视图与逻辑

1、页面导航

1.1、什么是页面导航

页面导航指的是页面之间的相互跳转。例如，浏览器中实现页面导航的方式有如下两种：
① 链接
② location.href

1.2、小程序中实现页面导航的两种方式

① 声明式导航
在页面上声明一个 导航组件
通过点击 组件实现页面跳转
② 编程式导航
调用小程序的导航 API，实现页面的跳转

1.3、声明式导航

1.3.1、导航到 tabBar 页面

tabBar 页面指的是被配置为 tabBar 的页面。
在使用 组件跳转到指定的 tabBar 页面时，需要指定 url 属性和 open-type 属性，其中：

1. url 表示要跳转的页面的地址，必须以 / 开头
2. open-type 表示跳转的方式，必须为switchTab
   示例代码如下：

```
<navigator url="/pages/message/message" open-type="switchTab">导航到消息页
</navigator>
```

1.3.2、导航到非 tabBar 页面

非 tabBar 页面指的是没有被配置为 tabBar 的页面。
在使用 组件跳转到普通的非 tabBar 页面时，则需要指定 url 属性和 open-type 属性，其中：

1. url 表示要跳转的页面的地址，必须以 / 开头
2. open-type 表示跳转的方式，必须为navigate
   示例代码如下：

```
<navigator url="/pages/info/info" open-type="navigate">导航到消息页</navigator>
```

注意：为了简便，在导航到非tabBar 页面时，open-type= " navigate " 属性可以省略。

1.3.3、 后退导航

如果要后退到上一页面或多级页面，则需要指定 open-type 属性和 delta 属性，其中：

1. open-type 的值必须是 navigateBack，表示要进行后退导航
2. delta 的值必须是数字，表示要后退的层级
   示例代码如下：

```
<navigator open-type="navigateBack" delta="1">导航到消息页</navigator>
```

注意：为了简便，如果只是后退到上一页面，则可以省略 delta 属性，因为其默认值就是 1。

1.4、编程式导航

1.4.1、导航到 tabBar 页面

调用 wx.switchTab(Object object) 方法，可以跳转到 tabBar 页面。其中 Object 参数对象的属性列表如下：

属性	类型	必选	说明
url	string	是	需要跳转的 tabBar 页面的路径，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）

示例代码如下：

```
gotoMessage() {
    wx.switchTab({
        url: 'pages/message/message'
    });
}
```

```
<button bindtip="gotoMessage()">
    跳转到消息页面
</button>
```

1.4.2、导航到非 tabBar 页面

调用 wx.navigateTo(Object object) 方法，可以跳转到非 tabBar 的页面。其中 Object 参数对象的属性列表
如下：

属性	类型	必选	说明
url	string	是	需要跳转的 非tabBar 页面的路径，路径后可以带参数
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）

示例代码如下：

```
gotoInfo() {
    wx.navigateTo({
        url: 'pages/message/message'
    });
}
```

```
<button bindtip="gotoInfo()">
    跳转到Info页面
</button>
```

1.4.3、后退导航

调用 wx.navigateBack(Object object) 方法，可以返回上一页面或多级页面。其中 Object 参数对象可选的
属性列表如下：

属性	类型	必选	说明	默认值
delta	number	否	返回的页面数，如果 delta 大于现有页面数，则返回到首页	1
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）	

```
goToBack() {
    wx.navigateBack();
}
```

```
<button bindtap="gotoBack()">
    后退
</button>
```

1.5、导航传参

1.5.1、 声明式导航传参

navigator 组件的 url 属性用来指定将要跳转到的页面的路径。同时，路径的后面还可以携带参数：

1. 参数与路径之间使用 ? 分隔
2. 参数键与参数值用 = 相连
3. 不同参数用 & 分隔
   代码示例如下：

```
<navigator url="/pages/info/info?name=zs&age=20">跳转到info页面</navigator>
```

1.5.2、编程式导航传参

x.navigateTo(Object object) 方法跳转页面时，也可以携带参数，代码示例如下：

```
gotoInfo2 () {
    wx.navigateTo({
        url: 'pages/info/info?name=zs&gender=男'
    });
}
```

```
<button bindtap="gotoInfo2">
    跳转到info页面
</button>
```

1.5.3、 在 onLoad 中接收导航参数

通过声明式导航传参或编程式导航传参所携带的参数，可以直接在 onLoad 事件中直接获取到，示例代码如下：

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

2、页面事件

2.1、下拉刷新事件

2.1.1、什么是下拉刷新

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

2.1.2、启用下拉刷新

启用下拉刷新有两种方式：
① 全局开启下拉刷新
在 app.json 的 window 节点中，将 enablePullDownRefresh 设置为 true
② 局部开启下拉刷新
在页面的 .json 配置文件中，将 enablePullDownRefresh 设置为 true
在实际开发中，推荐使用第 2 种方式，为需要的页面单独开启下拉刷新的效果。

2.1.3、配置下拉刷新窗口的样式

在全局或页面的 .json 配置文件中，通过 backgroundColor 和 backgroundTextStyle 来配置下拉刷新窗口
的样式，其中：
backgroundColor 用来配置下拉刷新窗口的背景颜色，仅支持16 进制的颜色值
backgroundTextStyle 用来配置下拉刷新 loading 的样式，仅支持 dark 和 light

2.1.4、监听页面的下拉刷新事件

在页面的 .js 文件中，通过 onPullDownRefresh() 函数即可监听当前页面的下拉刷新事件。
例如，在页面的 wxml 中有如下的 UI 结构，点击按钮可以让 count 值自增 +1：

```
countAdd() {
    this.setData({
        count: this.data.count + 1
    });
}
```

```
<view>count值为：{{count}}</view>
<button bindtap="countAdd()">
    +1
</button>
```

触发页面的下拉刷新事件的时候，如果要把 count 的值重置为 0，示例代码如下：

```
onPullDownRefresh:function() {
    this.setData({
        count: 0
    });
}
```

2.1.5、 停止下拉刷新的效果

当处理完下拉刷新后，下拉刷新的 loading 效果会一直显示，不会主动消失，所以需要手动隐藏下拉刷新的
loading 效果。此时，调用 wx.stopPullDownRefresh() 可以停止当前页面的下拉刷新。示例代码如下：

```
onPullDownRefresh:function() {
    this.setData({
        count:0
    })
}
wx.stopPullDownRefresh()
```

2.2、上拉触底事件

2.2.1、什么是上拉触底

上拉触底是移动端的专有名词，通过手指在屏幕上的上拉滑动操作，从而加载更多数据的行为。

2.2.2、监听页面的上拉触底事件

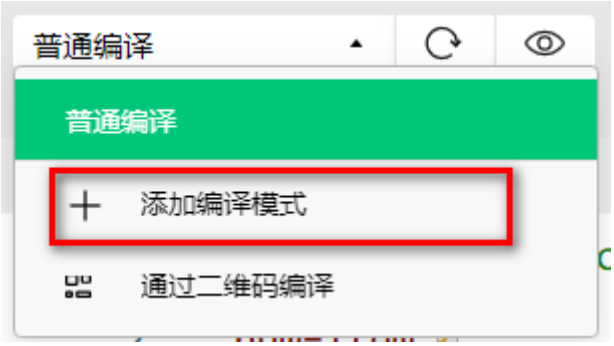
在页面的 .js 文件中，通过 onReachBottom() 函数即可监听当前页面的上拉触底事件。示例代码如下：

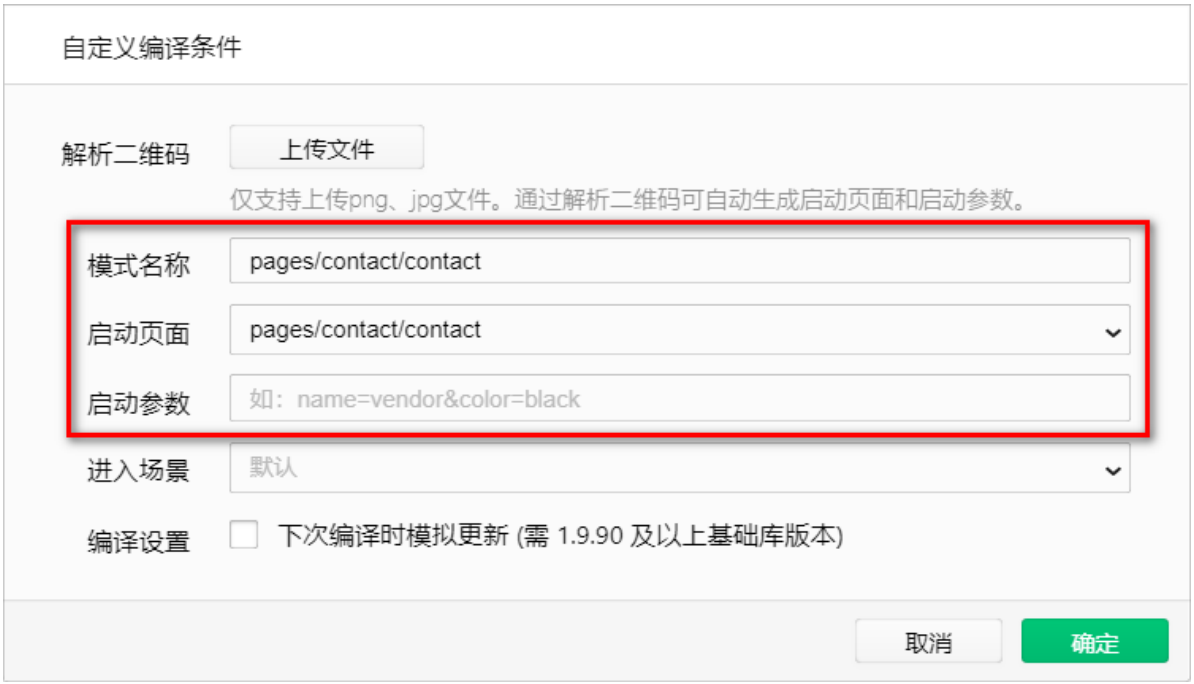
```
onReachBottom: function() {
    console.log('触发了上拉触底事件')
}
```

2.2.3、配置上拉触底距离

上拉触底距离指的是触发上拉触底事件时，滚动条距离页面底部的距离。
可以在全局或页面的 .json 配置文件中，通过 onReachBottomDistance 属性来配置上拉触底的距离。
小程序默认的触底距离是 50px，在实际开发中，可以根据自己的需求修改这个默认值。

2.2.4、扩展-自定义编译模式





3、生命周期

3.1、什么是生命周期

生命周期（Life Cycle）是指一个对象从创建-> 运行 -> 销毁的整个阶段，强调的是一个时间段。例如：

1. 张三出生，表示这个人生命周期的开始
2. 张三离世，表示这个人生命周期的结束
3. 中间张三的一生，就是张三的生命周期

我们可以把每个小程序运行的过程，也概括为生命周期：

1. 小程序的启动，表示生命周期的开始
2. 小程序的关闭，表示生命周期的结束
3. 中间小程序运行的过程，就是小程序的生命周期

3.2、生命周期的分类

在小程序中，生命周期分为两类，分别是：
① 应用生命周期
特指小程序从启动 -> 运行 -> 销毁的过程
② 页面生命周期
特指小程序中，每个页面的加载 -> 渲染 -> 销毁的过程

其中，页面的生命周期范围较小，应用程序的生命周期范围较大，如图所示：

3.3、什么是生命周期函数

生命周期函数：是由小程序框架提供的内置函数，会伴随着生命周期，自动按次序执行。
生命周期函数的作用：允许程序员在特定的时间点，执行某些特定的操作。例如，页面刚加载的时候，可以在
onLoad 生命周期函数中初始化页面的数据。
注意：生命周期强调的是时间段，生命周期函数强调的是时间点。

3.4、生命周期函数的分类

小程序中的生命周期函数分为两类，分别是：
① 应用的生命周期函数
特指小程序从启动 -> 运行 -> 销毁期间依次调用的那些函数
② 页面的生命周期函数
特指小程序中，每个页面从加载 -> 渲染 -> 销毁期间依次调用的那些函数

3.5、应用的生命周期函数

小程序的应用生命周期函数需要在 app.js 中进行声明，示例代码如下：

```
App({
    onLaunch: function(option){},
    onShow: function(option){},
    onHide:  function() {}
});
```

3.6、页面的生命周期函数

小程序的页面生命周期函数需要在页面的.js 文件中进行声明，示例代码如下：

```
Page({
    onLoad:function(options){},
    onShow: function(){},
    onReady:  function() {},
    onHide:  function() {}
    onUnload:function(){}
});
```

4、WXS脚本

4.1、概述

4.1.1、什么是 wxs

WXS（WeiXin Script）是小程序独有的一套脚本语言，结合 WXML，可以构建出页面的结构。

4.1.2、wxs 的应用场景

wxml 中无法调用在页面的.js 中定义的函数，但是，wxml 中可以调用 wxs 中定义的函数。因此，小程序中
wxs 的典型应用场景就是“过滤器”。

4.1.3、wxs 和 JavaScript 的关系*

虽然 wxs 的语法类似于 JavaScript，但是 wxs 和 JavaScript 是完全不同的两种语言：
① wxs 有自己的数据类型
　number 数值类型、string 字符串类型、boolean 布尔类型、object 对象类型、
　function 函数类型、array 数组类型、 date 日期类型、 regexp 正则
② wxs 不支持类似于 ES6 及以上的语法形式
　不支持：let、const、解构赋值、展开运算符、箭头函数、对象属性简写、etc...
　支持：var 定义变量、普通 function 函数等类似于 ES5 的语法
③ wxs 遵循 CommonJS 规范
　module 对象
　require() 函数
　module.exports 对象

4.2、基础语法

4.2.1、内嵌 wxs 脚本

wxs 代码可以编写在 wxml 文件中的 标签内，就像 Javascript 代码可以编写在 html 文件中的 标签内一样。
wxml 文件中的每个 标签，必须提供 module 属性，用来指定当前wxs 的模块名称，方便在wxml 中访问模块中的成员：

```
<view>{{m1.toUpper(username)}}</view>

<wxs module="m1">
    module.exports.toUpper = function(str) {
    return str.toUpperCase()
    }
</wxs>
```

4.2.2、 定义外联的 wxs 脚本

wxs 代码还可以编写在以.wxs 为后缀名的文件内，就像 javascript 代码可以编写在以 .js 为后缀名的文件中
一样。示例代码如下：

// tools.wxs

```
function toLower(str) {
    return str.toLowerCase();
}
module.export = {
    toLower: toLower
}
```

4.2.3、使用外联的 wxs 脚本

在 wxml 中引入外联的 wxs 脚本时，必须为 标签添加 module 和 src 属性，其中：

1. module 用来指定模块的名称
2. src 用来指定要引入的脚本的路径，且必须是相对路径
   示例代码如下：

```
<view>{{m2.toLower(country)}}</view>

<wxs src="../../utils/tools.wxs" module="m2"></wxs>
```

4.3、WXS 的特点

4.3.1、与 JavaScript 不同

为了降低 wxs（WeiXin Script）的学习成本， wxs 语言在设计时借大量鉴了JavaScript 的语法。但是本质上，
wxs 和 JavaScript 是完全不同的两种语言！

4.3.2、不能作为组件的事件回调

wxs 典型的应用场景就是“过滤器”，经常配合 Mustache 语法进行使用，例如：

```
<view>{{m2.toLower(country)}}</view>
```

但是，在 wxs 中定义的函数不能作为组件的事件回调函数。例如，下面的用法是错误的：

```
<button bindtap="m2.toLower">按钮</button>
```

4.3.3、隔离性

隔离性指的是 wxs 的运行环境和其他 JavaScript 代码是隔离的。体现在如下两方面：
① wxs 不能调用 js 中定义的函数
② wxs 不能调用小程序提供的API

4.3.4、性能好

1. 在 iOS 设备上，小程序内的 WXS 会比 JavaScript 代码快 2 ~ 20 倍
2. 在 android 设备上，二者的运行效率无差异

5、总结

① 能够知道如何实现页面之间的导航跳转
声明式导航、编程式导航
② 能够知道如何实现下拉刷新效果
enablePullDownRefresh、onPullDownRefresh
③ 能够知道如何实现上拉加载更多效果
onReachBottomDistance、onReachBottom
④ 能够知道小程序中常用的生命周期函数

1. 应用生命周期函数：onLaunch, onShow, onHide
2. 页面生命周期函数：onLoad, onShow, onReady, onHide, onUnload