第5章 小程序开发实例

5.1 准备工作

小程序从结构上可以分成一个 App 和多个 Page，这个 App 是对小程序整体或者全局信息的一个抽象封装；每个 Page 负责相应页面信息的抽象封装。其中，app 又可以分割成三个文件，分别描述小程序整体上的逻辑，以及全局配置和一些全局的公共样式表，如表 5.1 所示。

表 5.1 App 的 3 个文件

文件名	是否必需	说明
app.js	是	负责小程序逻辑
app.json	是	负责小程序公共配置
app.wxss	否	负责建立小程序公共样式表

每个 Page 对应的页面其相应代码又可以分成四个单独的文件(我们将要建立的页面名称命名为 about)，如表 5.2 所示。

表 5.2 一个小程序页面的 4 个文件

文件名	是否必需	说明
js	是	负责小程序页面逻辑
wxml	是	负责小程序页面结构
json	否	负责进行小程序页面配置
wxss	否	负责建立小程序页面样式表

这 4 个文件的名字都与页面的名称相同。需要注意的一点是：小程序包含的所有页面，每一个页面均要放在单独的目录中，所有这些目录又放在一个总的父目录中进行集中管理。

其中，JSON 对象格式的文本，不能是空白，至少应是一个空的 JSON 对象。如图 5.1 所示为创建一个空的 JSON 对象。

app.json 中还至少应配置一个属性，即 pages 属性，在该属性中通过一个字符串数组来配置这个小程序中用户可能访问到的每一个页面的路径，在我们即将建立的这个项目中，只需要配置唯一的 about 页面的访问路径，这个路径是 Pages 目录下 about 目录下的 about 页。每当新建一个页面时，都必须在 app.json 文件 pages 中将该新增页面的路径添加进

去，如图 5.2 所示。

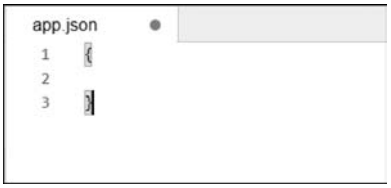

图 5.1　创建一个空的 JSON 对象

图 5.2　app.json 文件中页面路径配置信息

另外，页面的 ahout.js 函数不能是空白，至少需要调用 page 函数给 about 页面注册一个它自己的页面对象，可以注册一个空的页面对象，如图 5.3 所示。

图 5.3　注册空页面对象

新建的 about 页面上当前没有添加任何内容，此时需要在对应的结构文件即视图文件 about.wxml 中，添加一些组件元素来展示出对应的内容。类似于 HTML，可以使用最简单的基本组件元素，如用 text 元素来表示一个 Hello World 的文本；还可以通过 style 属性直接添加一些 inline 样式，如图 5.4 所示。

图 5.4　about.wxml 视图文件中直接使用 text 组件

也可以将样式代码从视图文件中抽离出来，放在一个单独的样式表中，即 WXSS 文件。定义一个样式规则，命名为 info，如图 5.5 所示。

如果要在视图文件 WXML 中应用，通过 class 取值来指定，如图 5.6 所示。

图 5.5　about.wxss 样式文件的定义

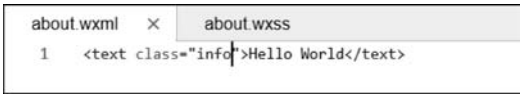

图 5.6　在 about.wxml 文件中引用 WXSS 文件中定义的样式

5.2　小程序生命周期

微信小程序的生命周期有两个，一个是 App 的生命周期，另一个是 Page 的生命周期。小程序的生命周期函数在 app.js 中调用，App(Object)函数用来注册一个小程序，接受一个 Object 参数，指定小程序的生命周期回调等，一般有 onLaunch 监听小程序初始化、onShow

监听小程序显示、onHide 监听小程序隐藏等生命周期回调函数，如表 5.3 所示。

表 5.3　小程序生命周期 Object 参数说明

属　性	类型	描　述	触发时机
onLaunch	function	生命周期回调——监听小程序初始化	小程序初始化完成时（全局只触发一次）
onShow	function	生命周期回调——监听小程序显示	小程序启动，或从后台进入前台显示时
onHide	function	生命周期回调——监听小程序隐藏	小程序从前台进入后台时
onError	function	错误监听函数	小程序发生脚本错误，或者 API 调用失败时触发
onPageNotFound	function	页面不存在监听函数	小程序要打开的页面不存在时触发
其他	any	开发者可以添加任意函数或数据到 Object 参数中，用 this 访问	

页面的生命周期函数是指每当进入或切换到一个新的页面时将会调用到的生命周期函数，Page(Object)函数用来注册一个页面。接受一个 Object 类型参数，其指定页面的初始数据、生命周期回调、事件处理函数等，如表 5.4 所示。

表 5.4　页面生命周期 Object 参数说明

属　性	类　型	描　述
data	Object	页面的初始数据
onLoad	function	生命周期回调——监听页面加载
onShow	function	生命周期回调——监听页面显示
onReady	function	生命周期回调——监听页面初次渲染完成
onHide	function	生命周期回调——监听页面隐藏
onUnload	function	生命周期回调——监听页面卸载
onPullDownRefresh	function	监听用户下拉动作
onReachBottom	function	页面上拉触底事件的处理函数
onShareAppMessage	function	用户单击右上角转发
onPageScroll	function	页面滚动触发事件的处理函数
onTabItemTap	function	当前是 Tab 页时，单击 Tab 时触发
其他	any	开发者可以添加任意函数或数据到 Object 参数中，用 this 访问

1. onLoad(Object query)

页面加载时触发。一个页面只会调用一次，可以在 onLoad 的参数中获取打开当前页面路径中的参数。

2. onShow()

页面显示/切入前台时触发。

3. onReady()

页面初次渲染完成时触发。一个页面只会调用一次，代表页面已经准备妥当，可以和视

图层进行交互。

4. onHide()

页面隐藏/切入后台时触发。如 navigateTo 或底部 Tab 切换到其他页面，小程序切入后台等。

5. onUnload()

页面卸载时触发。如 redirectTo 或 navigateBack 到其他页面时。

5.3 页面配置初探

任务 1：给 about 页添加一个标题，并配置为白底黑字的“关于”。

实现方法：在 about.json 文件中添加如下代码。

```
{
  "navigationBarTitleText":"关于",
  "navigationBarBackgroundColor": "#fff",
  "navigationBarTextStyle": "black"
}
```

第一个属性设置标题为“关于”，第二个属性配置导航栏的背景色为白色，第三个属性配置标题文本的样式为黑色。导航栏的背景色可以是任何颜色值，而标题文本的颜色只有 white 和 black 两种取值。

表 5.5 列出了页面的 JSON 文件中可以配置的页面属性信息，当前页面中配置的相关属性信息会覆盖 app.json 的 window 中相同的配置项。

表 5.5 JSON 文件中可以配置的页面属性信息

属 性 名	类型	默认值	说 明
navigationBarBackgroundColor	HexColor	#000000	导航栏背景颜色
navigationBarTextStyle	string	false	导航栏标题颜色，仅支持 black/white
navigationBarTitleText	string		导航栏标题文字内容
navigationStyle	string	default	导航栏样式
backgroundColor	HexColor	#ffffff	窗口的背景色
backgroundTextStyle	string	dark	下拉 loading 的样式，仅支持 dark/light
backgroundColorTop	string	#ffffff	顶部窗口的背景色，仅 iOS 支持
backgroundColorBottom	string	#ffffff	底部窗口的背景色，仅 iOS 支持
enablePullDownRefresh	boolean	false	是否开启当前页面下拉刷新
onReachBottomDistance	number	50	页面上拉触底事件触发时距页面底部距离，单位为 px
pageOrientation	string	portrait	屏幕旋转设置，支持 auto/portrait/landscape
disableScroll	boolean	false	设置为 true 则页面整体不能上下滚动
disableSwipeBack	boolean	false	禁止页面右滑手势返回
usingComponents	Object	否	页面自定义组件配置

任务 2：给 about 页添加页面结构和内容，通过对应的标记与元素来表达图片和文本。

前面章节提到过，WXML 也是通过框架提供的基础组件来表达内容元素，其中有一些属性，是任何一个 WXML 组件都可以设置的，比如

```
<text class="info" id="" style="" bindtap="" hidden="true" data-user-name="user">
        Hello World
</text>
```

中的 class 属性、id 属性、style 属性等，以及通过 bindtap 这样的方式来给该组件元素触发的事件绑定一个处理函数，也可以设置 hidden 属性来控制该元素是否隐藏，还可以通过 data-这样的属性来设置一些组件自定义的数据，这些自定义数据将会在事件触发的时候封装在事件对象中，传递给对应的事件处理函数进行处理。

还可以通过 image 组件完成图片的加载功能：

```
<image src="http://www."></image>
```

可以加载一个网页上的图片，也可以是一个本地的图片，该图片文件需要放到小程序项目的文件结构中，添加一个 images 目录放置一些需要使用到的图片文件，代码如：

```
<image src="/images/celavi.jpg"></image>
<text>CELAVI 餐厅和空中酒吧</text>
<text>Marina Bay Sands</text>
<text>小程序开发测试</text>
```

注意：image 组件也可以使用相对路径，以上代码运行后的界面如图 5.7 所示。

从图 5.7 显示效果来看，页面在渲染时是尽可能挤在同一行去展示，其原因在于 text 元素默认是 inline 元素，image 元素默认是 inline-block 元素。实际使用中，经常也需要将这样的多个元素放置在一个容器中，以便对由多个元素构成的整体做一个总的样式控制。在 HTML 中，是通过 div 来实现这样的容器元素，在 WXML 中则可以通过 view 元素来作容器元素，实现一个 view 元素包含一个 image 元素和三个 text 元素，同时也将图片和文本信息呈现出来，代码如下。

图 5.7　图片加载效果

```
<view>
   <image src="/images/celavi.jpg"></image>
   <text>CELAVI 餐厅和空中酒吧</text>
   <text>Marina Bay Sands</text>
   <text>小程序开发测试</text>
</view>
```

代码运行后的效果仍然如图 5.7 所示，各个组件元素的显示在布局上没有改观，如何设计布局样式来使组件元素的显示尽量美观呢？

5.4 快速实现基本布局——应用弹性盒子布局

先从一个简单的布局需求开始，具体要求如下。

(1) 从上往下，分行输出，而不是全部挤在一行输出，每个元素独占一行自上而下放置。

(2) 素材间隔均匀分布。

(3) 水平居中。

先给 view 元素添加一个临时的背景色，如前所述，需要修改样式文件 about.wxss，并在 WXML 文件中引用这些样式。在 about.wxss 中新建一个样式 container，代码如下。

```
.container{
  background-color: rgb(233, 227, 227);
}
```

小技巧：在 WXSS 样式文件中，经常需要定义各种颜色，如这里的 background-color，当需要选择想要的颜色时，此时可以先任意输入一组值，以 # 开头，如 #fff，然后移动鼠标至相应位置，此时就会出现颜色选择器，如图 5.8 所示。

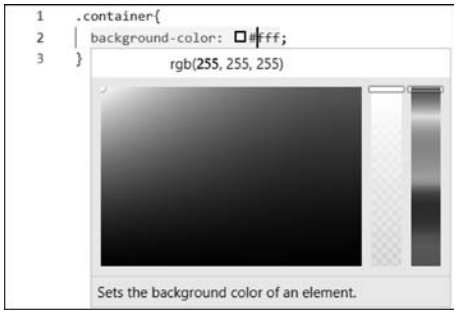

图 5.8 颜色选择器的使用

用鼠标单击调色板相应位置拾取想要定义的颜色，此时 background-color 的颜色值会以 rgb 三个分量的值呈现，如图 5.9 所示。

在 about.wxml 引入 wxss 中定义的 container 样式：

```
<view class = "container">
    ...
</view>
```

此时的渲染效果如图 5.10 所示。

图 5.9 颜色选择器的 rgb 三分量显示

图 5.10 引入 container 样式后的渲染效果

观察这里的 view 元素，它默认的高度取决于 4 个子元素的高度之和。在实际的页面效果中，如果希望 view 元素的高度等同于页面可见区域的高度，可以通过将高度设置为 100vh 来实现，vh 即 viewport height，100vh 相当于页面视口高度的百分之百，即相当于视口的高度。

5.4.1 传统布局的实现方式

1. 需要实现从上往下的布局需求

如果在 WXML 文件中不引入 view 组件，也不改变 text 组件的显示方式，此时的代码为：

```
<image src="/images/celavi.jpg"></image>
<text>CELAVI 餐厅和空中酒吧</text>
<text>Marina Bay Sands</text>
<text>小程序开发测试</text>
```

该段代码渲染出的结果将同图 5.7。那么如何实现文本从上往下显示的布局？如果将每个 text 元素从 inline 元素变成快捷元素，就可以实现每个元素独占一行从上往下放置。

这里先在 about.wxss 中添加一个样式 text，代码为：

```
.text{
  display:block;
}
```

然后在 about.wxml 文件中将该 text 样式在 text 组件中引入，代码如下。

```
<text class="text">CELAVI 餐厅和空中酒吧</text>
```

此时的实现效果如图 5.11 所示。

那么水平居中又该如何实现呢？

此时需要在 text 样式中增加一个 text-align 属性并将其值置为 center，此时的显示效果如图 5.12 所示。

图 5.11 引入 text 样式实现文本内容的分行显示

图 5.12 文本居中显示的 text-align 设置

2. 素材均匀分布

一种传统的实现方式为：先计算页面可见区域的总高度减去每个元素实际占的高度，得出在垂直方向上剩余的空间高度，将剩下的空间高度均匀地分配给每个元素的垂直外边距，通过设置每个子元素的下外边距来实现，这里暂且假定为 90rpx，具体页边距的设置可以根据实际显示效果做动态调整。

about.wxss 中的相应代码为：

```
image,text{
  margin-bottom:90rpx;
}
```

实现效果如图 5.13 所示。

图 5.13　图片及文字垂直方向均匀分布的实现效果

传统方式的特点如下。

(1) 布局目标可根据不同元素位置需要分别进行不同的属性赋值，有的属性需要赋值在一些子元素上，如 image、text，有的属性需要赋值在每一个子元素上，如 text 等。

(2) 传统布局依赖于页面结构与实际内容大小，当页面结构发生变化时，布局相关属性取值都将重新调整，从而不能足够灵活应对页面结构的变化。如本例中，如果加入的图片高度变大，每个元素向外边距的值则需要重新计算；如果视图中包含不止三个 text 元素，margin-bottom 也将重新计算，间距会相应减少。

5.4.2　弹性盒子布局

首先将拟使用弹性盒子布局的所有元素包括 image 和 text 均嵌套在一个 view 组件中，将其变成一个 flex container，方法是将该容器的 display 取值改为 flex。about.wxss 中的代码为：

```
.container{
  background-color: rgb(233, 227, 227);
  height:100vh;
  display:flex;
}
```

在 about.wxml 中引入 WXSS 中定义的 container 样式：

```
<view class = "container">
    <image src = "/images/celavi.jpg"></image>
    <text>CELAVI 餐厅和空中酒吧</text>
    <text>Marina Bay Sands </text>
    <text>小程序开发测试</text>
</view>
```

模拟器中渲染出的效果如图 5.14 所示。

如在 container 样式中增加一行代码：

```
flex-direction: column;
```

显示的效果同图 5.14，说明 flex-direction 默认主轴方向为从上往下。如果将 flex-direction 的值置为 row，则显示的效果如图 5.15 所示，此时的主轴方向为从左到右。

图 5.14　引入 flex 弹性盒子布局的显示效果

图 5.15　flex-direction 置为 row 时的显示效果

如果希望在垂直方向上各个元素的前后间隔均等，并且第一个元素和最后一个元素距离这个容器的边缘也都有一个相应的间隔，可通过置属性 justify-contentspace-around 为 space-around 来实现。属性 align-items 可用来控制元素的对齐方式，当 flex-direction 主轴方向定义为 row 时，align-items 分别取值 flex-start、flex-end、baseline 的显示效果如图 5.16 所示。

(a) align-items:flex-start　(b) align-items:flex-end　(c) align-items:baseline

图 5.16　align-items 取不同值时的显示效果

5.4.3　弹性盒子布局的优点

弹性盒子布局作为一种新的布局方式，实现的是一种整体布局，不仅直观高效，而且能够灵活地应对页面结构的变化，这种方式具有如下优点。

(1) WXSS 属性赋值相对统一,可在容器元素上做统一控制,容器内的所有组件元素都将具有统一赋值的显示属性。

(2) 布局方式灵活,可以应对页面结构的一些变化。如在页面结构中新增两个 text 元素,依然可以和原来的元素在垂直方向上保持均匀分布,从而实现一种整体控制,并没有去具体计算每个元素之间的间隔大小。

5.5 如何让元素大小适配不同宽度屏幕

本章中用到的图片在渲染时均未重置其显示尺寸,则保持为 image 元素默认的大小,即宽度为 320px,高度为 240px,如果希望宽高均为屏幕宽度的一半或其他尺寸,该如何实现呢?

当选择模拟器中的显示终端设备为 iPhone 5 时,其屏幕显示尺寸为 320×568,因此可将显示图片的宽、高均设置为 160。首先在 about.wxss 中新增一样式为:

```
.bar{
  width:160px;
  height: 160px;
}
```

在 about.wxml 文件中,修改 image 元素的 class 属性:

```
<image class = "bar" src = "/images/celavi.jpg"></image>
```

此时的显示效果如图 5.17 所示。

以上图片显示尺寸的设定是在选定显示设备型号的前提下,直接在 WXSS 文件中设置图片的宽高显示像素,如果想要实现图片的宽高始终保持为屏幕宽度的一半,这个时候显示像素这个绝对单位就不适用了,需要引入一个新的长度单位,它必须是一个相对于屏幕宽度大小的一个相对长度单位,小程序中称这个相对长度单位为 rpx。无论当前是哪种设备,它都统一规定这个设备上的屏幕宽度为 750rpx。如要实现图片的宽高均为屏幕宽度的一半,采用的相对单位就是 375rpx,如图 5.18 所示。

图 5.17 设置显示图片的尺寸大小

图 5.18 指定图文显示效果的布局样式

最后做一个简单的优化，把这个图片设计为一个圆形的图片，相应的文本可以实现加粗字体和字号变大的效果。

(1) 在 about.wxss 中增加图片的圆形设计。

将显示图片用到的组件 image 将要使用的样式 bar 修改为：

```
.bar{
  width:375rpx;
  height: 375rpx;
  border-radius: 50%
}
```

WXML 文件中 view 组件将要使用到的样式 container 的完整代码为：

```
.container{
  background-color: rgb(233, 227, 227);
  height:100vh;
  display:flex;
  flex-direction: column;
  justify-content: space-around;
  align-items:center
}
```

(2) 在 about.wxml 中增加文本的控制。

```
<view class="container">
    <image class="bar" src="/images/celavi.jpg"></image>
    <text style='font-weight:bold;font-size:60rpx'>CELAVI 餐厅和空中酒吧</text>
    <text>Marina Bay Sands</text>
    <text>小程序开发测试</text>
</view>
```

显示效果如图 5.19 所示。

图 5.19　圆形图片及指定显示样式的文本效果

5.6　新增“优惠推荐”promotion 页并快速调试

首先，需要添加 promotion 页对应的目录和文件(脚本文件 JS、配置文件 JSON、视图文件 WXML、样式表文件 WXSS)，在 promotion.json 文件中配置一个空的 JSON 对象；然后在脚本文件.js 中调用 Page 函数，给页面注册一个空对象{}；再给 WXML 添加“优惠推荐”所对应的 WXML 代码。

5.6.1　使用 navigator 组件——从 about 页跳转到 promotion 页

添加导航链接最简单的方式是使用 navigator 组件，主要是两个属性的使用，分别是 open-type 属性和 hove-class 属性。navigator 组件的属性在表 5.6 中予以列出。

表 5.6　navigator 组件的属性信息

属　性　名	类型	默　认　值	说　　明
target	string	self	在哪个目标上发生跳转，默认当前小程序，可选值：self/，miniProgram
url	string		当前小程序内的跳转链接
open-type	string	navigate	跳转方式
delta	number		当 open-type 为 navigateBack 时有效，表示回退的层数
app-id	string		当 target＝“miniProgram”时有效，要打开的小程序 appId
path	string		当 target＝“miniProgram”时有效，打开页面路径，如果为空则打开首页
extra-data	object		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
version	version	release	当 target＝“miniProgram”时有效，要打开的小程序版本，有效值：develop（开发版），trial（体验版），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
hover-class	string	navigator-hover	指定单击时的样式类，当 hover-class＝“none”时，没有单击态效果
hover-stop-propagation	boolean	false	指定是否阻止本节点的祖先节点出现单击态
hover-start-time	number	50	按住后多久出现单击态，单位：ms
hover-stay-time	number	600	手指松开后单击态保留时间，单位：ms
bindsuccess	string		当 target＝“miniProgram”时有效，跳转小程序成功
bindfail	string		当 target＝“miniProgram”时有效，跳转小程序失败
bindcomplete	string		当 target＝“miniProgram”时有效，跳转小程序完成

首先在 app. json 中添加 promotion 页面的路径“pages/promotion/promotion”。小程序的初始页面为 about，在 about 页上加一个导航链接，让它指向 promotion 页。

about. wxml 中的完整代码为：

```
<view class = "container">
    <image class = "bar" src = "/images/celavi. jpg"></image>
    <text style = 'font - weight:bold;font - size:60rpx'> CELAVI 餐厅和空中酒吧</text>
    <text> Marina Bay Sands </text>
    <text> Contact Us: + 65 65082188;商家优惠促销</text>
</view>
```

如果只是对“商家优惠促销”中的“优惠促销”，此时需要将文本进行拆分，分别为“商家”与“优惠促销”。

```
<text> Contact Us: + 65 65082188;商家</text>
<navigator url = '/pages/promotion/promotion'>优惠促销</navigator>
```

可以发现，此时的显示效果却是分成两行显示，如图 5.20 所示。

出现这种结果的原因，是因为< text >和< navigator >都是并列的关系，都是作为< view >元素的子元素，从上往下来进行放置。为了把它们仍然放在同一行，采取以下解决方案。

第 1 步：navigator 默认是块级元素，需要将其变成 inline 元素，**增加 style＝'display:inline'属性值**。

第 2 步：需要将一个< text >元素和一个< navigator >元素封装在一个< view >元素中。

经实测，两个步骤缺一不可，相应代码为：

```
<view>
  <text>Contact Us: +65 65082188;商家</text>
  <navigator style = 'display:inline' url = '/
pages/promotion/promotion'>优惠促销
</navigator>
  </view>
```

图 5.20　增加导航链接后分行显示效果

通过< view >元素的封装，仍然可以在同一行中显示为完整的一段话。只有对< navigator >元素中的“优惠促销”进行单击，才会从当前页面跳转到目标页。跳转页的左上角有一个“返回”按钮，可以返回到 about 页面。如果希望实现不能返回，则需要用到 navigator 元素的 open-type 属性。

置 open-type＝'redirect'（其默认值为 navigate，即返回跳转前的页面）。

第 3 步：如何给< navigator >元素添加一个单击态的样式效果？

实现单击态的样式效果需要用到 hove-class 属性，首先给< navigator >元素的 hove-class 赋值一个样式类：

```
hover-class = 'nav-hover'
```

然后在页面对应的样式表中去定义样式规则。如果希望导航链接被单击的时候，将它字体的颜色变为红色，在 about.wxss 中添加：

```
.nav-hover{
  color: red;
}
```

5.6.2　配置 tabBar——对若干一级页面的入口链接

如果小程序是一个多 Tab 应用，可通过配置顶部或底部导航栏来实现，基本上任何一个完整的小程序都会存在一个导航栏，小程序上官方文档要求 tabBar 中的 item 最少为两个，最多为五个。tabBar 标签栏配置可用于实现不同一级页面之间的快速任意切换，其本质实际上就是对若干一级页面的入口链接。

首先需要为每一个 Tab 准备两个 icon 图，分别对应这个 Tab 默认的时候使用的 icon

图和它被选中的时候使用的 icon 图(icon 图标可在网站 http://tool.58pic.com/tubiaobao/index.php 里找到)。

在 app.json 文件中添加如下代码。


```
"tabBar": {
    "list": [
      {
       "text": "关于",
       "pagePath": "pages/about/about",
       "iconPath": "images/icons/about.png",
       "selectedIconPath": "images/icons/selectedAbout.png",
      },
      {
        "text": "优惠促销",
        "pagePath": "pages/promotion/promotion",
        "iconPath": "images/icons/promotion.png",
        "selectedIconPath": "images/icons/selectedPromotion.png"
      },
      {
        "text": "联系我们",
        "pagePath": "pages/about/about",
        "iconPath": "images/icons/helpCenter.png",
        "selectedIconPath": "images/icons/selectedHelpCenter.png"
      }
    ]
  }
```

上述代码中的“iconPath”“selectedIconPath”分别表示默认的 icon 图和选中之后的 icon 图,编译后的运行效果如图 5.21 所示。

图 5.21　tabBar 的运行效果

但是当单击“优惠促销”文本时,navigator 元素跳转界面却没反应了,这是为什么呢? 在 app.json 中,通过 list 的第一个对象,将 about 页设置成了第一个 Tab 所链接到的页面,在 about 页面中单击 navigator 元素,此时不仅底部标签栏也做一个切换,切换到 promotion 页所对应的第 2 个 Tab,此时 navigator 元素中 open-type 的取值就不能是默认的 navigate,而应该是一个特殊的取值叫 switchTab。open-type 的有效取值如表 5.7 所示。

表 5.7　open-type 的有效取值

属　性　名	说　　明
navigate	保留当前页面,跳转到应用内的某个页面
redirect	关闭当前页面,跳转到应用内的某个页面
reLaunch	关闭所有页面,打开到应用内的某个页面
switchTab	跳转到 tabBar 页面,并关闭其他所有非 tabBar 页面

单击之后,实际上包含两个操作,第一个操作就是页面的跳转,第二个操作是底部 tabBar 的切换,此时跳转之后的显示界面如图 5.22 所示。跳转界面 promotion 的 WXML

与 WXSS 的代码分别为：

```
<view class="pContainer">
    <text>优惠推荐</text>
    <image class="dessert" src="/images/Celavi-Sling Bombe-Alaska.jpg"></image>
    <text>Singapore Sling Bombe-Alask</text>
</view>
```

及

```
.dessert{
  width:375rpx;
  height: 281rpx;
}

.pContainer{
  background-color: rgb(233, 227, 227);
  height:100vh;
  display:flex;
  flex-direction: column;
  align-items:center
}
```

图 5.22　跳转页面显示效果

5.6.3　数据绑定——从视图中抽离出数据

前面讲到的 about 页和 promotion 页，在两个页面的 WXML 代码中，相关的数据都是进行直接硬编码的，也就是数据直接在页面代码中进行赋值，硬编码数据适合于静态的数据。但实际应用中，更多的则是一些动态数据，如 promotion 页中的优惠推荐，在不同的时间节点，优惠商品是会发生变化的，如果采取硬编码方式，则需要频繁修改 WXML 代码，重新打包上传发布。甚至有些数据在编码的时候是无法获知的，需要在小程序运行过程中，动态地从服务器端去获取，然后再渲染输出到这个视图中进行显示。因此，应该提供一种机制能够让这个视图中的每一个部分与对应的数据做一个映射以便获取动态数据。

在小程序框架中，每个页面所需要的各种数据，都是集中在这个页面所注册的页面对象中集中定义的。每个页面都是在其对应的脚本文件中通过调用 Page 函数来给这个页面注册它所需要的页面对象。在页面对象中，通过 data 属性来定义该页面所需要的各种数据。

为了演示动态数据获取效果，先在 promotion.js 中添加如下代码。

```
Page({
   data:{
     promotionRecommend:{
       name:"Frozen Mochi",
       imagePath:"/images/Celavi-Frozen-Mochi.jpg"
     },
     count:123
   }
})
```

代码中的 promotionRecommend 数据代表了当前页面的一个内部状态，相当于该页面的内部状态变量。也可以再增加一个内部状态变量(数据)count，记录当前该推荐商品可供售出的份数。

接下来的问题是，数据定义后，如何将其绑定输出到视图中去？要把 count 的值渲染输出到视图页面上显示，需要通过一个双大括号进行数据绑定。在 promotion.wxml 中，可以将 count 变量绑定在 text 的内容上，如<text>此商品还剩{{count}}份可以售出</text>，此为最简单的一种数据绑定形式。

还有一些复杂的数据绑定，可以对内部状态变量进行一些运算或者进行一些组合来输出显示。比如在 promotion.js 的 data 中再增加一个评分的内部状态数据 score，然后在 promotion.wxml 视图文件中渲染输出。

```
<text>该商品用户评价打分：{{(score>=60)?"及格":"不及格"}}</text>
```

下面再来看下 promotion.js 中的数据 promotionRecommend 如何在视图文件中渲染输出，代码如下。

```
<image src='{{promotionRecommend.imagePath}}'></image>
<text>{{promotionRecommend.name}}</text>
```

提示：在开发者工具的调试器 AppData 中，可以很方便地对每个页面所包含的内部状态数据进行查看和调试，不仅可以查看这些内部状态变量，还可以修改它们，修改后在视图上就会有自动更新的效果，如图 5.23 所示。

图 5.23　在 AppData 中查看内部变量数据

5.6.4　条件渲染

条件渲染指条件成立时组件才渲染生成，本节主要介绍以下两个方面的内容。

(1) wx:if 属性的使用。

(2) 条件渲染 wx:if 与使用 hidden 属性的区别。

先在 promotion.js 中添加一个表示是否强烈推荐的内部状态变量 isHighlyRecommended，对于那些客户真正强烈推荐的商品，显示一个强烈推荐的红色标记。

然后在 promotion.wxml 文件中，定义“强烈推荐”的 text：

```
<text style="font-size:16px;color:red;">强烈推荐</text>
```

现在的需求是：根据实际返回的数据中的 isHighlyRecommended 是否为 true 来决定它是否要渲染生成。

此时可通过 wx:if 属性做数据绑定来实现。

```
<text wx:if="{{promotionRecommend.isHighlyRecommended}}"
                              style="font-size:16px;color:red;">强烈推荐</text>
```

这段代码表示，该 text 元素渲染生成的条件绑定到 promotionRecommend.isHighlyRecommended 属性上，根据该属性的值来决定是否渲染。

那么使用条件渲染与 hidden 属性有何区别？如以下代码使用的是 hidden 属性：

```
<text hidden="{{!promotionRecommend.isHighlyRecommended}}"
                    style="font-size:16px;color:red;">强烈推荐</text>
```

从语义上讲，使用 hidden 属性时，该元素总是要被渲染生成的（**可从调试器的 WXML 标签观察生成的节点树**），Hidden 属性只是控制了其可见性而已，如图 5.24 所示。

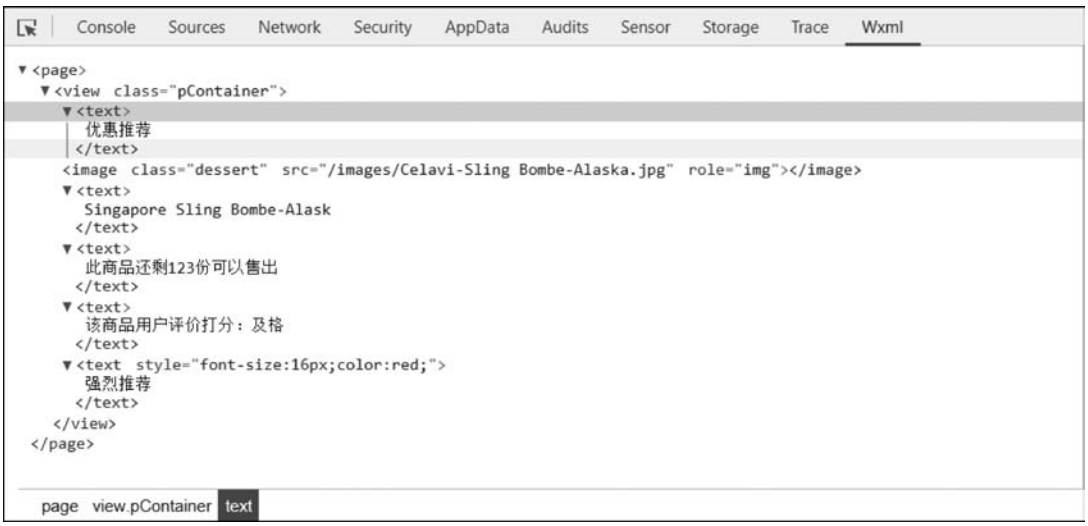

图 5.24　WXML 标签节点树

5.6.5　列表渲染

假如推荐列表中有多款商品，如何实现将对象数组中的多个商品对象渲染输出到视图页面中？

首先在 promotion.js 中将原来的 promotionRecommend 单个商品修改为列表商品的数组 promotionRecommendList，然后补充以下一些商品信息，代码如下。

```
Page({
   data:{
     promotionRecommendList:[
       {
         name:"Frozen Mochi",
         price:"￥59.9 元起",
```

```
            imagePath:"/images/Celavi - Frozen - Mochi.jpg",
            isHighlyRecommended:true
          },
          {
            name:"Hokkaido Scallop & Oyster Ceviche",
            price:"￥199.9 元起",
            imagePath: "/images/Celavi - Hokkaido Scallop & Oyster Ceviche.jpg",
          },
          {
            name: "Rose&Watermelon Petit Gateau",
            price: "￥169.9 元起",
            imagePath: "/images/celavi - Rose&Watermelon Petit Gateau.jpg",
          },
        ]
      }
})
```

第一种是最简单、实现起来最直接的方法，就是将 promotionRecommendList 商品列表中的数据一个个依次输出，promotion.wxml 中的相应代码为：

```
<view class = "pContainer">
  <view >
    <image class = "dessert"
   src = '{{promotionRecommendList[0].imagePath}}'  ></image >
    <text >{{promotionRecommendList[0].name}}</text >
    <text >{{promotionRecommendList[0].price}}</text >
    <text wx:if = "{{!promotionRecommendList[0].isHighlyRecommended}}"
                              style = "font - size:16px;color:red;">强烈推荐</text >
  </view >

  <view >
    <image class = "dessert" src = '{{promotionRecommendList[1].imagePath}}' ></image >
    <text >{{promotionRecommendList[1].name}}</text >
    <text >{{promotionRecommendList[1].price}}</text >
    <text wx:if = "{{!promotionRecommendList[1].isHighlyRecommended}}"
                              style = "font - size:16px;color:red;">强烈推荐</text >
  </view >

  <view >
     <image class = "dessert"
     src = '{{promotionRecommendList[2].imagePath}}'></image >
     <text >{{promotionRecommendList[2].name}}</text >
     <text >{{promotionRecommendList[2].price}}</text >
     <text wx:if = "{{!promotionRecommendList[2].isHighlyRecommended}}"
                              style = "font - size:16px;color:red;">强烈推荐</text >
  </view >
</view >
```

以上代码运行渲染出的效果如图 5.25 所示。

图 5.25　商品列表依次输出的效果

在实际开发中，这种输出方式不够灵活，因为在页面初始加载的时候，可能还不知道 server 端会返回多少个列表对象，WXML 中不知道要重复编写多少个这样的 view 结构代码，或者商品列表的个数会发生动态变化，此时这种依次输出的方式就不再适用，而需要采用程序设计中的循环控制思想，在视图中添加可以循环的控制结构，让框架可以自动对组件进行重复的渲染生成。

实现方法：将 view 元素的 wx:for 属性绑定到对应的数组上，此时 view 元素针对绑定数组中的每一个值都将重复地渲染生成一次。在 view 元素的结构代码中，通过内置的 item 循环控制变量来直接访问到本次所遍历到的数组中的值是哪一个值。

```
<view wx:for="{{promotionRecommendList}}">
     <image class="dessert" src='{{item.
imagePath}}'></image>
    <text>{{item.name}}</text>
    <text>{{item.price}}</text>
    <text wx:if="{{!item.isHighlyRecommended}}"
style="font-size:16px;color:red;">
                        强烈推荐</text>
</view>
```

上述代码中，将 view 元素的 wx:for 属性与对象数组 promotionRecommendList 进行绑定，通过 item 内置循环控制变量来对对象数组中的每一个元素进行遍历访问。可以发现，通过引入 wx:for 列表循环渲染，代码更加简洁，结构更为清晰。

如果要在视图中展示遍历到的促销商品的序号，则需要使用另一个循环控制变量 index（该变量在 promotion.js 数据 data 中定义，并初始化为 0），即当前所遍历到的 item 值在数组中的下标，代码如下。

```
<text>第{{index+1}}款：{{item.name}}</text>
```

图 5.25 中的图片文字排版布局有点儿乱，需要在此基础上做相应调整，使得图片文字分开居中显示，输出效果如图 5.26 所示。

图 5.26　改造后的列表渲染显示效果

原有 promotion.wxml 中的代码调整如下。

```
<view wx:for="{{promotionRecommendList}}">
      <view class="dessert">
```

```
            <image class="image" src='{{item.imagePath}}'></image>
        </view>
        <view class="promotion-details">
          <text style="font-size:13px;">第{{index+1}}款: {{item.name}}</text>
        </view>
        <view class="promotion-details">
          <text style="font-size:13px;color:blue">价格: {{item.price}}</text>
        </view>
        <view class="promotion-details">
          <text wx:if="{{!item.isHighlyRecommended}}" style="font-size:12px;color:red;">
                                                                    强烈推荐</text>
        </view>
    </view>
```

promotion.wxss 中的样式调整如下。

```
.dessert{
  display:flex;
  flex-direction: row;
  justify-content: center;
  flex-wrap:wrap
}
.image{
  width:350rpx;
  height:210rpx;
}
.promotion-details{
  display:flex;
  justify-content: center;
  align-items: center
}
```

其中,dessert 样式用于实现对象数组中促销商品图片的居中显示;image 样式用于重新定义渲染显示的图片尺寸大小;promotion-details 样式用于实现每一个商品详细文字信息的居中显示。

5.7 数据更新

逻辑层数据的更新操作需要在视图层能正确显示,微信小程序提供 this.setData 函数用于实现这一功能。为介绍该函数的用法,我们可以编写一些小的测试代码来进行讲解。

首先在编辑器目录树 pages 中新建一个目录,命名为 helpCenter,然后依次新建 helpCenter.js、helpCenter.json、helpCenter.wxml、helpCenter.wxss 四个文件,并在 app.json 文件中的 pages 对象数组中添加“pages/helpCenter/helpCenter”页面信息。同时还需将 app.json 文件中 tabBar 的 list 中“联系我们”关联的 pagePath 修改为我们新增的页面,相应代码为:

```
{
  "text": "联系我们",
```

```
  "pagePath": "pages/helpCenter/helpCenter",
  "iconPath": "images/icons/helpCenter.png",
  "selectedIconPath": "images/icons/selectedHelpCenter.png"
}
```

此时，当单击“联系我们”时，除了按钮图像会发生变化外，只有一个空白页面，需要往此空白页面中添加相应内容，以此来介绍逻辑层与视图层数据同步更新如何实现。

首先在 helpCenter. wxml 中新增一个 text 元素，它的内容绑定到内部状态变量 accessCount 上，该变量在 helpCenter. js 中 data 域进行定义，并初始化为 0，用于记录该页面被访问到的次数。

```
<view class = "container">
    <text >该服务中心页面被访问{{accessCount}}次</text >
    <button type = "primary" bindtap = 'f0' size = "mini"> Show Access Count </button >
</view >
```

其中，button 按钮用于对 accessCount 变量的取值进行显示输出。单击 Show Access Count 这个 button 按钮的代码在 helpCenter. js 中添加，如下。

```
f0:function(event){
    console.log(this.data.accessCount)
}
```

此处是用于读取内部状态变量 accessCount 的值，运行效果如图 5.27 所示。

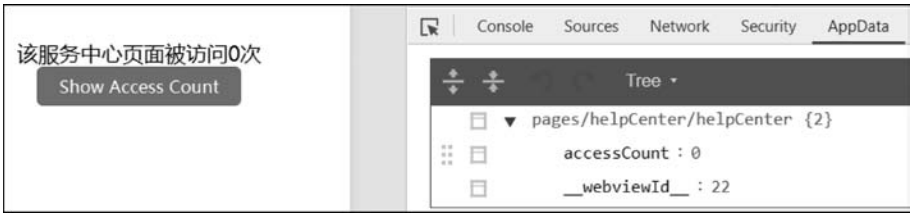

图 5.27　数据更新测试页面 1

以上代码是读取 data 中的 accessCount 变量的值并将其结果在 helpCenter. wxml 页面中渲染出来，如果是写 accessCount 变量该如何实现呢？比如现在要将 accessCount 变量的值递增 1，在 helpCenter. js 中将触发函数 f0 的代码修改为：

```
f0:function(event){
    this.data. accessCount = this.data. accessCount + 1
  }
```

分析图 5.28 可以发现，在 AppDaa 中，accessCount 的值随着单击次数的增加，该内部状态变量的值也发生了改变，但是视图层 WXML 界面的 accessCount 的值仍为初始化时定义的值 0，这是怎么回事呢？视图层的数据并没有同步更新为逻辑层的内部状态变量的值。

因此，这种试图通过对内部状态变量的直接赋值来实现变量值写入的方式是不能让这个框架自动更新到对应的视图部分的。同时还存在一个问题，就是当单击 Show Access Count 按钮后，AppData 标签并不能实时看到 accessCount 值的更新，此时需要单击一下其

他的标签再回到 AppData 标签，才会看到 accessCount 被更新后的值。

正确的实现内部状态变量赋值并自动渲染到视图层的方式是使用小程序提供的 this.setData()方法。

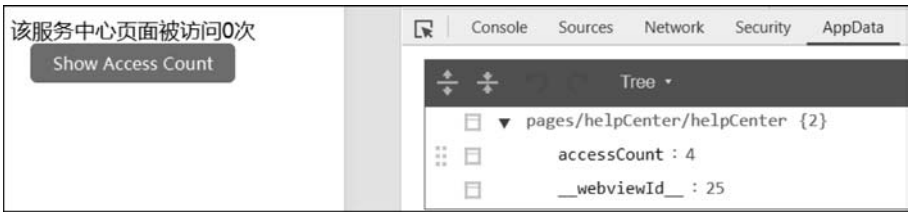

图 5.28　数据更新测试页面 2

将 helpCenter.js 中的相应代码修改为：

```
f0:function(event){
    this.setData({
      accessCount: this.data. accessCount + 1
    })
  }
```

重新编译运行，显示结果如图 5.29 所示。

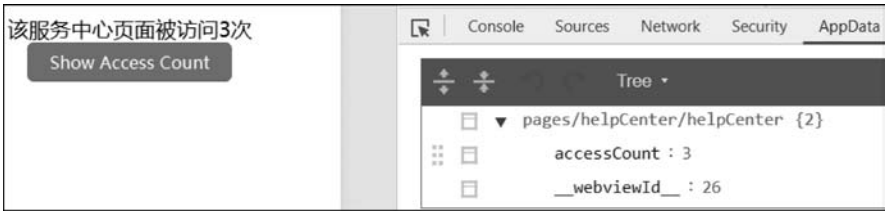

图 5.29　数据更新测试页面 3

可以发现，不管是在渲染页面 helpCenter.wxml 中的文字部分，还是在 AppData 中，内部状态数据的改变均实现了同步更新。

通过 **this.setData()**函数的调用，就是告诉框架，希望它来完成 accessCount 状态数据的更新，并且在更新完成之后，要让框架来通知这个视图层，对它所绑定到的这个内部状态变量的相关部分进行视图的更新。

this.setData 方法不仅可以更新一个已有的内部状态变量的取值，而且可以根据需要动态地来新增一个内部状态变量，也可以实现对内部状态数据中的某一小部分数据进行局部的更新。如在 helpCenter.wxml 中新增一个按钮“获取电子邮件信息”，当被单击时显示联系方式中的电子邮件，在该文件中添加以下代码。

```
<button type = "primary" bindtap = 'f1' size = "mini">获取电子邮件信息</button>
<text s>电子邮件: {{email}}</text>
```

然后在 helpCenter.js 中添加“获取电子邮件信息”按钮的触发函数代码。

```
f1: function (event) {
    this.setData({
```

```
    email: "celavi@gmail.com",
  })
}
```

上述代码 this.setData 函数中新增一个内部状态变量 email，该变量未在 data 中进行定义，通过 this.setData 函数定义并赋值。增加这两段代码运行后的初始界面及结果界面分别如图 5.30 和图 5.31 所示。

图 5.30　数据更新测试页面 4

图 5.31　数据更新测试页面 5

5.8　页面间跳转的实现机制

除了前面介绍的使用< navigator >标签跳转之外，小程序还提供 JS 事件跳转方式。

1. wx.navigateTo(Object)

API 中的导航 **wx.navigateTo(Object)** 功能，是保留当前页面，跳转到应用内的某个页面，但是不能跳到 tabBar 页面。使用 wx.navigateBack 即可以返回到原页面，小程序页面栈最多 10 层。其参数 Object 说明如表 5.8 所示。

表 5.8　wx.navigateTo(Object)参数说明

参　数	类　型	必　填	说　明
url	string	是	需要跳转的应用内非 tabBar 的页面路径，路径后可以带参数。参数与路径之间使用“?”分隔，参数键与参数值用“=”相连，不同参数用“&”分隔，如 'path? key = value&key2 = value2'
events	object	否	页面间通信接口，用于监听被打开页面发送到当前页面的数据
success	function	否	接口调用成功的回调函数
fail	function	否	接口调用失败的回调函数
complete	function	否	接口调用结束的回调函数(调用成功、失败都会执行)

2. wx.redirect(Object)

关闭当前页面，跳转到应用内的某个页面。但是不允许跳转到 tabBar 页面，其参数说明如表 5.9 所示。

表 5.9　wx.redirectTo(Object)参数说明

参　数	类　型	必　填	说　明
url	string	是	需要跳转的应用内非 tabBar 的页面路径，路径后可以带参数。参数与路径之间使用“?”分隔，参数键与参数值用“=”相连，不同参数用“&”分隔，如 'path? key = value&key2 = value2'
success	function	否	接口调用成功的回调函数
fail	function	否	接口调用失败的回调函数
complete	function	否	接口调用结束的回调函数(调用成功、失败都会执行)

3. wx.switchTab(Object)

跳转到 tabBar 页面，并关闭其他所有非 tabBar 页面，其参数说明如表 5.10 所示。

表 5.10　wx.switchTab(Object)参数说明

参　数	类　型	必　填	说　明
url	string	是	需要跳转的 tabBar 页面的路径(需在 app.json 的 tabBar 字段定义的页面)，路径后不能带参数
success	function	否	接口调用成功的回调函数
fail	function	否	接口调用失败的回调函数
complete	function	否	接口调用结束的回调函数(调用成功、失败都会执行)

我们在 promotion.wxml 中修改得到如下代码。

```
<view wx:for = "{{promotionRecommendList}}">
    <view class = "dessert" bindtap = "f1">
       <image class = "image" src = '{{item.imagePath}}'></image>
    </view>
    <view class = "promotion - details">
       <text style = "font - size:13px;">第{{index + 1}}款: {{item.name}}</text>
    </view>
    <view class = "promotion - details">
       <text style = "font - size:13px;color:blue">价格: {{item.price}}</text>
    </view>
    <view class = "promotion - details">
       <text wx:if = "{{!item.isHighlyRecommended}}" style = "font - size:12px;color:red;">
                                                          强烈推荐</text>
    </view>
  </view>
```

上述代码中的粗体字部分 **bindtap="f1"**即为添加代码，即当单击该 view 组件时，触发函数 f1，也即相当于单击 image 组件中的图片时，触发 f1。而函数 f1 在 promotion.js 中定义：

```
f1:function(event){
      wx.navigateTo({
        url:"/pages/details/details"
      })
}
```

跳转目标 details 为新建的一个界面，其 details.wxml 代码为：

```
<view class="container">
   <text bindtap="f2">这是一个显示促销商品详细信息的页面</text>
</view>
```

其中，text 组件的触发函数 f2，用于当用户单击到该文本时跳转回原来界面。f2 函数在 details.js 中定义。

```
Page({
  f2:function(event){
    wx.navigateBack({
    })
  }
})
```

对上述代码简单分析后不难发现，不管单击对象数组中的哪一个 image，跳转页面的显示结果都一样。也即当用户单击图 5.32 中任何一幅图片，跳转的页面都是图 5.33 的显示效果。

图 5.32　列表渲染显示结果

图 5.33　跳转页面显示效果

如果想要实现这种跳转效果，即在单击某个特定促销商品图片的时候，弹出与该商品相对应的详细信息，而不是同一个页面，该如何实现？

首先需要在 promotion.js 中对 promotionRecommendList 中的每一个对象新增一个 id，分别对应每一个商品的唯一 id 标识符，例如：

```
promotionRecommendList:[
{
  name:"Frozen Mochi",
```

```
    price:"￥59.9 元起",
    imagePath:"/images/Celavi - Frozen - Mochi.jpg",
    isHighlyRecommended:true,
    id:10
  },
  {
    name:"Hokkaido Scallop&Oyster Ceviche",
    price:"￥199.9 元起",
    imagePath: "/images/Celavi - Hokkaido Scallop & Oyster Ceviche.jpg",
    isHighlyRecommended: false,
    id: 11
  },
  {
    name: "Rose&Watermelon Petit Gateau",
    price: "￥169.9 元起",
    imagePath: "/images/celavi - Rose&Watermelon Petit Gateau.jpg",
    isHighlyRecommended: true,
    id: 12
  },
]
```

可以在视图页面中通过代码：

```
<text>{{item.id}}</text>
…
```

将该 id 展示出来。

接下来的任务是，如何在每一个 view 元素被单击的时候，将该商品的 id 值传递给对应的事件处理函数来处理？

实现方法：在 view 元素上定义一个对应的自定义数据属性（data-的方式）来记录该促销商品的 id：

```
<view class = "dessert" bindtap = "f1" data - promotion - id = "{{item.id}}">
```

然后在 promotion.js 中将 f1 的代码修改为：

```
f1:function(event){
    var promotionId = event.currentTarget.dataset.promotionId
        wx.navigateTo({
          url: '/pages/details/details?id = ' + promotionId,
    })
  }
```

注意：

（1）上述代码中的 data-promotion-id 为微信小程序特有的自定义属性设置，其语法为以 data-开头。

（2）如果写成 data-promotionid = “{{item.id}}”，或为 data-PROMOTIONID = “{{item.id}}”，在获取值时都为：

```
event.currentTarget.dataset.postid
```

即不管 data-后跟字符大小写，都将转换为小写。

(3) 如果写成 data-name-id 这种形式，在获取值的时候会自动去掉连字符，以驼峰方式去获取，变为 nameId，这就解释了上述代码中变量获取为什么是 promotionId，而不是其他形式。经实测，如果将

```
var promotionId = event.currentTarget.dataset.promotionId
```

中的 event.currentTarget.dataset.promotionId 改为 event.currentTarget.dataset.promotionid，控制台会输出 undefined 错误信息。写成其他形式均会报错。

(4) 上述代码中的 currentTarget 指当前单击的对象，dataset 是指自定义属性的集合。

以上代码运行后，当用户单击不同 view 组件中的图片对象时，控制台会输出相应的 id 值，如图 5.34 所示。

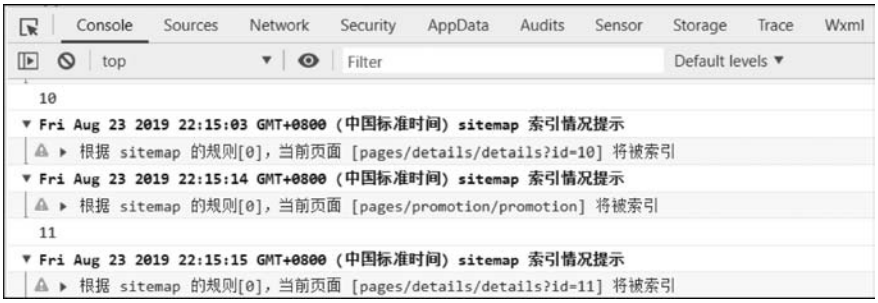

图 5.34　控制台输出不同 view 标签的 id 信息

上述代码中的 url：'/pages/details/details? id=' + promotionId 用于获取完整的跳转 URL，下面深入探讨完整 URL 的获取方式。

当 details 页被打开的时候，需要有一种机制，能知道在对应的**完整的 URL** 中被指定的 id 参数是多少。小程序框架在每次以这样完整 URL 的方式打开 details 页时，会首先调用 details 页注册的 onload 生命周期函数来对 details 页进行初始化，将“?”后面的 querystring 中的每个参数值进行解析，组合成一个对应的 JavaScript 对象，将该对象作为实参值传递给 onload 方法。

下面在 details.js 页面中添加如下代码。

```
Page({
  onLoad:function(options){
  console.log(options.id)
},
```

假如用户单击的是 promotion.wxml 中的第一个 view 标签，也就是 id=10 的促销商品，此时绑定事件触发函数 f1，其完整代码为，

```
f1:function(event){
      var promotionId = event.currentTarget.dataset.promotionId
      wx.navigateTo({
         url: '/pages/details/details?id=' + promotionId,
      })
}
```

触发函数 f1 先通过事件对象 event 获取到当前元素，也就是当前图片(id＝10)自定义的这个数据属性 promotionId 的取值(10)，以附带 id 为 10 的这样一个完整的 URL 来打开 details 页。

注意：这里梳理一下 promotion 页面与 details 页面关于 promotionId 变量的访问问题。promotion 与 details 是同一个小程序项目的两个不同页面，promotion 页面通过以下代码：

```
<view class = "dessert" bindtap = "f1">
   <image class = "image" src = '{{item.imagePath}}'></image>
</view>
```

在 view 组件中绑定了一个 f1 函数，该 f1 函数在 promotion.js 中定义，用于相应 view 组件单击时触发的事件，通过以下代码：

```
wx.navigateTo({
    url: '/pages/details/details?id = ' + promotionId,
})
```

将两个页面关联起来，也即 promotion 页面通过 url：'/pages/details/details? id＝' ＋ promotionId 语句指定其跳转页面为 details，details 是 promotion 的跳转目标页面。

小程序框架以这样一个完整的 URL 来打开 details 页，首先会将其中包含的 id＝10 的 querystring 解析成一个 **options 参数对象，然后调用 details 页，在对应的 Page Object 中定义 Onload 生命周期函数，并且给这个 Onload 生命周期函数传入一个刚才解析出来的 Options 参数对象**。Onload 函数在执行的过程中会将 options 中的 id 参数的取值打印出来。

```
Page({
  onLoad:function(options){
    console.log(options.id)
  }
})
```

这样一来，details 页在初始化的时候就能够获得它本次被打开时被指定的 promotionId 的值，也就是说，通过这种机制，details 页面获取了一个不在该页面中定义的外部变量的值。当然也可以在 details.js 文件中定义一个该文件本身的内部状态数据变量 pid，用于存放 promotion 页面传递过来的 promotionId 的值。

```
//details.js
Page({
  data:{
    pid:0
  },
  …
})
```

然后在 details 页面的 onLoad 函数中，采用 this.setData 函数对该页面的内部变量 pid 进行赋值。

```
//details.js
onLoad:function(options){
   //console.log(options.id)
    this.setData({
       pid:options.id
    })
},
```

可以发现，URL 中获取到的参数 promotionId＝11 成功保存到了 pid 内部变量中，如图 5.35 所示。

图 5.35　跳转目标页面 details 获取 promotion 页面传递过来的变量值

该 promotionId 也可以在之后的 details 页的视图渲染中，以数据绑定的方式进行渲染输出，如图 5.36 所示。

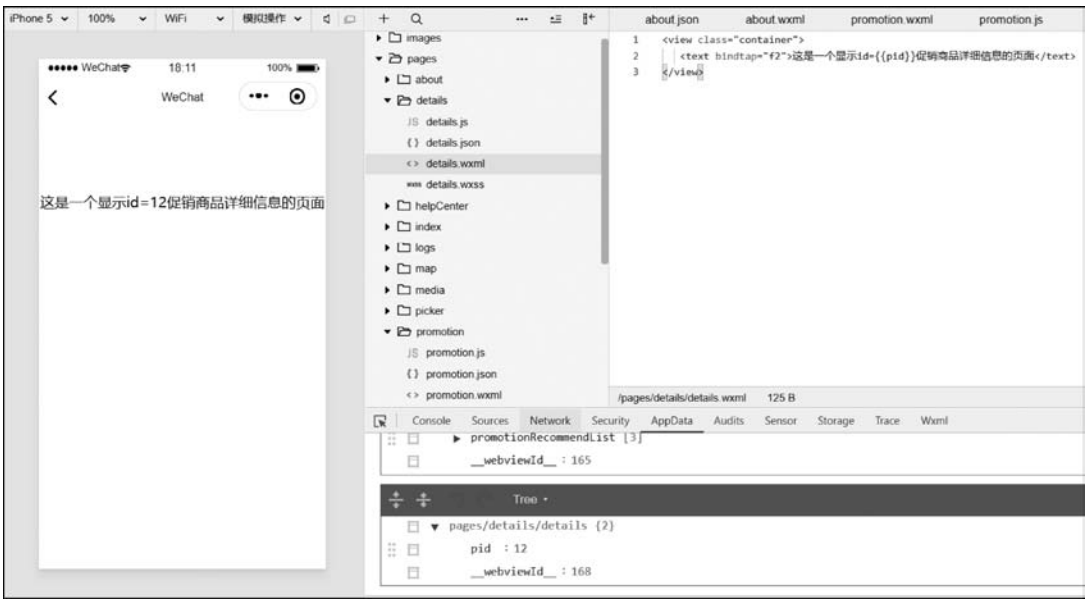

图 5.36　details 页面接收到 promotion 页面传递过来的变量并渲染显示

思 考 题

1. 小程序生命周期函数有哪些？触发的条件分别是什么？

2. 如果要实现图 5.18 中图片在左边显示，文字在右边显示的效果，如何定义 WXSS 样式文件？在 WXML 文件中如何布局？

3. 本章中关于页面信息的配置都是静态配置，如果页面数据需要在运行时动态地从服务器上获取，如何在小程序中动态地设置页面的标题？

4. 思考规划设计的小程序应用服务项目需要定义几个页面？页面之间如何实现跳转链接？需要用到几个 tabBar？并在小程序开发工具中予以实现。