层的内容中，已经提及小程序组件。作为视图层的基本组成单元，组件在小程序的开发中可谓必不可少。组件通常起于开始标签，止于结束标签。在两个标签之间可以包括该组件的属性和内容，其语法格式如图 3-1 所示。

```
<标签 属性 = "值">//组件的开始标签
内容              //组件的内容
<标签>            //组件的结束标签
```

图 3-1 组件的格式

组件的属性可以为布尔值、字符串、数组和数字等。通常把组件都具有的属性称为公共属性，如属性的唯一标识 id，属性的样式 class 和内联样式 style 等。官方文档提供的所有组件的公共属性如表 3-1 所示。

表 3-1 组件的公共属性

属性名	类 型	描 述	说 明
id	String	组件的唯一标识	保持整个页面唯一
class	String	组件的样式类	在对应的 WXSS 中定义的样式类
style	String	组件的内联样式	可以动态设置的内联样式
hidden	Boolean	组件是否显示	所有组件默认显示
data— *	Any	自定义属性	组件上触发相应的事件时，会发送给事件处理函数
bind * /catch *	EventHandler	组件的事件	详见事件

除了这些公共属性以外，有些组件还有自定义的特殊属性，本章接下来将会介绍这些组件的特殊属性。

2 组件的分类

根据官方文档提供的资料，可将组件按功能进行大致分类，具体如表 3-2 所示。

表 3-2 组件的分类

序号	组件的类别	包含的组件	说 明
1	视图容器组件	view scroll-view swiper swiper-item movable-area movable-view cover-view cover-image	
2	基础内容组件	icon text rich-text progress	
3	表单组件	label 和 button radio 和 radio-group checkbox 和 checkbox-group input、textarea 和 editor picker、picker-view 和 picker-view-column slider switch form	input 在 focus 时表现为原生组件(native-component)
4	导航组件	functional-page-navigator navigator	

续表

序号	组件的类别	包含的组件	说　明
5	媒体组件	audio image video camera live-player live-pusher	camera、video、live-player、live-pusher 是由客户端创建的原生组件
6	地图组件	map	由客户端创建的原生组件
7	画布组件	canvas	由客户端创建的原生组件
8	其他组件	ad official-account open-data web-view	

如表中说明所示，小程序中的部分组件是由客户端创建的原生组件。由于原生组件脱离在 WebView 渲染流程外，因此在使用时有以下限制：

（1）原生组件的层级是最高的，所以页面中的其他组件无论设置 z-index 为多少，都无法覆盖在原生组件上。后插入的原生组件可以覆盖之前的原生组件。

（2）原生组件还无法在 picker-view 中使用。

基础库 2.4.4 以下版本，原生组件不支持在 scroll-view、swiper、movable-view 中使用。

（3）部分 CSS 样式无法应用于原生组件，例如：

① 无法对原生组件设置 CSS 动画；

② 无法定义原生组件为 position：fixed；

③ 不能在父级节点使用 overflow：hidden 来裁剪原生组件的显示区域。

（4）原生组件的事件监听不能使用 bind:eventname 的写法，只支持 bindeventname。原生组件也不支持 catch 和 capture 的事件绑定方式。

（5）原生组件会遮挡 vConsole 弹出的调试面板。

原生组件是用 Web 组件模拟的，因此很多情况并不能很好地还原真机的表现，建议开发者在使用到原生组件时尽量在真机上进行调试。

对于原生组件的层级问题，小程序做了以下处理：

（1）为了解决原生组件层级最高的限制，小程序专门提供了 cover-view 和 cover-image 组件，它们可以覆盖在部分原生组件上面。这两个组件也是原生组件，但是使用限制与其他原生组件有所不同。

（2）为了解决原生组件的层级问题，引入了同层渲染。在支持同层渲染后，原生组件与其他组件可以随意叠加，有关层级的限制将不再存在。但需要注意的是，组件内部仍由原生渲染，样式一般还是对原生组件内部无效。当前 video、map 组件已支持同层渲染。

（3）为了可以调整原生组件之间的相对层级位置，小程序在 v2.7.0 及以上版本支持在样式中声明 z-index 来指定原生组件的层级。该 z-index 仅调整原生组件之间的层级顺序，其层级仍高于其他非原生组件。

3.2 视图容器组件

主要的视图容器组件如表 3-3 所示。

表 3-3 视图容器组件

序号	组件名称	说明
1	view	视图容器
2	scroll-view	可滚动视图区域
3	swiper	滑块视图容器
4	movable-view	可移动的视图容器
5	cover-view	覆盖在原生组件之上的文本视图

3.2.1 view

view 是静态的视图容器，其语法格式如图 3-2 所示。

```
<view>

</view>
```

图 3-2 view 组件的格式

view 的属性如表 3-4 所示。

表 3-4 view 组件的属性

序号	属性	说明
1	hover-class	指定按下去的样式类。当 hover-class="none"时，没有点击态效果。该属性类型是 string，默认值为 none
2	hover-stop-propagation	指定是否阻止本节点的祖先节点出现点击态。该属性类型是 boolean，默认值为 false
3	hover-start-time	按住后多久出现点击态，单位毫秒。该属性类型是 number，默认值为 50
4	hover-stay-time	手指松开后点击态保留时间，单位毫秒。该属性类型是 number，默认值为 400

注意：如果需要使用滚动视图，请使用 scroll-view。

如图 3-3 所示为 view 组件的示例代码。

```
<view class="section">
  <view class="section__title">flex-direction: row</view>
  <view class="flex-wrp" style="flex-direction:row;">
    <view class="flex-item bc_green">1</view>
```

图 3-3 view 组件的示例代码

```
      <view class = "flex - item bc_red"> 2 </view >
      <view class = "flex - item bc_blue"> 3 </view >
    </view >
  </view >
  <view class = "section">
    <view class = "section__title"> flex - direction: column </view >
    <view class = "flex - wrp" style = "height: 300px;flex - direction:column;">
      <view class = "flex - item bc_green"> 1 </view >
      <view class = "flex - item bc_red"> 2 </view >
      <view class = "flex - item bc_blue"> 3 </view >
    </view >
  </view >
```

图 3-3 view 组件的示例代码(续)

该代码包括若干个 view 组件,两个 class 值为 section 的 view 组件是并列关系,在第一个 view 中,class 为 section__title 的 view 和 class 为 flex-wrp 的 view 是并列的,它们都是内嵌在 class 为 section 的 view 中,与其为父子关系,而 class 为 flex-item bc_green、flex-item bc_red 和 flex-item bc_blue 的三个 view 与 class 为 flex-wrp 的 view 也是父子嵌套关系。图 3-4 显示了 view 组件的示例代码中 view 的关系。

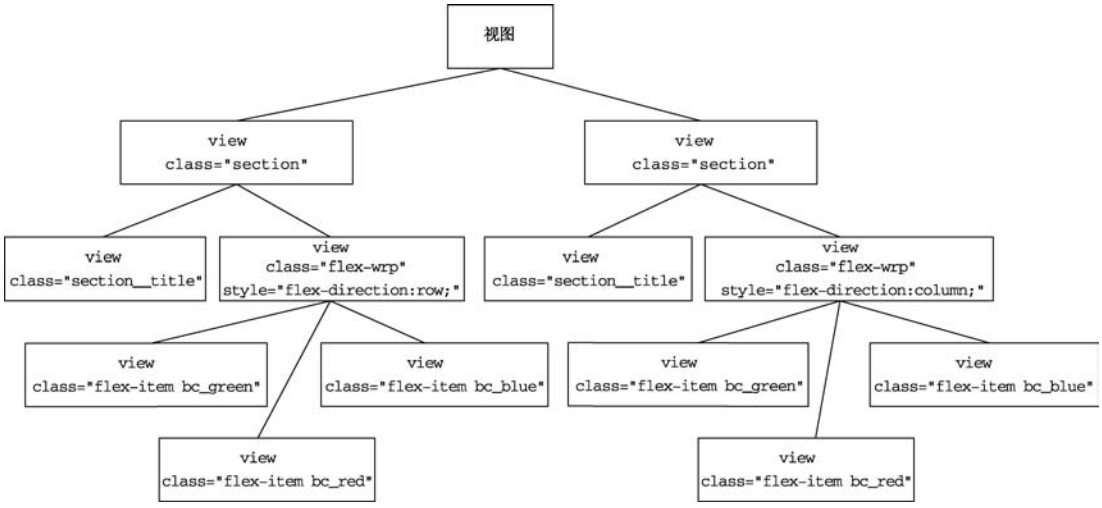

图 3-4 示例代码中 view 的关系

关于 view 组件的更多内容,可访问以下链接。

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

注意:要运行 view 的示例代码,请在上述链接中"示例代码"下方单击"在开发者工具中预览效果"超链接,或者在微信开发者工具中直接打开教材配套的本章代码运行。

3.2.2 scroll-view

scroll-view 是可滚动的视图区域，其语法格式如图 3-5 所示。

```
<scroll-view>

</scroll-view>
```

图 3-5 scroll-view 组件的格式

scroll-view 的属性如表 3-5 所示。

表 3-5 scroll-view 组件的属性

序号	属性	说明
1	scroll-x	允许横向滚动。该属性类型是 boolean，默认值为 false
2	scroll-y	允许纵向滚动。该属性类型是 boolean，默认值为 false
3	upper-threshold	距顶部/左边多远时，触发 scrolltoupper 事件。该属性类型是 number/string，默认值为 50
4	lower-threshold	距底部/右边多远时，触发 scrolltolower 事件。该属性类型是 number/string，默认值为 50
5	scroll-top	设置竖向滚动条位置。该属性类型是 number/string
6	scroll-left	设置横向滚动条位置。该属性类型是 number/string
7	scroll-into-view	值应为某子元素 id(id 不能以数字开头)。设置哪个方向可滚动，则在哪个方向滚动到该元素。该属性类型是 string
8	scroll-with-animation	在设置滚动条位置时使用动画过渡。该属性类型是 boolean，默认值为 false
9	enable-back-to-top	iOS 点击顶部状态栏、Android 双击标题栏时，滚动条返回顶部，只支持竖向。该属性类型是 boolean，默认值为 false
10	enable-flex	启用 flexbox 布局。开启后，当前节点声明了 display: flex 就会成为 flex container，并作用于其子节点。该属性类型是 boolean，默认值为 false
11	scroll-anchoring	开启 scroll anchoring 特性，即控制滚动位置不随内容变化而抖动，仅在 iOS 下生效，Android 下可参考 CSS overflow-anchor 属性。该属性类型是 boolean，默认值为 false
12	bindscrolltoupper	滚动到顶部/左边时触发。该属性类型是 eventhandle
13	bindscrolltolower	滚动到底部/右边时触发。该属性类型是 eventhandle
14	bindscroll	滚动时触发，event.detail = {scrollLeft, scrollTop, scrollHeight, scrollWidth, deltaX, deltaY}。该属性类型是 eventhandle

注意：

(1) 上述属性都不是必须填写的。

(2) 使用竖向滚动时，需要给 scroll-view 一个固定高度，通过 WXSS 设置 height。组件属性的长度单位默认为 px，从 2.4.0 起支持传入单位(rpx/px)。

(3) 基础库 2.4.0 以下不支持嵌套 textarea、map、canvas、video 组件。

(4) scroll-into-view 的优先级高于 scroll-top。

(5) 在滚动 scroll-view 时会阻止页面回弹，所以在 scroll-view 中滚动，是无法触发 onPullDownRefresh。

(6) 若要使用下拉刷新，请使用页面的滚动，而不是 scroll-view，这样也能通过点击顶部状态栏回到页面顶部。

如图 3-6 所示为 scroll-view 组件的示例代码。

```
//视图层
<view class="section">
  <view class="section__title">vertical scroll</view>
  <scroll-view scroll-y style="height: 100px;" bindscrolltoupper="upper" bindscrolltolower="lower" bindscroll="scroll" scroll-into-view="{{toView}}" scroll-top="{{scrollTop}}">
    <view id="green" class="scroll-view-item bc_green"></view>
    <view id="red" class="scroll-view-item bc_red"></view>
    <view id="yellow" class="scroll-view-item bc_yellow"></view>
    <view id="blue" class="scroll-view-item bc_blue"></view>
  </scroll-view>
  <view class="btn-area">
    <button size="mini" bindtap="tap">click me to scroll into view</button>
    <button size="mini" bindtap="tapMove">click me to scroll</button>
  </view>
</view>

<view class="section section_gap">
  <view class="section__title">horizontal scroll</view>
  <scroll-view class="scroll-view_H" scroll-x style="width: 100%">
    <view id="green" class="scroll-view-item_H bc_green"></view>
    <view id="red" class="scroll-view-item_H bc_red"></view>
    <view id="yellow" class="scroll-view-item_H bc_yellow"></view>
    <view id="blue" class="scroll-view-item_H bc_blue"></view>
  </scroll-view>
</view>

//逻辑层
var order = ['red', 'yellow', 'blue', 'green', 'red']
Page({
  data: {
    toView: 'red',
    scrollTop: 100
  },
  upper: function(e) {
    console.log(e)
  },
  lower: function(e) {
    console.log(e)
  },
  scroll: function(e) {
    console.log(e)
  },
  tap: function(e) {
    for (var i = 0; i < order.length; ++i) {
      if (order[i] === this.data.toView) {
        this.setData({
          toView: order[i + 1]
```

图 3-6　scroll-view 组件的示例代码

```
            })
            break
          }
        }
      },
      tapMove: function(e) {
        this.setData({
          scrollTop: this.data.scrollTop + 10
        })
      }
    })
```

图 3-6 scroll-view 组件的示例代码(续)

图 3-7 显示了 scroll-view 组件的示例代码中 view 与 scroll-view 的关系。

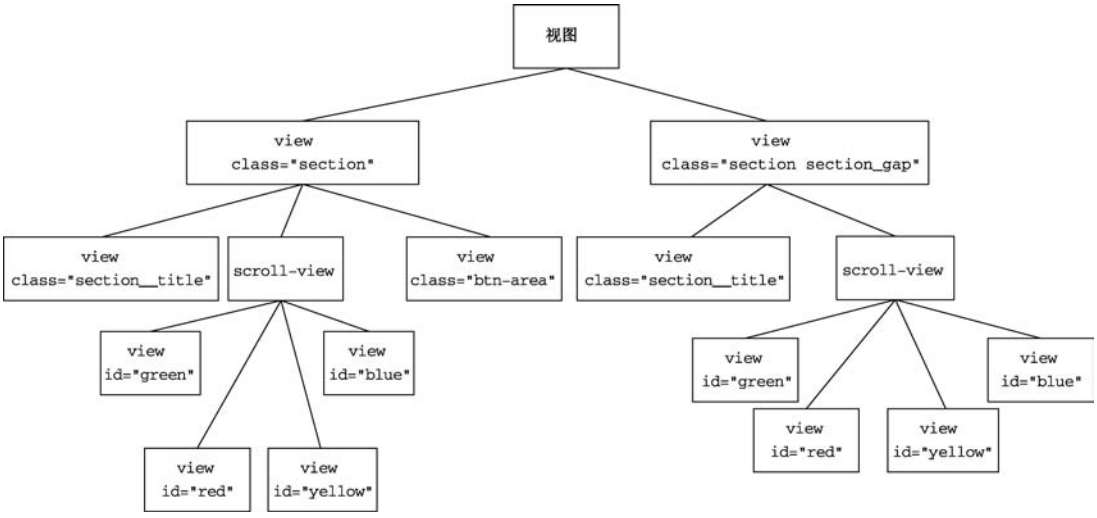

图 3-7 示例代码中 view 与 scroll-view 的关系

若点击 click me to scroll into view 的按钮,将触发其绑定的 tap 事件,即 scroll-view 内的四个 view 循环显示,这四个 view 的颜色分别对应为 green(绿色)、red(红色)、yellow(黄色)和 blue(蓝色)四种颜色。这一变化是通过视图层代码 scroll-into-view="{{toView}}"来实现的,toView 的值在 tap 事件中动态变化,刚好对应 scroll-view 中四个子元素(即 view)的 id(即"green"、"red"、"yellow"和"blue")。

从代码中可以看到,由于 toView: 'red',所以默认为红色。点击 click me to scroll into view 的按钮时,颜色依次变为(黄色)、blue(蓝色)和 green(绿色)。在 scroll-view 中设置了 scroll-y 属性(即允许纵向滚动),因此 view 是纵向滚动的。若将 style="height: 100px;"修改为 style="height: 150px;",可以看到更为明显的滚动效果。

有兴趣的读者可以运行本章配套的代码,直观感受 scroll-view 的用法。

关于 scroll-view 组件的更多内容,可访问以下链接。

https://developers.weixin.qq.com/miniprogram/dev/component/scroll-view.html

3.2.3 swiper 和 swiper-item

swiper 是滑块视图容器，其中只可放置 swiper-item 组件，否则会导致未定义的行为。它们的语法格式如图 3-8 所示。

```
<swiper>
    <swiper-item>

    </swiper-item>
</swiper>
```

图 3-8 swiper 和 swiper-item 组件的格式

swiper 的属性如表 3-6 所示。

表 3-6 swiper 组件的属性

序号	属性	说明
1	indicator-dots	是否显示面板指示点。该属性类型是 boolean，默认值为 false
2	indicator-color	指示点颜色。该属性类型是 color，默认值为 rgba(0，0，0，.3)
3	indicator-active-color	当前选中的指示点颜色。该属性类型是 color，默认值为 #000000
4	autoplay	是否自动切换。该属性类型是 boolean，默认值为 false
5	current	当前所在滑块的 index。该属性类型是 number，默认值为 0
6	interval	自动切换时间间隔。该属性类型是 number，默认值为 5000
7	duration	滑动动画时长。该属性类型是 number，默认值为 500
8	circular	是否采用衔接滑动。该属性类型是 boolean，默认值为 false
9	vertical	滑动方向是否为纵向。该属性类型是 boolean，默认值为 false
10	previous-margin	前边距，可用于露出前一项的一小部分，接收 px 和 rpx 值。该属性类型是 string，默认值为"0px"
11	next-margin	后边距，可用于露出后一项的一小部分，接受 px 和 rpx 值。该属性类型是 string，默认值为"0px"
12	display-multiple-items	同时显示的滑块数量。该属性类型是 number，默认值为 1
13	skip-hidden-item-layout	是否跳过未显示的滑块布局，设为 true 可优化复杂情况下的滑动性能，但会丢失隐藏状态滑块的布局信息。该属性类型是 boolean，默认值为 false
14	easing-function	指定 swiper 切换缓动动画类型。该属性类型是 string，默认值为 "default"
15	bindchange	current 改变时会触发 change 事件，event.detail={current，source}。该属性类型是 eventhandle
16	bindtransition	swiper-item 的位置发生改变时会触发 transition 事件，event.detail ={dx：dx，dy：dy}。该属性类型是 eventhandle
17	bindanimationfinish	动画结束时会触发 animationfinish 事件，event.detail 同上。该属性类型是 eventhandle

其中 easing-function 的合法值如表 3-7 所示。

表 3-7 easing-function 的合法值

序号	值	说 明
1	default	默认缓动函数
2	linear	线性动画
3	easeInCubic	缓入动画
4	easeOutCubic	缓出动画
5	easeInOutCubic	缓入缓出动画

另外，swiper 组件从 1.4.0 开始，change 事件增加了 source 字段（用于表示导致变更的原因），其可能的值如下：

（1）autoplay 自动播放导致 swiper 变化；

（2）touch 用户滑动引起 swiper 变化；

（3）其他原因将用空字符串表示。

如果在 bindchange 的事件回调函数中使用 setData 改变 current 值，则有可能导致 setData 被不停地调用，因而通常情况下请在改变 current 值前先检测 source 字段来判断其值变化是否是由于用户触摸引起。

图 3-9 为 swiper 和 swiper-item 组件的示例代码。

```
//视图层
<swiper indicator-dots="{{indicatorDots}}"
  autoplay="{{autoplay}}" interval="{{interval}}" duration="{{duration}}">
  <block wx:for="{{imgUrls}}">
    <swiper-item>
      <image src="{{item}}" class="slide-image" width="355" height="150"/>
    </swiper-item>
  </block>
</swiper>
<button bindtap="changeIndicatorDots"> indicator-dots </button>
<button bindtap="changeAutoplay"> autoplay </button>
<slider bindchange="intervalChange" show-value min="500" max="2000"/>
<slider bindchange="durationChange" show-value min="1000" max="10000"/>

//逻辑层
Page({
  data: {
    imgUrls: [
      'https://images.unsplash.com/photo-1551334787-21e6bd3ab135?w=640',
      'https://images.unsplash.com/photo-1551214012-84f95e060dee?w=640',
      'https://images.unsplash.com/photo-1551446591-142875a901a1?w=640'
    ],
    indicatorDots: false,
    autoplay: false,
    interval: 5000,
```

图 3-9 swiper 和 swiper-item 组件的示例代码

```
    duration: 1000
  },
  changeIndicatorDots: function(e) {
    this.setData({
      indicatorDots: !this.data.indicatorDots
    })
  },
  changeAutoplay: function(e) {
    this.setData({
      autoplay: !this.data.autoplay
    })
  },
  intervalChange: function(e) {
    this.setData({
      interval: e.detail.value
    })
  },
  durationChange: function(e) {
    this.setData({
      duration: e.detail.value
    })
  }
})
```

图 3-9　swiper 和 swiper-item 组件的示例代码(续)

示例代码中 swiper 组件的 indicator-dots、autoplay、interval 和 duration 属性均从逻辑层中获取。按钮 indicator-dots 和 autoplay 分别绑定 changeIndicatorDots 和 changeAutoplay 事件。两个 slider 分别对应 intervalChange 和 durationChange，用于改变 interval 和 duration 的值，其中 interval 的值从 500 变化到 2000，duration 的值从 1000 变化到 10 000。

swiper-item 从基础库 1.0.0 开始支持，低版本需做兼容处理，它仅可放置在 swiper 组件中，宽高自动设置为 100%，它的属性如表 3-8 所示。

表 3-8　swiper-item 组件的属性

序号	属性	说　明
1	item-id	swiper-item 的标识符。该属性类型是 string

关于 swiper 和 swiper-item 组件的更多内容，可访问以下链接。

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

https://developers.weixin.qq.com/miniprogram/dev/component/swiper-item.html

注意：若读者在上述链接对应的 swiper.html 中点击“在开发者工具中预览效果”，代码与图 3-9 不完全一样，请参见本章配套的名为“ex030203_swiper_在开发者工具中预览效果”文件夹下的代码。图 3-9 中的代码存放在本章代码的文件夹下。

3.2.4 movable-area 和 movable-view

movable-view 是可移动的视图容器，在页面中可以拖曳滑动。movable-view 必须在 movable-area 组件中，并且必须是直接子节点，否则不能移动，它们的语法格式如图 3-10 所示。

```
<movable-area>
    <movable-view>
    </movable-view>
</movable-area>
```

图 3-10 movable-area 和 movable-view 组件的格式

movable-view 的属性如表 3-9 所示。

表 3-9 movable-view 组件的属性

序号	属 性	说 明
1	direction	movable-view 的移动方向，属性值有 all、vertical、horizontal、none。该属性类型是 string，默认值为 none
2	inertia	movable-view 是否带有惯性。该属性类型是 boolean，默认值为 false
3	out-of-bounds	超过可移动区域后，movable-view 是否还可以移动。该属性类型是 boolean，默认值为 false
4	x	定义 x 轴方向的偏移，如果 x 的值不在可移动范围内，会自动移动到可移动范围；改变 x 的值会触发动画。该属性类型是 number
5	y	定义 y 轴方向的偏移，如果 y 的值不在可移动范围内，会自动移动到可移动范围；改变 y 的值会触发动画。该属性类型是 number
6	damping	阻尼系数，用于控制 x 或 y 改变时的动画和过界回弹的动画，值越大移动越快。该属性类型是 number，默认值为 20
7	friction	摩擦系数，用于控制惯性滑动的动画，值越大摩擦力越大，滑动越快停止；必须大于 0，否则会被设置成默认值。该属性类型是 number，默认值为 2
8	disabled	是否禁用。该属性类型是 boolean，默认值为 false
9	scale	是否支持双指缩放，默认缩放手势生效区域是在 movable-view 内。该属性类型是 boolean，默认值为 false
10	scale-min	定义缩放倍数最小值。该属性类型是 number，默认值为 0.5
11	scale-max	定义缩放倍数最大值。该属性类型是 number，默认值为 10
12	scale-value	定义缩放倍数，取值范围为 0.5～10
13	animation	是否使用动画。该属性类型是 boolean，默认值为 true
14	bindchange	拖动过程中触发的事件，event.detail = {x, y, source}。该属性类型是 eventhandle
15	bindscale	缩放过程中触发的事件，event.detail = {x, y, scale}，x 和 y 字段在 2.1.0 之后支持。该属性类型是 eventhandle
16	htouchmove	初次手指触摸后移动为横向的移动时触发，如果 catch 此事件，则意味着 touchmove 事件也被 catch。该属性类型是 eventhandle
17	vtouchmove	初次手指触摸后移动为纵向的移动时触发，如果 catch 此事件，则意味着 touchmove 事件也被 catch。该属性类型是 eventhandle

其中 bindchange 事件返回的 source 表示产生移动的原因，具体如表 3-10 所示。

表 3-10 source 的值和说明

序号	值	说明
1	touch	拖动
2	touch-out-of-bounds	超出移动范围
3	out-of-bounds	超出移动范围后的回弹
4	friction	惯性
5	空字符串	setData

注意：movable-view 默认为绝对定位，top 和 left 属性为 0px，它的 width 和 height 属性默认为 10px。

movable-area 是 movable-view 的可移动区域，从基础库 1.2.0 开始支持，低版本需做兼容处理。它的属性如表 3-11 所示。

表 3-11 source 的值和说明

序号	属性	说明
1	scale-area	当里面的 movable-view 设置为支持双指缩放时，设置此值可将缩放手势生效区域修改为整个 movable-area。该属性类型是 boolean，默认值为 false

注意：

(1) movable-area 必须设置 width 和 height 属性，不设置默认为 10px。

(2) 当 movable-view 小于 movable-area 时，movable-view 的移动范围是在 movable-area 内。

(3) 当 movable-view 大于 movable-area 时，movable-view 的移动范围必须包含 movable-area(x 轴方向和 y 轴方向分开考虑)。

图 3-11 为 movable-area 和 movable-view 组件的示例代码。

```
//视图层
<view class="section">
  <view class="section__title">movable-view 区域小于 movable-area</view>
  <movable-area style="height: 200px; width: 200px; background: red;">
    <movable-view style="height: 50px; width: 50px; background: blue;" x="{{x}}" y="{{y}}" direction="all">
    </movable-view>
  </movable-area>
  <view class="btn-area">
    <button size="mini" bindtap="tap">click me to move to (30px, 30px)</button>
  </view>
  <view class="section__title">movable-view 区域大于 movable-area</view>
```

图 3-11 movable-area 和 movable-view 组件的示例代码

```
  <movable-area style="height: 100px; width: 100px; background: red;">
    <movable-view style="height: 200px; width: 200px; background: blue;" direction="all">
    </movable-view>
  </movable-area>
  <view class="section__title">可放缩</view>
  <movable-area style="height: 200px; width: 200px; background: red;" scale-area>
    <movable-view style="height: 50px; width: 50px; background: blue;" direction="all"
bindchange="onChange" bindscale="onScale" scale scale-min="0.5" scale-max="4" scale-
value="2">
    </movable-view>
  </movable-area>
</view>

//逻辑层
Page({
  data: {
    x: 0,
    y: 0
  },
  tap: function(e) {
    this.setData({
      x: 30,
      y: 30
    });
  },
  onChange: function(e) {
    console.log(e.detail)
  },
  onScale: function(e) {
    console.log(e.detail)
  }
})
```

图 3-11 movable-area 和 movable-view 组件的示例代码(续)

在示例代码中,当 movable-view 区域小于 movable-area 时,通过在按钮上绑定 tap 事件,可以把 movable-view 移动到 movable-area 中位置(30px, 30px)处;当 movable-view 区域大于 movable-area 时,movable-area 被 movable-view 完全覆盖;将 movable-area 的 scale-area 属性设置为真,并设置了 movable-view 的 scale 为真,scale-min 和 scale-max 分别设置为 0.5 和 4,scale-value 设置为 2,此时 movable-view 实际高度(height)为 height * scale-value=50px * 2=100px,高度(width)为 width * scale-value=50px * 2=100px。若将 scale-value 设置为 4,则 movable-view 的实际宽高将与 movable-area 一致。

关于 movable-area 和 movable-view 组件的更多内容,可访问以下链接。

https://developers.weixin.qq.com/miniprogram/dev/component/movable-area.html

https://developers.weixin.qq.com/miniprogram/dev/component/movable-view.html

注意:若读者在上述链接对应的 movable-area.html 中点击“在开发者工具中预览效果”,代码与图 3-11 不完全一样,请参见教材配套的本章代码(文件夹名为“ex030204_movable-area_在开发者工具中预览效果”)。图 3-11 中的代码存放在本章代码的文件夹下。

3.2.5 cover-view 和 cover-image

cover-view 是覆盖在原生组件之上的文本视图，可覆盖的原生组件包括 map、video、canvas、camera、live-player、live-pusher。该组件属性的长度单位默认为 px，从基础库 2.4.0 起支持传入单位(rpx/px)。

cover-view 只支持嵌套 cover-view、cover-image，在 cover-view 中可使用 button。它的语法格式如图 3-12 所示。

```
<cover-view>

</cover-view>
```

图 3-12 cover-view 组件的格式

cover-view 的属性如表 3-12 所示。

表 3-12 cover-view 组件的属性

序号	属　性	说　明
1	scroll-top	用于设置顶部滚动偏移量，仅在设置了 overflow-y：scroll 成为滚动元素后生效。该属性类型为 number/string

使用 cover-view 组件时，请注意：

(1) 从基础库 1.6.0 起，cover-view 支持 css opacity 和 css transition 动画，transition-property 只支持 transform (translateX，translateY)与 opacity。

(2) 从基础库 1.9.90 起，cover-view 支持 overflow：scroll，但不支持动态更新 overflow；最外层 cover-view 支持 position：fixed；支持插在 view 等标签下。在此之前只可嵌套在原生组件 map、video、canvas、camera 内，避免嵌套在其他组件内。

(3) 从基础库 2.1.0 起支持设置 scale rotate 的 css 样式，包括 transition 动画。

(4) 从基础库 2.2.4 起支持 touch 相关事件，也可使用 hover-class 设置点击态。

(5) cover-view 和 cover-image 的 aria-role 仅可设置为 button，读屏模式下才可以点击，并朗读出“按钮”；为空时可以聚焦，但不可点击。

(6) 事件模型遵循冒泡模型，但不会冒泡到原生组件。

(7) 文本建议都套上 cover-view 标签，避免排版错误。

(8) 只支持基本的定位、布局、文本样式。不支持设置单边的 border、background-image、shadow、overflow：visible 等。

(9) 建议子节点不要溢出父节点。

(10) 支持使用 z-index 控制层级。

(11) 默认设置的样式有：white-space：nowrap；line-height：1.2；display：block。

注意：自定义组件嵌套 cover-view 时，自定义组件的 slot 及其父节点暂不支持通过 wx:if 控制显隐，否则会导致 cover-view 不显示。

图 3-13 为 cover-view 和 cover-image 组件的示例代码。

```
//index.wxml
<video id="myVideo" src="http://wxsnsdy.tc.qq.com/105/20210/snsdyvideodownload?filekey=
30280201010421301f0201690402534804102ca905ce620b1241b726bc41dcff44e00204012882540400&bizid=
1023&hy=SH&fileparam=302c020101042530230204136ffd93020457e3c4ff02024ef202031e8d7f02030-
f42400204045a320a0201000400" controls="{{false}}" event-model="bubble">
  <cover-view class="controls">
    <cover-view class="play" bindtap="play">
      <cover-image class="img" src="/path/to/icon_play.jpg" />
    </cover-view>
    <cover-view class="pause" bindtap="pause">
      <cover-image class="img" src="/path/to/icon_pause.jpg" />
    </cover-view>
    <cover-view class="time">00:00</cover-view>
  </cover-view>
</video>

//index.wxss
.controls {
  position: relative;
  top: 50%;
  height: 50px;
  margin-top: -25px;
  display: flex;
}
.play,.pause,.time {
  flex: 1;
  height: 100%;
}
.time {
  text-align: center;
  background-color: rgba(0, 0, 0, .5);
  color: white;
  line-height: 50px;
}
.img {
  width: 40px;
  height: 40px;
  margin: 5px auto;
}

//index.js
Page({
  onReady() {
    this.videoCtx = wx.createVideoContext('myVideo')
  },
  play() {
    this.videoCtx.play()
  },
  pause() {
    this.videoCtx.pause()
  }
})
```

图 3-13 cover-view 和 cover-image 组件的示例代码

图 3-14 显示了示例代码中 video、cover-view 与 cover-image 的关系。原生组件 video 在最下面，cover-view 组件覆盖在 video 组件之上，在这个 cover-view 组件中放置了三个 cover-view 组件，其中前两个 cover-view 组件分别用于放置 cover-image 组件，在这两个 cover-image 组件中分别放置了播放和暂停的图片。

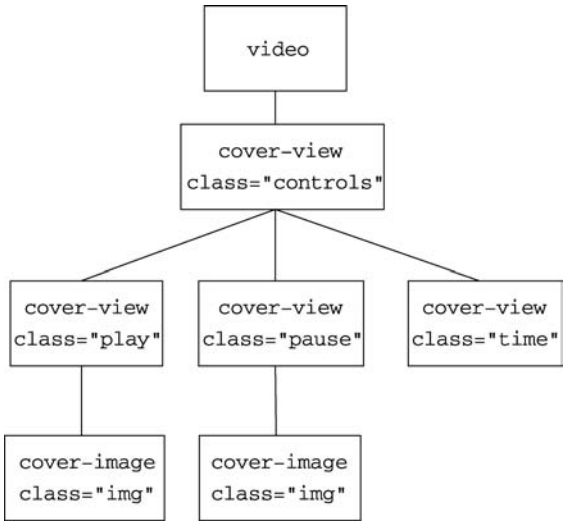

图 3-14 示例代码中控件间的关系

cover-image 是覆盖在原生组件之上的图片视图组件。它可以覆盖的原生组件同 cover-view 一样，还支持嵌套在 cover-view 里。基础库从 1.4.0 开始支持，低版本需做兼容处理。它的语法格式如图 3-15 所示。

```
<cover-image>

</cover-image>
```

图 3-15 cover-image 组件的格式

cover-image 的属性如表 3-13 所示。

表 3-13 cover-image 组件的属性

序号	属　性	说　明
1	src	图标路径，支持临时路径、网络地址(1.6.0 起支持)、云文件 ID(2.2.3 起支持)。暂不支持 base64 格式。该属性的类型为 string
2	bindload	图片加载成功时触发。该属性的类型为 eventhandle
3	binderror	图片加载失败时触发。该属性的类型为 eventhandle

关于 cover-view 和 cover-image 组件的更多内容，可访问以下链接。

https://developers.weixin.qq.com/miniprogram/dev/component/cover-image.html

https://developers.weixin.qq.com/miniprogram/dev/component/cover-image.html

3.3 基础内容组件

3.3.1 icon

icon 是图标组件，其语法格式如图 3-16 所示。

```
<icon>

</icon>
或
<icon      />
```

图 3-16 icon 组件的格式

icon 的属性如表 3-14 所示。

表 3-14 icon 组件的属性

序号	属 性	说 明
1	type	icon 的类型，有效值：success、success_no_circle、info、warn、waiting、cancel、download、search、clear。该字段是必填字段，类型为 string
2	size	icon 的大小。该字段的类型为 number/string，默认值为 23
3	color	icon 的颜色，同 css 的 color。该字段的类型为 string

如图 3-17 所示为 icon 组件的示例代码。

```
//视图层
<view class="group">
  <block wx:for="{{iconSize}}">
    <icon type="success" size="{{item}}"/>
  </block>
</view>

<view class="group">
  <block wx:for="{{iconType}}">
    <icon type="{{item}}" size="40"/>
  </block>
</view>

<view class="group">
  <block wx:for="{{iconColor}}">
    <icon type="success" size="40" color="{{item}}"/>
  </block>
</view>
//逻辑层
Page({
  data: {
```

图 3-17 icon 组件的示例代码

```
    iconSize: [20, 30, 40, 50, 60, 70],
    iconColor: [
      'red', 'orange', 'yellow', 'green', 'rgb(0,255,255)', 'blue', 'purple'
    ],
    iconType: [
      'success', 'success_no_circle', 'info', 'warn', 'waiting', 'cancel', 'download', 'search', 'clear'
    ]
  }
})
```

图 3-17　icon 组件的示例代码(续)

从代码中可以看出,视图层有三个 view,在每个 view 里都放了 block 用于展示 icon 组件的用法。第一个 view 的 block 里的 icon 类型为 success,大小从 20 到 70 依次变大;第二个 view 的 block 里的 icon 的大小为 40,类型分别为 success、success_no_circle、info、warn、waiting、cancel、download、search 和 clear;第三个 view 的 block 里的 icon 类型为 success,大小为 40,颜色依次为 red、orange、yellow、green、rgb(0,255,255)、blue 和 purple。

关于 icon 组件的更多内容,可访问以下链接。

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

读者既可以在上述链接页面内看到示例代码的效果,也可以点击“在开发者工具中预览效果”,在微信开发者工具中查看代码和模拟器上运行的效果,还可以导入教材配套的代码(在本章文件夹 ex030301_icon 中)运行之。

3.3.2　text

text 是文本组件,其语法格式如图 3-18 所示。

```
<text>

</text>
```

图 3-18　text 组件的格式

text 的属性如表 3-15 所示。

表 3-15　text 组件的属性

序号	属　性	说　明
1	selectable	文本是否可选。该字段的类型为 boolean,默认值为 false
2	space	显示连续空格。该字段的类型为 string
3	decode	是否解码。该字段的类型为 boolean,默认值为 false

其中 space 的合法值如表 3-16 所示。

表 3-16　space 的合法值

序号	属　性	说　明
1	ensp	中文字符空格一半大小
2	emsp	中文字符空格大小
3	nbsp	根据字体设置的空格大小

使用 text 组件时，请注意：

(1) 各个操作系统的空格标准不一致；

(2) decode 属性可以解析的字符有 、&lt、&gt、&amp、&apos、 和 ；

(3) text 组件内只支持 text 嵌套；

(4) 除了文本节点以外的其他节点都无法长按选中；

(5) 当基础库版本低于 2.1.0 时，text 组件内嵌的 text style 设置可能不会生效。

如图 3-19 所示为 text 组件的示例代码。

```
//视图层
<view class = "btn - area">
  <view class = "body - view">
    <text >{{text}}</text >
    <button bindtap = "add"> add line </button >
    <button bindtap = "remove"> remove line </button >
  </view >
</view >
//逻辑层
var initData = 'this is first line\nthis is second line'
var extraLine = [];
Page({
  data: {
    text: initData
  },
  add: function(e) {
    extraLine.push('other line')
    this.setData({
      text: initData + '\n' + extraLine.join('\n')
    })
  },
  remove: function(e) {
    if (extraLine.length > 0) {
      extraLine.pop()
      this.setData({
        text: initData + '\n' + extraLine.join('\n')
      })
    }
  }
})
```

图 3-19 text 组件的示例代码

在示例代码中，有两个 view 组件，一个 text 组件和两个 button 组件。其中 class＝"body-view"的 view 嵌套在 class＝"btn-area"的 view 中，text 组件和 button 组件自上而下的置于 class＝"body-view"的 view 中，这三个组件是并列关系。text 组件的内容是动态获取的，第一个 button 组件绑定了 add 事件，若单击该按钮(即 add line)将触发 add 函数，会在 text 组件中新增一行(内容为 other line)；第二个 button 组件绑定了 remove 事件，若单击该按钮(即 remove line)将触发 remove 函数，只要 text 组件中存在新增行，均会移除一行(内容为 other line)。

注意：因为变量 initData 的内容 this is first line\nthis is second line 不写入变量 extraLine 中，所以它们也不会被移除。

关于 text 组件的更多内容，可访问以下链接。

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

3.3.3 rich-text

rich-text 是富文本组件，其语法格式如图 3-20 所示。

```
<rich-text>

</rich-text>
```

图 3-20 rich-text 组件的格式

rich-text 的属性如表 3-17 所示。

表 3-17 rich-text 组件的属性

序号	属 性	说 明
1	nodes	节点列表/HTML String。该属性的类型为 array/string，其默认值为[]
2	space	显示连续空格。该属性的类型为 string

1 nodes 属性

属性 nodes 支持两种节点，它们通过 type 来区分，分别是元素节点(type = node*)和文本节点(type = text*)，其中默认是元素节点。

1) 元素节点

元素节点的属性如表 3-18 所示。

表 3-18 元素节点的属性

序号	属 性	说 明
1	name	节点的标签名。支持部分受信任的 HTML 节点。该属性的类型为 string，为必填字段
2	attrs	节点的属性。支持部分受信任的属性，遵循 Pascal 命名法。该属性的类型为 object
3	children	节点的子节点列表。结构和 nodes 一致。该属性的类型为 array

受信任的 HTML 节点及属性如表 3-19 所示。

表 3-19 受信任的 HTML 节点及属性

序号	节 点	属 性
1	a, abbr, address, article, aside, b, bdi	
2	bdo	dir
3	big, blockquote, br, caption, center, cite, code	

续表

序号	节　　点	属　　性
4	col，colgroup	span，width
5	dd，del，div，dl，dt，em，fieldset，font，footer，h1，h2，h3，h4，h5，h6，header，hr，i	
6	img	alt，src，height，width
7	ins，label，legend，li，mark，nav，	
8	ol	start，type
9	p，pre，q，rt，ruby，s，section，small，span，strong，sub，sup	
10	table	width
11	td，th，tr	colspan，height，rowspan，width
12	tbody，tfoot，thead，tt，u，ul	

表中列举的受信任的 HTML 节点，全局支持 class 和 style 属性，但不支持 id 属性。

对于 rich-text 组件及其属性，使用时还要注意以下几点：

（1）nodes 属性不推荐使用 string 类型，性能会有所下降。

（2）rich-text 组件内屏蔽所有节点的事件。

（3）attrs 属性不支持 id，支持 class。

（4）name 属性大小写不敏感。

（5）如果使用了不受信任的 HTML 节点，该节点及其所有子节点将会被移除。

（6）img 标签仅支持网络图片。

（7）如果在自定义组件中使用 rich-text 组件，那么仅自定义组件的 wxss 样式对 rich-text 中的 class 生效。

2）文本节点

文本节点的属性如表 3-20 所示。

表 3-20　文本节点的属性

属　　性	说　　明
text	节点的文本。支持部分受信任的 HTML 节点。该属性的类型为 string，为必填字段

2 space 属性

属性 space 的合法值如表 3-21 所示。

表 3-21　space 的合法值

序号	属　　性	说　　明
1	ensp	中文字符空格一半大小
2	emsp	中文字符空格大小
3	nbsp	根据字体设置的空格大小

如图 3-21 所示为 rich-text 组件的示例代码。

```
//视图层
<rich-text nodes="{{nodes}}" bindtap="tap"></rich-text>
//逻辑层
Page({
  data: {
    nodes: [{
      name: 'div',
      attrs: {
        class: 'div_class',
        style: 'line-height: 60px; color: red;'
      },
      children: [{
        type: 'text',
        text: 'Hello World!'
      }]
    }]
  },
  tap() {
    console.log('tap')
  }
})
```

图 3-21 rich-text 组件的格式

在视图层中的代码只有一个 rich-text 组件，它绑定了 tap 事件，并从逻辑层中动态获取 nodes 属性对应地数据。属性 nodes 中放置了元素节点的 name、attrs 和 children 属性，其中 children 为文本节点，text 属性为“Hello World!”，属性 attrs 的 style 为“line-height：60px；color：red；”，读者可运行该代码查看最终效果。

关于 rich-text 组件的更多内容，可访问以下链接。

https://developers.weixin.qq.com/miniprogram/dev/component/rich-text.html

注意：若读者在上述链接中“示例代码”下方点击“在开发者工具中预览效果”超链接，将会在微信开发者工具中打开与图 3-21 不完全一样的代码，具体请参见教材配套的本章代码(文件夹名为“ex030303_rich-text_在开发者工具中预览效果”)。

图 3-21 中的代码存放在本章代码的文件夹下。

3.3.4 progress

progress 是进度条组件，其语法格式如图 3-22 所示。

```
<progress>

</progress>
或
<progress        />
```

图 3-22 progress 组件的格式

如表 3-22 所示为 progress 组件的属性，其长度单位默认为 px，从基础库 2.4.0 起支持传入单位(rpx/px)。

表 3-22 progress 组件的属性

序号	属 性	说 明
1	percent	百分比 0～100。该属性的类型为 number
2	show-info	在进度条右侧显示百分比。该属性的类型为 boolean，默认值为 false
3	border-radius	圆角大小。该属性的类型为 number/string，默认值为 0
4	font-size	右侧百分比字体大小。该属性的类型为 number/string，默认值为 16
5	stroke-width	进度条线的宽度。该属性的类型为 number/string，默认值为 6
6	color	进度条颜色(请使用 activeColor)。该属性的类型为 string，默认值为 #09BB07
7	activeColor	已选择的进度条的颜色。该属性的类型为 string，默认值为 #09BB07
8	backgroundColor	未选择的进度条的颜色。该属性的类型为 string，默认值为 #EBEBEB
9	active	进度条从左往右的动画。该属性的类型为 boolean，默认值为 false
10	active-mode	backwards：动画从头播；forwards：动画从上次结束点接着播。该属性的类型为 string，默认值为 backwards
11	duration	进度增加 1%所需毫秒数。该属性的类型为 number，默认值为 30
12	bindactiveend	动画完成事件。该属性的类型为 eventhandle

如图 3-23 所示为 progress 组件的示例代码。

```
<progress percent="20" show-info />
<progress percent="40" stroke-width="12" />
<progress percent="60" color="pink" />
<progress percent="80" active />
```

图 3-23 progress 组件的示例代码

上述代码中一共有四个 progress 组件，第一个 progress 组件设置了在右侧显示百分比，第二个 progress 组件设置了进度条线宽度，第三个 progress 组件设置了进度条颜色，第四个 progress 组件设置了从左往右的动画。

关于 progress 组件的更多内容，可访问以下链接。

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

3.4 表单组件

3.4.1 label 和 button

1 label

label 用来改进表单组件的可用性，其语法格式如图 3-24 所示。

```
<label>

</label>
```

图 3-24 label 组件的格式

使用 for 属性找到对应的 id，或者将控件放在该标签下，当点击时，就会触发对应的控件。for 优先级高于内部控件，内部有多个控件的时候默认触发第一个控件。目前可以绑定的控件有：button、checkbox、radio、switch。

表 3-23 所示为 label 组件的属性。

表 3-23 label 组件的属性

属性	说　明
for	用于绑定控件的 id。该属性的数据类型是 string

图 3-25 为 label 组件的示例代码。

```
//视图层
<view class="section section_gap">
<view class="section__title">表单组件在 label 内</view>
<checkbox-group class="group" bindchange="checkboxChange">
  <view class="label-1" wx:for="{{checkboxItems}}">
    <label>
      <checkbox hidden value="{{item.name}}" checked="{{item.checked}}"></checkbox>
      <view class="label-1__icon">
        <view class="label-1__icon-checked" style="opacity:{{item.checked ? 1: 0}}"></
view>
      </view>
      <text class="label-1__text">{{item.value}}</text>
    </label>
  </view>
</checkbox-group>
</view>

<view class="section section_gap">
<view class="section__title">label 用 for 标识表单组件</view>
<radio-group class="group" bindchange="radioChange">
  <view class="label-2" wx:for="{{radioItems}}">
    <radio id="{{item.name}}" hidden value="{{item.name}}" checked="{{item.checked}}"></
radio>
    <view class="label-2__icon">
      <view class="label-2__icon-checked" style="opacity:{{item.checked ? 1: 0}}"></
view>
    </view>
    <label class="label-2__text" for="{{item.name}}"><text>{{item.name}}</text></label
>
  </view>
</radio-group>
</view>

.label-1, .label-2{
    margin-bottom: 15px;
}
```

图 3-25 label 组件的示例代码

```
.label-1__text, .label-2__text {
    display: inline-block;
    vertical-align: middle;
}

.label-1__icon {
    position: relative;
    margin-right: 10px;
    display: inline-block;
    vertical-align: middle;
    width: 18px;
    height: 18px;
    background: #fcfff4;
}

.label-1__icon-checked {
    position: absolute;
    top: 3px;
    left: 3px;
    width: 12px;
    height: 12px;
    background: #1aad19;
}


.label-2__icon {
    position: relative;
    display: inline-block;
    vertical-align: middle;
    margin-right: 10px;
    width: 18px;
    height: 18px;
    background: #fcfff4;
    border-radius: 50px;
}

.label-2__icon-checked {
    position: absolute;
    left: 3px;
    top: 3px;
    width: 12px;
    height: 12px;
    background: #1aad19;
    border-radius: 50%;
}

.label-4_text{
    text-align: center;
    margin-top: 15px;
```

图 3-25 label 组件的示例代码(续)

```
}

.label-1, .label-2{
    margin-bottom: 15px;
}
.label-1__text, .label-2__text {
    display: inline-block;
    vertical-align: middle;
}

.label-1__icon {
    position: relative;
    margin-right: 10px;
    display: inline-block;
    vertical-align: middle;
    width: 18px;
    height: 18px;
    background: #fcfff4;
}

.label-1__icon-checked {
    position: absolute;
    top: 3px;
    left: 3px;
    width: 12px;
    height: 12px;
    background: #1aad19;
}


.label-2__icon {
    position: relative;
    display: inline-block;
    vertical-align: middle;
    margin-right: 10px;
    width: 18px;
    height: 18px;
    background: #fcfff4;
    border-radius: 50px;
}

.label-2__icon-checked {
    position: absolute;
    left: 3px;
    top: 3px;
    width: 12px;
    height: 12px;
    background: #1aad19;
    border-radius: 50%;
```

图 3-25　label 组件的示例代码（续）

```
}

.label-4_text{
    text-align: center;
    margin-top: 15px;
}

//逻辑层
Page({
  data: {
    checkboxItems: [
      { name: 'USA', value: '美国' },
      { name: 'CHN', value: '中国', checked: 'true' },
      { name: 'BRA', value: '巴西' },
      { name: 'JPN', value: '日本', checked: 'true' },
      { name: 'ENG', value: '英国' },
      { name: 'TUR', value: '法国' },
    ],
    radioItems: [
      { name: 'USA', value: '美国' },
      { name: 'CHN', value: '中国', checked: 'true' },
      { name: 'BRA', value: '巴西' },
      { name: 'JPN', value: '日本' },
      { name: 'ENG', value: '英国' },
      { name: 'TUR', value: '法国' },
    ],
    hidden: false
  },
  checkboxChange: function (e) {
    var checked = e.detail.value
    var changed = {}
    for (var i = 0; i < this.data.checkboxItems.length; i++) {
      if (checked.indexOf(this.data.checkboxItems[i].name) !== -1) {
        changed['checkboxItems[' + i + '].checked'] = true
      } else {
        changed['checkboxItems[' + i + '].checked'] = false
      }
    }
    this.setData(changed)
  },
  radioChange: function (e) {
    var checked = e.detail.value
    var changed = {}
    for (var i = 0; i < this.data.radioItems.length; i++) {
      if (checked.indexOf(this.data.radioItems[i].name) !== -1) {
        changed['radioItems[' + i + '].checked'] = true
      } else {
        changed['radioItems[' + i + '].checked'] = false
      }
    }
    this.setData(changed)
  }
})
```

图 3-25 label 组件的示例代码（续）

从视图层代码中可以看到 label 控件内嵌套了 checkbox、view 和 text 等控件，层次结构比较复杂，具体如图 3-26 所示。

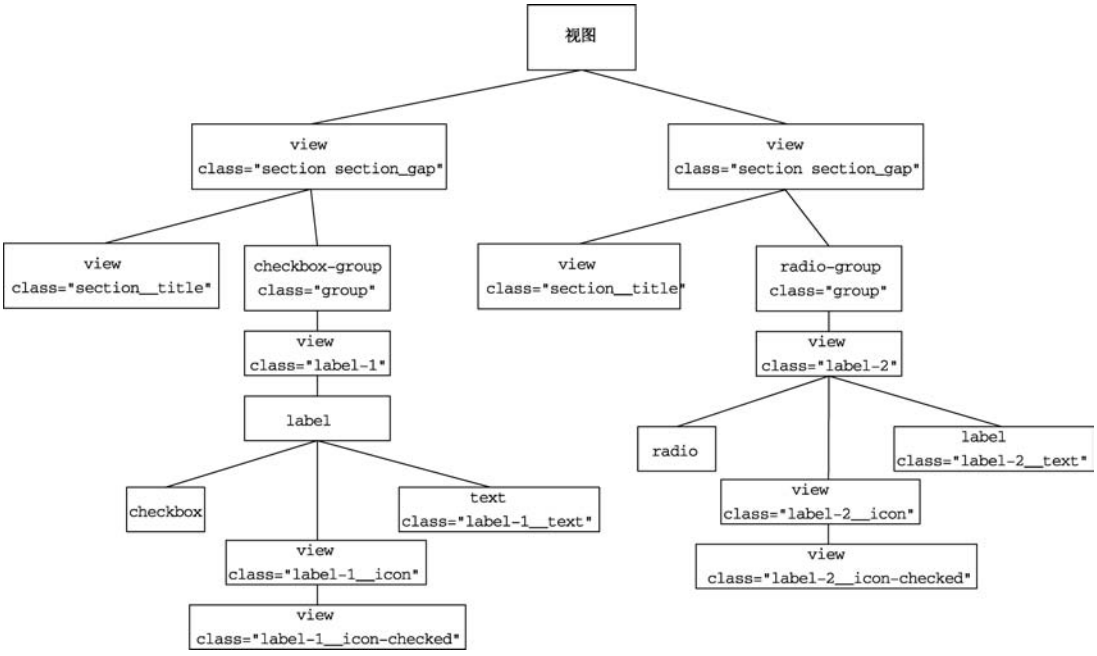

图 3-26 label 组件的示例代码中控件的关系

2 button

button 是按钮组件，其语法格式如图 3-27 所示。

```
<button>

</button>
```

图 3-27 button 组件的格式

如表 3-24 所示为 button 组件的属性。

表 3-24 button 组件的属性

序号	属　性	说　明
1	size	按钮的大小。该属性的类型为 string，默认值为 default（即默认尺寸），该属性还可以取值 mini（即小尺寸）
2	type	按钮的样式类型。该属性的类型为 string，默认值为 default（即白色），该属性还可以取值 primary（即绿色）和 warn（即红色）
3	plain	按钮是否镂空，背景色透明。该属性的类型为 boolean，默认值为 false
4	disabled	是否禁用。该属性的类型为 boolean，默认值为 false
5	loading	名称前是否带 loading 图标。该属性的类型为 boolean，默认值为 false
6	form-type	用于 form 组件，点击分别会触发 form 组件的 submit（即提交表单）或 reset（即重置表单）事件。该属性的类型为 string

续表

序号	属　性	说　明
7	open-type	微信开放能力。该属性的类型为 string
8	hover-class	指定按钮按下去的样式类。当 hover-class="none"时，没有点击态效果。该属性的类型为 string，默认值为 button-hover
9	hover-stop-propagation	指定是否阻止本节点的祖先节点出现点击态。该属性的类型为 boolean，默认值为 false
10	hover-start-time	按住后多久出现点击态，单位为毫秒。该属性的类型为 number，默认值为 20
11	hover-stay-time	手指松开后点击态保留时间，单位为毫秒。该属性的类型为 number，默认值为 20
12	lang	指定返回用户信息的语言，取值可以为 zh_CN(即简体中文)、zh_TW(即繁体中文)和 en(即英文)。该属性的类型为 string，默认值为 en
13	session-from	会话来源，open-type="contact"时有效。该属性的类型为 string
14	send-message-title	会话内消息卡片标题，open-type="contact"时有效。该属性的类型为 string，默认值为当前标题
15	send-message-path	会话内消息卡片点击跳转小程序路径，open-type="contact"时有效。该属性的类型为 string，默认值为当前分享路径
16	send-message-img	会话内消息卡片图片，open-type="contact"时有效。该属性的类型为 string，默认值为截图
17	app-parameter	打开 APP 时，向 APP 传递的参数，open-type=launchApp 时有效。该属性的类型为 string
18	show-message-card	是否显示会话内消息卡片，设置此参数为 true，用户进入客服会话会在右下角显示“可能要发送的小程序”提示，用户点击后可以快速发送小程序消息，open-type="contact"时有效。该属性的类型为 boolean，默认值为 false
19	bindgetuserinfo	用户点击该按钮时，会返回获取到的用户信息，回调的 detail 数据与 wx.getUserInfo 返回的一致，open-type="getUserInfo"时有效。该属性的类型为 eventhandle
20	bindcontact	客服消息回调，open-type="contact"时有效。该属性的类型为 eventhandle
21	bindgetphonenumber	获取用户手机号回调，open-type=getPhoneNumber 时有效。该属性的类型为 eventhandle
22	binderror	当使用开放能力时，发生错误的回调，open-type=launchApp 时有效。该属性的类型为 eventhandle
23	bindopensetting	在打开授权设置页后回调，open-type=openSetting 时有效。该属性的类型为 eventhandle
24	bindlaunchapp	打开 APP 成功的回调，open-type=launchApp 时有效。该属性的类型为 eventhandle

属性 open-type 的合法值如表 3-25 所示。

表 3-25 open-type 的合法值

序号	值	说 明
1	contact	打开客服会话，如果用户在会话中点击消息卡片后返回小程序，可以从 bindcontact 回调中获得具体信息(消息的页面路径 path 和对应的参数 query)
2	share	触发用户转发。例如：用户点击按钮后触发 Page. onShareAppMessage 事件
3	getPhoneNumber	获取用户手机号，可以从 bindgetphonenumber 回调中获取到用户信息。例如：用户可以通过 bindgetphonenumber 事件回调获取到微信服务器返回的加密数据，然后在第三方服务端结合 session_key 以及 app_id 进行解密获取手机号
4	getUserInfo	获取用户信息，可以从 bindgetuserinfo 回调中获取到用户信息
5	launchApp	打开 APP，可以通过 app-parameter 属性设定向 APP 传的参数。通过 binderror 可以监听打开 APP 的错误事件
6	openSetting	打开授权设置页
7	feedback	打开“意见反馈”页面，用户可提交反馈内容并上传日志，开发者可以登录小程序管理后台后进入左侧菜单“客服反馈”页面获取到反馈内容

使用 button 组件，请注意以下几点：

(1) button-hover 默认为{background-color：rgba(0，0，0，0.1)；opacity：0.7；}。

(2) bindgetphonenumber 从 1.2.0 开始支持，但是在 1.5.3 以下版本中无法使用 wx.canIUse 进行检测，建议使用基础库版本进行判断。

(3) 在 bindgetphonenumber 等返回加密信息的回调中调用 wx.login 登录，可能会刷新登录态。此时服务器使用 code 换取的 sessionKey 不是加密时使用的 sessionKey，导致解密失败。建议开发者提前进行 login；或者在回调中先使用 checkSession 进行登录态检查，避免 login 刷新登录态。

(4) 从 2.1.0 起，button 可作为原生组件的子节点嵌入，以便在原生组件上使用 open-type 的能力。

(5) 目前设置了 form-type 的 button 只会对当前组件中的 form 有效。若将 button 封装在自定义组件中，而将 form 放在自定义组件外，这会使这个 button 的 form-type 失效。

图 3-28 为 button 组件的示例代码。

```
//视图层
<button type="default" size="{{defaultSize}}" loading="{{loading}}" plain="{{plain}}"
        disabled="{{disabled}}" bindtap="default" hover-class="other-button-hover">
default </button>
<button type="primary" size="{{primarySize}}" loading="{{loading}}" plain="{{plain}}"
        disabled="{{disabled}}" bindtap="primary"> primary </button>
<button type="warn" size="{{warnSize}}" loading="{{loading}}" plain="{{plain}}"
        disabled="{{disabled}}" bindtap="warn"> warn </button>
<button bindtap="setDisabled">点击设置以上按钮 disabled 属性</button>
<button bindtap="setPlain">点击设置以上按钮 plain 属性</button>
```

图 3-28 button 组件的示例代码

```
<button bindtap="setLoading">点击设置以上按钮 loading 属性</button>
<button open-type="contact">进入客服会话</button>
<button open-type="getUserInfo" lang="zh_CN" bindgetuserinfo="onGotUserInfo">获取用户信息</button>

/** wxss **/
/** 修改 button 默认的点击态样式类 **/
.button-hover {
  background-color: red;
}
/** 添加自定义 button 点击态样式类 **/
.other-button-hover {
  background-color: blue;
}
button {margin: 10px;}

//逻辑层
var types = ['default', 'primary', 'warn']
var pageObject = {
  data: {
    defaultSize: 'default',
    primarySize: 'default',
    warnSize: 'default',
    disabled: false,
    plain: false,
    loading: false
  },
  setDisabled: function (e) {
    this.setData({
      disabled: !this.data.disabled
    })
  },
  setPlain: function (e) {
    this.setData({
      plain: !this.data.plain
    })
  },
  setLoading: function (e) {
    this.setData({
      loading: !this.data.loading
    })
  },
  onGotUserInfo: function (e) {
    console.log(e.detail.errMsg)
    console.log(e.detail.userInfo)
    console.log(e.detail.rawData)
  },
}

for (var i = 0; i < types.length; ++i) {
```

图 3-28 button 组件的示例代码（续）

```
  (function (type) {
    pageObject[type] = function (e) {
      var key = type + 'Size'
      var changedData = {}
      changedData[key] =
        this.data[key] === 'default'? 'mini' : 'default'
      this.setData(changedData)
    }
  })(types[i])
}

Page(pageObject)
```

图 3-28 button 组件的示例代码(续)

视图层的代码中有 8 个 button,第一个 button 的 type 为 default,第二个 button 的 type 为 primary,第三个 button 的 type 为 warn,第四个 button 绑定了 setDisabled 函数,由于前三个 button 的 disabled 属性都是动态获取的,因此若按下第四个 button,将会触发 setDisabled 函数,从而使前三个 button 的 disabled 属性的值为当前值的非(!),即若当前值为 true,则变为 false,反之亦然。若反复按下第四个 button,前三个 button 就会在可用和不可用之间循环切换。

第五个 button 绑定了 setPlain 函数,由于前三个 button 的 plain 属性都是动态获取的,因此若按下第四个 button,将会触发 setPlain 函数,从而使前三个 button 的 plain 属性的值为当前值的非(!),即若当前值为 true,则变为 false,反之亦然。若反复按下第五个 button,前三个 button 就会在背景色透明和背景色不透明之间循环切换。

第六个 button 绑定了 setLoading 函数,由于前三个 button 的 loading 属性都是动态获取的,因此若按下第四个 button,将会触发 setLoading 函数,从而使前三个 button 的 loading 属性的值为当前值的非(!),即若当前值为 true,则变为 false,反之亦然。若反复按下第六个 button,前三个 button 就会在名称前带 loading 图标和名称前不带 loading 图标之间循环切换。

第七个 button 的 open-type 为 contact,按下它可以打开客服会话。

第八个 button 的 open-type 为 getUserInfo,按下它将通过 bindgetuserinfo 回调获取到用户信息,onGotUserInfo 函数将在控制台输出 errMsg、userInfo 和 rawData。

关于 label 和 button 组件的更多内容,可访问以下链接。

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

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

3.4.2 radio 和 radio-group

radio 是单选项目组件,radio-group 是单项选择器组件,它的内部由多个 radio 组成。radio 和 radio-group 的语法格式如图 3-29 所示。

```
<radio-group>
    <radio>

    </radio>
</radio-group>
```

图 3-29 radio 和 radio-group 组件的格式

如表 3-26 所示为 radio 组件的属性。

表 3-26 radio 组件的属性

序号	属 性	说 明
1	value	radio 标识。当该 radio 选中时，radio-group 的 change 事件会携带 radio 的 value。该属性的类型为 string
2	checked	当前是否选中。该属性的类型为 boolean，默认值为 false
3	disabled	是否禁用。该属性的类型为 boolean，默认值为 false
4	color	radio 的颜色，同 css 的 color。该属性的类型为 string，默认值为 #09BB07

如表 3-27 所示为 radio-group 组件的属性。

表 3-27 radio-group 组件的属性

属 性	说 明
bindchange	radio-group 中选中项发生改变时触发 change 事件，detail={value:[选中的 radio 的 value 的数组]}。该属性的类型为 EventHandle

图 3-30 为 radio 和 radio-group 组件的示例代码。

```
//视图层
<view class="page">
    <view class="page__hd">
        <text class="page__title">radio</text>
        <text class="page__desc">单选框</text>
    </view>
    <view class="page__bd">
        <view class="section section_gap">
            <radio-group class="radio-group" bindchange="radioChange">
                <radio class="radio" wx:for-items="{{items}}" wx:key="name" value="
{{item.name}}" checked="{{item.checked}}">
                    <text>{{item.value}}</text>
                </radio>
            </radio-group>
        </view>
    </view>
</view>
//逻辑层
Page({
  data: {
    items: [
```

图 3-30 radio 和 radio-group 组件的格式

```
        { name: 'USA', value: '美国' },
        { name: 'CHN', value: '中国', checked: 'true' },
        { name: 'BRA', value: '巴西' },
        { name: 'JPN', value: '日本' },
        { name: 'ENG', value: '英国' },
        { name: 'FRA', value: '法国' },
      ]
    },
    radioChange: function (e) {
      console.log('radio 发生 change 事件,携带 value 值为: ', e.detail.value)
    }
  })
```

图 3-30　radio 和 radio-group 组件的格式(续)

从视图层代码可以看到,radio 组件嵌入在 radio-group 组件中,text 组件嵌入在 radio 组件中。radio 和 text 组件的数据(如 radio 组件的 value 和 checked 的值,text 组件的值)都是从逻辑层动态获得。由于 radio-group 组件绑定了 radioChange 事件,所以在改变选择项时将会触发这一事件,在控制台输出改变后选择的值。

关于 radio 和 radio-group 组件的更多内容,可访问以下链接。

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

https://developers.weixin.qq.com/miniprogram/dev/component/radio-group.html

3.4.3　checkbox 和 checkbox-group

checkbox 是多选项目组件,checkbox-group 是多项选择器组件,它的内部由多个 checkbox 组成。checkbox 和 checkbox-group 的语法格式如图 3-31 所示。

```
<checkbox-group>
    <checkbox>

    </checkbox>
</checkbox-group>
```

图 3-31　checkbox 和 checkbox-group 组件的格式

如表 3-28 所示为 checkbox 组件的属性。

表 3-28　checkbox 组件的属性

序号	属　性	说　明
1	value	checkbox 标识,选中时触发 checkbox-group 的 change 事件,并携带 checkbox 的 value。该属性的类型为 string
2	disabled	是否禁用。该属性的类型为 boolean,默认值为 false
3	checked	当前是否选中,可用来设置默认选中。该属性的类型为 boolean,默认值为 false
4	color	checkbox 的颜色,同 css 的 color。该属性的类型为 string,默认值为 #09BB07

如表 3-29 所示为 checkbox-group 组件的属性。

表 3-29 checkbox-group 组件的属性

属 性	说 明
bindchange	checkbox-group 中选中项发生改变时触发 change 事件，detail = {value:[选中的 checkbox 的 value 的数组]}。该属性的类型为 EventHandle

图 3-32 为 checkbox 和 checkbox-group 组件的示例代码。

```
//视图层
<checkbox-group bindchange="checkboxChange">
  <label class="checkbox" wx:for="{{items}}">
    <checkbox value="{{item.name}}" checked="{{item.checked}}"/>{{item.value}}
  </label>
</checkbox-group>
//逻辑层
Page({
  data: {
    items: [
      { name: 'USA', value: '美国' },
      { name: 'CHN', value: '中国', checked: 'true' },
      { name: 'BRA', value: '巴西' },
      { name: 'JPN', value: '日本' },
      { name: 'ENG', value: '英国' },
      { name: 'TUR', value: '法国' },
    ]
  },
  checkboxChange: function (e) {
    console.log('checkbox 发生 change 事件，携带 value 值为：', e.detail.value)
  }
})
```

图 3-32 checkbox 和 checkbox-group 组件的示例代码

在上述视图层代码中，label 组件内嵌在 checkbox-group 组件中，checkbox 内嵌在 label 组件中，其数据从逻辑层动态获取。checkbox-group 组件绑定了 checkboxChange 函数，当改变 checkbox 当前值时，将会触发该事件，从而在控制台上输出 value。

关于 checkbox 和 checkbox-group 组件的更多内容，可访问以下链接。

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

https://developers.weixin.qq.com/miniprogram/dev/component/checkbox-group.html

3.4.4 input、textarea 和 editor

1 input

input 是输入框组件，它是原生组件，使用时请注意相关限制。它的语法格式如图 3-33 所示。

```
< input >

</input >
或
< input        />
```

图 3-33　input 组件的格式

如表 3-30 所示为 input 组件的属性。

表 3-30　input 组件的属性

序号	属　性	说　明
1	value	输入框的初始内容。该属性的类型为 string，为必填字段
2	type	input 的类型。该属性的类型为 string，默认值为 text（文本输入键盘）。该属性的值还可以为 number（数字输入键盘）、idcard（身份证输入键盘）和 digit（带小数点的数字键盘）
3	password	是否是密码类型。该属性的类型为 boolean，默认值为 false
4	placeholder	输入框为空时占位符。该属性的类型为 string，为必填字段
5	placeholder-style	指定 placeholder 的样式。该属性的类型为 string，为必填字段
6	placeholder-class	指定 placeholder 的样式类。该属性的类型为 string，默认值为 input-placeholder
7	disabled	是否禁用。该属性的类型为 boolean，默认值为 false
8	maxlength	最大输入长度，设置为－1 的时候不限制最大长度。该属性的类型为 number，默认值为 140
9	cursor-spacing	指定光标与键盘的距离，取 input 距离底部的距离和 cursor-spacing 指定的距离的最小值作为光标与键盘的距离。该属性的类型为 number，默认值为 140
10	auto-focus	（即将废弃，请直接使用 focus）自动聚焦，拉起键盘。该属性的类型为 boolean，默认值为 false
11	focus	获取焦点。该属性的类型为 boolean，默认值为 false
12	confirm-type	设置键盘右下角按钮的文字，仅在 type＝'text'时生效。该属性的类型为 string，默认值为 done（右下角按钮为“完成”）。该属性的值还可以为 send（右下角按钮为“发送”）、search（右下角按钮为“搜索”）、next（右下角按钮为“下一个”）和 go（右下角按钮为“前往”）
13	confirm-hold	点击键盘右下角按钮时是否保持键盘不收起。该属性的类型为 boolean，默认值为 false
14	cursor	指定 focus 时的光标位置。该属性的类型为 number
15	selection-start	光标起始位置，自动聚集时有效，需与 selection-end 搭配使用。该属性的类型为 number，默认值为－1
16	selection-end	光标结束位置，自动聚集时有效，需与 selection-start 搭配使用。该属性的类型为 number，默认值为－1
17	adjust-position	键盘弹起时，是否自动上推页面。该属性的类型为 boolean，默认值为 true
18	hold-keyboard	focus 时，点击页面的时候不收起键盘。该属性的类型为 boolean，默认值为 false
19	bindinput	键盘输入时触发，event. detail＝{value，cursor，keyCode}，keyCode 为键值，2.1.0 起支持，处理函数可以直接 return 一个字符串，将替换输入框的内容。该属性的类型为 eventhandle

续表

序号	属　性	说　明
20	bindfocus	输入框聚焦时触发，event. detail＝{value，height}，height 为键盘高度。该属性的类型为 eventhandle
21	bindblur	输入框失去焦点时触发，event. detail＝{value：value}。该属性的类型为 eventhandlev
22	bindconfirm	点击完成按钮时触发，event. detail＝{value：value}。该属性的类型为 eventhandle
23	bindkeyboardheightchange	键盘高度发生变化的时候触发此事件，event. detail＝{height：height，duration：duration}。该属性的类型为 eventhandle

使用 input 组件时，还需要注意以下问题。

（1）confirm-type 的最终表现与手机输入法本身的实现有关，部分安卓系统输入法和第三方输入法可能不支持或不完全支持；

（2）input 组件是一个原生组件，字体是系统字体，所以无法设置 font-family；

（3）在 input 聚焦期间，避免使用 css 动画；

（4）对于将 input 封装在自定义组件中、而 form 在自定义组件外的情况，form 将不能获得这个自定义组件中 input 的值。此时需要使用自定义组件的内置 behaviors wx://form-field；

（5）键盘高度发生变化，keyboardheightchange 事件可能会多次触发，开发者对于相同的 height 值应该忽略掉；

（6）微信版本 6.3.30，focus 属性设置无效，placeholder 在聚焦时出现重影问题。

图 3-34 为 input 组件的示例代码。

```
<view class="page-body">
  <view class="page-section">
    <view class="weui-cells__title">可以自动聚焦的 input</view>
    <view class="weui-cells weui-cells_after-title">
      <view class="weui-cell weui-cell_input">
        <input class="weui-input" auto-focus placeholder="将会获取焦点"/>
      </view>
    </view>
  </view>
  <view class="page-section">
    <view class="weui-cells__title">控制最大输入长度的 input</view>
    <view class="weui-cells weui-cells_after-title">
      <view class="weui-cell weui-cell_input">
        <input class="weui-input" maxlength="10" placeholder="最大输入长度为 10" />
      </view>
    </view>
  </view>
  <view class="page-section">
    <view class="weui-cells__title">实时获取输入值: {{inputValue}}</view>
    <view class="weui-cells weui-cells_after-title">
```

图 3-34　input 组件的示例代码

```
        <view class="weui-cell weui-cell_input">
          <input class="weui-input" maxlength="10" bindinput="bindKeyInput" placeholder="输
入同步到 view 中"/>
        </view>
      </view>
    </view>
    <view class="page-section">
      <view class="weui-cells__title">控制输入的 input</view>
      <view class="weui-cells weui-cells_after-title">
        <view class="weui-cell weui-cell_input">
          <input class="weui-input" bindinput="bindReplaceInput" placeholder="连续的两个
1 会变成 2" />
        </view>
      </view>
    </view>
    <view class="page-section">
      <view class="weui-cells__title">控制键盘的 input</view>
      <view class="weui-cells weui-cells_after-title">
        <view class="weui-cell weui-cell_input">
          <input class="weui-input" bindinput="bindHideKeyboard" placeholder="输入 123 自
动收起键盘" />
        </view>
      </view>
    </view>
    <view class="page-section">
      <view class="weui-cells__title">数字输入的 input</view>
      <view class="weui-cells weui-cells_after-title">
        <view class="weui-cell weui-cell_input">
          <input class="weui-input" type="number" placeholder="这是一个数字输入框" />
        </view>
      </view>
    </view>
    <view class="page-section">
      <view class="weui-cells__title">密码输入的 input</view>
      <view class="weui-cells weui-cells_after-title">
        <view class="weui-cell weui-cell_input">
          <input class="weui-input" password type="text" placeholder="这是一个密码输入框" />
        </view>
      </view>
    </view>
    <view class="page-section">
      <view class="weui-cells__title">带小数点的 input</view>
      <view class="weui-cells weui-cells_after-title">
        <view class="weui-cell weui-cell_input">
          <input class="weui-input" type="digit" placeholder="带小数点的数字键盘"/>
        </view>
      </view>
    </view>
    <view class="page-section">
```

图 3-34　input 组件的示例代码（续）

```
    <view class="weui-cells__title">身份证输入的 input</view>
    <view class="weui-cells weui-cells_after-title">
      <view class="weui-cell weui-cell_input">
        <input class="weui-input" type="idcard" placeholder="身份证输入键盘" />
      </view>
    </view>
  </view>
  <view class="page-section">
    <view class="weui-cells__title">控制占位符颜色的 input</view>
    <view class="weui-cells weui-cells_after-title">
      <view class="weui-cell weui-cell_input">
        <input class="weui-input" placeholder-style="color:#F76260" placeholder="占位
符字体是红色的" />
      </view>
    </view>
  </view>
</view>
//逻辑层
Page({
  data: {
    focus: false,
    inputValue: ''
  },
  bindKeyInput: function (e) {
    this.setData({
      inputValue: e.detail.value
    })
  },
  bindReplaceInput: function (e) {
    var value = e.detail.value
    var pos = e.detail.cursor
    var left
    if (pos !== -1) {
      // 光标在中间
      left = e.detail.value.slice(0, pos)
      // 计算光标的位置
      pos = left.replace(/11/g, '2').length
    }

    // 直接返回对象,可以对输入进行过滤处理,同时可以控制光标的位置
    return {
      value: value.replace(/11/g, '2'),
      cursor: pos
    }

    // 或者直接返回字符串,光标在最后边
    // return value.replace(/11/g,'2'),
  },
  bindHideKeyboard: function (e) {
```

图 3-34 input 组件的示例代码(续)

```
        if (e.detail.value === '123') {
          // 收起键盘
          wx.hideKeyboard()
        }
      }
    })
```

图 3-34　input 组件的示例代码(续)

上述代码中包括多个 view 和 input 控件,这些控件的关系如图 3-35 所示。

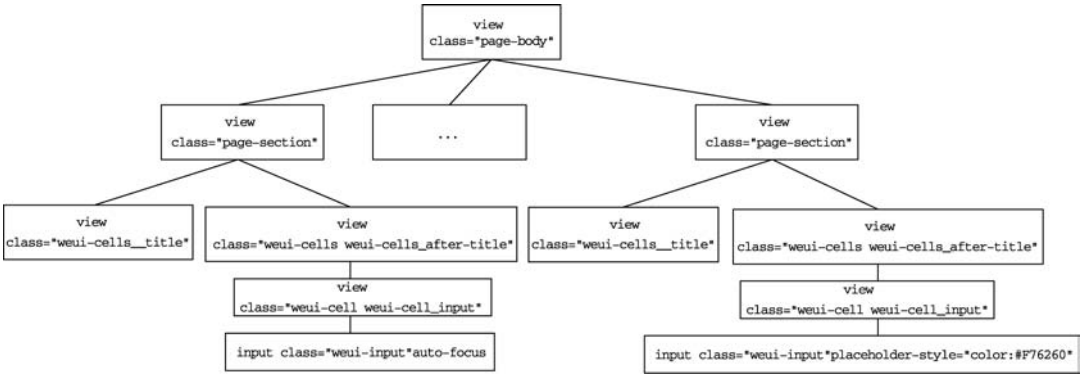

图 3-35　input 组件的示例代码中控件的关系

上述视图层代码中,在 class="page-body"的 view 中一共内嵌 10 个 class="page-section"的 view。对于每一个 class="page-section"的 view,又分别内嵌一个 class="weui-cells__title"的 view 和一个 class="weui-cells weui-cells_after-title"的 view,在 class="weui-cells weui-cells_after-title"的 view 里,内嵌了 class="weui-cell weui-cell_input"的 view,在其中内嵌了 class="weui-input"的 input。因此,在 class="page-body"的 view 中一共有 10 个不同类型的 input。

2 textarea

textarea 是多行输入框组件,它是原生组件,使用时请注意相关限制。它的语法格式如图 3-36 所示。

```
<textarea>

</textarea>
或
<textarea        />
```

图 3-36　textarea 组件的格式

如表 3-31 所示为 textarea 组件的属性。

表 3-31　textarea 组件的属性

序号	属　性	说　明
1	value	输入框的内容。该属性的类型的 string
2	placeholder	输入框为空时占位符。该属性的类型的 string

续表

序号	属　性	说　明
3	placeholder-style	指定 placeholder 的样式，目前仅支持 color，font-size 和 font-weight。该属性的类型的 string
4	placeholder-class	指定 placeholder 的样式类。该属性的类型的 string，默认值为 textarea-placeholder
5	disabled	是否禁用。该属性的类型的 boolean，默认值为 false
6	maxlength	最大输入长度，设置为－1 的时候不限制最大长度。该属性的类型的 number，默认值为 140
7	auto-focus	自动聚焦，拉起键盘。该属性的类型的 boolean，默认值为 false
8	focus	获取焦点。该属性的类型的 boolean，默认值为 false
9	auto-height	是否自动增高，设置 auto-height 时，style. height 不生效。该属性的类型的 boolean，默认值为 false
10	fixed	如果 textarea 是在一个 position:fixed 的区域，需要显示指定属性 fixed 为 true。该属性的类型的 boolean，默认值为 false
11	cursor-spacing	指定光标与键盘的距离。取 textarea 距离底部的距离和 cursor-spacing 指定的距离的最小值作为光标与键盘的距离。该属性的类型的 number，默认值为 0
12	cursor	指定 focus 时的光标位置。该属性的类型的 number，默认值为－1
13	show-confirm-bar	是否显示键盘上方带有“完成”按钮那一栏。该属性的类型的 boolean，默认值为 true
14	selection-start	光标起始位置，自动聚集时有效，需与 selection-end 搭配使用。该属性的类型的 number，默认值为－1
15	selection-end	光标结束位置，自动聚集时有效，需与 selection-start 搭配使用。该属性的类型的 number，默认值为－1
16	adjust-position	键盘弹起时，是否自动上推页面。该属性的类型的 boolean，默认值为 true
17	hold-keyboard	focus 时，点击页面的时候不收起键盘。该属性的类型的 boolean，默认值为 false
18	bindfocus	输入框聚焦时触发，event. detail＝{value，height}，height 为键盘高度。该属性的类型为 eventhandle
19	bindblur	输入框失去焦点时触发，event. detail＝{value，cursor}。该属性的类型为 eventhandle
20	bindlinechange	输入框行数变化时调用，event. detail＝{height:0，heightRpx:0，lineCount:0}。该属性的类型为 eventhandle
21	bindinput	当键盘输入时，触发 input 事件，event. detail＝{value，cursor，keyCode}，keyCode 为键值，目前工具还不支持返回 keyCode 参数。bindinput 处理函数的返回值并不会反映到 textarea 上。该属性的类型为 eventhandle
22	bindconfirm	点击完成时，触发 confirm 事件，event. detail＝{value:value}。该属性的类型为 eventhandle
23	bindkeyboardheightchange	键盘高度发生变化的时候触发此事件，event. detail＝{height:height，duration:duration}。该属性的类型为 eventhandle

使用 textarea 组件时，请注意以下几点。

(1) textarea 的 blur 事件会晚于页面上的 tap 事件，如果需要在 button 的点击事件获取

textarea，可以使用 form 的 bindsubmit。

（2）不建议在多行文本上对用户的输入进行修改，所以 textarea 的 bindinput 处理函数并不会将返回值反映到 textarea 上。

（3）键盘高度发生变化，keyboardheightchange 事件可能会多次触发，开发者对于相同的 height 值应该忽略掉。

（4）微信版本 6.3.30，textarea 在列表渲染时，新增加的 textarea 在自动聚焦时的位置计算错误。

图 3-37 为 textarea 组件的示例代码。

```
<view class="section">
  <textarea bindblur="bindTextAreaBlur" auto-height placeholder="自动变高" />
</view>
<view class="section">
  <textarea placeholder="placeholder 颜色是红色的" placeholder-style="color:red;" />
</view>
<view class="section">
  <textarea placeholder="这是一个可以自动聚焦的 textarea" auto-focus />
</view>
<view class="section">
  <textarea placeholder="这个只有在按钮点击的时候才聚焦" focus="{{focus}}" />
  <view class="btn-area">
    <button bindtap="bindButtonTap">使得输入框获取焦点</button>
  </view>
</view>
<view class="section">
  <form bindsubmit="bindFormSubmit">
    <textarea placeholder="form 中的 textarea" name="textarea"/>
    <button form-type="submit"> 提交 </button>
  </form>
</view>

//textarea.js
Page({
  data: {
    height: 20,
    focus: false
  },
  bindButtonTap: function() {
    this.setData({
      focus: true
    })
  },
  bindTextAreaBlur: function(e) {
    console.log(e.detail.value)
  },
  bindFormSubmit: function(e) {
    console.log(e.detail.value.textarea)
  }
})
```

图 3-37　textarea 组件的格式

上述代码中包括多个 view 和 textarea 控件，这些控件的关系如图 3-38 所示。

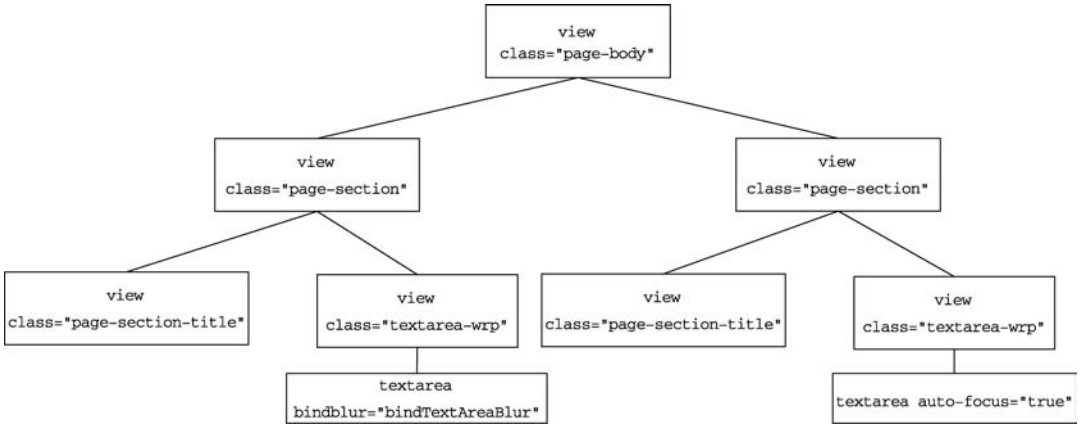

图 3-38 textarea 组件的示例代码中控件的关系

上述视图层代码中，在 class="page-body"的 view 中内嵌两个 class="page-section"的 view。对于每一个 class="page-section"的 view，分别内嵌一个 class="page-section-title"的 view 和一个 class="textarea-wrp"的 view，在 class=" textarea-wrp"的 view 里内嵌了一个 textarea。

3 editor

editor 是富文本编辑器组件，可以对图片、文字进行编辑。它的语法格式如图 3-39 所示。

```
<editor>

</editor>
```

图 3-39 editor 组件的格式

编辑器导出内容支持带标签的 html 和纯文本的 text，编辑器内部采用 delta 格式进行存储。通过 setContents 接口设置内容时，解析插入的 html 可能会由于一些非法标签导致解析错误，建议开发者在小程序内使用时通过 delta 进行插入。

富文本组件内部引入了一些基本的样式使得内容可以正确的展示，开发时可以进行覆盖。需要注意的是，在其他组件或环境中使用富文本组件导出的 html 时，需要额外引入 editor.css（参考 https://github.com/wechat-miniprogram/editor-style 或教材配套的本章代码），并维护如图 3-40 所示结构。

```
<ql-container>
    <ql-editor>

    </ql-editor>
</ql-container>
```

图 3-40 在其他组件或环境中使用富文本组件

注意：图片控件仅初始化时设置有效。

如表 3-32 所示为 editor 组件的属性。

表 3-32 editor 组件的属性

序号	属　性	说　明
1	read-only	设置编辑器为只读。该属性的类型的 boolean，默认值为 false
2	placeholder	提示信息。该属性的类型的 string
3	show-img-size	点击图片时显示图片大小控件。该属性的类型的 boolean，默认值为 false
4	show-img-toolbar	点击图片时显示工具栏控件。该属性的类型的 boolean，默认值为 false
5	show-img-resize	点击图片时显示修改尺寸控件。该属性的类型的 boolean，默认值为 false
6	bindready	编辑器初始化完成时触发。该属性的类型的 eventhandle
7	bindfocus	编辑器聚焦时触发，event. detail＝{html，text，delta}。该属性的类型的 eventhandle
8	bindblur	编辑器失去焦点时触发，detail ＝ {html，text，delta}。该属性的类型的 eventhandle
9	bindinput	编辑器内容改变时触发，detail ＝ {html，text，delta}。该属性的类型的 eventhandle
10	bindstatuschange	通过 Context 方法改变编辑器内样式时触发，返回选区已设置的样式。该属性的类型的 eventhandle

注意：编辑器内支持部分 HTML 标签和内连样式，不支持 class 和 id。对于不支持的标签会被忽略，< div >会被转换为< p >储存。

编辑器支持的标签如表 3-33 所示。

表 3-33 编辑器支持的标签

序号	类型	节　点
1	行内元素	< span > < strong > < b > < ins > < em > < i > < u > < a > < del > < s > < sub > < sup > < img >
2	块级元素	< p > < h1 > < h2 > < h3 > < h4 > < h5 > < h6 > < hr > < ol > < ul > < li >

编辑器支持的内联样式如表 3-34 所示。

表 3-34 编辑器支持的内联样式

序号	类型	节　点
1	块级样式	text-align direction margin margin-top margin-left margin-right margin-bottom padding padding-top padding-left padding-right padding-bottom line-height text-indent
2	行内样式	font font-size font-style font-variant font-weight font-family letter-spacing text-decoration color background-color

使用编辑器组件时，请注意以下问题。

(1) 使用 catchtouchend 绑定事件则不会使编辑器失去焦点(2.8.3)。

(2) 插入的 html 中事件绑定会被移除。

(3) formats 中的 color 属性会统一以 hex 格式返回。

(4) 粘贴时仅纯文本内容会被复制进编辑器。

(5) 插入 html 到编辑器内时，编辑器会删除一些不必要的标签，以保证内容的统一。例

如 xxx 会改写为 xxx 。

(6) 编辑器聚焦时页面会被上推,系统行为以保证编辑区可见。

图 3-41 为 editor 组件在视图层的示例代码(完整项目见本章配套代码文件夹 ex030404_editor)。

```
<view class="container" style="height:{{editorHeight}}px;">
  <editor id="editor" class="ql-container" placeholder="{{placeholder}}"
bindstatuschange="onStatusChange" bindready="onEditorReady">
  </editor>
</view>

<view class="toolbar" catchtouchend="format" hidden="{{keyboardHeight > 0 ? false : true}}"
style="bottom: {{isIOS ? keyboardHeight : 0}}px">
  <i class="iconfont icon-charutupian" catchtouchend="insertImage"></i>
  <i class="iconfont icon-format-header-2 {{formats.header === 2 ? 'ql-active' : ''}}"
data-name="header" data-value="{{2}}"></i>
  <i class="iconfont icon-format-header-3 {{formats.header === 3 ? 'ql-active' : ''}}"
data-name="header" data-value="{{3}}"></i>
  <i class="iconfont icon-zitijiacu {{formats.bold ? 'ql-active' : ''}}" data-name="bold"></i>
  <i class="iconfont icon-zitixieti {{formats.italic ? 'ql-active' : ''}}" data-name=
"italic"></i>
  <i class="iconfont icon-zitixiahuaxian {{formats.underline ? 'ql-active' : ''}}" data-
name="underline"></i>
  <i class="iconfont icon-checklist" data-name="list" data-value="check"></i>
  <i class="iconfont icon-youxupailie {{formats.list === 'ordered' ? 'ql-active' : ''}}"
data-name="list" data-value="ordered"></i>
  <i class="iconfont icon-wuxupailie {{formats.list === 'bullet' ? 'ql-active' : ''}}" data-
name="list" data-value="bullet"></i>
</view>
```

图 3-41 editor 组件的示例代码

关于 input、textarea 和 editor 组件的更多内容,可访问以下链接。

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

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

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

3.4.5 picker、picker-view 和 picker-view-column

1 picker

picker 是从底部弹起的滚动选择器组件。它的语法格式如图 3-42 所示。

```
<picker>

</picker>
```

图 3-42 picker 组件的格式

如表 3-35 所示为 picker 组件的属性。

表 3-35 picker 组件的属性

序号	属性	说明
1	mode	选择器类型。该属性的类型是 string，默认值是 selector（普通选择器），该属性的值还可以为 multiSelector（多列选择器）、time（时间选择器）、date（日期选择器）和 region（省市区选择器）
2	disabled	是否禁用。该属性的类型是 boolean，默认值是 false
3	bindcancel	取消选择时触发。该属性的类型是 eventhandle

对于取值不同的 mode，picker 拥有不同的属性，具体如下。

1）mode 为 selector

当 mode 为 selector，其属性如表 3-36 所示。

表 3-36 mode 为 selector 时 picker 组件的属性

序号	属性	说明
1	range	mode 为 selector 或 multiSelector 时，range 有效。该属性类型为 array 或 object array，默认值为[]
2	range-key	当 range 是一个 Object Array 时，通过 range-key 来指定 Object 中 key 的值作为选择器显示内容。该属性类型为 string
3	value	表示选择了 range 中的第几个（下标从 0 开始）。该属性类型为 number，默认值为 0
4	bindchange	value 改变时触发 change 事件，event. detail = {value}。该属性类型为 eventhandle

2）mode 为 multiSelector

当 mode 为 multiSelector，其属性如表 3-37 所示。

表 3-37 mode 为 mutiSelector 时 picker 组件的属性

序号	属性	说明
1	range	mode 为 selector 或 multiSelector 时，range 有效。该属性类型为 array 或 object array，默认值为[]
2	range-key	当 range 是一个 Object Array 时，通过 range-key 来指定 Object 中 key 的值作为选择器显示内容。该属性类型为 string
3	value	表示选择了 range 中的第几个（下标从 0 开始）。该属性类型为 array，默认值为[]
4	bindchange	value 改变时触发 change 事件，event. detail = {value}。该属性类型为 eventhandle
5	bindcolumnchange	列改变时触发。该属性类型为 eventhandle

3）mode 为 time

当 mode 为 time，其属性如表 3-38 所示。

表 3-38 mode 为 time 时 picker 组件的属性

序号	属 性	说 明
1	value	表示选中的时间，格式为"hh:mm"。该属性类型为 string
2	start	表示有效时间范围的开始，字符串格式为"hh:mm"。该属性类型为 string
3	end	表示有效时间范围的结束，字符串格式为"hh:mm"。该属性类型为 string
4	bindchange	value 改变时触发 change 事件，event. detail = {value}。该属性类型为 eventhandle

4）mode 为 date

当 mode 为 date，其属性如表 3-39 所示。

表 3-39 mode 为 date 时 picker 组件的属性

序号	属 性	说 明
1	value	表示选中的日期，格式为"YYYY-MM-DD"。该属性类型为 string，默认值为 0
2	start	表示有效日期范围的开始，字符串格式为"YYYY-MM-DD"。该属性类型为 string
3	end	表示有效日期范围的结束，字符串格式为"YYYY-MM-DD"。该属性类型为 string
4	fields	表示选择器的粒度，有效值为 year(选择器粒度为年)，month(选择器粒度为月份)和 day(选择器粒度为天)。该属性类型为 string，默认值为 day
5	bindchange	value 改变时触发 change 事件，event. detail = {value}。该属性类型为 eventhandle

5）mode 为 region

当 mode 为 region，其属性如表 3-40 所示。

表 3-40 mode 为 region 时 picker 组件的属性

序号	属 性	说 明
1	value	表示选中的省市区，默认选中每一列的第一个值。该属性类型为 array，默认值为[]
2	custom-item	可为每一列的顶部添加一个自定义的项。该属性类型为 string
3	bindchange	value 改变时触发 change 事件，event. detail = {value, code, postcode}，其中字段 code 是统计用区划代码，postcode 是邮政编码。该属性类型为 eventhandle

图 3-43 为 picker 组件的视图层示例代码。

```
<view class="section">
  <view class="section__title">普通选择器</view>
  <picker bindchange="bindPickerChange" value="{{index}}" range="{{array}}">
    <view class="picker">
      当前选择: {{array[index]}}
```

图 3-43 picker 组件的视图层示例代码

```
      </view>
    </picker>
  </view>
  <view class="section">
    <view class="section__title">多列选择器</view>
    <picker mode="multiSelector" bindchange="bindMultiPickerChange" bindcolumnchange="bindMultiPickerColumnChange" value="{{multiIndex}}" range="{{multiArray}}">
      <view class="picker">
        当前选择：{{multiArray[0][multiIndex[0]]}},{{multiArray[1][multiIndex[1]]}},{{multiArray[2][multiIndex[2]]}}
      </view>
    </picker>
  </view>
  <view class="section">
    <view class="section__title">时间选择器</view>
    <picker mode="time" value="{{time}}" start="09:01" end="21:01" bindchange="bindTimeChange">
      <view class="picker">
        当前选择：{{time}}
      </view>
    </picker>
  </view>

  <view class="section">
    <view class="section__title">日期选择器</view>
    <picker mode="date" value="{{date}}" start="2015-09-01" end="2017-09-01" bindchange="bindDateChange">
      <view class="picker">
        当前选择：{{date}}
      </view>
    </picker>
  </view>
  <view class="section">
    <view class="section__title">省市区选择器</view>
    <picker mode="region" bindchange="bindRegionChange" value="{{region}}" custom-item="{{customItem}}">
      <view class="picker">
        当前选择：{{region[0]}},{{region[1]}},{{region[2]}}
      </view>
    </picker>
  </view>
```

图 3-43　picker 组件的视图层示例代码（续）

上述代码中 picker 组件分别为普通选择器、多列选择器、时间选择器、日期选择器和省市区选择器。

2 picker-view 和 picker-view-column

picker-view 是嵌入页面的滚动选择器，picker-view-column 是滚动选择器子项，它仅可放置于 picker-view 中，其子节点的高度会自动设置成与 picker-view 的选中框的高度一致。它们的语法格式如图 3-44 所示。

```
<picker-view>
    <picker-view-column>

    </picker-view-column>
</picker-view>
```

图 3-44 picker-view 和 picker-view-column 组件的格式

如表 3-41 所示为 picker-view 组件的属性。

表 3-41 picker-view 组件的属性

序号	属 性	说 明
1	value	数组中的数字依次表示 picker-view 内的 picker-view-column 选择的第几项(下标从 0 开始),数字大于 picker-view-column 可选项长度时,选择最后一项。该属性的类型为 Array.<number>
2	indicator-style	设置选择器中间选中框的样式。该属性的类型为 string
3	indicator-class	设置选择器中间选中框的类名。该属性的类型为 string
4	mask-style	设置蒙层的样式。该属性的类型为 string
5	mask-class	设置蒙层的类名。该属性的类型为 string
6	bindchange	滚动选择时触发 change 事件,event.detail={value};value 为数组,表示 picker-view 内的 picker-view-column 当前选择的是第几项(下标从 0 开始)。该属性的类型为 eventhandle
7	bindpickstart	当滚动选择开始时候触发事件。该属性的类型为 eventhandle
8	bindpickend	当滚动选择结束时候触发事件。该属性的类型为 eventhandle

注意:滚动时在 iOS 自带振动反馈,可在系统设置—>声音与触感—>系统触感反馈中关闭。

图 3-45 为 picker-view 和 picker-view-column 组件的示例代码。

```
//视图层
<view>
  <view>{{year}}年{{month}}月{{day}}日</view>
  <picker-view indicator-style="height: 50px;" style="width: 100%; height: 300px;" value="{{value}}" bindchange="bindChange">
    <picker-view-column>
      <view wx:for="{{years}}" style="line-height: 50px">{{item}}年</view>
    </picker-view-column>
    <picker-view-column>
      <view wx:for="{{months}}" style="line-height: 50px">{{item}}月</view>
    </picker-view-column>
    <picker-view-column>
      <view wx:for="{{days}}" style="line-height: 50px">{{item}}日</view>
    </picker-view-column>
  </picker-view>
```

图 3-45 picker-view 和 picker-view-column 组件的示例代码

```
</view>
//逻辑层
const date = new Date()
const years = []
const months = []
const days = []

for (let i = 1990; i <= date.getFullYear(); i++) {
  years.push(i)
}

for (let i = 1; i <= 12; i++) {
  months.push(i)
}

for (let i = 1; i <= 31; i++) {
  days.push(i)
}

Page({
  data: {
    years: years,
    year: date.getFullYear(),
    months: months,
    month: 2,
    days: days,
    day: 2,
    value: [9999, 1, 1],
  },
  bindChange: function (e) {
    const val = e.detail.value
    this.setData({
      year: this.data.years[val[0]],
      month: this.data.months[val[1]],
      day: this.data.days[val[2]]
    })
  }
})
```

图 3-45 picker-view 和 picker-view-column 组件的示例代码(续)

在示例代码中,picker-view 组件中嵌入了三个 picker-view-column 组件,其中第一个 picker-view-column 组件显示年份,第二个 picker-view-column 组件显示月份,第三个 picker-view-column 组件显示日。由于 picker-view 组件绑定了 bindChange 函数,因此在三个 picker-view-column 组件中分别选择年月日时,将会显示在最上方的 view 上。

关于 picker、picker-view 和 picker-view-column 组件的更多内容,可访问以下链接。

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

https://developers.weixin.qq.com/miniprogram/dev/component/picker-view.html

https://developers.weixin.qq.com/miniprogram/dev/component/picker-view-column.html

3.4.6 slider

slider 是滑动选择器组件。它的语法格式如图 3-46 所示。

```
<slider>

</slider>
或
<slider        />
```

图 3-46 slider 组件的格式

如表 3-42 所示为 slider 组件的属性。

表 3-42 slider 组件的属性

序号	属 性	说 明
1	min	最小值。该属性的类型为 number，默认值为 0
2	max	最大值。该属性的类型为 number，默认值为 100
3	step	步长，取值必须大于 0，并且可被（max-min）整除。该属性的类型为 number，默认值为 1
4	disabled	是否禁用。该属性的类型为 boolean，默认值为 false
5	value	当前取值。该属性的类型为 number，默认值为 0
6	color	背景条的颜色（请使用 backgroundColor）。该属性的类型为 color，默认值为 #e9e9e9
7	selected-color	已选择的颜色（请使用 activeColor）。该属性的类型为 color，默认值为 #1aad19
8	activeColor	已选择的颜色。该属性的类型为 color，默认值为 #1aad19
9	backgroundColor	背景条的颜色。该属性的类型为 color，默认值为 #e9e9e9
10	block-size	滑块的大小，取值范围为 12～28。该属性的类型为 number，默认值为 28
11	block-color	滑块的颜色。该属性的类型为 color，默认值为 #ffffff
12	show-value	是否显示当前 value。该属性的类型为 boolean，默认值为 false
13	bindchange	完成一次拖动后触发的事件，event. detail = {value}。该属性的类型为 eventhandle
14	bindchanging	拖动过程中触发的事件，event. detail = {value}。该属性的类型为 eventhandle

如图 3-47 所示为 slider 的示例代码。

```
<view class = "page">
    <view class = "page__hd">
        <text class = "page__title"> slider </text>
        <text class = "page__desc">滑块</text>
    </view>
    <view class = "page__bd">
        <view class = "section section_gap">
```

图 3-47 slider 组件的示例代码

```
            <text class="section__title">设置 left/right icon</text>
            <view class="body-view">
                <slider bindchange="slider1change" left-icon="cancel" right-icon=
"success_no_circle"/>
            </view>
        </view>

        <view class="section section_gap">
            <text class="section__title">设置 step</text>
            <view class="body-view">
                <slider bindchange="slider2change" step="5"/>
            </view>
        </view>

        <view class="section section_gap">
            <text class="section__title">显示当前 value</text>
            <view class="body-view">
                <slider bindchange="slider3change" show-value/>
            </view>
        </view>

        <view class="section section_gap">
            <text class="section__title">设置最小/最大值</text>
            <view class="body-view">
                <slider bindchange="slider4change" min="50" max="200" show-value/>
            </view>
        </view>
    </view>
</view>
//逻辑层
var pageData = {}
for (var i = 1; i < 5; ++i) {
  (function (index) {
    pageData['slider${index}change'] = function (e) {
      console.log('slider${index}发生 change 事件,携带值为', e.detail.value)
    }
  })(i);
}
Page(pageData)
```

图 3-47　slider 组件的示例代码(续)

示例代码中有四个 slider,每一个 slider 都绑定了 bindchange 事件。在微信开发者工具中,第一个 slider 无法实现设置 left 或 right 的 icon;第二个 slider 设置了其步长(每拖动一次移动的距离)为 5;第三个 slider 设置了显示滑块当前值;第四个 slider 不仅设置了显示滑块当前值,还设置了滑块的最小和最大值。

关于 slider 组件的更多内容,可访问以下链接。

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

3.4.7 switch

switch 是开关选择器组件。它的语法格式如图 3-48 所示。

```
<switch>

</switch>
或
<switch      />
```

图 3-48 switch 组件的格式

如表 3-43 所示为 switch 组件的属性。

表 3-43 switch 组件的属性

序号	属 性	说 明
1	checked	是否选中。该属性的类型为 boolean，默认值为 false
2	disabled	是否禁用。该属性的类型为 boolean，默认值为 false
3	type	switch 的样式，其有效值为 switch 或 checkbox。该属性的类型为 string，默认值为 switch
4	color	switch 的颜色，同 css 的 color。该属性的类型为 string，默认值为 #04BE02
5	bindchange	checked 改变时触发 change 事件，event.detail={ value}。该属性的类型为 eventhandle

注意：switch 类型切换时在 iOS 自带振动反馈，可在系统设置—>声音与触感—>系统触感反馈中关闭。

图 3-49 为 switch 组件的示例代码。

```
//视图层
<view class = "page">
    <view class = "page__hd">
        <text class = "page__title"> switch </text >
        <text class = "page__desc">开关</text >
    </view >
    <view class = "page__bd">
        <view class = "section section_gap">
            <view class = "section__title"> type = "switch"</view >
            <view class = "body - view">
                <switch checked = "{{switch1Checked}}" bindchange = "switch1Change"/>
            </view >
        </view >

        <view class = "section section_gap">
```

图 3-49 switch 组件的示例代码

```
            <view class="section__title">type="checkbox"</view>
            <view class="body-view">
                <switch type="checkbox" checked="{{switch2Checked}}" bindchange=
"switch2Change"/>
            </view>
        </view>
    </view>
</view>
//逻辑层
var pageData = {
  data: {
    switch1Checked: true,
    switch2Checked: false,
    switch1Style: '',
    switch2Style: 'text-decoration: line-through'
  }
}
for (var i = 1; i <= 2; ++i) {
  (function (index) {
    pageData['switch${index}Change'] = function (e) {
      console.log('switch${index}发生 change 事件,携带值为', e.detail.value)
      var obj = {}
      obj['switch${index}Checked'] = e.detail.value
      this.setData(obj)
      obj = {}
      obj['switch${index}Style'] = e.detail.value ? '' : 'text-decoration: line-through'
      this.setData(obj)
    }
  })(i)
}
Page(pageData)
```

图 3-49 switch 组件的示例代码(续)

示例代码中有两个 switch 组件,分别绑定了 switch1Change 和 switch2Change,逻辑层代码中统一处理了这两个事件。第一个 switch 的 type 为默认的 switch;第二个 switch 的 type 为 checkbox。

关于 switch 组件的更多内容,可访问以下链接。

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

3.4.8 form

form 是表单组件,当点击 form 表单中 form-type 为 submit 的 button 组件时,会将表单组件中提供给用户输入的 switch、input、checkbox、slider、radio 和 picker 的组件的 value 值提交(此时需要在表单内各组件中加上 name 用作 key)。它的语法格式如图 3-50 所示。

```
<form>

</form>
```

图 3-50 form 组件的格式

如表 3-44 所示为 form 组件的属性。

表 3-44 form 组件的属性

序号	属 性	说 明
1	report-submit	是否返回 formId 用于发送模板消息。该属性的类型为 boolean，默认值为 false
2	report-submit-timeout	等待一段时间(毫秒数)以确认 formId 是否生效。如果未指定这个参数，formId 有很小的概率是无效的(如遇到网络失败的情况)。指定这个参数将可以检测 formId 是否有效，以这个参数的时间作为这项检测的超时时间。如果失败，将返回 requestFormId:fail 开头的 formId。该属性的类型为 number，默认值为 0
3	bindsubmit	携带 form 中的数据触发 submit 事件，event.detail = {value : {'name': 'value'},formId: ''}。该属性的类型为 eventhandle
4	bindreset	表单重置时会触发 reset 事件。该属性的类型为 eventhandle

如图 3-51 所示为表单组件的示例代码。

```
//视图层
<form bindsubmit="formSubmit" bindreset="formReset">
  <view class="section section_gap">
    <view class="section__title">switch</view>
    <switch name="switch"/>
  </view>
  <view class="section section_gap">
    <view class="section__title">slider</view>
    <slider name="slider" show-value></slider>
  </view>

  <view class="section">
    <view class="section__title">input</view>
    <input name="input" placeholder="please input here" />
  </view>
  <view class="section section_gap">
    <view class="section__title">radio</view>
    <radio-group name="radio-group">
      <label><radio value="radio1"/>radio1</label>
      <label><radio value="radio2"/>radio2</label>
    </radio-group>
  </view>
  <view class="section section_gap">
    <view class="section__title">checkbox</view>
    <checkbox-group name="checkbox">
      <label><checkbox value="checkbox1"/>checkbox1</label>
      <label><checkbox value="checkbox2"/>checkbox2</label>
    </checkbox-group>
  </view>
  <view class="btn-area">
```

图 3-51 form 组件的示例代码

```
      <button formType = "submit"> Submit </button>
      <button formType = "reset"> Reset </button>
    </view>
  </form>
  //逻辑层
  Page({
    formSubmit: function (e) {
      console.log('form 发生了 submit 事件,携带数据为: ', e.detail.value)
    },
    formReset: function () {
      console.log('form 发生了 reset 事件')
    }
  })
```

图 3-51　form 组件的示例代码（续）

示例代码中包括的组件较多，这些组件间的关系如图 3-52 所示。

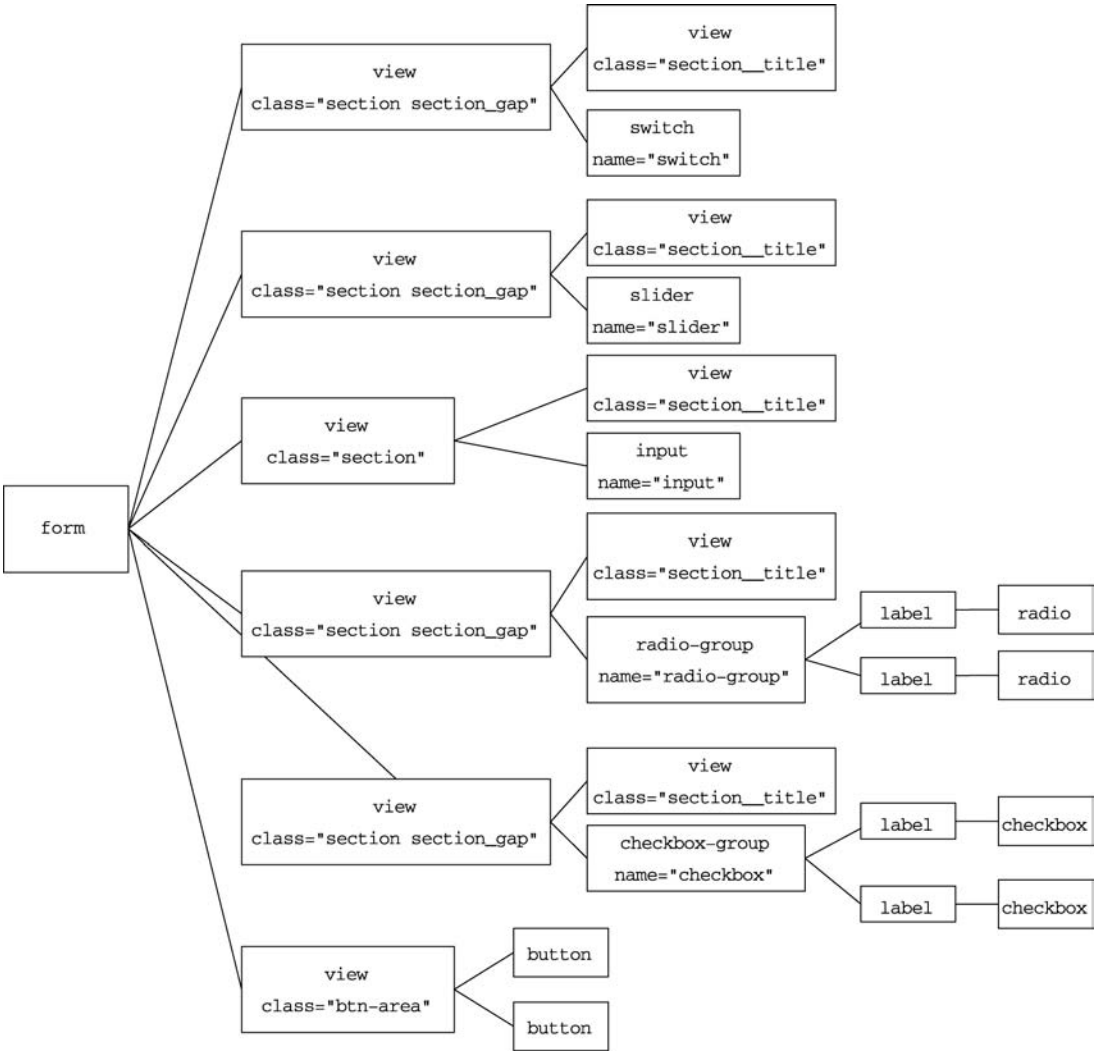

图 3-52　示例代码中组件关系

关于 form 组件的更多内容，可访问以下链接。

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

3.5 导航组件

3.5.1 functional-page-navigator

functional-page-navigator 组件仅在插件中有效，用于跳转到插件功能页。它的语法格式如图 3-53 所示。

```
<functional-page-navigator>

</functional-page-navigator>
```

图 3-53 functional-page-navigator 组件的格式

如表 3-45 所示为 functional-page-navigator 组件的属性。

表 3-45 functional-page-navigator 组件的属性

序号	属 性	说 明
1	version	跳转到的小程序版本，线上版本必须设置为 release。该属性的类型为 string，默认值为 release(正式版)。该属性的值还可以为 develop(开发版)和 trial(体验版)
2	name	要跳转到的功能页。该属性的类型为 string，其合法值为 loginAndGetUserInfo(用户信息功能页)和 requestPayment(支付功能页)
3	args	功能页参数，参数格式与具体功能页相关。该属性的类型为 object
4	bindsuccess	功能页返回，且操作成功时触发，detail 格式与具体功能页相关。该属性的类型为 eventhandler
5	bindfail	功能页返回，且操作失败时触发，detail 格式与具体功能页相关。该属性的类型为 eventhandler
6	bindcancel	因用户操作从功能页返回时触发。该属性的类型为 eventhandler

使用 functional-page-navigator 组件时，请注意以下问题。

(1) 功能页是插件所有者小程序中的一个特殊页面，开发者不能自定义这个页面的外观；

(2) 在功能页展示时，一些与界面展示相关的接口将被禁用(接口调用返回 fail)；

(3) 这个组件本身可以在开发者工具中使用，但功能页的跳转目前不支持在开发者工具中调试，请在真机上测试。

图 3-54 为组件 functional-page-navigator 的示例代码。

```
//视图层
<functional-page-navigator name="loginAndGetUserInfo" bind:success="loginSuccess">
  <button>登录到插件</button>
</functional-page-navigator>

//逻辑层
```

图 3-54 functional-page-navigator 组件的示例代码

```
Component({
  methods: {
    loginSuccess: function(e) {
      console.log(e.detail.code)      // wx.login 的 code
      console.log(e.detail.userInfo) // wx.getUserInfo 的 userInfo
    }
  }
})
```

图 3-54　functional-page-navigator 组件的示例代码(续)

关于 functional-page-navigator 组件的更多内容,可访问以下链接。

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

3.5.2　navigator

navigator 是页面链接组件,它的语法格式如图 3-55 所示。

```
<navigator>

</navigator>
```

图 3-55　navigator 组件的格式

如表 3-46 所示为 navigator 组件的属性。

表 3-46　navigator 组件的属性

序号	属　性	说　明
1	target	在哪个目标上发生跳转,默认当前小程序。该属性的类型为 string,默认值为 self(当前小程序)。该属性的值还可以是 miniProgram(其他小程序)
2	url	当前小程序内的跳转链接。该属性的类型为 string
3	open-type	跳转方式。该属性的类型为 string,默认值为 navigate(对应 wx.navigateTo 或 wx.navigateToMiniProgram 的功能)。该属性的合法值还有 redirect(对应 wx.redirectTo 的功能),switchTab(对应 wx.switchTab 的功能),reLaunch(对应 wx.reLaunch 的功能),navigateBack(对应 wx.navigateBack 的功能)和 exit(退出小程序,target="miniProgram"时生效)
4	delta	当 open-type 为'navigateBack'时有效,表示回退的层数。该属性的类型为 number,默认值为 1
5	app-id	当 target="miniProgram"时有效,要打开的小程序 appId。该属性的类型为 string
6	path	当 target="miniProgram"时有效,打开的页面路径,如果为空则打开首页。该属性的类型为 string
7	extra-data	当 target="miniProgram"时有效,需要传递给目标小程序的数据,目标小程序可在 App.onLaunch(),App.onShow()中获取到这份数据。该属性的类型为 object

续表

序号	属　性	说　明
8	version	当 target="miniProgram"时有效，要打开的小程序版本。该属性的类型为 string，默认值为 release(正式版，仅在当前小程序为开发版或体验版时此参数有效；如果当前小程序是正式版，则打开的小程序必定是正式版)。该属性的值还可以为 develop(开发版)和 trial(体验版)
9	hover-class	指定点击时的样式类，当 hover-class="none"时，没有点击态效果。该属性的类型为 string，默认值为 navigator-hover
10	hover-stop-propagation	指定是否阻止本节点的祖先节点出现点击态。该属性的类型为 boolean，默认值为 false
11	hover-start-time	按住后多久出现点击态，单位为毫秒。该属性的类型为 number，默认值为 50
12	hover-stay-time	手指松开后点击态保留时间，单位为毫秒。该属性的类型为 number，默认值为 600
13	bindsuccess	当 target="miniProgram"时有效，跳转小程序成功。该属性的类型为 string
14	bindfail	当 target="miniProgram"时有效，跳转小程序失败。该属性的类型为 string
15	bindcomplete	当 target="miniProgram"时有效，跳转小程序完成。该属性的类型为 string

使用 navigator 组件，要注意以下限制。

(1) 需要用户确认跳转。从 2.3.0 版本开始，在跳转至其他小程序前，将统一增加弹窗，询问是否跳转，用户确认后才可以跳转其他小程序。如果用户点击取消，则回调 fail cancel。

(2) 每个小程序可跳转的其他小程序数量限制为不超过 10 个。从 2.4.0 版本以及指定日期(具体待定)开始，开发者提交新版小程序代码时，如使用了跳转其他小程序功能，则需要在代码配置中声明将要跳转的小程序名单，限定不超过 10 个，否则将无法通过审核。该名单可在发布新版时更新，不支持动态修改。调用此接口时，所跳转的 appId 必须在配置列表中，否则回调 fail appId "${appId}" is not in navigateToMiniProgramAppIdList。

(3) 在开发者工具上调用此 API 并不会真实地跳转到另外的小程序，但是开发者工具会校验本次调用跳转是否成功。

(4) 开发者工具上支持被跳转的小程序处理接收参数的调试。

(5) navigator-hover 默认为{background-color：rgba(0，0，0，0.1)；opacity：0.7；}，navigator 的子节点背景色应为透明色。

图 3-56 为 navigator 的示例代码。

```
//视图层
<!-- index.wxml -->
<view class="btn-area">
  <navigator url="/page/navigate/navigate?title=navigate" hover-class="navigator-hover">跳转到新页面</navigator>
  <navigator url="../../redirect/redirect/redirect?title=redirect" open-type="redirect" hover-class="other-navigator-hover">在当前页打开</navigator>
```

图 3-56　navigator 组件的示例代码

```
  <! -- < navigator url = "/page/index/index" open - type = "switchTab" hover - class = "other -
navigator - hover">切换 Tab </navigator >
  < navigator target = "miniProgram" open - type = "navigate" app - id = "" path = "" extra - data = ""
version = "release">打开绑定的小程序</navigator > -- >
</view >

<! -- navigator.wxml -- >
< view style = "text - align:center"> {{title}} </view >
< view > 点击左上角返回回到之前页面 </view >
<! -- redirect.wxml -- >
< view style = "text - align:center"> {{title}} </view >
< view > 点击左上角返回回到上级页面 </view >
Page({
  onLoad: function(options) {
    this.setData({
      title: options.title
    })
  }
})
/ ** index.wxss ** /
.navigator - hover {
  color:blue;
}
.other - navigator - hover {
  color:red;
}
```

图 3-56 navigator 组件的示例代码(续)

示例代码中有两个 navigator 组件,第一个组件的跳转方式为默认,因此会跳转到新页面;第二个组件的跳转方式设置为 redirect,即在当前页面打开(不跳转)。

注意:被注释的代码中也有两个 navigator 组件,其中 open-type 为"switchTab",需要读者修改 app.json 文件增加对应的页面才可以使切换 Tab 生效,同理 open-type 为"navigate"时,也需要读者设置 app-id 和 path 等信息才可以实现打开绑定的小程序。

关于 navigator 组件的更多内容,可访问以下链接。

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

3.6 媒体组件

3.6.1 audio

audio 是音频组件,它的语法格式如图 3-57 所示。该组件从 1.6.0 版本开始不再维护,建议使用能力更强的 wx.createInnerAudioContext 接口。

```
<audio>

</audio>
```

图 3-57 audio 组件的格式

如表 3-47 所示为 audio 组件的属性。

表 3-47 audio 组件的属性

序号	属 性	说 明
1	id	audio 组件的唯一标识符。该属性的类型为 string
2	src	要播放音频的资源地址。该属性的类型为 string
3	loop	是否循环播放。该属性的类型为 boolean，默认值为 false
4	controls	是否显示默认控件。该属性的类型为 boolean，默认值为 false
5	poster	默认控件上的音频封面的图片资源地址，如果 controls 属性值为 false 则设置 poster 无效。该属性的类型为 string
6	name	默认控件上的音频名字，如果 controls 属性值为 false 则设置 name 无效。该属性的类型为 string，默认值为未知音频
7	author	默认控件上的作者名字，如果 controls 属性值为 false 则设置 author 无效。该属性的类型为 string，默认值为未知作者
8	binderror	当发生错误时触发 error 事件，detail={errMsg:MediaError.code}。该属性的类型为 eventhandle。MediaError.code 返回的错误码可以为 1(获取资源被用户禁止)，2(网络错误)，3(解码错误)和 4(不合适资源)
9	bindplay	当开始/继续播放时触发 play 事件。该属性的类型为 eventhandle
10	bindpause	当暂停播放时触发 pause 事件。该属性的类型为 eventhandle
11	bindtimeupdate	当播放进度改变时触发 timeupdate 事件，detail={currentTime，duration}。该属性的类型为 eventhandle
12	bindended	当播放到末尾时触发 ended 事件。该属性的类型为 eventhandle

注意：创建 audio 上下文 AudioContext 对象的接口是 wx.createAudioContext(string id，Object this)，其中 string id 为 audio 组件的 id，Object this 是当前组件实例的 this，用以操作自定义组件内 audio 组件。从基础库 1.6.0 开始，本接口停止维护，请使用 wx.createInnerAudioContext 代替。

图 3-58 为 audio 组件的示例代码。

```
//视图层代码
<audio poster="{{poster}}" name="{{name}}" author="{{author}}" src="{{src}}" id="
myAudio" controls loop></audio>

<button type="primary" bindtap="audioPlay">播放</button>
<button type="primary" bindtap="audioPause">暂停</button>
<button type="primary" bindtap="audio14">设置当前播放时间为 14 秒</button>
<button type="primary" bindtap="audioStart">回到开头</button>

//逻辑层代码
```

图 3-58 audio 组件的示例代码

```
Page({
  onReady: function (e) {
    // 使用 wx.createAudioContext 获取 audio 上下文 context
    this.audioCtx = wx.createAudioContext('myAudio')
  },
  data: {
    poster: 'http://y.gtimg.cn/music/photo_new/T002R300x300M000003rsKF44GyaSk.jpg?max_age=
2592000',
    name: '此时此刻',
    author: '许巍',
    src: 'http://ws.stream.qqmusic.qq.com/M500001VfvsJ21xFqb.mp3?guid=ffffffff82def4af4-
b12b3cd9337d5e7&uin=346897220&vkey=6292F51E1E384E06DCBDC9AB7C49FD713D632D313AC4858BACB8-
DDD29067D3C601481D36E62053BF8DFEAF74C0A5CCFADD6471160CAF3E6A&fromtag=46',
  },
  audioPlay: function () {
    this.audioCtx.play()
  },
  audioPause: function () {
    this.audioCtx.pause()
  },
  audio14: function () {
    this.audioCtx.seek(14)
  },
  audioStart: function () {
    this.audioCtx.seek(0)
  }
})
```

图 3-58 audio 组件的示例代码(续)

视图层代码中有一个 audio 组件和四个 button 组件。四个 button 组件分别对应播放、暂停、设置当前播放时间为 14 秒和回到开头这四个功能,它们是通过绑定逻辑层的 audioPlay,audioPause,audio14 和 audioStart 实现的。

关于 audio 组件的更多内容,可访问以下链接。

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

注意:若读者在上述链接对应的 audio.html 中点击“在开发者工具中预览效果”,代码与图 3-58 不完全一样,请参见本章配套的名为“ex030601_audio_在开发者工具中预览效果”文件夹下的代码。图 3-58 中的代码存放在本章代码的文件夹下。

3.6.2 image

image 是图片组件,它支持 JPG、PNG、SVG 格式,该组件从 2.3.0 起支持云文件 ID,它的语法格式如图 3-59 所示。

```
<image>

</image>
```

图 3-59 image 组件的格式

如表 3-48 所示为 image 组件的属性。

表 3-48 image 组件的属性

序号	属 性	说 明
1	src	图片资源地址。该属性的类型为 string
2	mode	图片裁剪、缩放的模式。该属性的类型为 string，默认值为 scaleToFill
3	lazy-load	图片懒加载，在即将进入一定范围（上下三屏）时才开始加载。该属性的类型为 boolean，默认值为 false
4	show-menu-by-longpress	开启长按图片显示识别小程序码菜单。该属性的类型为 boolean，默认值为 false
5	binderror	当错误发生时触发，event.detail = {errMsg}。该属性的类型为 eventhandle
6	bindload	当图片载入完毕时触发，event.detail = {height, width}。该属性的类型为 eventhandle

如表 3-49 所示为 mode 属性的合法取值和说明。

表 3-49 mode 属性的合法取值和说明

序号	值	说 明
1	scaleToFill	缩放模式，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。scaleToFill 为默认值
2	aspectFit	缩放模式，保持纵横比缩放图片，使图片的长边能完全显示出来。也就是说，可以完整地将图片显示出来
3	aspectFill	缩放模式，保持纵横比缩放图片，只保证图片的短边能完全显示出来。也就是说，图片通常只在水平或垂直方向是完整的，另一个方向将会发生截取
4	widthFix	缩放模式，宽度不变，高度自动变化，保持原图宽高比不变
5	top	裁剪模式，不缩放图片，只显示图片的顶部区域
6	bottom	裁剪模式，不缩放图片，只显示图片的底部区域
7	center	裁剪模式，不缩放图片，只显示图片的中间区域
8	left	裁剪模式，不缩放图片，只显示图片的左边区域
9	right	裁剪模式，不缩放图片，只显示图片的右边区域
10	top left	裁剪模式，不缩放图片，只显示图片的左上边区域
11	top right	裁剪模式，不缩放图片，只显示图片的右上边区域
12	bottom left	裁剪模式，不缩放图片，只显示图片的左下边区域
13	bottom right	裁剪模式，不缩放图片，只显示图片的右下边区域

使用 image 组件时，要注意以下问题。

（1）image 组件默认宽度 300px、高度 225px。

（2）image 组件中二维码/小程序码图片不支持长按识别。仅在 wx.previewImage 中支持长按识别。

图 3-60 为 image 组件的示例代码。

```
//视图层
<view class="page">
  <view class="page__hd">
    <text class="page__title">image</text>
    <text class="page__desc">图片</text>
  </view>
  <view class="page__bd">
    <view class="section section_gap" wx:for="{{array}}" wx:for-item="item">
      <view class="section__title">{{item.text}}</view>
      <view class="section__ctn">
        <image style="width: 200px; height: 200px; background-color: #eeeeee;" mode="
{{item.mode}}" src="{{src}}"></image>
      </view>
    </view>
  </view>
</view>
//逻辑层
Page({
  data: {
    array: [{
      mode: 'scaleToFill',
      text: 'scaleToFill: 不保持纵横比缩放图片,使图片完全适应'
    }, {
      mode: 'aspectFit',
      text: 'aspectFit: 保持纵横比缩放图片,使图片的长边能完全显示出来'
    }, {
      mode: 'aspectFill',
      text: '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'
    }, {
      mode: 'top',
      text: 'top: 不缩放图片,只显示图片的顶部区域'
    }, {
      mode: 'bottom',
      text: 'bottom: 不缩放图片,只显示图片的底部区域'
    }, {
      mode: 'center',
      text: 'center: 不缩放图片,只显示图片的中间区域'
    }, {
      mode: 'left',
      text: 'left: 不缩放图片,只显示图片的左边区域'
    }, {
      mode: 'right',
      text: 'right: 不缩放图片,只显示图片的右边边区域'
    }, {
      mode: 'top left',
      text: 'top left: 不缩放图片,只显示图片的左上边区域'
    }, {
      mode: 'top right',
      text: 'top right: 不缩放图片,只显示图片的右上边区域'
```

图 3-60　image 组件的示例代码

```
      }, {
        mode: 'bottom left',
        text: 'bottom left: 不缩放图片,只显示图片的左下边区域'
      }, {
        mode: 'bottom right',
        text: 'bottom right: 不缩放图片,只显示图片的右下边区域'
      }],
      src: '../resources/cat.jpg'
    },
    imageError: function(e) {
      console.log('image3 发生 error 事件,携带值为', e.detail.errMsg)
    }
  })
```

图 3-60 image 组件的示例代码(续)

示例代码中展示了 image 组件不同 mode 下图片显示的效果,有兴趣的读者可以运行教材配套的代码感受下。

关于 image 组件的更多内容,可访问以下链接。

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

3.6.3 video

video 是视频组件,它的语法格式如图 3-61 所示。

```
<video>

</video>
```

图 3-61 video 组件的格式

如表 3-50 所示为 video 组件的属性。

表 3-50 video 组件的属性

序号	属 性	说 明
1	src	要播放视频的资源地址,支持云文件 ID(2.3.0)。该属性的类型为 string,为必填属性
2	duration	指定视频时长。该属性的类型为 number
3	controls	是否显示默认播放控件(播放/暂停按钮、播放进度、时间)。该属性的类型为 boolean,默认值为 true
4	danmu-list	弹幕列表。该属性的类型为 Array.<object>
5	danmu-btn	是否显示弹幕按钮,只在初始化时有效,不能动态变更。该属性的类型为 boolean,默认值为 false
6	enable-danmu	是否展示弹幕,只在初始化时有效,不能动态变更。该属性的类型为 boolean,默认值为 false
7	autoplay	是否自动播放。该属性的类型为 boolean,默认值为 false
8	loop	是否循环播放。该属性的类型为 boolean,默认值为 false

续表

序号	属　性	说　明
9	muted	是否静音播放。该属性的类型为 boolean，默认值为 false
10	initial-time	指定视频初始播放位置。该属性的类型为 number，默认值为 0
11	page-gesture	在非全屏模式下，是否开启亮度与音量调节手势（废弃，见 vslide-gesture）。该属性的类型为 boolean，默认值为 false
12	direction	设置全屏时视频的方向，不指定则根据宽高比自动判断。该属性的类型为 number，其合法值为 0（正常竖向），90（屏幕逆时针 90 度）和－90（屏幕顺时针 90 度）
13	show-progress	若不设置，宽度大于 240 时才会显示。该属性的类型为 boolean，默认值为 true
14	show-fullscreen-btn	是否显示全屏按钮。该属性的类型为 boolean，默认值为 true
15	show-play-btn	是否显示视频底部控制栏的播放按钮。该属性的类型为 boolean，默认值为 true
16	show-center-play-btn	是否显示视频中间的播放按钮。该属性的类型为 boolean，默认值为 true
17	enable-progress-gesture	是否开启控制进度的手势。该属性的类型为 boolean，默认值为 true
18	object-fit	当视频大小与 video 容器大小不一致时，视频的表现形式。该属性的类型为 string，默认值为 contain（即包含）。该属性的值还可以为 fill（即填充）和 cover（即覆盖）
19	poster	视频封面的图片网络资源地址或云文件 ID（2.3.0）。若 controls 属性值为 false 则设置 poster 无效。该属性的类型为 string
20	show-mute-btn	是否显示静音按钮。该属性的类型为 boolean，默认值为 false
21	title	视频的标题，全屏时在顶部展示。该属性的类型为 string
22	play-btn-position	播放按钮的位置。该属性的类型为 string，默认值为 bottom（即 controls bar 上）。该属性的值还可以为 center（即视频中间）
23	enable-play-gesture	是否开启播放手势，即双击切换播放/暂停。该属性的类型为 boolean，默认值为 false
24	auto-pause-if-navigate	当跳转到其他小程序页面时，是否自动暂停本页面的视频。该属性的类型为 boolean，默认值为 true
25	auto-pause-if-open-native	当跳转到其他微信原生页面时，是否自动暂停本页面的视频。该属性的类型为 boolean，默认值为 true
26	vslide-gesture	在非全屏模式下，是否开启亮度与音量调节手势（同 page-gesture）。该属性的类型为 boolean，默认值为 false
27	vslide-gesture-in-fullscreen	在全屏模式下，是否开启亮度与音量调节手势。该属性的类型为 boolean，默认值为 true
28	ad-unit-id	视频前贴广告单元 ID，该属性的类型为 string
29	bindplay	当开始/继续播放时触发 play 事件。该属性的类型为 eventhandle
30	bindpause	当暂停播放时触发 pause 事件。该属性的类型为 eventhandle
31	bindended	当播放到末尾时触发 ended 事件。该属性的类型为 eventhandle
32	bindtimeupdate	播放进度变化时触发，event.detail = {currentTime, duration}。触发频率 250ms 一次。该属性的类型为 eventhandle

续表

序号	属　　性	说　　明
33	bindfullscreenchange	视频进入和退出全屏时触发，event.detail = {fullScreen, direction}，direction 有效值为 vertical 或 horizontal。该属性的类型为 eventhandle
34	bindwaiting	视频出现缓冲时触发。该属性的类型为 eventhandle
35	binderror	视频播放出错时触发。该属性的类型为 eventhandle
36	bindprogress	加载进度变化时触发，只支持一段加载。event.detail = {buffered}，百分比。该属性的类型为 eventhandle

使用 video 组件时，请注意以下方面。

(1) video 默认宽度 300px、高度 225px，可通过 wxss 设置宽高。

(2) 从 2.4.0 起 video 支持同层渲染。对于 iOS，video 支持 mp4、mov、m4v、3gp、avi 和 m3u8 视频格式，支持 H.264、HEVC 和 MPEG-4 编码格式；对于 Android，video 支持 mp4、3gp、m3u8 和 webm 视频格式，支持 H.264、HEVC、MPEG-4 和 VP9 编码格式。

图 3-62 为 video 组件的示例代码。

```
//视图层
<view class="section tc">
  <video src="{{src}}" controls></video>
  <view class="btn-area">
    <button bindtap="bindButtonTap">获取视频</button>
  </view>
</view>

<view class="section tc">
  <video id="myVideo" src="http://wxsnsdy.tc.qq.com/105/20210/snsdyvideodownload?filekey=
30280201010421301f0201690402534804102ca905ce620b1241b726bc41dcff44e00204012882540400&bizid=
1023&hy=SH&fileparam=302c020101042530230204136ffd93020457e3c4ff02024ef202031e8d7f02030f
42400204045a320a0201000400" danmu-list="{{danmuList}}" enable-danmu danmu-btn controls>
</video>
  <view class="btn-area">
    <button bindtap="bindButtonTap">获取视频</button>
    <input bindblur="bindInputBlur"/>
    <button bindtap="bindSendDanmu">发送弹幕</button>
  </view>
</view>

//逻辑层
function getRandomColor () {
  let rgb = []
  for (let i = 0 ; i < 3; ++i){
    let color = Math.floor(Math.random() * 256).toString(16)
    color = color.length == 1 ? '0' + color : color
    rgb.push(color)
  }
```

图 3-62　video 组件的示例代码

```
    return '#' + rgb.join('')
  }
  Page({
    onReady: function (res) {
      this.videoContext = wx.createVideoContext('myVideo')
    },
    inputValue: '',
    data: {
      src: '',
      danmuList: [
        {
          text: '第 1s 出现的弹幕',
          color: '#ff0000',
          time: 1
        },
        {
          text: '第 3s 出现的弹幕',
          color: '#ff00ff',
          time: 3
      }]
    },
    bindInputBlur: function(e) {
      this.inputValue = e.detail.value
    },
    bindButtonTap: function() {
      var that = this
      wx.chooseVideo({
        sourceType: ['album', 'camera'],
        maxDuration: 60,
        camera: ['front','back'],
        success: function(res) {
          that.setData({
            src: res.tempFilePath
          })
        }
      })
    },
    bindSendDanmu: function () {
      this.videoContext.sendDanmu({
        text: this.inputValue,
        color: getRandomColor()
      })
    }
  })
```

图 3-62　video 组件的示例代码（续）

示例代码中的组件的关系如图 3-63 所示。

示例代码中的第一个 video 需要通过按下 button 获取，第二个 video 则是从指定 URL 获取的一段视频，可以通过在 input 中输入相应的内容并按下 button 发送弹幕。

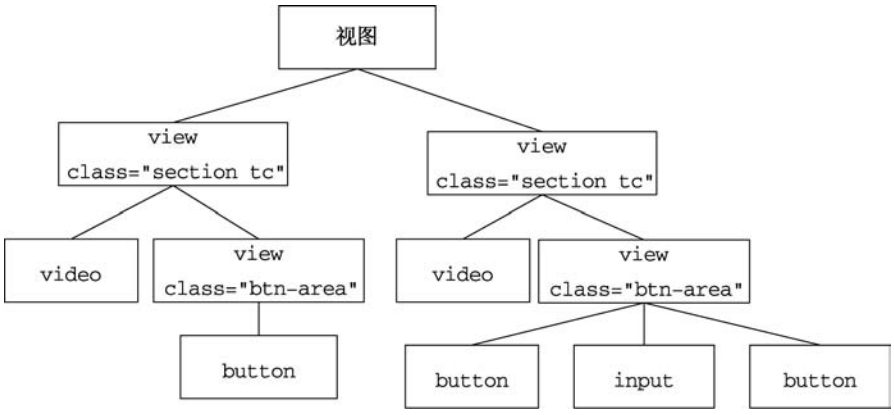

图 3-63 示例代码中组件的关系

关于 video 组件的更多内容，可访问以下链接。

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

注意：若读者在上述链接对应的 video.html 中点击“在开发者工具中预览效果”，代码与图 3-62 不完全一样，请参见本章配套的名为“ex030603_video_在开发者工具中预览效果”的文件夹下的代码。图 3-62 中的代码存放在本章代码的文件夹下。

3.6.4 camera

camera 是系统相机组件，使用时需要用户授权(scope.camera)，升级微信客户端至 6.7.3 该组件将具有扫码功能。它的语法格式如图 3-64 所示。

```
<camera>

</camera>
```

图 3-64 camera 组件的格式

如表 3-51 所示为 camera 组件的属性。

表 3-51 camera 组件的属性

序号	属 性	说 明
1	mode	应用模式，只在初始化时有效，不能动态变更。该属性的类型为 string，默认值为 normal(相机模式)。该属性的值还可以为 scanCode(扫码模式)
2	device-position	摄像头朝向。该属性的类型为 string，默认值为 back(后置)。该属性的值还可以为 front(前置)
3	flash	闪光灯，值为 auto(自动)、on(打开)、off(关闭)和 torch(常亮)。该属性的类型为 string，默认值为 auto
4	frame-size	指定期望的相机帧数据尺寸。该属性的类型为 string，默认值为 medium(中尺寸帧数据)。该属性的值还可以为 small(小尺寸帧数据)和 large(大尺寸帧数据)

续表

序号	属　　性	说　　明
5	bindstop	摄像头在非正常终止时触发，如退出后台等情况。该属性的类型为 eventhandle
6	binderror	用户不允许使用摄像头时触发。该属性的类型为 eventhandle
7	bindinitdone	相机初始化完成时触发。该属性的类型为 eventhandle
8	bindscancode	在扫码识别成功时触发，仅在 mode＝"scanCode"时生效。该属性的类型为 eventhandle

使用 camera 组件时，还要注意以下问题。

(1) 同一页面只能插入一个 camera 组件。

(2) 请注意原生组件使用限制。

(3) onCameraFrame 接口根据 frame-size 返回不同尺寸的原始帧数据，与 Camera 组件展示的图像不同，其实际像素值由系统决定。

图 3-65 是 camera 组件的示例代码。

```
//视图层
<camera device - position = "back" flash = "off" binderror = "error" style = "width: 100 %;
height: 300px;"></camera>
<button type = "primary" bindtap = "takePhoto">拍照</button>
<view>预览</view>
<image mode = "widthFix" src = "{{src}}"></image>
//逻辑层
Page({
  takePhoto() {
    const ctx = wx.createCameraContext()
    ctx.takePhoto({
      quality: 'high',
      success: (res) => {
        this.setData({
          src: res.tempImagePath
        })
      }
    })
  },
  error(e) {
    console.log(e.detail)
  }
})
```

图 3-65　camera 组件的示例代码

示例代码中有一个 camera 组件、一个 button 组件和一个 image 组件，其中 button 组件绑定了 takePhoto 事件用于拍照并将照片显示在 image 组件中。

关于 camera 组件的更多内容，可访问以下链接。

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

注意：若读者在上述链接对应的 camera.html 中点击“在开发者工具中预览效果”，代码与图 3-65 不完全一样，请参见本章配套的名为“ex030605_camera_在开发者工具中预览效果”的文件夹下的代码。图 3-65 中的代码在存放本章代码的文件夹下。

3.6.5 live-player

live-player 是实时音视频播放组件，它的语法格式如图 3-66 所示。

```
<live-player>

</live-player>
或
<live-player            />
```

图 3-66 live-player 组件的格式

live-player 暂时只针对如表 3-52 所示的国内主体类目的小程序开放，需要先通过类目审核，然后在小程序管理后台“开发”-“接口设置”中自助开通该组件权限。

表 3-52 国内小程序主体类目

序号	一级类目/主体类型	二 级 类 目	说 明
1	社交	直播	涉及娱乐性质，如明星直播、生活趣事直播、宠物直播等。选择该类目后首次提交代码审核，需经当地互联网主管机关审核确认，预计审核时长 7 天左右
2	教育	在线视频课程	网课、在线培训、讲座等教育类直播
3	医疗	互联网医院，公立医院	问诊、大型健康讲座等直播
4	金融	银行、信托、基金、证券/期货、证券、期货投资咨询、保险、征信业务、新三板信息服务平台、股票信息服务平台（港股/美股）、消费金融	金融产品视频客服理赔、金融产品推广直播等
5	汽车	汽车预售服务	汽车预售、推广直播
6	政府主体账号	/	政府相关工作推广直播、领导讲话直播等
7	工具	视频客服	不涉及以上几类内容的一对一视频客服服务，如企业售后一对一视频服务等

如表 3-53 所示为 live-player 组件的属性。

表 3-53 live-player 组件的属性

序号	属 性	说 明
1	src	音视频地址。目前仅支持 flv、rtmp 格式。该属性的类型是 string
2	mode	模式。该属性的类型是 string，默认值为 live(直播)。该属性的值还可以为 RTC(实时通话，该模式时延更低)

续表

序号	属　性	说　明
3	autoplay	自动播放。该属性的类型是 boolean，默认值为 false
4	muted	是否静音。该属性的类型是 boolean，默认值为 false
5	orientation	画面方向。该属性的类型是 string，默认值为 vertical(竖直)。该属性的值还可以为 horizontal(水平)
6	object-fit	填充模式，可选值有 contain(图像长边填满屏幕，短边区域会被填充)，fillCrop(图像铺满屏幕，超出显示区域的部分将被截掉)。该属性的类型是 string，默认值为 contain
7	background-mute	进入后台时是否静音(已废弃，默认退台静音)。该属性的类型是 boolean，默认值为 false
8	min-cache	最小缓冲区，单位 s(RTC 模式推荐 0.2s)。该属性的类型是 number，默认值为 1
9	max-cache	最大缓冲区，单位 s(RTC 模式推荐 0.8s)。该属性的类型是 number，默认值为 3
10	sound-mode	声音输出方式。该属性的类型是 string，默认值为 speaker(扬声器)。该属性的值还可以为 ear(听筒)
11	auto-pause-if-navigate	当跳转到其他小程序页面时，是否自动暂停本页面的实时音视频播放。该属性的类型是 boolean，默认值为 true
12	auto-pause-if-open-native	当跳转到其他微信原生页面时，是否自动暂停本页面的实时音视频播放。该属性的类型是 boolean，默认值为 true
13	bindstatechange	播放状态变化事件，detail={code}。该属性的类型是 eventhandle
14	bindfullscreenchange	全屏变化事件，detail={direction，fullScreen}。该属性的类型是 eventhandle
15	bindnetstatus	网络状态通知，detail={info}。该属性的类型是 eventhandle

如表 3-54 所示为状态码 code 的取值及释义。

表 3-54　状态码 code 的取值及释义

序号	代　码	说　明
1	2001	已经连接服务器
2	2002	已经连接 RTMP 服务器，开始拉流
3	2003	网络接收到首个视频数据包(IDR)
4	2004	视频播放开始
5	2005	视频播放进度
6	2006	视频播放结束
7	2007	视频播放 Loading
8	2008	解码器启动
9	2009	视频分辨率改变
10	－2301	网络断连，且经多次重连抢救无效，更多重试请自行重启播放
11	－2302	获取加速拉流地址失败
12	2101	当前视频帧解码失败
13	2102	当前音频帧解码失败
14	2103	网络断连，已启动自动重连
15	2104	网络来包不稳：可能是下行带宽不足，或由于主播端出流不均匀

续表

序号	代　码	说　明
16	2105	当前视频播放出现卡顿
17	2106	硬解启动失败，采用软解
18	2107	当前视频帧不连续，可能丢帧
19	2108	当前流硬解第一个 I 帧失败，SDK 自动切软解
20	3001	RTMP-DNS 解析失败
21	3002	RTMP 服务器连接失败
22	3003	RTMP 服务器握手失败
23	3005	RTMP 读/写失败

如表 3-55 所示为网络状态数据 info 的键名及释义。

表 3-55　网络状态数据 info 的键名及释义

序号	键　名	说　明
1	videoBitrate	当前视频编/码器输出的比特率，单位 kbps
2	audioBitrate	当前音频编/码器输出的比特率，单位 kbps
3	videoFPS	当前视频帧率
4	videoGOP	当前视频 GOP，也就是每两个关键帧(I 帧)间隔时长，单位 s
5	netSpeed	当前的发送/接收速度
6	netJitter	网络抖动情况，抖动越大，网络越不稳定
7	videoWidth	视频画面的宽度
8	videoHeight	视频画面的高度

live-player 组件默认宽度 300px、高度 225px，可通过 wxss 设置宽高。开发者工具上暂时不支持该组件。图 3-67 为 live-player 组件的示例代码。

```
< live - player src = " https://domain/pull_stream" mode = " RTC" autoplay bindstatechange = "
statechange" binderror = "error" style = "width: 300px; height: 225px;" />
Page({
  statechange(e) {
    console.log('live - player code:', e.detail.code)
  },
  error(e) {
    console.error('live - player error:', e.detail.errMsg)
  }
})
```

图 3-67　live-player 组件的示例代码

关于 live-player 组件的更多内容，可访问以下链接。

https://developers.weixin.qq.com/miniprogram/dev/component/live-player.html

注意：若读者在上述链接对应的 live-player.html 中点击“在开发者工具中预览效果”，代码与图 3-67 不完全一样，请参见本章配套的名为“ex030605_live-player_在开发者工具中预览效果”文件夹下的代码。图 3-67 中的代码存放在本章代码的文件夹下。

3.6.6 live-pusher

live-pusher 是实时音视频录制组件，它的语法格式如图 3-68 所示。

```
<live-pusher>

</live-pusher>
或
<live-pusher              />
```

图 3-68 live-pusher 组件的格式

和 live-player 一样，live-pusher 暂时也只针对国内主体类目的小程序开放，需要先通过类目审核，然后在小程序管理后台“开发”-“接口设置”中自助开通该组件权限（需要用户授权 scope.camera 和 scope.record）。

如表 3-56 所示为 live-pusher 组件的属性。

表 3-56 live-pusher 组件的属性

序号	属　性	说　明
1	url	推流地址。目前仅支持 flv、rtmp 格式。该属性的类型是 string
2	mode	模式。该属性的类型是 string，默认值为 RTC（实时通话）。该属性的值还可以为 SD（标清）、HD（高清）和 FHD（超清）
3	autopush	自动推流。该属性的类型是 boolean，默认值为 false
4	muted	是否静音。该属性的类型是 boolean，默认值为 false
5	enable-camera	开启摄像头。该属性的类型是 boolean，默认值为 true
6	auto-focus	自动聚集。该属性的类型是 boolean，默认值为 true
7	orientation	画面方向。该属性的类型是 string，默认值为 vertical（竖直）。该属性的值还可以为 horizontal（水平）
8	beauty	美颜，取值范围 0-9，0 表示关闭。该属性的类型是 number，默认值为 0
9	whiteness	美白，取值范围 0-9，0 表示关闭。该属性的类型是 number，默认值为 0
10	aspect	宽高比，可选值有 3:4，9:16。该属性的类型是 string，默认值为 9:16
11	min-bitrate	最小码率。该属性的类型是 number，默认值为 200
12	max-bitrate	最大码率。该属性的类型是 number，默认值为 1000
13	audio-quality	高音质（48KHz）或低音质（16KHz），值为 high，low。该属性的类型是 string，默认值为 high
14	waiting-image	进入后台时推流的等待画面。该属性的类型是 string
15	waiting-image-hash	等待画面资源的 MD5 值。该属性的类型是 string
16	zoom	调整焦距。该属性的类型是 boolean，默认值为 false
17	device-position	前置或后置，值为 front，back。该属性的类型是 string，默认值为 front
18	background-mute	进入后台时是否静音。该属性的类型是 boolean，默认值为 false
19	mirror	设置推流画面是否镜像，产生的效果在 live-player 显现。该属性的类型是 boolean，默认值为 false
20	bindstatechange	状态变化事件，detail={code}。该属性的类型是 eventhandle
21	bindnetstatus	网络状态通知，detail={info}。该属性的类型是 eventhandle
22	binderror	渲染错误事件，detail={errMsg, errCode}。该属性的类型是 eventhandle

续表

序号	属　性	说　明
23	bindbgmstart	背景音开始播放时触发。该属性的类型是 eventhandle
24	bindbgmprogress	背景音进度变化时触发，detail＝{progress，duration}。该属性的类型是 eventhandle
25	bindbgmcomplete	背景音播放完成时触发。该属性的类型是 eventhandle

如表 3-57 所示为错误码(errCode)的取值及释义。

表 3-57　错误码(errCode)的取值及释义

序号	代　码	说　明
1	10001	用户禁止使用摄像头
2	10002	用户禁止使用录音
3	10003	背景音资源(BGM)加载失败
4	10004	等待画面资源(waiting-image)加载失败

如表 3-58 所示为状态码 code 的取值及说明。

表 3-58　状态码 code 的取值及说明

序号	代　码	说　明
1	1001	已经连接推流服务器
2	1002	已经与服务器握手完毕，开始推流
3	1003	打开摄像头成功
4	1004	录屏启动成功
5	1005	推流动态调整分辨率
6	1006	推流动态调整码率
7	1007	首帧画面采集完成
8	1008	编码器启动
9	－1301	打开摄像头失败
10	－1302	打开麦克风失败
11	－1303	视频编码失败
12	－1304	音频编码失败
13	－1305	不支持的视频分辨率
14	－1306	不支持的音频采样率
15	－1307	网络断连，且经多次重连抢救无效，更多重试请自行重启推流
16	－1308	开始录屏失败，可能是被用户拒绝
17	－1309	录屏失败，不支持的 Android 系统版本，需要 5.0 以上的系统
18	－1310	录屏被其他应用打断了
19	－1311	Android Mic 打开成功，但是录不到音频数据
20	－1312	录屏动态切横竖屏失败
21	1101	网络状况不佳：上行带宽太小，上传数据受阻
22	1102	网络断连，已启动自动重连
23	1103	硬编码启动失败，采用软编码
24	1104	视频编码失败
25	1105	新美颜软编码启动失败，采用老的软编码

续表

序号	代　码	说　明
26	1106	新美颜软编码启动失败，采用老的软编码
27	3001	RTMP-DNS 解析失败
28	3002	RTMP 服务器连接失败
29	3003	RTMP 服务器握手失败
30	3004	RTMP 服务器主动断开，请检查推流地址的合法性或防盗链有效期
31	3005	RTMP 读/写失败

注意：

(1) live-pusher 组件中代码为 3001、3002、3003 和 3005 的状态码与 live-player 组件中的一致。

(2) live-pusher 组件网络状态数据(info)的键名和释义与 live-player 组件的相同。

图 3-69 为 live-pusher 组件的示例代码。

```
< live - pusher url = " https://domain/push_ stream" mode = " RTC" autopush bindstatechange =
"statechange" style = "width: 300px; height: 225px;" />
Page({
  statechange(e) {
    console.log('live - pusher code:', e.detail.code)
  }
})
```

图 3-69　live-pusher 组件的示例代码

关于 live-pusher 组件的更多内容，可访问以下链接。

https://developers.weixin.qq.com/miniprogram/dev/component/live-pusher.html

注意：若读者在上述链接对应的 live-pusher.html 中点击“在开发者工具中预览效果”，代码与图 3-69 不完全一样，请参见本章配套的名为“ex030606_live-pusher_在开发者工具中预览效果”文件夹下的代码。图 3-69 中的代码在存放本章代码的文件夹下。

3.7　地图组件

map 是地图组件，它的语法格式如图 3-70 所示。

```
< map >

</map >
```

图 3-70　map 组件的格式

个性化地图能力可在小程序后台“开发-开发者工具-腾讯位置服务”申请开通，详见《小程序个性地图使用指南》(https://lbs.qq.com/product/miniapp/guide/)。小程序内地图组件

应使用同一 subkey，可通过 layer-style（地图官网设置的样式 style 编号）属性选择不同的地图风格。组件属性的长度单位默认为 px，从 2.4.0 起支持传入单位（rpx/px）。

如表 3-59 所示为 map 组件的属性。

表 3-59 map 组件的属性

序号	属 性	说 明
1	longitude	中心经度。该属性的类型为 number，为必填字段
2	latitude	中心纬度。该属性的类型为 number，为必填字段
3	scale	缩放级别，取值范围为 3～20。该属性的类型为 number，默认值为 16
4	markers	标记点，用于在地图上显示标记的位置。该属性的类型为 Array.＜marker＞
5	covers	此属性即将移除，请使用 markers。该属性的类型为 Array.＜cover＞
6	polyline	路线。该属性的类型为 Array.＜polyline＞
7	circles	圆。该属性的类型为 Array.＜circle＞
8	controls	控件（即将废弃，建议使用 cover-view 代替）。该属性的类型为 Array.＜control＞
9	include-points	缩放视野以包含所有给定的坐标点。该属性的类型为 Array.＜point＞
10	show-location	显示带有方向的当前定位点。该属性的类型为 boolean，默认值为 false
11	polygons	多边形。该属性的类型为 Array.＜polygon＞
12	subkey	个性化地图使用的 key。该属性的类型为 string
13	layer-style	个性化地图配置的 style，不支持动态修改。该属性的类型为 number，默认值为 1
14	rotate	旋转角度，范围 0～360，地图正北和设备 y 轴角度的夹角。该属性的类型为 number，默认值为 0
15	skew	倾斜角度，范围 0～40，关于 z 轴的倾角。该属性的类型为 number，默认值为 0
16	enable-3D	展示 3D 楼块（工具暂不支持）。该属性的类型为 boolean，默认值为 false
17	show-compass	显示指南针。该属性的类型为 boolean，默认值为 false
18	show-scale	显示比例尺，工具暂不支持。该属性的类型为 boolean，默认值为 false
19	enable-overlooking	开启俯视。该属性的类型为 boolean，默认值为 false
20	enable-zoom	是否支持缩放。该属性的类型为 boolean，默认值为 true
21	enable-scroll	是否支持拖动。该属性的类型为 boolean，默认值为 true
22	enable-rotate	是否支持旋转。该属性的类型为 boolean，默认值为 false
23	enable-satellite	是否开启卫星图。该属性的类型为 boolean，默认值为 false
24	enable-traffic	是否开启实时路况。该属性的类型为 boolean，默认值为 false
25	setting	配置项。该属性的类型为 object
26	bindtap	点击地图时触发。该属性的类型为 eventhandle
27	bindmarkertap	点击标记点时触发，e.detail＝{markerId}。该属性的类型为 eventhandle
28	bindcontroltap	点击控件时触发，e.detail＝{controlId}。该属性的类型为 eventhandle
29	bindcallouttap	点击标记点对应的气泡时触发 e.detail＝{markerId}。该属性的类型为 eventhandle
30	bindupdated	在地图渲染更新完成时触发。该属性的类型为 eventhandle
31	bindregionchange	视野发生变化时触发。该属性的类型为 eventhandle。视野改变时，regionchange 会触发两次，返回的 type 值分别为 begin（视野变化开始）和 end（视野变化结束）。从 2.8.0 起 begin 阶段返回 causedBy（导致视野变化的原因），有效值为 gesture（手势触发）和 update（接口触发）。从 2.3.0 起 end 阶段返回 causedBy，有效值为 drag（拖动导致）、scale（缩放导致）、update（调用更新接口导致）rotate，skew 仅在 end 阶段返回
32	bindpoitap	点击地图 poi 点时触发，e.detail＝{name，longitude，latitude}。该属性的类型为 eventhandle

1 setting

提供 setting 对象统一设置地图配置。同时对于一些动画属性如 rotate 和 skew，通过 setData 分开设置时无法同时生效，需通过 setting 统一修改，如图 3-71 所示。

```
// 默认值
const setting = {
  skew: 0,
  rotate: 0,
  showLocation: false,
  showScale: false,
  subKey: '',
  layerStyle: -1,
  enableZoom: true,
  enableScroll: true,
  enableRotate: false,
  showCompass: false,
  enable3D: false,
  enableOverlooking: false,
  enableSatellite: false,
  enableTraffic: false,
}

this.setData({
  // 仅设置的属性会生效，其他的不受影响
  setting: {
    enable3D: true,
    enableTraffic: true
  }
})
```

图 3-71　setting 的用法

2 marker

marker 用于在地图上显示标记的位置，如表 3-60 所示为其属性及说明。

表 3-60　marker 的属性及说明

序号	属　　性	说　　明
1	id	标记点 id，marker 点击事件回调会返回此 id。该属性的类型为 number，建议为每个 marker 设置上 number 类型 id，保证更新 marker 时有更好的性能
2	latitude	经度。该属性的类型为 number，为必填字段。浮点数，范围－90～90
3	longitude	纬度。该属性的类型为 number，为必填字段。浮点数，范围－180～180
4	title	标注点名。点击时显示，callout 存在时将被忽略。该属性的类型为 string
5	zIndex	显示层级。该属性的类型为 number
6	iconPath	显示的图标。项目目录下的图片路径，支持相对路径写法，以'/'开头则表示相对小程序根目录；也支持临时路径和网络图片(2.3.0)。该属性的类型为 string
7	rotate	旋转角度。顺时针旋转的角度，范围 0～360，默认为 0。该属性的类型为 number
8	alpha	标注的透明度。默认为 1，无透明，范围 0～1。该属性的类型为 number

续表

序号	属　性	说　明
9	width	标注图标宽度。默认为图片实际宽度。该属性的类型为 number/string
10	height	标注图标高度。默认为图片实际高度。该属性的类型为 number/string
11	callout	自定义标记点上方的气泡窗口。该属性的类型为 Object
12	label	为标记点旁边增加标签。该属性的类型为 Object
13	anchor	经纬度在标注图标的锚点,默认底边中点。该属性的类型为 Object。{x, y},x 表示横向(0-1),y 表示竖向(0-1)。{x: .5, y: 1} 表示底边中点
14	aria-label	无障碍访问,(属性)元素的额外描述。该属性的类型为 string

marker 上的气泡 callout 的属性及说明如表 3-61 所示。

表 3-61　callout 的属性及说明

序号	属　性	说　明
1	content	文本。该属性的类型为 string
2	color	文本颜色。该属性的类型为 string
3	fontSize	文字大小。该属性的类型为 number
4	borderRadius	边框圆角。该属性的类型为 number
5	borderWidth	边框宽度。该属性的类型为 number
6	borderColor	边框颜色。该属性的类型为 string
7	bgColor	背景色。该属性的类型为 string
8	padding	文本边缘留白。该属性的类型为 string
9	display	'BYCLICK':点击显示; 'ALWAYS':常显。该属性的类型为 string
10	textAlign	文本对齐方式。有效值: left, right, center。该属性的类型为 string

marker 上的气泡 label 的属性及说明如表 3-62 所示。

表 3-62　label 的属性及说明

序号	属　性	说　明
1	content	文本。该属性的类型为 string
2	color	文本颜色。该属性的类型为 string
3	fontSize	文字大小。该属性的类型为 number
4	x	label 的坐标(废弃)。该属性的类型为 number
5	y	label 的坐标(废弃)。该属性的类型为 number
6	anchorX	label 的坐标,原点是 marker 对应的经纬度。该属性的类型为 number
7	anchorY	label 的坐标,原点是 marker 对应的经纬度。该属性的类型为 number
8	borderWidth	边框宽度。该属性的类型为 number
9	borderColor	边框颜色。该属性的类型为 string
10	borderRadius	边框圆角。该属性的类型为 number
11	bgColor	背景色。该属性的类型为 string
12	padding	文本边缘留白。该属性的类型为 number
13	textAlign	文本对齐方式。有效值: left, right, center。该属性的类型为 string

3 polyline

polyline 用于指定一系列坐标点,从数组第一项连线至最后一项,其属性和说明如表 3-63 所示。

表 3-63 polyline 的属性及说明

序号	属性	说明
1	points	经纬度数组。该属性的类型为 array，为必填字段。例如：[{latitude：0，longitude：0}]
2	color	线的颜色。该属性的类型为 string(十六进制)
3	width	线的宽度。该属性的类型为 number
4	dottedLine	是否虚线。该属性的类型为 boolean，默认值为 false
5	arrowLine	带箭头的线。该属性的类型为 boolean，默认值为 false，开发者工具暂不支持该属性
6	arrowIconPath	更换箭头图标。该属性的类型为 string，在 arrowLine 为 true 时生效
7	borderColor	线的边框颜色。该属性的类型为 string
8	borderWidth	线的厚度。该属性的类型为 number

4 polygon

polygon 用于指定一系列坐标点，根据 points 坐标数据生成闭合多边形，其属性和说明如表 3-64 所示。

表 3-64 polygon 的属性及说明

序号	属性	说明
1	points	经纬度数组，该属性的类型为 array，为必填字段。例如：[{latitude：0，longitude：0}]
2	strokeWidth	描边的宽度，该属性的类型为 number
3	strokeColor	描边的颜色，该属性的类型为 string(十六进制)
4	fillColor	填充颜色，该属性的类型为 string(十六进制)
5	zIndex	设置多边形 Z 轴数值，该属性的类型为 number

5 circle

circle 用于在地图上显示圆，如表 3-65 所示。

表 3-65 circle 的属性及说明

序号	属性	说明
1	latitude	纬度，该属性的类型为 number(浮点数，范围－90～90)，为必填字段
2	longitude	经度，该属性的类型为 number(浮点数，范围－180～180)
3	color	描边的颜色，该属性的类型为 string(十六进制)
4	fillColor	填充颜色，该属性的类型为 string(十六进制)
5	radius	半径，该属性的类型为 number，为必填字段
6	strokeWidth	描边的宽度，该属性的类型为 number

6 control

control 用于在地图上显示控件，控件不随着地图移动。如表 3-66 所示为 control 组件的属性和说明。

表 3-66 control 的属性及说明

序号	属 性	说 明
1	id	控件 id。在控件点击事件回调会返回此 id，该属性的类型为 number
2	position	控件在地图的位置。控件相对地图位置，该属性的类型为 object，为必填字段
3	iconPath	显示的图标。项目目录下的图片路径，支持相对路径写法，以'/'开头则表示相对小程序根目录；也支持临时路径。该属性的类型为 string，为必填字段
4	clickable	是否可点击。默认不可点击，该属性的类型为 boolean，为必填字段

注意：control 组件即将废弃，请使用 cover-view。

control 组件的属性 position 的属性如表 3-67 所示。

表 3-67 position 的属性及说明

序号	属 性	说 明
1	left	距离地图的左边界多远。该属性的类型为 number，默认为 0
2	top	距离地图的上边界多远。该属性的类型为 number，默认为 0
3	width	控件宽度。该属性的类型为 number，默认为图片宽度
4	height	控件高度。该属性的类型为 number，默认为图片高度

7 scale

scale(比例尺)属性取值范围及对应的比例尺如表 3-68 所示。

表 3-68 scale 的属性及说明

序号	scale	比 例
1	3	1000km
2	4	500km
3	5	200km
4	6	100km
5	7	50km
6	8	50km
7	9	20km
8	10	10km
9	11	5km
10	12	2km
11	13	1km
12	14	500m
13	15	200m
14	16	100m
15	17	50m
16	18	50m
17	19	20m
18	20	10m

如图 3-72 所示为 map 组件的示例代码。

```
//视图层
<!-- map.wxml -->
<map id="map" longitude="113.324520" latitude="23.099994" scale="14" controls=
"{{controls}}" bindcontroltap="controltap" markers="{{markers}}" bindmarkertap="markertap"
polyline="{{polyline}}" bindregionchange="regionchange" show-location style="width: 100%;
height: 300px;"></map>
//逻辑层
Page({
  data: {
    markers: [{
      iconPath: "/resources/others.png",
      id: 0,
      latitude: 23.099994,
      longitude: 113.324520,
      width: 50,
      height: 50
    }],
    polyline: [{
      points: [{
        longitude: 113.3245211,
        latitude: 23.10229
      }, {
        longitude: 113.324520,
        latitude: 23.21229
      }],
      color:"#FF0000DD",
      width: 2,
      dottedLine: true
    }],
    controls: [{
      id: 1,
      iconPath: '/resources/location.png',
      position: {
        left: 0,
        top: 300 - 50,
        width: 50,
        height: 50
      },
      clickable: true
    }]
  },
  regionchange(e) {
    console.log(e.type)
  },
  markertap(e) {
    console.log(e.markerId)
  },
  controltap(e) {
    console.log(e.controlId)
  }
})
```

图 3-72　map 组件的示例代码

使用地图组件时，请注意以下问题。

（1）个性化地图暂不支持在微信开发者工具中调试。请先使用微信客户端进行测试。

（2）地图中的颜色值 color/borderColor/bgColor 等需使用 6 位(8 位)十六进制表示，8 位时后两位表示 alpha 值，如：#000000AA。

（3）地图组件的经纬度必填，如果不填经纬度则默认值是北京的经纬度。

（4）map 组件使用的经纬度是火星坐标系，调用 wx.getLocation 接口需要指定 type 为 gcj02。

（5）从 2.8.0 起 map 支持同层渲染，更多请参考原生组件使用限制。

（6）请注意原生组件使用限制。

关于 map 组件的更多内容，可访问以下链接。

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

注意：若读者在上述链接中“示例代码”下方点击“在开发者工具中预览效果”超链接，将会在微信开发者工具中打开与图 3-72 不完全一样的代码，具体请参见教材配套的本章代码(文件夹名为“ex0307_map_在开发者工具中预览效果”)。

图 3-72 中的代码存放在本章代码的文件夹下。

3.8 画布组件

canvas 是画布组件，它的语法格式如图 3-73 所示。

```
<canvas>

</canvas>
```

图 3-73 canvas 组件的格式

表 3-69 所示为画布组件的属性。

表 3-69 画布组件的属性

序号	属性	说明
1	type	指定 canvas 类型，当前仅支持 webgl。该属性的类型为 string
2	canvas-id	canvas 组件的唯一标识符，若指定了 type 则无需再指定该属性。该属性的类型为 string
3	disable-scroll	当在 canvas 中移动时且有绑定手势事件时，禁止屏幕滚动以及下拉刷新。该属性的类型为 boolean，默认值为 false
4	bindtouchstart	手指触摸动作开始。该属性的类型为 eventhandle
5	bindtouchmove	手指触摸后移动。该属性的类型为 eventhandle
6	bindtouchend	手指触摸动作结束。该属性的类型为 eventhandle
7	bindtouchcancel	手指触摸动作被打断，如来电提醒、弹窗。该属性的类型为 eventhandle
8	bindlongtap	手指长按 500ms 之后触发，触发了长按事件后进行移动不会触发屏幕的滚动。该属性的类型为 eventhandle
9	binderror	当发生错误时触发 error 事件，detail = {errMsg}。该属性的类型为 eventhandle

如图 3-74 所示为 canvas 组件的示例代码。

```
//视图层
<!-- canvas.wxml -->
<canvas style="width: 300px; height: 200px;" canvas-id="firstCanvas"></canvas>
<!-- 当使用绝对定位时,文档流后边的 canvas 的显示层级高于前边的 canvas -->
<canvas style="width: 400px; height: 500px;" canvas-id="secondCanvas"></canvas>
<!-- 因为 canvas-id 与前一个 canvas 重复,该 canvas 不会显示,并会发送一个错误事件到
AppService -->
<canvas style="width: 400px; height: 500px;" canvas-id="secondCanvas" binderror=
"canvasIdErrorCallback"></canvas>
//逻辑层
Page({
  canvasIdErrorCallback: function (e) {
    console.error(e.detail.errMsg)
  },
  onReady: function (e) {
    // 使用 wx.createContext 获取绘图上下文 context
    var context = wx.createCanvasContext('firstCanvas')

    context.setStrokeStyle("#00ff00")
    context.setLineWidth(5)
    context.rect(0, 0, 200, 200)
    context.stroke()
    context.setStrokeStyle("#ff0000")
    context.setLineWidth(2)
    context.moveTo(160, 100)
    context.arc(100, 100, 60, 0, 2 * Math.PI, true)
    context.moveTo(140, 100)
    context.arc(100, 100, 40, 0, Math.PI, false)
    context.moveTo(85, 80)
    context.arc(80, 80, 5, 0, 2 * Math.PI, true)
    context.moveTo(125, 80)
    context.arc(120, 80, 5, 0, 2 * Math.PI, true)
    context.stroke()
    context.draw()
  }
})
```

图 3-74　canvas 组件的示例代码

如图 3-75 所示为 canvas 组件的 type 为 2D 时的示例代码。

```
<!-- canvas.wxml -->
  <canvas type="2d" id="myCanvas"></canvas>
// canvas.js
Page({
  onReady() {
    const query = wx.createSelectorQuery()
```

图 3-75　canvas 2D 的示例代码

```
    query.select('#myCanvas')
      .fields({ node: true, size: true })
      .exec((res) => {
        const canvas = res[0].node
        const ctx = canvas.getContext('2d')

        const dpr = wx.getSystemInfoSync().pixelRatio
        canvas.width = res[0].width * dpr
        canvas.height = res[0].height * dpr
        ctx.scale(dpr, dpr)

        ctx.fillRect(0, 0, 100, 100)
      })
  }
})
```

图 3-75 canvas 2D 的示例代码(续)

如图 3-76 所示为 canvas 组件的 type 为 WebGL 时的示例代码。

```
<canvas type="webgl" id="myCanvas"></canvas>
Page({
  onReady() {
    const query = wx.createSelectorQuery()
    query.select('#myCanvas').node().exec((res) => {
      const canvas = res[0].node
      const gl = canvas.getContext('webgl')
      console.log(gl)
    })
  }
})
```

图 3-76 type 为 WebGL 时的示例代码

使用 canvas 组件时,请注意以下问题。

(1) canvas 标签默认宽度 300px、高度 150px。

(2) 同一页面中的 canvas-id 不可重复,如果使用一个已经出现过的 canvas-id,该 canvas 标签对应的画布将被隐藏并不再正常工作。

(3) 请注意原生组件使用限制。

(4) 当前暂不支持存在多个 WebGL 实例。

(5) 开发者工具中默认关闭了 GPU 硬件加速,可在开发者工具的设置中开启“硬件加速”,提高 WebGL 的渲染性能。

(6) 避免设置过大的宽高,在安卓下会有 crash 的问题。

关于 canvas 组件的更多内容,可访问以下链接。

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

注意：若读者在上述链接中“示例代码”下方点击“在开发者工具中预览效果”超链接，将会在微信开发者工具中打开与图 3-74、图 3-75 和图 3-76 不完全一样的代码，具体请参见教材配套的本章代码（文件夹名分别为“ex0308_canvas_在开发者工具中预览效果”、“ex0308_canvas_2d_在开发者工具中预览效果”和“ex0308_canvas_webgl_在开发者工具中预览效果”）。

图 3-74 中的代码，图 3-75 中的代码，图 3-76 中的代码均存放在本章代码的文件夹下。

3.9 其他组件

3.9.1 开放能力的组件

1 ad

ad 是用于 Banner 广告的组件，它的语法格式如图 3-77 所示。

```
<ad>

</ad>
```

图 3-77 ad 组件的格式

如表 3-70 所示为 ad 组件的属性。

表 3-70 ad 组件的属性

序号	属性	说明
1	unit-id	广告单元 id，可在小程序管理后台的流量主模块新建。该属性的类型为 string，为必填字段
2	ad-intervals	广告自动刷新的间隔时间，单位为秒，参数值必须大于等于 30（该参数不传入时 Banner 广告不会自动刷新）。该属性的类型为 number
3	bindload	广告加载成功的回调。该属性的类型为 eventhandle
4	binderror	广告加载失败的回调，event.detail = {errCode: 1002}。该属性的类型为 eventhandle
5	bindclose	广告关闭的回调。该属性的类型为 eventhandle

错误码是通过 binderror 回调获取到的错误信息，表 3-71 为错误代码、表示的异常情况、理由和解决方案。

表 3-71 错误码相关信息表

序号	代码	异常情况	理由	解决方案
1	1000	后端错误调用失败	该项错误不是开发者的异常情况	一般情况下忽略一段时间即可恢复

续表

序号	代码	异常情况	理　由	解决方案
2	1001	参数错误	使用方法错误	可以前往 developers. weixin. qq. com 确认具体教程(小程序和小游戏分别有各自的教程,可以在顶部选项中,“设计”一栏的右侧进行切换
3	1002	广告单元无效	可能是拼写错误、或者误用了其他 APP 的广告 ID	请重新前往 mp. weixin. qq. com 确认广告位 ID
4	1003	内部错误	该项错误不是开发者的异常情况	一般情况下忽略一段时间即可恢复
5	1004	无适合的广告	广告不是每一次都会出现,这次没有出现可能是由于该用户不适合浏览广告	属于正常情况,且开发者需要针对这种情况做形态上的兼容
6	1005	广告组件审核中	你的广告正在被审核,无法展现广告	请前往 mp. weixin. qq. com 确认审核状态,且开发者需要针对这种情况做形态上的兼容
7	1006	广告组件被驳回	你的广告审核失败,无法展现广告	请前往 mp. weixin. qq. com 确认审核状态,且开发者需要针对这种情况做形态上的兼容
8	1007	广告组件被驳回	你的广告能力已经被封禁,封禁期间无法展现广告	请前往 mp. weixin. qq. com 确认小程序广告封禁状态
9	1008	广告单元已关闭	该广告位的广告能力已经被关闭	请前往 mp. weixin. qq. com 重新打开对应广告位的展现

使用 ad 组件时,请注意以下问题。

(1) 在无广告展示时,ad 标签不会占用高度。

(2) ad 组件不支持触发 bindtap 等触摸相关事件。

(3) 目前可以给 ad 标签设置 wxss 样式调整广告宽度,以使广告与页面更融洽,但请遵循小程序流量主应用规范。

(4) 监听到 error 回调后,开发者可以针对性的处理,比如隐藏广告组件的父容器,以保证用户体验,但不要移除广告组件,否则将无法收到 bindload 的回调。

2 official-account

official-account 是公众号关注组件。当用户扫小程序码打开小程序时,开发者可在小程序内配置公众号关注组件,方便用户快捷关注公众号,它也可以嵌套在原生组件内。

它的语法格式如图 3-78 所示。

```
<official-account>

</official-account>
```

图 3-78　official-account 组件的格式

如表 3-72 所示为 official-account 组件的属性。

表 3-72 official-account 组件的属性

序号	属性	说明
1	bindload	组件加载成功时触发。该属性的类型为 EventHandle
2	binderror	组件加载失败时触发。该属性的类型为 EventHandle

如表 3-73 所示为 detail 对象的属性。

表 3-73 detail 对象的属性

序号	属性	说明
1	status	状态码。该属性的类型为 Number，它的有效值分别为－2(网络错误)，－1(数据解析错误)，0(加载成功)，1(小程序关注公众号功能被封禁)，2(关联公众号被封禁)，3(关联关系解除或未选中关联公众号)，4(未开启关注公众号功能)，5(场景值错误)，6(重复创建)
2	errMsg	错误信息。该属性的类型为 Number

在使用 official-account 组件时，请注意以下问题。

(1) 使用组件前，需前往小程序后台，在“设置”—>“关注公众号”中设置要展示的公众号(设置的公众号需与小程序主体一致)。

(2) 在一个小程序的生命周期内，只有从以下场景进入小程序，才具有展示引导关注公众号组件的能力：

① 当小程序从扫小程序码场景(场景值 1047，场景值 1124)打开时。

② 当小程序从聊天顶部场景(场景值 1089)中的「最近使用」内打开时，若小程序之前未被销毁，则该组件保持上一次打开小程序时的状态。

③ 当从其他小程序返回小程序(场景值 1038)时，若小程序之前未被销毁，则该组件保持上一次打开小程序时的状态。

(3) 为便于开发者调试，从基础库 2.7.3 版本起，开发版小程序增加以下场景展示公众号组件：即开发版小程序从扫二维码(场景值 1011)打开。

(4) 组件限定最小宽度为 300px，高度为定值 84px。

(5) 每个页面只能配置一个该组件。

3 open-data

open-data 是用于展示微信开放的数据的组件，它的语法格式如图 3-79 所示。

```
<open-data>

</open-data>
```

图 3-79 open-data 组件的格式

如表 3-74 所示为 open-data 组件的属性。

表 3-74 open-data 组件的属性

序号	属 性	说 明
1	type	开放数据类型，该属性的类型为 string。该属性的值可以为 groupName（拉取群名称），userNickName（用户昵称），userAvatarUrl（用户头像），userGender（用户性别），userCity（用户所在城市），userProvince（用户所在省份），userCountry（用户所在国家）和 userLanguage（用户的语言）
2	open-gid	群 id，当 type＝"groupName" 时生效。该属性的类型为 string。关于 open-gid 的获取请使用 wx.getShareInfo
3	lang	当 type＝"user＊"时生效，以哪种语言展示 userInfo，该属性的类型为 string，默认值为 en（英文）。该属性还可以取值 zh_CN（简体中文）或 zh_TW（繁体中文）
4	default-text	数据为空时的默认文案。该属性的类型为 string
5	default-avatar	用户头像为空时的默认图片，支持相对路径和网络图片路径。该属性的类型为 string
6	binderror	群名称或用户信息为空时触发。该属性的类型为 eventhandle

注意：只有当前用户在此群内才能拉取到群名称。

图 3-80 为 open-data 的示例代码。

```
<open-data type="groupName" open-gid="xxxxxx"></open-data>
<open-data type="userAvatarUrl"></open-data>
<open-data type="userGender" lang="zh_CN"></open-data>
```

图 3-80 open-data 组件的示例代码

4 web-view

承载网页的容器。会自动铺满整个小程序页面，个人类型的小程序暂不支持使用。客户端从 6.7.2 版本开始，navigationStyle：custom 对 web-view 组件无效。如表 3-75 所示为 web-view 组件的属性。

表 3-75 web-view 组件的属性

序号	属 性	说 明
1	src	web-view 指向网页的链接。可打开关联的公众号的文章，其他网页需登录小程序管理后台配置业务域名。该属性的类型为 string
2	bindmessage	网页向小程序 postMessage 时，会在特定时机（小程序后退、组件销毁、分享）触发并收到消息。e.detail＝{data}，data 是多次 postMessage 的参数组成的数组。该属性的类型为 eventhandler
3	bindload	网页加载成功时候触发此事件。e.detail＝{src}。该属性的类型为 eventhandler
4	binderror	网页加载失败的时候触发此事件。e.detail＝{src}。该属性的类型为 eventhandler

web-view 网页中可使用 JSSDK 1.3.2 提供的接口返回小程序页面。支持的接口及说明如表 3-76 所示

表 3-76 web-view 网页中可用于返回小程序页面的接口

序号	接口名	说　明
1	wx.miniProgram.navigateTo	参数与小程序接口一致
2	wx.miniProgram.navigateBack	参数与小程序接口一致
3	wx.miniProgram.switchTab	参数与小程序接口一致
4	wx.miniProgram.reLaunch	参数与小程序接口一致
5	wx.miniProgram.redirectTo	参数与小程序接口一致
6	wx.miniProgram.postMessage	向小程序发送消息，会在特定时机（小程序后退、组件销毁、分享）触发组件的 message 事件
7	wx.miniProgram.getEnv	获取当前环境

图 3-81 为 web-view 的示例代码。

```
//视图层
<view class="page-body">
  <view class="page-section page-section-gap">
    <web-view src="https://mp.weixin.qq.com/"></web-view>
  </view>
</view>
//逻辑层
Page({})
```

图 3-81 web-view 组件的示例代码

web-view 网页中仅支持如表 3-77 所示的 JSSDK 接口。

表 3-77 web-view 网页支持的 JSSDK 接口

序号	接口模块	接口说明	具体接口
1	判断客户端是否支持 js		checkJSApi
2	图像接口	拍照或上传	chooseImage
		预览图片	previewImage
		上传图片	uploadImage
		下载图片	downloadImage
		获取本地图片	getLocalImgData
3	音频接口	开始录音	startRecord
		停止录音	stopRecord
		监听录音自动停止	onVoiceRecordEnd
		播放语音	playVoice
		暂停播放	pauseVoice
		停止播放	stopVoice
		监听语音播放完毕	onVoicePlayEnd
		上传接口	uploadVoice
		下载接口	downloadVoice
4	智能接口	识别音频	translateVoice

续表

序号	接口模块	接口说明	具体接口
5	设备信息	获取网络状态	getNetworkType
6	地理位置	使用内置地图打开地点	openLocation
		获取地理位置	getLocation
7	摇一摇周边	开启 ibeacon	startSearchBeacons
		关闭 ibeacon	stopSearchBeacons
		监听 ibeacon	onSearchBeacons
8	微信扫一扫	调起微信扫一扫	scanQRCode
9	微信卡券	拉取使用卡券列表	chooseCard
		批量添加卡券接口	addCard
		查看微信卡包的卡券	openCard
10	长按识别	小程序圆形码	无

用户分享时可获取当前 web-view 的 URL，即在 onShareAppMessage 回调中返回 webViewUrl 参数，图 3-82 为示例代码。

```
Page({
  onShareAppMessage(options) {
    console.log(options.webViewUrl)
  }
})
```

图 3-82 回调中返回 webViewUrl 参数的示例代码

在网页内可通过 window.__wxjs_environment 变量判断是否在小程序环境，建议在 WeixinJSBridgeReady 回调中使用，也可以使用 JSSDK 1.3.2 提供的 getEnv 接口。图 3-83 为示例代码。

```
// web-view 下的页面内
function ready() {
  console.log(window.__wxjs_environment === 'miniprogram') // true
}
if (!window.WeixinJSBridge || !WeixinJSBridge.invoke) {
  document.addEventListener('WeixinJSBridgeReady', ready, false)
} else {
  ready()
}

// 或者
wx.miniProgram.getEnv(function(res) {
  console.log(res.miniprogram) // true
})
```

图 3-83 在 WeixinJSBridgeReady 中回调或使用 getEnv 接口的示例代码

从微信 7.0.0 开始，可以通过判断 userAgent 中包含 miniProgram 字样来判断小程序 web-view 环境。从微信 7.0.3 开始，webview 内可以通过以下方式判断小程序是否在前台，

如图 3-84 代码所示。

```
WeixinJSBridge.on('onPageStateChange', function(res) {
  console.log('res is active', res.active)
})
```

图 3-84 判断小程序是否在前台的示例代码

使用 web-view 组件时，请注意以下问题。

(1) 网页内 iframe 的域名也需要配置到域名白名单。

(2) 开发者工具上，可以在 web-view 组件上通过右击—>调试，打开 web-view 组件的调试。

(3) 每个页面只能有一个 web-view，web-view 会自动铺满整个页面，并覆盖其他组件。

(4) web-view 网页与小程序之间不支持除 JSSDK 提供的接口之外的通信。

(5) 在 iOS 中，若存在 JSSDK 接口调用无响应的情况，可在 web-view 的 src 后面加个 #wechat_redirect 解决。

(6) 避免在链接中带有中文字符，在 iOS 中会有打开白屏的问题，建议加一下 encodeURIComponent。

关于 ad、official-account、open-data 和 web-view 组件的更多内容，可访问以下链接。

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

https://developers.weixin.qq.com/miniprogram/dev/component/official-account.html

https://developers.weixin.qq.com/miniprogram/dev/component/open-data.html

https://developers.weixin.qq.com/miniprogram/dev/component/web-view.html

3.9.2 无障碍访问的组件

为了更好地满足视障人士对于小程序的访问需求，基础库自 2.7.1 起，支持部分 ARIA 标签。无障碍特性在读屏模式下可以访问，iOS 可通过设置—>通用—>辅助功能—>旁白打开。

以 view 组件为例，开发者可以增加 aria-role 和 aria-label 属性。其中 aria-role 表示组件的角色，当设置为'img'时，读屏模式下聚焦后系统会朗读出'图像'。设置为'button'时，聚焦后系统朗读出'按钮'。aria-label 表示组件附带的额外信息，聚焦后系统会自动朗读出来。

小程序已经内置了一些无障碍的特性，对于非原生组件，开发者可以添加以下无障碍标签，如表 3-78 所示。

表 3-78 无障碍标签

aria-hidden	aria-role	aria-label	aria-checked	aria-disabled
aria-describedby	aria-expanded	aria-haspopup	aria-selected	aria-required
aria-orientation	aria-valuemin	aria-valuemax	aria-valuenow	aria-readonly
aria-multiselectable	aria-controls	tabindex	aria-labelledby	ria-orientation
aia-multiselectable	aria-labelledby			

图 3-85 为无障碍访问的示例代码。

```
<view aria-role="button"  aria-label="提交表单">提交</view>
```

图 3-85 无障碍访问的示例代码

使用无障碍访问的组件，请注意以下方面。

(1) Android 和 iOS 读屏模式下设置 aria-role 后朗读的内容不同系统之间会有差异。

(2) 可设置的 aria-role 可参看 Using Aria 中的 Widget Roles，部分 role 的设置在移动端可能无效。

关于无障碍访问的更多内容，可访问以下链接。

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

3.10 小结

作为视图层的基本组成单元，组件在小程序的开发中可谓必不可少。本章从组件的概念讲起，然后依据其功能分类，详细介绍了视图容器组件、基础内容组件、表单组件、导航组件、媒体组件、地图组件、画布组件和其他组件。

对于视图容器组件中的 view、scroll-view 和 swiper 组件，表单组件中的 label、button、radio、checkbox、input 和 form 组件等，基础内容组件中的 icon、text 和 progress 组件，读者必须熟练掌握其用法。读者在学习这些组件时，尽可能与后续章节中的实际项目开发结合起来，这样可以达到事半功倍的效果。对于其他组件的学习，可以根据自己的实际情况而定。