第二单元 小程序逻辑层

CONTENT

目录

2.1

CHAPTER

小程序全局配置文件

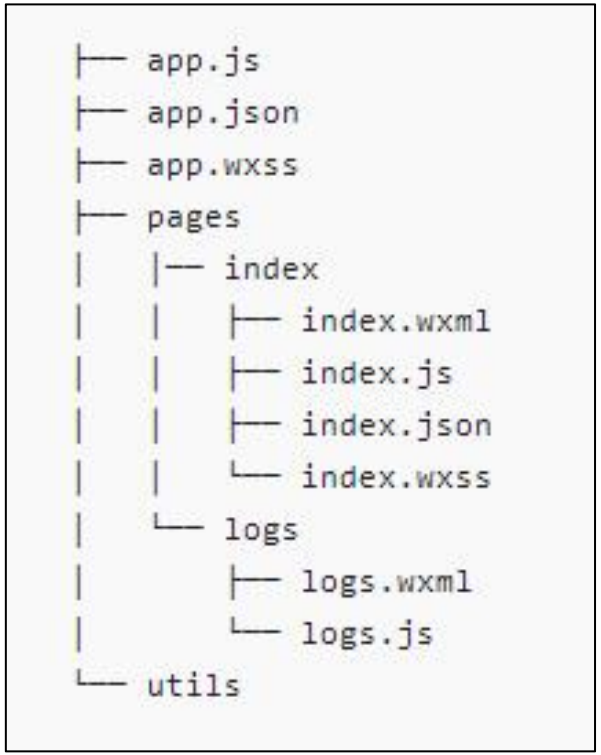
一、任务描述

开发者根据实际需要，重新修改定义全局配置文件app.json中的window属性，修改window属性中各种样式实现自己想要的效果。

二、导入知识点

pages用于指定小程序由哪些页面组成，每一项都对应一个页面的路径（含文件名）信息。文件名不需要写文件后缀，框架会自动去寻找对应位置的 .json, .js, .wxml, .wxss 四个文件进行处理。小程序中新增/减少页面，都需要对 pages 数组进行修改。

小程序的目录结构主要包含项目配置文件、主体文件、页面文件和其他文件。本任务将基于任务1.7创建第一个微信小程序的项目，对代码文件的构成展开分析。开发目录如图所示。

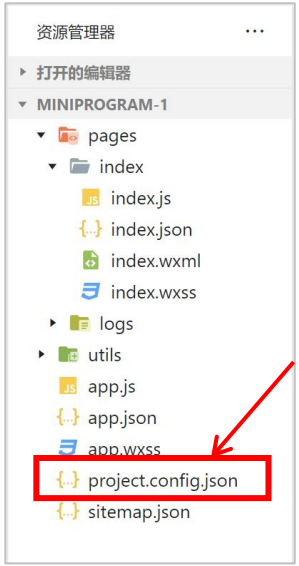


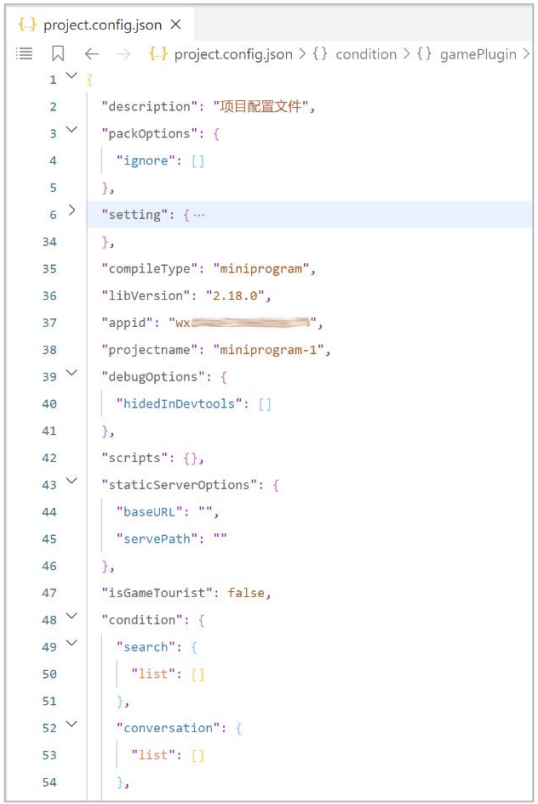
二、导入知识点

1.项目配置文件

项目配置文件是project.config.json，每个小程序在新建时都会自动生成一个project.config.json文件，该文件直接位于项目根目录下，如左图所示。

其内部代码可用来定义小程序的项目名称、AppID等内容，如右图所示。

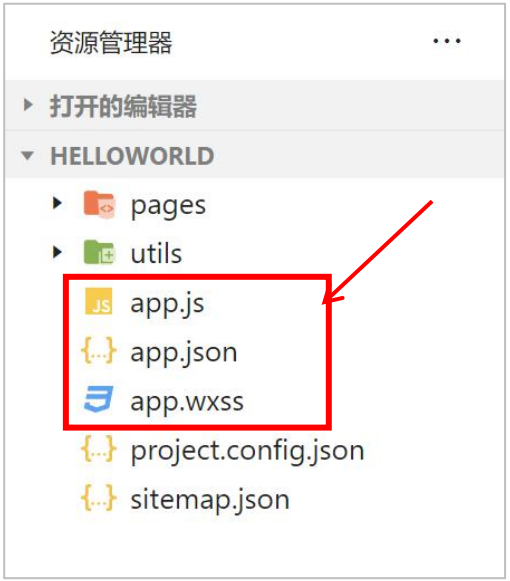




二、导入知识点

2.主体文件

小程序主体文件名称均为app，同样直接位于项目根目录下，如图所示。



二、导入知识点

2.主体文件

主体文件app根据后缀名不同分为3种类型的文件分别是app.json、app.js和app.wxss。

app.json是小程序的全局配置文件，主要包含了小程序所有页面文件的路径地址、导航栏样式等，必填文件，如图所示。

app.json

app.json > ...

```
{
  "pages":[
    "pages/index/index",
    "pages/logs/logs"
  ],
  "window":{
    "backgroundTextStyle":"light",
    "navigationBarBackgroundColor": "#fff",
    "navigationBarTitleText": "Weixin",
    "navigationBarTextStyle":"black"
  },
  "style": "v2",
  "sitemapLocation": "sitemap.json"
}
```

二、导入知识点

2.主体文件

小程序项目主要包含了pages和window两个属性。事实上，除了pages和window，app.json还可以配置tabBar、networkTimeout及debug等属性，这些属性的具体说明，如表所示。

属性	类型	说明
pages	String Array	必填属性，用于记录小程序所有页面文件的路径地址。其中如果pages中包含多个页面，pages/index/index将默认为小程序的初始页面
window	Object	可选属性，用于设置页面的窗口表现，例如导航栏的背景颜色、标题文字内容以及文字颜色等
tabBar	Object	可选属性，用于设置页面底部tab工具条的表现
networkTimeout	Object	可选属性，用于设置各种网络请求的超时时间
debug	Boolean	可选属性，用于设置是否开启调试模式

二、导入知识点

2.主体文件

（1）pages属性：对应的值是数组形式，数组的每一项都是以字符串形式记录小程序页面的路径地址。表示当前共有两个页面，分别是index和log页面，并且其中的index页面被默认为小程序的初始页面。

```
"pages":[
  "pages/index/index",
  "pages/logs/logs"
],
```

二、导入知识点

2.主体文件

（2）window属性：对应的值是对象形式，其中包括了小程序页面顶端导航栏的背景颜色、标题文字内容以及文字颜色等属性，具体可以包含的对象属性，如表所示。

属性	类型	默认值	描述
navigationBarBackgroundColor	HexColor	#000000	导航栏背景颜色，默认值表示黑色，也可以简写为#000
navigationBarTextStyle	String	white	导航栏标题颜色，默认值表示白色，该属性值只能是white或black
navigationBarTitleText	String		导航栏标题文字内容，默认为无文字内容
navigationStyle	String	default	导航栏样式，只支持default或custom,其中custom用于自定义导航栏内容，只保留右上角的小图标
backgroundColor	HexColor	#ffffff	窗口的背景颜色，默认值表示白色，也可以简写为#ffff
backgroundTextStyle	String	dark	下拉加载的样式，该属性值只能是dark或light
backgroundColorTop	String	#ffffff	顶部窗口的背景颜色，只有iOS有效
backgroundColorBottom	String	#ffffff	底部窗口的背景颜色，只有iOS有效
enablePullDownRefresh	Boolean	false	是否开启下拉刷新功能
onReachBottomDistance	Number	50	页面上拉触底事件触发时距页面底部的距离，单位为像素（px）

三、实现效果

通过修改app.json文件，把顶部窗口的背景颜色变为红色，导航栏标题文字内容改为“微信小程序开发教程”，导航栏标题颜色变为白色，实现如图所示的效果。

三、实现效果

通过修改app.json文件，把顶部窗口的背景颜色变为红色，导航栏标题文字内容改为“微信小程序开发教程”，导航栏标题颜色变为白色，实现如图所示的效果。

四、任务实现

对app.json文件中的window属性进行修改。如，顶部窗口的背景颜色"navigationBarBackgroundColor"设置为"#f00"红色，导航栏标题文字"navigationBarTitleText"设置为“微信小程序开发教程”，导航栏标题颜色"navigationBarTextStyle"变为"white"白色，实现代码如图所示。

app.json

```
1  {
2    "pages":[
3      "pages/index/index",
4      "pages/logs/logs"
5    ],
6    "window":{
7      "backgroundTextStyle":"light",
8      "navigationBarBackgroundColor": "#f00",
9      "navigationBarTitleText": "微信小程序开发教程",
10     "navigationBarTextStyle":"white"
11   },
12   "style": "v2",
13   "sitemapLocation": "sitemap.json"
14 }
```

- 第8行：将导航栏背景颜色从白色”#ffffff”改为红色”#f00”。
- 第9行：将导航栏文字内容从“WeChat”改为“微信小程序开发教程”。
- 第10行：将导航栏文字颜色从黑色”black”改为白色”white”。

2.2

CHAPTER

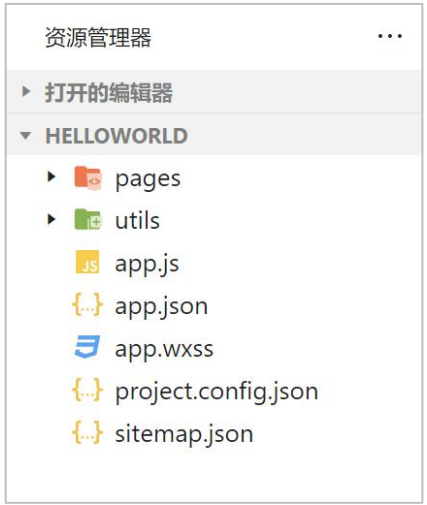
小程序tabBar栏

一、任务描述

开发者根据实际需要，定义全局配置文件app.json中的tabBar属性，通过添加tabBar属性，实现窗口底部tab栏多个页面的切换，实现自己想要的多页面效果。

二、导入知识点

小程序的全局配置文件是app.json，主要包含了小程序所有页面的路径地址、导航栏样式等内容，页面的路径地址、导航栏样式为必填信息，如图所示。



app.json

```
{
  "pages":[
    "pages/index/index",
    "pages/logs/logs"
  ],
  "window":{
    "backgroundTextStyle":"light",
    "navigationBarBackgroundColor": "#fff",
    "navigationBarTitleText": "Weixin",
    "navigationBarTextStyle":"black"
  },
  "tabBar": {
    "position": "bottom",
    "borderStyle": "white",
    "list": [
      {
        "pagePath": "     ",
        "text": "    ",
        "iconPath": "     ",
        "selectedIconPath": "      "
      }
    ]
  },
  "style": "v2",
  "sitemapLocation": "sitemap.json"
}
```

二、导入知识点

如果小程序是一个多tab应用，即客户端窗口的底部有一个tab栏，进行切换页面，可以通过tabBar配置项指定tab栏的表现，以及tab切换时显示的对应页面。tabBar的属性值如表所示。

属性	类型	必填	默认值	说明
color	HexColor	是		tab上的文字默认颜色
selectedColor	HexColor	是		tab上的文字选中时的颜色
backgroundColor	HexColor	是		tab的背景色
borderStyle	String	否	black	tabBar上边框的颜色，仅支持black、white
list	Array	是		tab的列表
position	String	否	bottom	tabBar的位置，仅支持bottom、top

二、导入知识点

其中，list接收一个数组，只能配置最少2个、最多5个tab。tab按数组的顺序排序，每项都是一个对象，其属性值如表所示。

属性	类型	必填	描述
pagePath	String	是	页面路径，必须在pages中先定义
text	String	是	tab上按钮的文字
iconPath	String	否	图标路径，icon大小限制为40KB,建议尺寸为81px×81px，不支持网络图片
selectedIconPath	String	否	选中时的图标路径，icon大小限制为40KB,建议尺寸为81pxx81px，不支持网络图片

二、导入知识点

注意：当position属性值为top时iconPath和selectedIconPath属性无效，不显示图标。同时，iconPath和selectedIconPath属性不是必填内容，如下代码所示：

```
"tabBar": {
    "list": [
      {
        "pagePath": "pages/index/index",
        "text": "首页"
      },
      {
        "pagePath": "pages/phonebook/phonebook",
        "text": "通讯录"
      }
    ]
  }
}
```

app.json

app.json > ...

```
{
  "pages":[
    "pages/index/index",
    "pages/logs/logs"
  ],
  "window":{
    "backgroundTextStyle":"light",
    "navigationBarBackgroundColor": "#fff",
    "navigationBarTitleText": "Weixin",
    "navigationBarTextStyle":"black"
  },
  "tabBar": {
    "position": "bottom",
    "borderStyle": "white",
    "list": [
      {
        "pagePath": "    ",
        "text": "   ",
        "iconPath": "    ",
        "selectedIconPath": "     "
      }
    ]
  },
  "style": "v2",
  "sitemapLocation": "sitemap.json"
}
```

二、导入知识点

下面将用图例的方式把tabBar中list属性值的对应关系展示出来，如图所示。

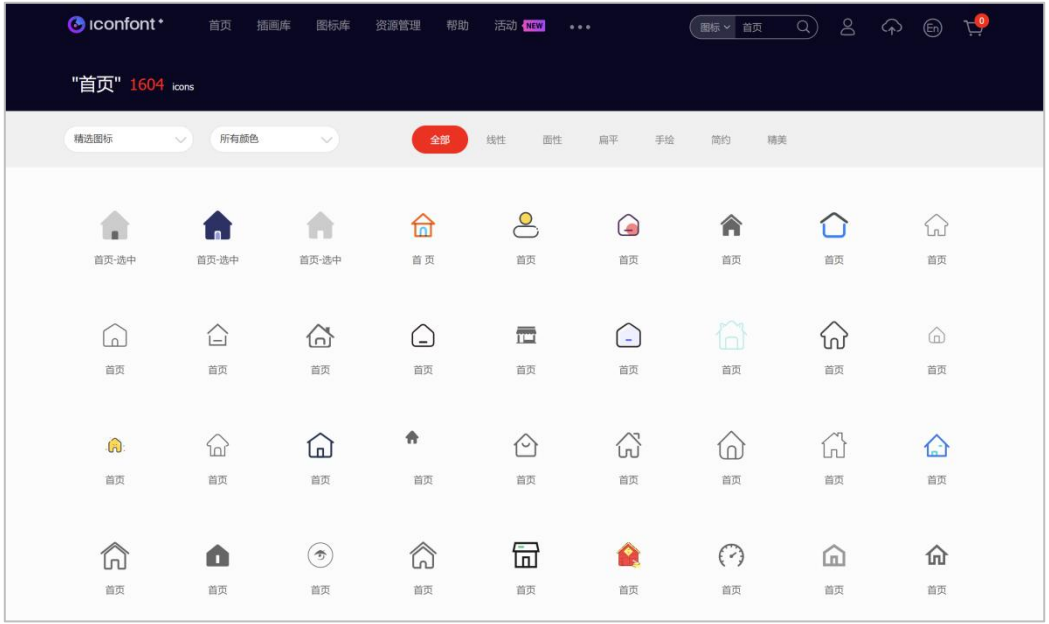
三、实现效果

通过编写app.json文件中tabBar中list属性值，设置成四个tab栏显示，并且可以相互选中切换。没选中的icon图标的颜色和文字颜色就变成绿色(#1AAD16)，如图所示的效果。

四、任务实现

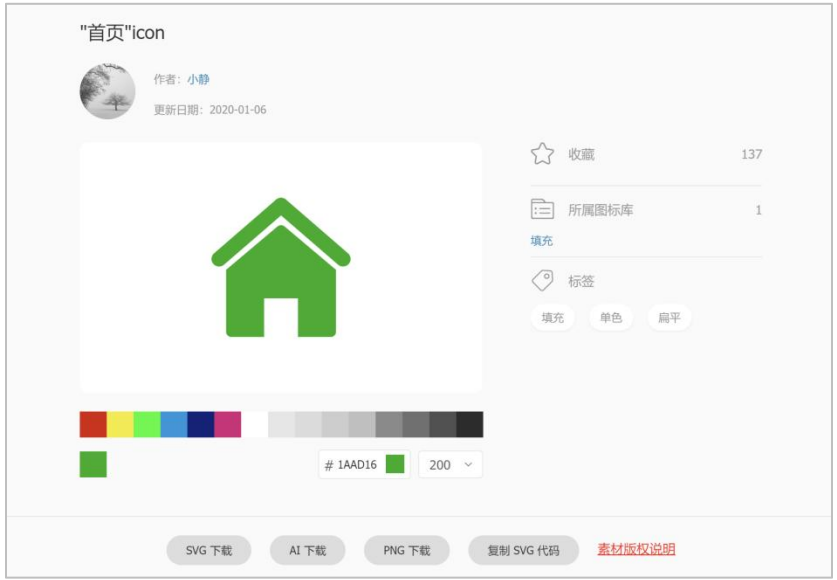
首先，进入阿里巴巴矢量图库网站https://www.iconfont.cn/，在“搜索图标”栏上查找需要下载的图标。

如在“搜索图标”栏上输入“首页”，就会出现很多“首页”icon图标，如图所示。



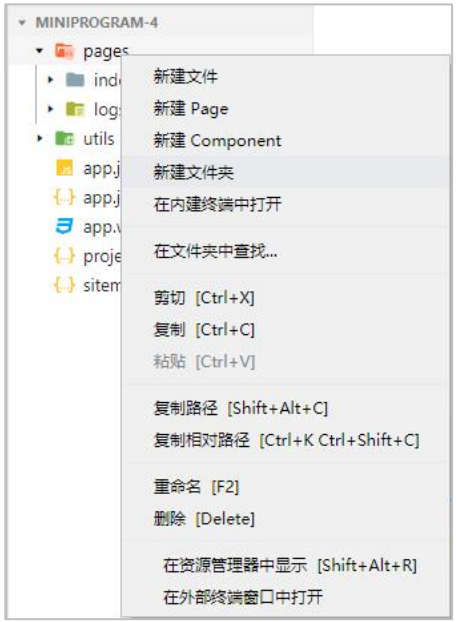
四、任务实现

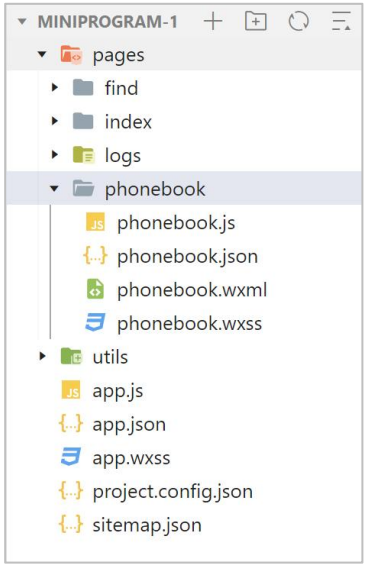
根据需求选中一个“首页”icon图标，单击“下载”，弹出对话框，分别选中灰色“#515151”和绿色“#1AAD16”进行PNG格式的图标下载，以下载“#1AAD16”绿色图标为例，如图所示。



四、任务实现

然后，除pages目录下的index和logs外，需另新建“phonebook”通讯录和“find”发现2个页面文件，如图所示。





四、任务实现

新建Page完成后，在app.json文件中的"pages"列表中会自动出现"pages/phonebook/phonebook"和"pages/find/find"，如图所示。

app.json

app.json > { } tabBar > [ ] list > { } 0 >

```
1  {
2    "pages":[
3      "pages/index/index",
4      "pages/logs/logs",
5      "pages/phonebook/phonebook",
6      "pages/find/find"
7    ],
```

四、任务实现

最后，在app.json中编写”tabBar”属性list列表，代码如下：

```
  "tabBar": {
    "list": [
      {
        "pagePath": "pages/index/index",
        "text": "首页",
        "iconPath": "images/home.png",
        "selectedIconPath": "images/home-2.png"
      },
      {
        "pagePath": "pages/phonebook/phonebook",
        "text": "通讯录",
        "iconPath": "images/phonebook.png",
        "selectedIconPath": "images/phonebook-2.png"
      },
      {
        "pagePath": "pages/find/find",
        "text": "发现",
        "iconPath": "images/find.png",
        "selectedIconPath": "images/find-2.png"
      },
      {
        "pagePath": "pages/logs/logs",
        "text": "我",
        "iconPath": "images/me.png",
        "selectedIconPath": "images/me-2.png"
      }
    ]
  },
  "style": "v2",
  "sitemapLocation": "sitemap.json"
}
```

2.3

CHAPTER

小程序逻辑文件

一、任务描述

小程序开发框架的逻辑层又称为App Service,是由JavaScript编写和实现的。开发者写的所有代码最后将被打包成一份 JavaScript程序,并在小程序启动的时候运行，直到小程序被销毁。

逻辑层的主要作用是处理数据后发送给视图层渲染以及接收视图层的事件反馈。为了更方便地进行项目开发，小程序在JavaScript的基础上进行了一些优化。

一、任务描述

其提供了不同的方法和接口用于不同的文件。

（1）App()方法用于整个应用程序的注册。

（2）Page()方法用于单独页面的注册。

（3）getApp()方法用于获取整个应用实例。

（4）getCurrentPages()方法用于获取当前页面实例。

（5）提供丰富的微信原生API，可以方便地获取微信用户信息、本地存储、扫一扫、微信支付、微信运动等特殊功能。

下面就App()方法和Page()方法的创建做一讲解。

二、导入知识点

一、App()方法

App.js文件是小程序的全局逻辑文件也是整个小程序的入口文件，它控制整个小程序生命周期的文件。

1.默认app.js代码

当新建一个项目时，默认的app.js代码如图所示。

app.js

app.js > ...

```
// app.js
App({
  onLaunch() {
    // 展示本地存储能力
    const logs = wx.getStorageSync('logs') || []
    logs.unshift(Date.now())
    wx.setStorageSync('logs', logs)

    // 登录
    wx.login({
      success: res => {
        // 发送 res.code 到后台换取 openId, sessionKey, unionId
      }
    })
  },
  globalData: {
    userInfo: null
  }
})
```

二、导入知识点

一、App()方法

省略app.js中具体的函数内容后将得到以下代码框架，代码框架如下：

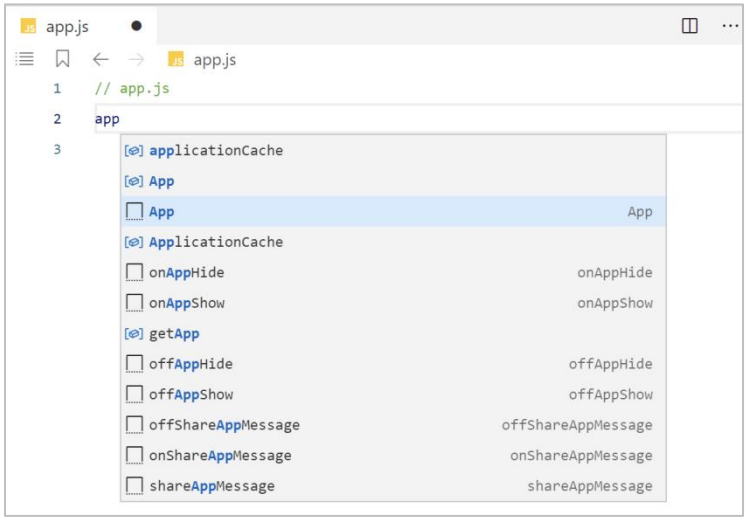
```
// app.js
App({
  onLaunch() { ... ... },
  globalData: { ... ...  }
})
```

二、导入知识点

一、App()方法

自动生成app()代码：

用户可以使用微信web开发者工具在空白app.js文件中直接输入关键词app，此时会自动出现提示列表，如图所示。



注意：App()方法只能写在小程序根目录下的app.js文件中，并且只能注册1个。

二、导入知识点

一、App()方法

选择提示列表中的第三项，直接按回车键就可以自动生成带有生命周期全套函数的代码结构，如图所示。

事实上，App()中的这些函数均为可选函数，开发者可以根据实际需要删除其中的部分函数，或保留这些函数但空着不填充内容。

app.js

app.js > ...

```
// app.js
App({
  /**
   * 当小程序初始化完成时，会触发 onLaunch（全局只触发一次）
   */
  onLaunch: function () {

  },
  /**
   * 当小程序启动，或从后台进入前台显示，会触发 onShow
   */
  onShow: function (options) {

  },
  /**
   * 当小程序从前台进入后台，会触发 onHide
   */
  onHide: function () {

  },
  /**
   * 当小程序发生脚本错误，或者 api 调用失败时，会触发 onError 并带上错误信息
   */
  onError: function (msg) {

  }
})
```

二、导入知识点

一、App()方法

App.js用App()来实现对整个程序的注册，同时App()里面还实现了对小程序生命周期的监控函数（onLaunch(), onShow(), onHide()）。对App()内部参数说明如表所示。

属性	类型	说明	描述
onLaunch()	Function	生命周期函数--监听小程序初始化	当小程序初始化完成时，会触发onLaunch（全局只触发一次）
onShow()	Function	生命周期函数--监听小程序显示	当小程序启动或从后台进入前台显示，会触发onShow函数
onHide()	Function	生命周期函数--监听小程序隐藏	当小程序从前台进入后台，会触发onHide函数
onError()	Function	错误监听函数	当小程序发送脚本错误时，会触发onError函数
onPageNotFound()	Function	页面不存在函数	当小程序需要打开的页面不存在时触发
其他自定义函数	Any	开发者可以添加任意的函数或数据到Object 参数中，用 this 可以访问	

二、导入知识点

二、Page()方法

小程序在每个页面JS文件中通过使用Page(OBJECT)方法进行页面注册，该方法可以用于指定小程序页面的生命周期函数。

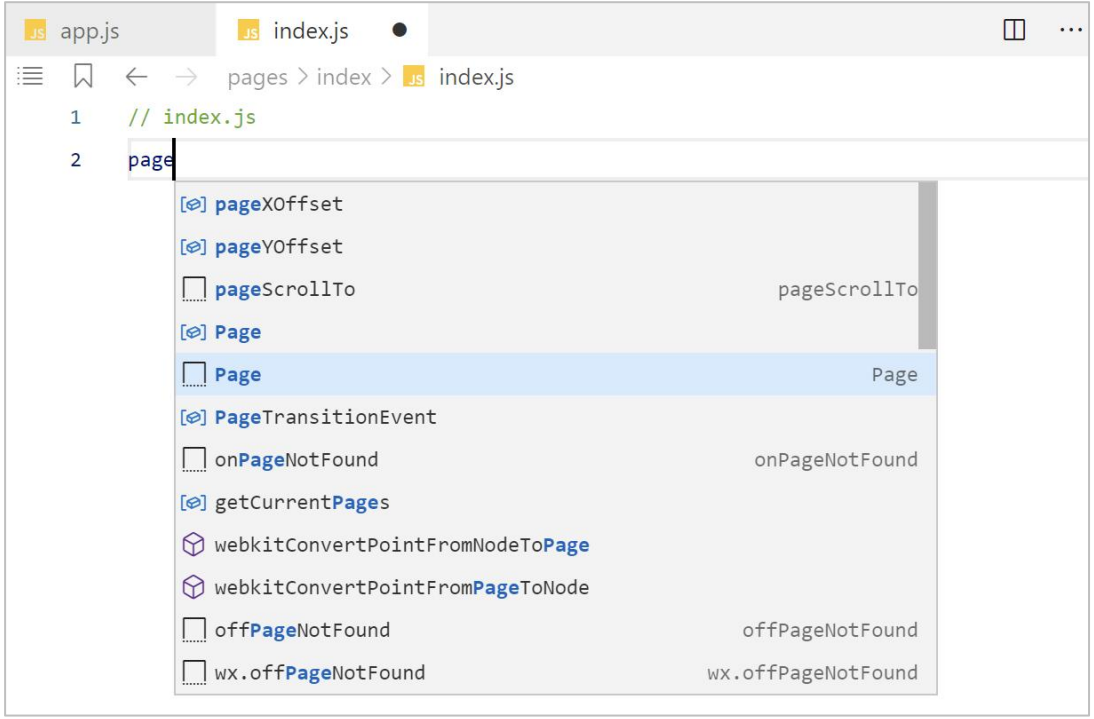
小程序中每一个页面可以放在一个文件夹中，这个文件夹中一般包括4个文件分别是.js、.json、.wxml和.wxss。

注意：这四个文件的名字最好和文件夹的名字一致，这样便于在代码框架中自动查找。

二、导入知识点

二、Page()方法

例如，用户在微信web开发者工具的index.js文件中，直接输入关键词page，此时会自动出现提示列表，如图所示。



二、导入知识点

二、Page()方法

每个页面都需要注册，index.js用Page()这个函数来注册一个页面，它接受一个object参数，用这个参数来指定页面的初始数据，生命周期函数，事件处理函数。对Page( )内部参数说明如表所示。

属性	类型	说明
data	Object	页面的初始数据
onLoad	Function	生命周期函数 - - 监听页面加载
onReady	Function	生命周期函数 - - 监听页面初次渲染完成
onShow	Function	生命周期函数 - - 监听页面显示
onHide	Function	生命周期函数 - - 监听页面隐藏
onUnload	Function	生命周期函数 - - 监听页面卸载
onPullDownRefresh	Function	页面相关事件处理函数 - - 监听用户下拉动作

二、导入知识点

二、Page()方法

选择提示列表中的第五项，直接按回车键就可以自动生成带有生命周期全套函数的代码结构，创建完成后的index.js代码如图所示。

app.js　index.js

pages > index > index.js > ...

```
// index.js
Page({
  /**
   * 页面的初始数据
   */
  data: {

  },
  /**
   * 生命周期函数--监听页面加载
   */
  onLoad: function (options) {

  },
  /**
   * 生命周期函数--监听页面初次渲染完成
   */
  onReady: function () {

  },
  /**
   * 生命周期函数--监听页面显示
   */
  onShow: function () {

  },
  /**
   * 生命周期函数--监听页面隐藏
   */
  onHide: function () {

  },
  /**
   * 生命周期函数--监听页面卸载
   */
  onUnload: function () {

  },
  /**
   * 页面相关事件处理函数--监听用户下拉动作
   */
  onPullDownRefresh: function () {

  },
  /**
   * 页面上拉触底事件的处理函数
   */
  onReachBottom: function () {

  },
  /**
   * 用户点击右上角分享
   */
  onShareAppMessage: function () {

  }
})
```

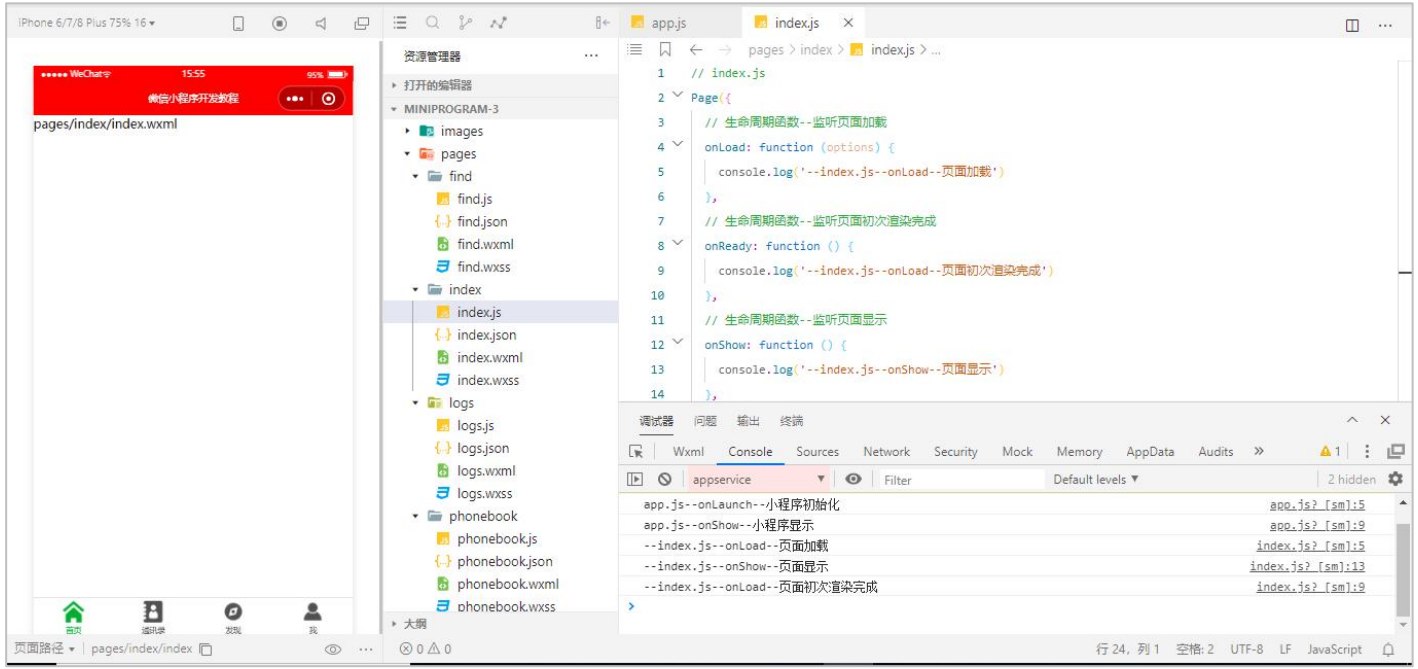
2.4

CHAPTER

小程序生命周期执行顺序

一、任务描述

设计一个小程序，测试小程序各个页面和函数的执行顺序。



二、导入知识点

小程序开发框架的逻辑层使用JavaScript引擎为小程序提供开发者JavaScript代码的运行环境以及微信小程序的特有功能。

逻辑层将数据进行处理后发送给视图层，同时接收视图层的事件反馈。开发者编写的所有代码最终将会打包成一份JavaScript文件，并在小程序启动的时候运行，直到小程序销毁。

二、导入知识点

1.小程序应用生命周期

每个小程序都需要在app.js中调用App方法注册小程序实例，绑定生命周期回调函数、错误监听和页面不存在监听函数等。

App(Object object)函数用于注册小程序，该函数必须在app.js中调用，必须调用且只能调用一次，如图所示。

app.js

app.js > ...

```
// app.js
App({
  /**
   * 当小程序初始化完成时，会触发 onLaunch（全局只触发一次）
   */
  onLaunch: function () {

  },
  /**
   * 当小程序启动，或从后台进入前台显示，会触发 onShow
   */
  onShow: function (options) {

  },
  /**
   * 当小程序从前台进入后台，会触发 onHide
   */
  onHide: function () {

  },
  /**
   * 当小程序发生脚本错误，或者 api 调用失败时，会触发 onError 并带上错误信息
   */
  onError: function (msg) {

  }
})
```

二、导入知识点

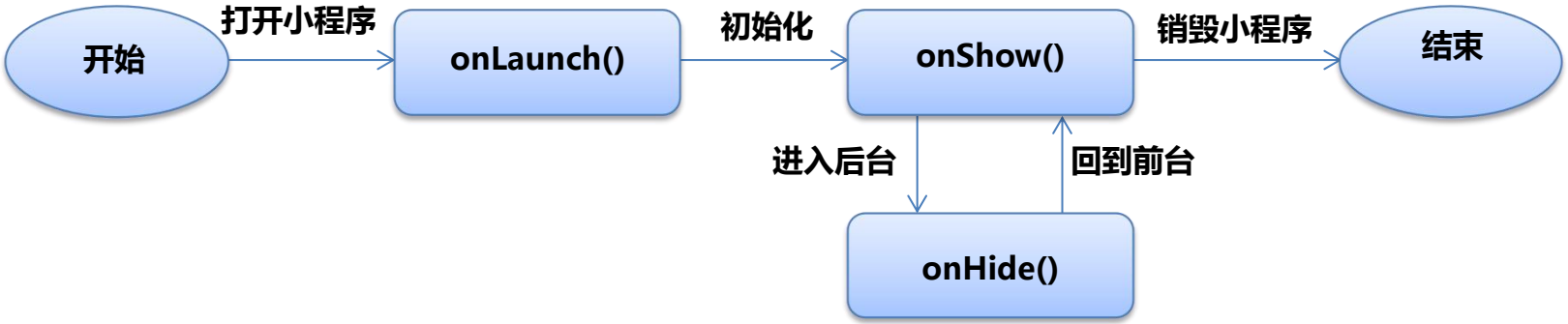
1.小程序应用生命周期

函数参数Object object的属性说明如表所示。

属性	类型	必填	描述
onLaunch	Function	否	生命周期函数 - 监听小程序初始化
onShow	Function	否	生命周期函数 - 监听小程序显示
onHide	Function	否	生命周期函数 - 监听小程序隐藏
onError	Function	否	错误监听函数
onPageNotFound	Function	否	页面不存在函数
其他自定义函数	Any	否	开发者可以添加任意的函数或数据到Object 参数中，用 this 可以访问

二、导入知识点

小程序应用与页面有各自的生命周期函数，它们在使用过程中也会互相影响。小程序应用生命周期，如图所示。



二、导入知识点

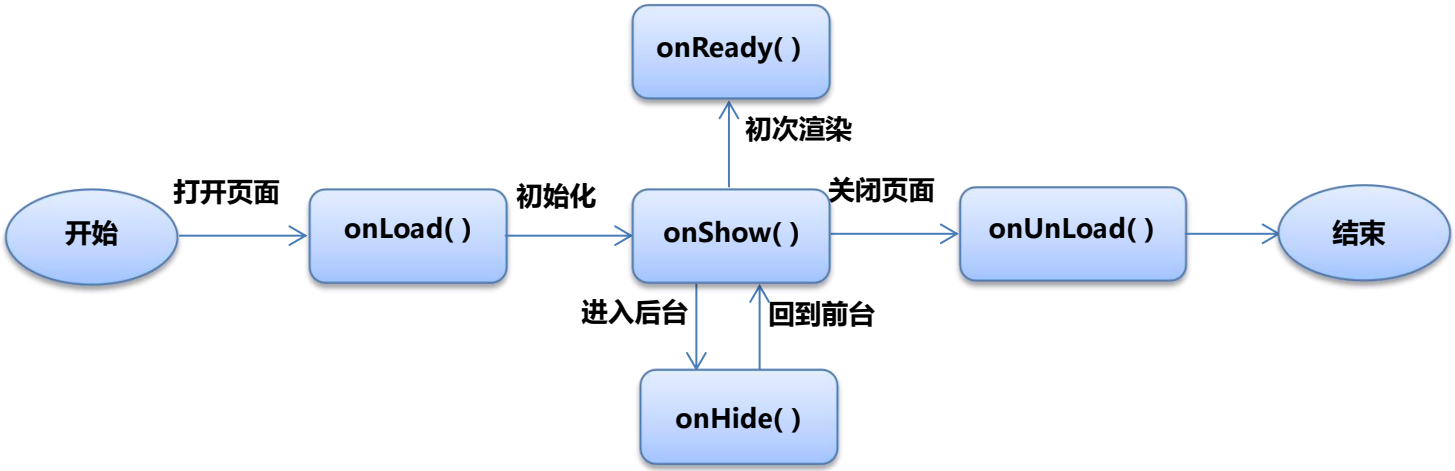
2.小程序页面生命周期

对于小程序中的每个页面，都需要在页面对应的js文件中调用Page方法注册页面，指定页面的初始数据、生命周期回调、事件处理函数等。函数Page(Object object)用于注册小程序中的一个页面。其参数Object object指定页面的初始数据、生命周期回调、事件处理函数等，其主要属性说明如表所示。

属性	类型	说明
data	Object	页面的初始数据
onLoad	Function	生命周期函数 - - 监听页面加载
onShow	Function	生命周期函数 - - 监听页面显示
onReady	Function	生命周期函数 - - 监听页面初次渲染完成
onHide	Function	生命周期函数 - - 监听页面隐藏
onUnload	Function	生命周期函数 - - 监听页面卸载
其他	Any	开发者可以添加任意的函数或数据到 object 参数中，用 this 可以访问

二、导入知识点

小程序在被打开时会首先触发onLaunch()进行程序启动，完成后调用onShow()准备展示页面，如果被切换进入后台会调用onHide()，直到下次程序在销毁前重新被唤起会再次调onShow()。小程序页面生命周期，如图所示。



二、导入知识点

3.小程序整个生命周期

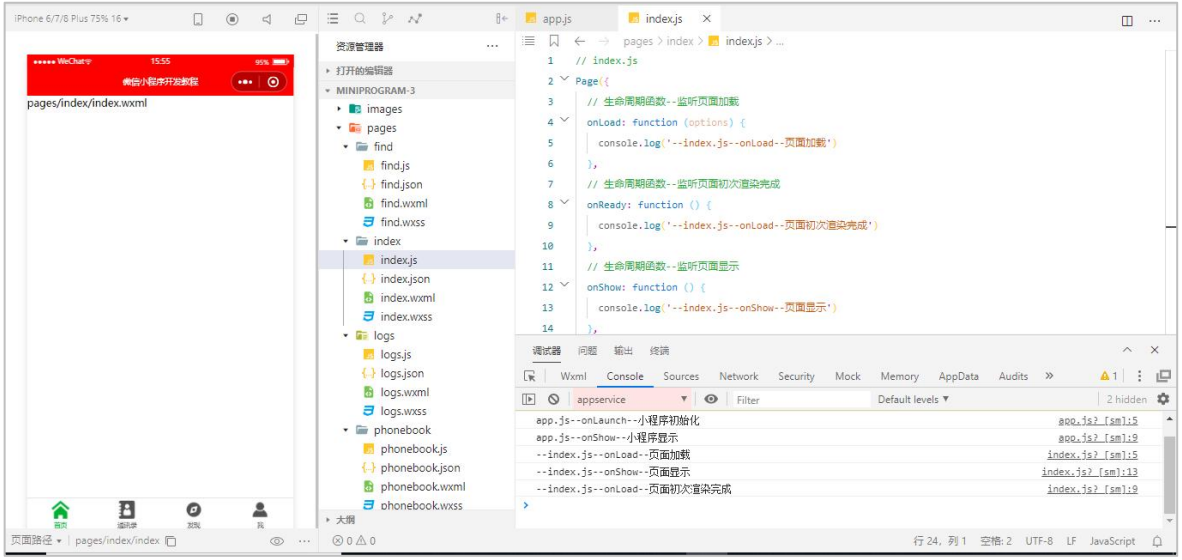
在小程序应用生命周期调用完onShow()以后就准备触发小程序页面生命周期了。页面初次打开会依次触发onLoad()、onShow()、onReady()这3个函数。

同样，如果被切换到后台，会调用页面onHide(),从后台被唤醒会调用页面onShow()。直到页面关闭会调用onUnload(),下次打开还会依次触发onLoad()、onShow()、onReady()这3个函数。小程序整个生命周期如图所示。

三、实现效果

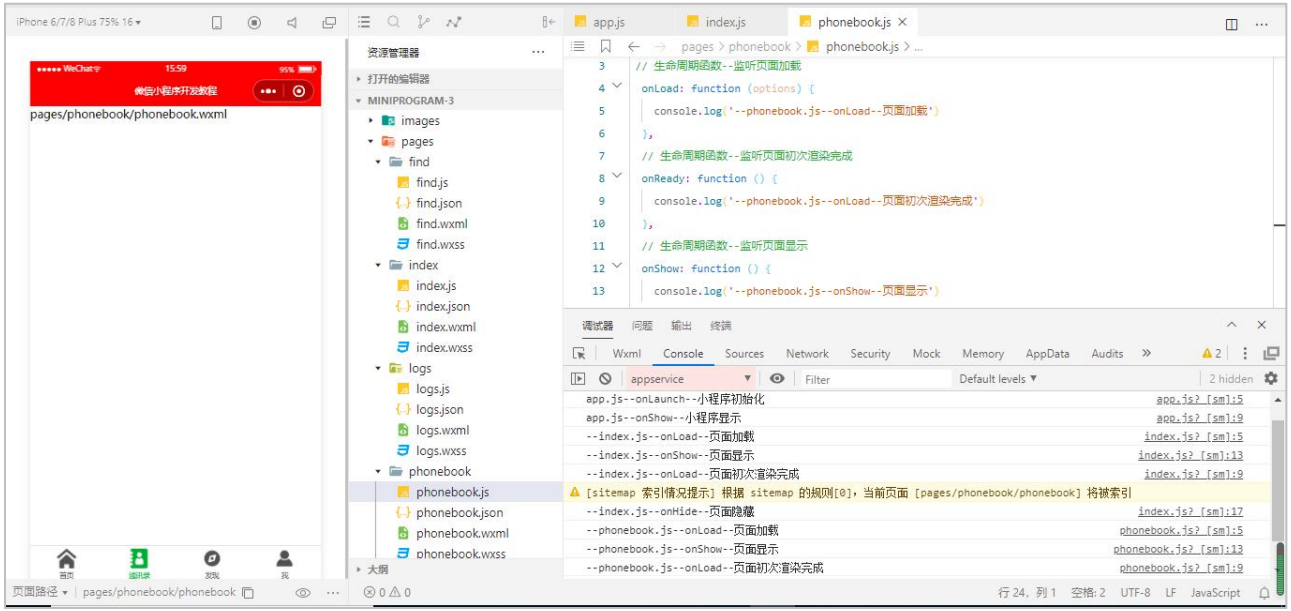
根据任务描述可以实现如下运行效果：

（1）案例运行后，在Console页面显示的程序运行顺序如图所示。从图中可以看出，小程序运行后首先执行app.js文件中的onLaunch函数和onShow函数，然后再执行index.js中的onLoad函数、onShow函数和onReady函数。



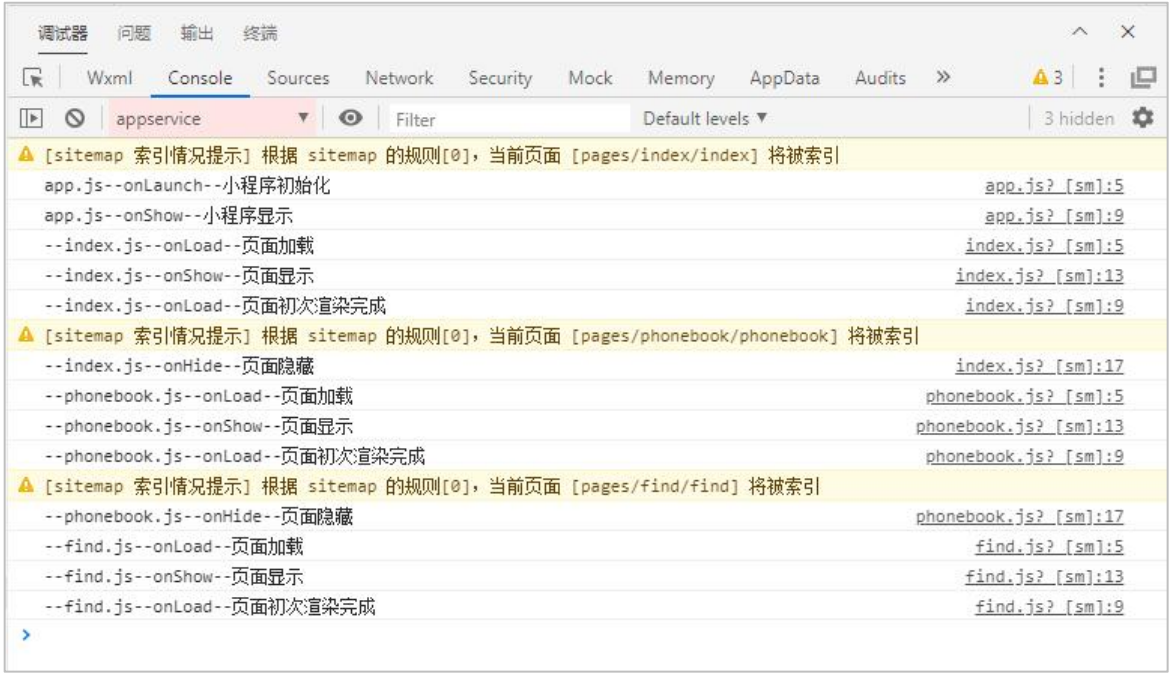
三、实现效果

（2）当点击“通讯录”标签后，首先隐藏index.js页面，然后phonebook.js页面被加载、显示和渲染，如图所示。



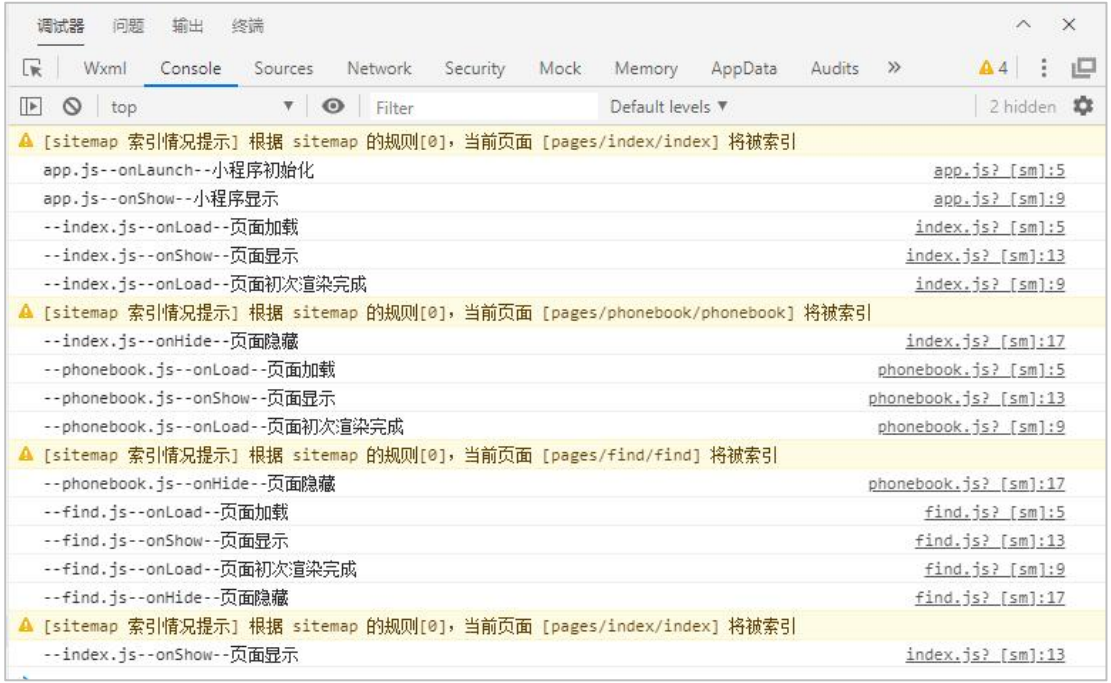
三、实现效果

（3）当点击“发现”标签后，此时首先隐藏phonebook.js页面，然后find.js显示和渲染，如图所示。



三、实现效果

（4）当点击如图所示的“切后台”按钮后，小程序首先执行find.js中的onHide函数，然后再执行app.js中index.js的onShow函数，如图所示。



四、任务实现

在“任务2.2 小程序tabBar栏”基础上完成以下内容。

（1）编写app.js文件代码。在App函数的对象参数中添加如下函数：onLaunch、onShow和onHide，每个函数都通过调用console.log函数来在Console窗口中显示程序执行的位置。app.js文件：

```
// app.js
App({
  //当小程序初始化完成时，会触发 onLaunch（全局只触发一次）
  onLaunch: function () {
    console.log('app.js--onLaunch--小程序初始化')
  },
  //当小程序启动，或从后台进入前台显示，会触发 onShow
  onShow: function (options) {
    console.log('app.js--onShow--小程序显示')
  },
  //当小程序从前台进入后台，会触发 onHide
  onHide: function () {
    console.log('app.js--onHide--小程序隐藏')
  }
})
```

四、任务实现

（2）编写index.js文件代码。在该文件中添加如下函数：onLoad、onReady、onShow、onHide、onUnload，在每个函数中通过调用console.log函数来显示程序执行的位置。index.js文件：

```
// index.js
Page({
  // 生命周期函数--监听页面加载
  onLoad: function (options) {
    console.log('--index.js--onLoad--页面加载')
  },
  // 生命周期函数--监听页面初次渲染完成
  onReady: function () {
    console.log('--index.js--onLoad--页面初次渲染完成')
  },
  // 生命周期函数--监听页面显示
  onShow: function () {
    console.log('--index.js--onShow--页面显示')
  },
  // 生命周期函数--监听页面隐藏
  onHide: function () {
    console.log('--index.js--onHide--页面隐藏')
  },
  //生命周期函数--监听页面卸载
  onUnload: function () {
    console.log('--index.js--onUnLoad--页面隐藏')
  }
})
```

四、任务实现

（3）编写phonebook.js文件代码。在该文件中添加如下函数：onLoad、onReady、onShow、onHide、onUnload，在每个函数中通过调用console.log函数来显示程序执行的位置。phonebook.js文件：

```
// pages/phonebook/phonebook.js
Page({
 // 生命周期函数--监听页面加载
  onLoad: function (options) {
    console.log('--phonebook.js--onLoad--页面加载')
  },
  // 生命周期函数--监听页面初次渲染完成
  onReady: function () {
    console.log('--phonebook.js--onLoad--页面初次渲染完成')
  },
  // 生命周期函数--监听页面显示
  onShow: function () {
    console.log('--phonebook.js--onShow--页面显示')
  },
  // 生命周期函数--监听页面隐藏
  onHide: function () {
    console.log('--phonebook.js--onHide--页面隐藏')
  },
  //生命周期函数--监听页面卸载
  onUnload: function () {
    console.log('--phonebook.js--onUnLoad--页面隐藏')
  }
})
```

四、任务实现

（4）编写find.js文件代码。在该文件中添加如下函数：onLoad、onReady、onShow、onHide、onUnload,在每个函数中通过调用console.log函数来显示程序执行的位置。find.js文件：

```
// pages/find/find.js
Page({
  // 生命周期函数--监听页面加载
  onLoad: function (options) {
    console.log('--find.js--onLoad--页面加载')
  },
  // 生命周期函数--监听页面初次渲染完成
  onReady: function () {
    console.log('--find.js--onLoad--页面初次渲染完成')
  },
  // 生命周期函数--监听页面显示
  onShow: function () {
    console.log('--find.js--onShow--页面显示')
  },
  // 生命周期函数--监听页面隐藏
  onHide: function () {
    console.log('--find.js--onHide--页面隐藏')
  },
  //生命周期函数--监听页面卸载
  onUnload: function () {
    console.log('--find.js--onUnLoad--页面隐藏')
  }
 })
```

2.5

CHAPTER

setData视图渲染

一、任务描述

在页面文件中能使用setData()函数将js中data数据渲染到视图界面，并且能调用函数去改变视图渲染的内容。

二、导入知识点

在Page()方法中，setData()函数用于将数据异步从逻辑层发送到视图层WXML页面上，同时也可以同步更新data属性中的数据值，并改变对应的this.data 的值。setData()参数格式如下：

```
setData(data, callback);
```

// data需为可 JSON 化的数据，callback在 setData 对界面渲染完毕后调用

二、导入知识点

Page 下面的任何一个方法内都可以使用 setData 方法，它接收两个参数。其参数说明如表所示：

属性	类型	必填	描述
data	Object	是	要更新的一个或多个数据，格式为{key1:value1, key2:value2,..., keyN:valueN}
callback	Function	否	setData引起的界面更新渲染完毕后的回调函数

- **第一个参数Object data，它是必传的，数据类型是Object，即这次要改变的数据。**
- **第二个参数Function callback，它是回调函数，非必填的，它所代表的含义是setData引起的界面更新渲染完毕后的回调函数。**

三、实现效果

根据此任务描述可以做出如图所示的效果。

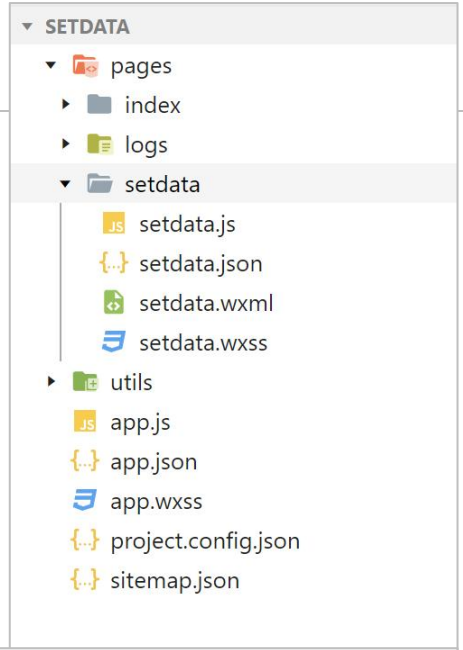
页面载入初始时，页面显示是“小程序开发教程” 的文字，如图（a）所示。当单击“更改data数据”按钮时，页面显示的文字发生改变，变为“微信小程序创始人张小龙” ，如图（b）所示，页面渲染的内容发生改变。

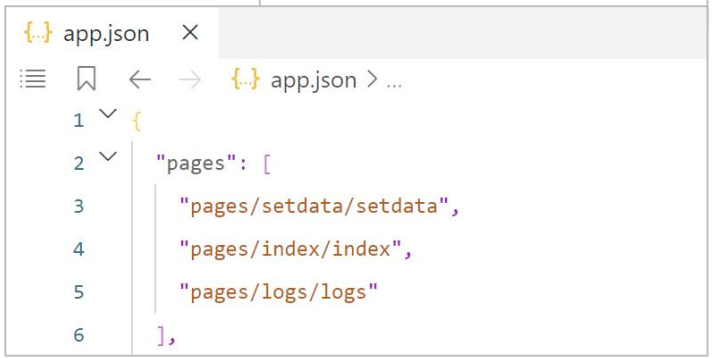
图（a）

图（b）

四、任务实现

（1）创建一个 page 页面名为setdata的文件，并且把app.json中的pages属性中的顺序进行变换，选中pages/setdata/setdata，按住“Alt+向上”把该页面的位置调整到第一位。如图所示。





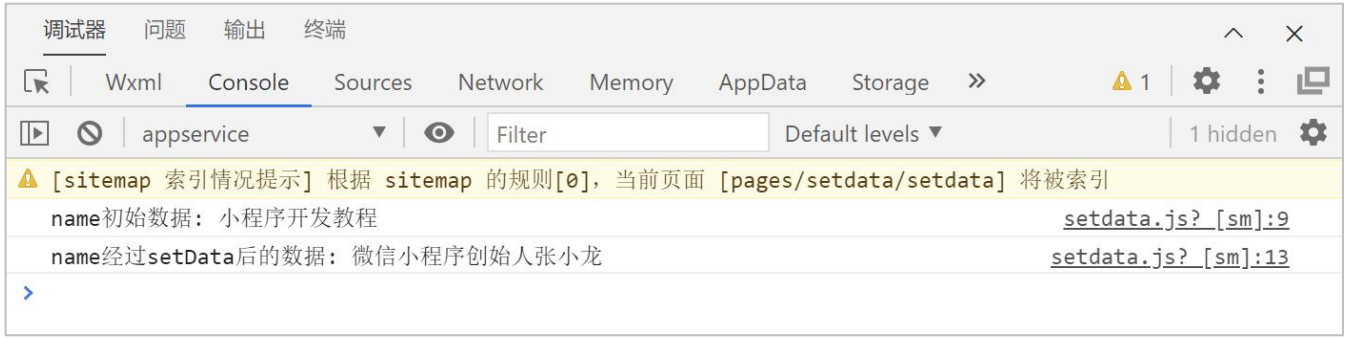
四、任务实现

（2）编写setdata.wxml文件代码。

（3）编写setdata.wxss文件代码。

（4）编写setdata.js文件代码。

console调试器setdata数据变化:



2.6 CHAPTER

变量和函数的作用域及模块化

一、任务描述

设计一个小程序，在index.js文件中调用app.js文件、index.js文件和util.js文件变量和函数，从而实现对全局变量和函数、本文件定义的变量和函数以及其他模块中定义的变量和函数的引用。

二、导入知识点

1.文件作用域

在小程序的任意JS文件中声明的变量和函数只在该文件中有效，不同的JS文件中可以声明相同名字的变量和函数，不会互相影响。

如果需要跨页面进行数据共享，可以在app.js中定义全局变量，然后在其他JS文件中使用getApp()来获取和更新。例如在app.js设置全局变量globalData中msg的值。

```
//app.js
App({
  globalData: {
    msg: ' Goodbye 2018'  //这是一个全局变量
  }
})

//test.js
var app = getApp()
app.globalData.msg = 'Hello 2019' //全局变量被更新
```

二、导入知识点

2.模块化

程序支持将一些公共JavaScript代码放在一个单独的JS文件中，作为一个公共模块可以被其他JS文件调用。模块只能通过module.exports或者exports才能对外提供接口。

例如，在根目录下新建utils文件夹并创建公共JS文件common.js。

```
//common.js
function sayHello(name) {
  console.log('Hello ${name} ! ')
}
function sayGoodbye(name) {
  console.log('Goodbye ${name} ! ')
}
module.exports.sayHello = sayHello //推荐使用这种
exports.sayGoodbye = sayGoodbye
```

二、导入知识点

页面JS中使用require引用common.js文件，此时就可以调用其中的函数。

```
//test.js
var common=require('../../utils/common.js') //目前暂时不支持绝对路径地址
Page({
  hello: function() {
    common.sayHello('2019')
  },
  goodbye: function() {
    common.sayGoodbye('2018')
  }
})}
module.exports.sayHello = sayHello //推荐使用这种
exports.sayGoodbye = sayGoodbye
```

三、实现效果

在app.js文件中定义的变量和函数是全局变量和函数，可以在任何js文件中进行引用。根据案例描述运行结果如图所示的效果。

四、任务实现

在index.wxml中设置了六个调用变量分别是msg1、msg2 、msg3、msg4、msg5和msg6，这六个变量需要通过index.js把值渲染到视图层中。通过index.js文件中调用app.js文件，调用index.js文件本身以及调用util.js文件。变量和函数如图展示了index.wxml、index.js、app.js和utils.js之间的调用关系。

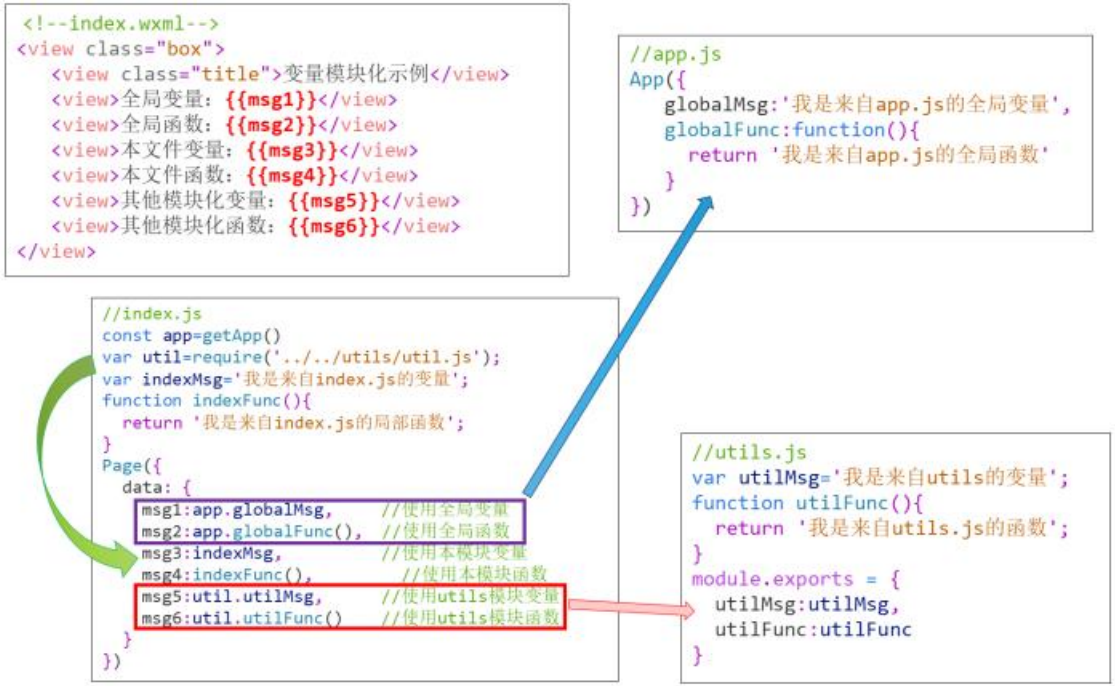


图 2.33 index.js、app.js 和 utils.js 之间调用关系图

四、任务实现

（1）编写index.wxml文件代码。

（2）编写index.wxss文件代码。

（3）编写index.js文件代码。

（4）编写app.js文件代码。

四、任务实现

代码首先获取全局应用实例和utils模块实例，并定义了本模块的变量和函数，然后使用了全局变量、全局函数、utils模块变量、utils模块函数、本模块变量和本模块函数。

①如果在本模块中引用app.js中定义的全局变量和函数，就必须定义全局对象并利用getApp()函数给该对象赋值。本案例中利用const app=getApp()语句定义了全局对象app,并利用app.globalMsg和app.globalFunc()引用app.js文件中定义的变量和函数，并赋值给msg1和msg2。

四、任务实现

②如果在本模块中引用模块内定义的变量和函数，在data中直接引用就可以了，如本例中的msg3和msg4直接引用了indexMsg和indexFunc()。

③如果在本模块中引用其他模块，如本案例中的util模块中定义的变量和函数，首先需要利用require函数引入util文件，本案例中利用var util=require('../utils/util.js')创建util实例，然后利用util引用util.js文件中定义的变量和函数，本例通过util.utilMsg和util.

utilFunc()引用了util.js文件中定义的变量和函数给msg5和msg6赋值。

四、任务实现

（5）在pages文件夹下添加utils文件夹，并在其中添加util.js文件，然后编写util.js文件代码。该文件定义了1个变量和1个函数，如果要在其他js文件中引用这个变量和函数，就必须通过module.exports或exports来输出该变量和函数。util.js文件：

```
//utils.js
var utilMsg='我是来自 utils 的变量';
function utilFunc(){
  return '我是来自 utils.js 的函数';
}
module.exports = {
  utilMsg:utilMsg,
  utilFunc:utilFunc
}
```

四、任务实现

在JS文件中声明的变量和函数只在该文件中有效；不同文件中可以声明相同名字的变量和函数，不会互相影响。通过全局函数getApp()可以获取全局的应用实例，如果需要全局的数据，可以在App()中设置。

可以将一些公共的代码抽离成为一个单独的js文件作为一个模块。模块通过module.exports或者exports对外暴露接口，在需要这些模块的文件中，使用require(path)将公共代码引入。主要注意的是path为相对路径，暂时不支持绝对路径。

感 谢 聆 听