第 7 章

React Native 独立组件高级篇

通过第 6 章的学习，你已经掌握了许多 React Native 中跨平台的独立组件。然而 iOS 平台与 Android 平台毕竟差异很大，在 React Native 中还定义了许多专用于某种平台的组件。本章我们将主要介绍这类组件的应用。

一款完整的应用程序几乎不可能只有一个界面，本章你还将学习如何在 React Native 中管理界面路由，实现界面的跳转切换。通过本章的学习，相信你的 React Native 开发能力会再上一个台阶。

7.1 时间选择器 DatePickerIOS 组件的应用

在前面章节中，已经学习了选择器 Picker 组件。其实在 iOS 平台上，React Native 还提供了一个专门用来选择日期时间的组件 DatePickerIOS。在 HelloWorld 工程的 Demo 文件夹下新建一个命名为 DatePickerIOSDemo.js 的文件，在其中编写如下代码：

```
import React,{Component} from 'react';
import {DatePickerIOS} from 'react-native';
export default class DatePickerIOSDemo extends Component{
    constructor(props){
        super(props);
        this.state = {
            date:new Date()
        }
    }
    render(){
        return(
            <DatePickerIOS style={{width:300,height:200,left:10,top:30}}
            date={this.state.date}
```

```
                mode={'time'}
                onDateChange={(newDate)=>{
                    this.setState({
                        date:newDate
                    });
                }}/>
            );
        }
    }
```

DatePickerIOS 组件有 3 种模式，分别为 date、time 和 datetime。修改 index.ios.js 文件后运行工程，效果如图 7-1 所示。

需要注意，DatePickerIOS 组件是一个受控组件，用户的操作无法改变 DatePickerIOS 组件的选择值，开发者在接收到用户选择回调后，需要手动修改 DatePickerIOS 组件的 date 属性。DatePickerIOS 组件扩展于 View 组件，因此 View 组件的所有属性都支持，其他常用属性如表 7-1 所示。

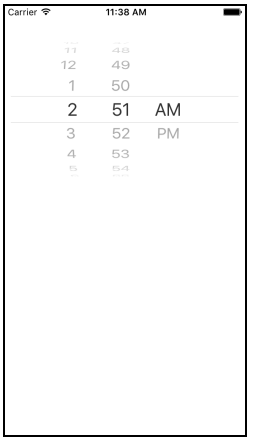

图 7-1　DatePickerIOS 组件样式

表 7-1　DatePickerIOS 组件的其他常用属性

属性名	解释	平台	值类型或可选值
date	当前选中的日期	iOS	Date 对象
maximumDate	可选的最大日期	iOS	Date 对象
minimumDate	可选的最小日期	iOS	Date 对象
mimuteInterval	可选的时间分钟间隔	iOS	可选数值为 1、2、3、4、5、6、10、12、15、20、30
mode	选择器组件模式	iOS	枚举字符串 • date：日期模式 • time：时间模式 • datetime：日期时间模式
onDateChange	当用户修改日期时回调的函数	iOS	函数，会传入用户选择的 Date 日期对象
timeZoneOffsetInMinutes	设置时区差，单位是分钟	iOS	数值，用来指定时区的分钟差，例如东八区可以设置 8*60

7.2　DrawerLayoutAndroid 抽屉组件的应用

在安卓设备上，很容易见到各式各样的抽屉视图。抽屉视图常常可以通过在设备屏幕边缘滑

动手势来打开，在 React Native 中，提供了 DrawerLayoutAndroid 组件来创建抽屉视图。

在 HelloWorld 工程的 Demo 文件夹下新建一个命名为 DrawerLayoutAndroidDemo.js 的文件，在其中编写如下代码：

```
    import React,{Component} from 'react';
    import {DrawerLayoutAndroid,View,Text} from 'react-native';
    export default class DrawerLayoutAndroidDemo extends Component{
        render(){
            return(
            <DrawerLayoutAndroid
            drawerWidth={150}
            drawerPosition={DrawerLayoutAndroid.positions.Left}
            renderNavigationView={()=>{
                return(
                    <View style={{flex: 1, backgroundColor: '#fff'}}>
                        <Text style={{margin: 10, fontSize: 15, textAlign:
'left'}}>抽屉视图</Text>
                </View>
                );
            }}>
                <View style={{flex: 1, alignItems: 'center'}}>
                <Text style={{margin: 10, fontSize: 15, textAlign: 'right'}}>
Hello</Text>
                <Text style={{margin: 10, fontSize: 15, textAlign: 'right'}}>
World!</Text>
                </View>
            </DrawerLayoutAndroid>
            );
        }
    }
```

DrawerLayoutAndroid 组件是一个容器组件，其中的子组件会渲染在显示内容中，renderNavigationView 属性用来设置抽屉视图，修改 index.android.js 文件后，运行工程，效果如图 7-2 所示。

通过在屏幕上使用左划和右划手势，可以对抽屉进行打开与收起操作。DrawerLayoutAndroid 组件扩展于 View 组件，因此可以使用所有 View 组件可用的属性。DrawerLayoutAndroid 组件中的常用属性如表 7-2 所示。

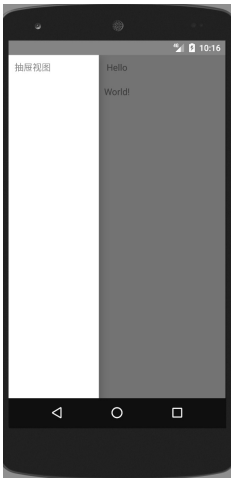

图 7-2　DrawerLayoutAndroid 组件效果

表 7-2　DrawerLayoutAndroid 组件常用属性

属性名	解释	平台	值类型或可选值
keyboardDismissMode	设置在拖拽过程中是否隐藏	Android	枚举字符串 none：不隐藏键盘 on-drag：拖拽时隐藏
drawerPosition	设置抽屉的弹出方向	Android	有如下两种可选值： • DrawerConsts.DrawerPosition.Left：从左侧滑出 • DrawerConsts.DrawerPosition.Right：从右侧滑出
drawerWidth	设置抽屉宽度	Android	数值
drawerLockMode	设置抽屉锁定状态，设置后，只能够通过函数方法来开关，不能通过手势开关	Android	枚举字符串： • unlocked：不锁定 • locked-closed：关闭 • locked-open：开启
onDrawerSlide	在用户滑动抽屉时会不断调用此回调	Android	函数
onDrawerStateChanged	当抽屉状态发生改变时调用的回调	Android	函数，会传入如下 3 种状态之一： • idle：空闲状态，无任何交互 • dragging：拖拽中 • settling：停靠中
onDrawerClose	当抽屉关闭时调用的回调	Android	函数
renderNavigationView	用来渲染抽屉视图	Android	函数，需要返回组件
statusBarBackgroundColor	设置状态栏背景色	Android	特定颜色字符串

可以使用如表 7-3 所示的方法通过代码来实现抽屉的开关。

表 7-3　实现抽屉开关的方法

方法名	解释	平台	参数
openDrawer	打开抽屉	Android	无
closeDrawer	关闭抽屉	Android	无

7.3　进度条组件的应用

无论在 iOS 平台还是 Android 平台都有进度条这一原生控件，在进行数据加载、网络请求、音视频播放等任务时，进度条都起到了不可或缺的提示作用。然而在 React Native 中，进度条组件却不是跨平台的，针对 iOS 与 Android 平台，React Native 分别提供了用于创建进度条的组件。我们可以通过一些兼容平台技巧来同时在双平台上进行进度条组件的应用。

7.3.1 通过文件名分平台加载组件

React Native 支持通过规范的命名文件来使其自适应平台加载代码。前面我们编写的文件都是通用格式的文件，如果需要只在 iOS 平台加载，需要添加.ios 作为后缀名，同样如果需要只在 Android 平台加载，则需要添加.android 作为后缀名。在 React Native 中，iOS 平台的进度条组件定义为 ProgressViewIOS，Android 平台定义的进度条组件为 ProgressBarAndroid。

在 HelloWorld 工程的 Demo 文件夹下面新建两个文件，分别命名为 ProgressBar.ios.js 与 ProgressBar.android.js。当你在其他文件中引用 ProgressBar 文件时，系统会自动判断平台取合适的 JavaScript 文件。将 ProgressBar.ios.js 文件实现如下：

```
import React,{Component} from 'react';
import {ProgressViewIOS} from 'react-native';
export default class ProgressBar extends Component{
    render(){
        return(
            <ProgressViewIOS style={{top:100}}
            progressTintColor='red'
            progressViewStyle={'default'}
            progress={0.5}
            />
        );
    }
}
```

将 ProgressBar.android.js 文件实现如下：

```
import React,{Component} from 'react';
import {ProgressBarAndroid,View} from 'react-native';
export default class ProgressBar extends Component{
    render(){
        return(
            <ProgressBarAndroid style={{top:100}}
            color='red'
            styleAttr={"Horizontal"}
            progress={0.5}
            indeterminate={false}/>
        );
    }
}
```

将 index.ios.js 与 index.android.js 文件修改如下：

```
import React, { Component } from 'react';
import {
  AppRegistry
} from 'react-native';
```

```
import ProgressBar from './Demo/ProgressBar';
export default class HelloWorld extends Component {
  render() {
    return (<ProgressBar />);
  }
}
AppRegistry.registerComponent('HelloWorld', () => HelloWorld)
```

在两个平台上运行工程，效果分别如图 7-3 与图 7-4 所示。

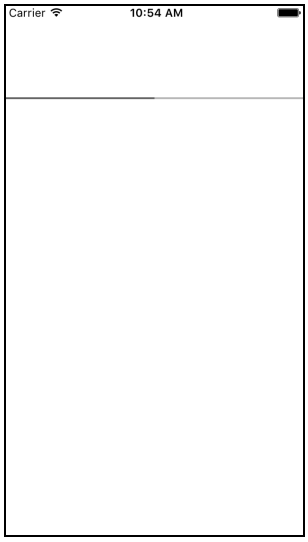

图 7-3　iOS 平台下的进度条组件

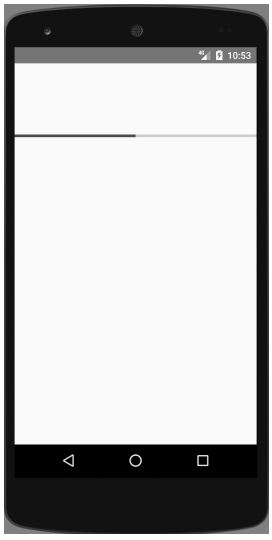

图 7-4　Android 平台下的进度条组件

7.3.2　ProgressBarAndroid 组件常用属性

ProgressBarAndroid 组件是 Android 平台上的进度条组件，扩展于 View 组件并且支持多种风格样式，除了可以使用所有 View 组件可用的属性外，其他常用属性如表 7-4 所示。

表 7-4　ProgressBarAndroid 组件其他常用属性

属性名	解释	平台	值类型或可选值
color	进度条颜色	Android	特定的颜色字符串
progress	当前进度，需要当 indeterminate 设置为 false 并且 styleAttr 为 Horizontal 风格时使用	Android	数值（0～1 之间）
indeterminate	是否显示不确定的进度	Android	布尔值
styleAttr	设置进度条风格	Android	枚举字符串 • Horizontal：水平 • Small：小圆圈 • Large：大圆圈 • Inverse：逆序圆圈 • SmallInverse：小逆序圆圈 • LargeInverse：大逆序圆圈

7.3.3 ProgressViewIOS 组件常用属性

ProgressViewIOS 组件用于在 iOS 平台上创建进度条，其同样也是扩展于 View 组件，因此可以使用所有 View 组件的属性，除此之外，常用属性如表 7-5 所示。

表 7-5 ProgressViewIOS 组件常用属性

属性名	解释	平台	值类型或可选值
progress	当前进度值	iOS	数值（0～1）
progressImage	设置进度条图片	iOS	图片素材
progressTintColor	设置进度条颜色	iOS	特定的颜色字符串
progressViewStyle	设置进度条风格	iOS	枚举字符串： • default • bar
trackImage	设置进度条未走过进度的图片	iOS	图片素材
trackTintColor	设置进度条未走过进度的颜色	iOS	特定的颜色字符串

7.4 SegmentedControlIOS 组件的应用

SegmentedControl 是 iOS 平台上的一个原生控件，又称为分段控制器。SegmentedControlIOS 组件是 React Native 对 iOS 平台中分段控制器的包装。它常用在进行模块选择的功能中。

在 HelloWorld 工程的 Demo 文件夹下新建一个命名为 SegmentedControlIOSDemo.js 的文件，在其中编写如下代码：

```
import React,{Component} from 'react';
import {SegmentedControlIOS} from 'react-native';
export default class SegmentedControlIOSDemo extends Component{
    render(){
        return(
            <SegmentedControlIOS
            style={{top:100}}
            onChange={()=>{
                console.log("onChange");
            }}
            onValueChange={(value)=>{
                console.log(value);
            }}
            selectedIndex={0}
            tintColor='red'
```

```
                values={['one','two','three']}/>
            );
        }
    }
```

修改 index.ios.js 文件后，运行工程，效果如图 7-5 所示。

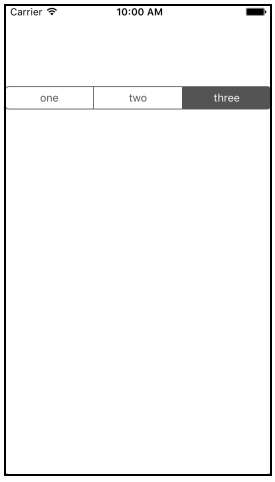

图 7-5　SegmentedControlIOS 组件效果

当用户单击分段控制器中的某一段时可以实现选中切换。SegmentedControlIOS 组件扩展于 View 组件，因此可以使用所有 View 组件可用的属性，除此之外的常用属性如表 7-6 所示。

表 7-6　SegmentedControlIOS 组件常用属性

属性名	解释	平台	值类型或可选值
enabled	设置组件是否可以交互	iOS	布尔值
momentary	设置是否不保持选中状态，如果设置为 true，则分段控制器不会保持选中状态，但相应回调依然会调用	iOS	布尔值
onChange	当用户单击了某段会调用的回调	iOS	函数
onValueChange	当用户单击某段时调用，会将选中的值作为参数传入	iOS	函数
selectedIndex	设置初始选中的段下标	iOS	数值
tintColor	设置分段控制器颜色	iOS	特定的颜色字符串
values	设置每段的值，会显示为标题	iOS	数组，其中的元素需要为字符串类型

7.5　Android 平台上的工具条组件

React Native 中提供了 ToolbarAndroid 组件，用来在 Android 平台上创建工具条。ToolbarAndroid 组件十分灵活，不仅提供了丰富的属性供我们对工具条进行定制，还可以完全定义自己的工具条组件。

在 HelloWorld 工程的 Demo 文件夹下面新建一个命名为 ToolbarAndroidDemo.js 的文件，在其中编写如下代码：

```
import React,{Component} from 'react';
import {ToolbarAndroid,View} from 'react-native';
export default class ToolbarAndroidDemo extends Component{
    render(){
        return(
            <ToolbarAndroid
            style={{height:56,backgroundColor:'green'}}
            logo={require('./../source/logo.png')}
            navIcon={require('./../source/setting.png')}
            overflowIcon={require('./../source/setting.png')}
            subtitle='React Native'
            subtitleColor='blue'
            title='ToolbarAndroid'
            titleColor='red'
            actions={[
                {
                    title:'setting1',
                    icon:require('./../source/timg.jpeg'),
                    show:'always',
                    showWithText:true
                },
                {
                    title:'setting2',
                    icon:require('./../source/timg.jpeg'),
                    show:'ifRoom',
                    showWithText:true
                },
                {
                    title:'setting3',
                    icon:require('./../source/timg.jpeg'),
                    show:'never',
                    showWithText:true
                },
            ]}
            onActionSelected={(position)=>{
                console.log(position);
            }}
            onIconClicked={()=>{
                console.log('icon');
            }}
            />
        );
    }
}
```

ToolbarAndroid 组件大致分为 3 个部分：导航图标部分，标题部分和功能列表部分。其中，导航图标部分在左，标题部分在中间，功能列表部分在右侧。功能列表也支持进行折叠显示。修改 index.android.js 文件后，运行工程，效果如图 7-6 所示。

上面的示例代码使用的是默认的工具条样式，是使用单标签创建的。同样，也可以使用双标签来创建完全自定的工具条，示例如下：

```
                <ToolbarAndroid style={{backgroundColor:'red',height:50,
marginTop:50}}
                logo={require('./../source/logo.png')}>
                    <Switch />
                    <Text>Switch</Text>
                </ToolbarAndroid>
```

运行工程效果如图 7-7 所示。

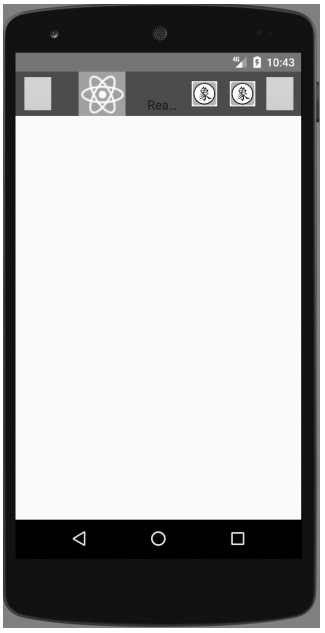

图 7-6　ToolbarAndroid 组件效果

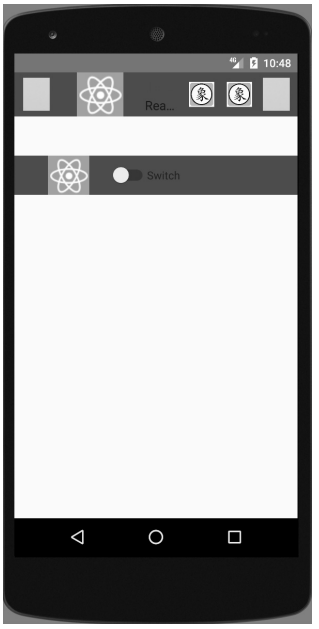

图 7-7　定制 ToolbarAndroid 组件

ToolbarAndroid 组件扩展于 View 组件，默认所有 View 组件的属性 ToolbarAndroid 组件都可以使用。ToolbarAndroid 组件常用属性如表 7-7 所示。

表 7-7　ToolbarAndroid 组件常用属性

属性名	解释	平台	值类型或可选值
navIcon	设置导航图标	Android	本地图片素材
logo	设置工具条图标	Android	本地图片素材
onIconClicked	当单击导航图片时调用的回调	Android	函数

（续表）

属性名	解释	平台	值类型或可选值

actions	设置功能按钮数组	Android	数组，其中为如下格式的对象：
			{
			title：功能按钮标题
			icon：功能按钮图标
			show：是否隐藏
			sowWithText：是否显示文案
			}
			其中 show 属性可选值如下：
			• always：总是显示
			• ifRoom：能放下就显示
			• never：不显示

onActionSelected	设置单击功能按钮后的回调函数	Android	函数，会将选中的功能按钮在数组中的下标传入
overflowIcon	设置溢出的功能按钮被集成到列表中的入口图标	Android	本地图片素材
subtitle	设置工具条副标题	Android	字符串
subtitleColor	设置工具条副标题颜色	Android	特定格式的颜色字符串
title	设置工具条标题	Android	字符串
titleColor	设置工具条标题颜色	Android	特定格式的颜色字符串

需要注意，所有 ToolbarAndroid 组件中使用到的图片素材必须为本地素材，不可以使用远程图像（虽然在开发环境可能可以使用，但正式打包时依然会有问题）。

7.6 Navigator 导航控制器

Navigatior 组件是 React Native 中最为重要的几个组件之一。有了 Navigatior 组件，你可以轻松地实现界面间的切换跳转。Navigator 组件并非为一个独立的视图，其实质是一种界面管理器，其管理着一组视图的切换。在学习 Navigator 组件之前，先回忆一下栈这种数据结构。

栈是编程中一种重要的数据结构，与其对应的还有队列这种数据结构。你可以将栈理解为只有一面开口的容器，数据的进出遵守“后进先出，先进后出”这种原则。队列则是一种双面开口的容器，数据的进出遵守“先进先出，后进后出”原则。栈和队列数据存储的示例图如图 7-8 所示。

通常，数据的入栈操作也叫 push，数据的出栈操作也叫 pop。在 Navigatior 组件中，栈存储的其实就是每个界面。

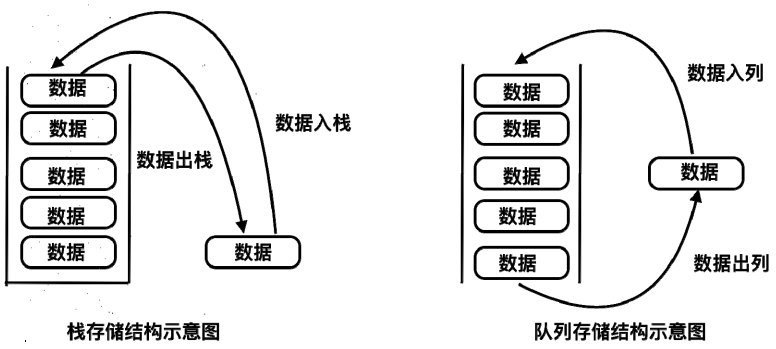

图 7-8　栈与队列存储示意图

7.6.1　Navigatior 牛刀小试

在 HelloWorld 工程的 Demo 文件夹下新建一个命名为 NavigatiorDemo.js 的文件，在其中编写如下代码：

```
    import React,{Component} from 'react';
    import {Navigator,View,Text,Button} from 'react-native';
    export default class NavigatorDemo extends Component{
        render(){
            return(
                <Navigator initialRoute={{name:'first',index:0}}
                renderScene={(route,navigatior)=>{
                    return this.renderScene(route,navigatior);
                }}/>
            );
        }
        renderScene(route,navigatior){
            return(
                <View style={{flex:1,backgroundColor:'red'}}>
                    <Text style={{marginTop:30,fontSize:20,textAlign:'center'}}
>{route.name}</Text>
                    <Button title="push" onPress={()=>{
                        navigatior.push({name:'Scene'+(route.index+1),
index:route.index+1});
                    }}/>
                    <Button title="pop" onPress={()=>{
                        if (route.index>0) {
                            navigatior.pop();
                        }
                    }}/>
                </View>
            );
        }
    }
```

上面的示例代码中只使用到了 Navigator 组件的两个属性——initialRoute 属性和 renderScene 属性，initialRoute 属性设置导航组件的初始路由，renderScene 属性用来提供要渲染的界面。运行工程，单击 push 和 pop 按钮，可以观察到界面的切换效果。

路由在 Navigator 组件中是十分重要的，你需要通过路由来切换对应的界面。试想一下，打开浏览器，想要浏览一个网页时需要输入此网页的地址，一个地址对应一个网页。Navigator 组件界面的切换也是采用这样的设计思路，你可以为每个路由设置名称与下标。renderScene 方法中会传入路由信息，其需要根据路由信息来渲染指定的界面。Navigator 实例的 push 与 pop 方法分别用来界面的入栈与出栈操作，你也可以简单理解，push 方法将弹出一个新的界面覆盖在原界面上，而 pop 操作则是将当前的界面移除，将其下面的界面显示出来。

抽 丝 剥 茧

需要注意，如果 React Native 版本大于 0.44，那么上面的代码可能会无法运行，原因在于 React Native 0.44 版本后将 Navigator 组件移动到了一个单独的库中，你需要在根目录中使用如下命令进行安装：

```
yarn add react-native-deprecated-custom-components
```

并且使用如下方式进行导入：

```
import { Navigator } from 'react-native-deprecated-custom-components'
```

7.6.2 Navigator 属性配置

Navigator 组件中提供了丰富的配置属性，如表 7-8 所示。

表 7-8 Navigator 组件的配置属性

属性名	解释	平台	值类型或可选值
configureScene	配置界面弹出的效果	通用	函数，会传入如下格式参数： (route, routeStack) 其中 route 为当前跳转的路由，routeStack 为路由栈数组。需要返回如下指定值（所有值都需要加 Navigator.SceneConfigs 前缀）： • PushFromRight：从右侧压入 • FloatFromRight：右侧浮入 • FloatFromLeft：左侧浮入 • FloatFromBottom:下侧浮入 • FloatFromBottomAndroid：下侧闪入 • FadeAndroid：闪烁进入 • HorizontalSwipeJump：水平滑入 • HorizontalSwipeJumpFromRight：右侧水平滑入 • VerticalUpSwipeJump：竖直滑入 • VerticalDownSwipeJump：竖直滑入

（续有）

属性名	解释	平台	值类型或可选值
renderScene	用来渲染指定路由场景	通用	函数，会传入如下参数： (route, navigator) route 为当前跳转的路由，navigator 为导航组件实例，需要返回组件对象
initialRoute	定义启动时加载的路由	通用	自定义路由对象
initialRouteStack	提供一个初始化的路由数组	通用	数组，其中为自定义的路由对象
onWillFocus	界面切换前会被调用的回调，会将新界面的路由作为参数传入	通用	函数，会将切换界面的路由作为参数传递
onDidFocus	界面切换完成后回调的函数	通用	函数，会将当前界面的路由传入
navigationBar	设置在界面切换过程中使用保持的导航栏	通用	需要返回组件对象
navigator	设置父导航控件	通用	需要返回 Navigator 对象
sceneStyle	设置每个界面容器的风格	通用	同 View 组件的 style 属性，这个属性设置后会作用在所有路由界面中

7.6.3　Navigator 实例方法解析

Navigator 组件中提供了大量的用来转换界面的方法，如表 7-9 所示。

表 7-9　Navigator 组件的方法

方法名	解释	平台	参数
getCurretRoutes	获取当前路由栈数组	通用	无
jumpBack	跳回上一个路由，但是依然保存当前路由	通用	无
jumpForward	与 jumpBack 对应，如果通过 jumpBack 跳回上一个路由，路由会被保存，使用这个方法可以再跳回下一个路由	通用	无
jumpTo	跳转到指定路由，不销毁当前的路由	通用	路由对象
push	跳转到新的界面	通用	路由对象
pop	跳转回上一个界面，并且将当前界面和路由销毁	通用	无
replace	用一个新的路由替换当前的界面	通用	路由对象
replaceAtIndex	用新的路由替换掉路由栈数组中指定位置的路由	通用	需要两个参数，第一个为路由对象，第二个为要替换的路由在栈数组中的下标
replacePrevious	替换掉之前的路由界面	通用	路由对象
resetTo	跳转到新的场景，并且重置路由栈	通用	路由对象
immediatelyResetRouteStack	用新的路由栈来重置 Navigator	通用	路由对象数组

（续表）

方法名	解释	平台	参数
popToRoute	跳转回之前的某个路由位置，这个位置之后的所有路由界面都会销毁	通用	路由对象
popToTop	跳转回导航的第一个界面，之后所有的路由界面都会被销毁	通用	无

7.7 iOS 平台的导航控制器 NavigatorIOS 组件

如果你熟悉 iOS 原生开发，那么你一定知道 UINavigationController 这个类，在 iOS 原生开发中会时常使用到 UINavagationController 来组织导航栈结构的界面切换。React Native 中提供了与其对应的 NavigatorIOS 组件。

7.7.1 使用 NavigatorIOS 组件

在 HelloWorld 工程的 Demo 文件夹下面新建一个命名为 NavigatorIOSDemo.js 的文件，在其中编写如下代码：

```
import React, {
    Component
} from 'react';
import {
    NavigatorIOS,
    View,
    Text,
    Button
} from 'react-native';
export default class NavigatorIOSDemo extends Component {
    render() {
        return (
            <NavigatorIOS initialRoute = {
                {
                    component: MyScene,
                    title: 'First',
                    passProps:{
                        param:"Secen",
                        index:0
                    }
                }
            }
            style = {
                {
                    flex: 1
```

```
                    }
                }
                />
            );
        }
    }
    class MyScene extends Component {
        render() {
            return (
                <View>
                    <Text style={{marginTop:70,textAlign:'center',fontSize:22}}>
                        {this.props.param+this.props.index}
                    </Text>
                    <Button title = "push"
                        onPress = {
                            () => {
                                this.props.navigator.push({
                                title: "Other",
                                component:MyScene,
                                passProps:{
                                    param:"Secen",
                                    index:this.props.index+1
                                }
                            });
                        }
                    }
                />
                </View> )
            ;
        }
    }
```

修改 index.ios.js 文件后在 iOS 平台运行工程，效果如图 7-9 所示。

NavigatorIOS 组件自带一个顶部的导航栏，导航栏中间会显示标题，如果当前界面不是导航器的根界面，那么导航栏上还会默认显示一个返回按钮，当用户单击返回按钮或者使用右滑手势时，可以实现界面的返回操作（可以单击 push 进行试验）。

关于上面的示例代码，我们还需要深入地解释一下，和 Navigator 组件不同的是，NavigatorIOS 组件的路由配置有着严格的要求，同时功能也更加强大。首先 NavigatorIOS 组件的 initialRoute 属性用来设置导航器的初始路由，即导航器加载出来时默认显示的界面。在 NavigatorIOS 组件中，路由对象是被严格定义的，其中可以添加的属性和意义如表 7-10 所示。

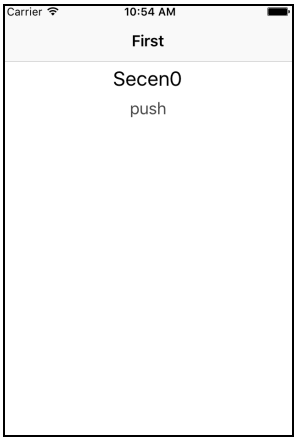

图 7-9　NavigatorIOS 组件效果

表 7-10　在 NavigatorIOS 组件路由对象属性

属性名	解释	平台	值类型或可选值
component	必选属性，设置渲染界面的组件类	iOS	类函数对象
title	设置界面标题，这个标题会被渲染在导航栏中间	iOS	字符串
titleImage	设置标题图标，如果设置了这个属性，会在导航栏中间显示所设置的图标	iOS	本地图片素材
passProps	用来传递参数	iOS	自定义对象
backButtonIcon	设置返回按钮图标	iOS	本地图片素材
backButtonTitle	设置返回按钮标题	iOS	字符串
leftButtonTitle	提供一个左侧功能按钮，设置按钮标题	iOS	字符串
leftButtonIcon	提供一个左侧功能按钮，设置按钮图标	iOS	本地图片素材
leftButtonSystemIcon	提供一个系统风格的左侧功能按钮	iOS	枚举字符串： • done：完成按钮 • cancel：取消按钮 • edit：编辑按钮 • save：存储按钮 • add：添加按钮 • compose：整理按钮 • reply：重复按钮 • action：活动按钮 • organize：组织按钮 • bookmarks：书库按钮 • search：搜索按钮 • refresh：刷新按钮 • stop：停止按钮 • camera：相机按钮 • trash：删除按钮 • play：播放按钮 • paush：暂停按钮 • rewind：回退按钮 • undo：撤销按钮 • redo：重做按钮
onLeftButtonPress	左侧功能按钮被单击时的回调函数	iOS	函数
rightButtonIcon	设置右侧功能按钮图标	iOS	本地图片素材
rightButtonTitle	设置右侧功能按钮标题	iOS	字符串
rightButtonSystemIcon	提供一个系统风格的右侧功能按钮	iOS	枚举字符串，同 rleftButtonSystemIcon
onRightButtonPress	右侧功能按钮被单击时回调的函数	iOS	函数

（续表）

属性名	解释	平台	值类型或可选值
wrapperStyle	设置当前路由对应界面中容器视图的风格	iOS	style 对象
navigationBarHidden	设置是否隐藏导航栏	iOS	布尔值
shadowHidden	设置是否隐藏导航栏下边线	iOS	布尔值
tintColor	设置导航栏按钮颜色	iOS	颜色字符串
barTintColor	设置导航栏背景颜色	iOS	颜色字符串
titleTextColor	设置导航栏标题颜色	iOS	颜色字符串
translucent	设置导航栏是否半透明	iOS	布尔值

正如表 7-10 所示，NavigatorIOS 组件的路由对象提供了极为丰富的可配置属性，当进行界面渲染时，component 属性注册的组件类会被自动进行实例化，并且会对界面组件实例添加一些属性：route、navigator 以及 passProps 所提供的自定义对象中的属性。在 MyScene 类中，你可以使用 this.props.navigator 来获取导航实例，也可以使用 this.props.route 获取当前路由对象，还可以使用 this.props.index 直接获取 passProps 传递的属性。

7.7.2　NavigatorIOS 属性与方法解析

NavigatorIOS 组件中常用属性如表 7-11 所示。

表 7-11　NavigatorIOS 组件常用属性

属性名	解释	平台	值类型或可选值
navigationBarHidden	设置导航栏是否隐藏	iOS	布尔值
shadowHidden	设置是否显示导航栏下边线	iOS	布尔值
itemWrapperStyle	设置导航中界面组件的容器视图风格	iOS	style 对象
tintColor	设置导航按钮颜色	iOS	颜色字符串
barTintColor	设置导航栏背景色	iOS	颜色字符串
titleTextColor	设置导航标题颜色	iOS	颜色字符串
translucent	设置导航栏是否半透明	iOS	布尔值
interactivePopGestureEnabled	设置是否开启右滑自动返回上一界面手势	iOS	布尔值
initialRoute	初始路由	iOS	路由对象

NavigatorIOS 组件提供的界面切换方法与 Navigator 组件基本一致，只是传入的路由参数必须是严格的 NavigatorIOS 路由对象，方法如表 7-12 所示。

表 7-12　NavigatorIOS 路由对象方法

方法名	解释	平台	参数
push	切换到一个新的界面	iOS	路由对象
pop	返回上一个界面	iOS	无
popN	返回到前 N 个界面	iOS	数值，当传入 1 时效果和 pop 一致

（续表）

方法名	解释	平台	参数
replaceAtIndex	替换导航栈中某个路由	iOS	传参格式如下： (route,index) • route：路由对象 • index：替换导航栈中的路由下标
replace	替换当前界面的路由	iOS	路由对象
replacePrevious	替换上一个界面的路由	iOS	路由对象
popToTop	返回到导航栈根路由界面	iOS	无
popToRoute	返回到导航栈中的指定路由	iOS	路由对象
replacePreviousAndPop	替换上一个界面的路由并且执行 pop 操作	iOS	路由对象
resetTo	重置导航栈	iOS	路由对象

抽 丝 剥 茧

NavigatorIOS 组件中最重要的概念便是路由，路由配置负责对界面渲染、设置导航栏样式以及为数据传递提供支持。

7.8 标签栏 TabBarIOS 组件

本节将介绍 React Native 中最后一个常用的视图管理组件 TabBarIOS。正如其名，TabBarIOS 组件只能应用于 iOS 平台，其是对 iOS 原生控件 UITabBarBarController 的 React Native 版实现。所谓标签栏其实际也是用于界面的切换，与 Navigator 不同的是导航管理的界面是有层次结构的，而标签栏管理的界面是并列结构，其界面的切换并没有父子关系。

在 HelloWorld 工程的 Demo 文件夹下新建一个命名为 TabBarIOSDemo.js 的文件，在其中编写如下代码：

```
import React,{Component} from 'react';
import {TabBarIOS,View,Text} from 'react-native';
export default class TabBarIOSDemo extends Component{
    constructor(props){
        super(props);
        this.state={
            item1Selected:true,
            item2Selected:false
        }
    }
    render(){
        return(
            <TabBarIOS
```

```
                barTintColor="green"
                style={{height:49}}
                tintColor="red"
                unselectedItemTintColor="white"
                translucent={false}>
                    <TabBarIOS.Item
                    title="历史"
                    systemIcon="history"
                    selected={this.state.item1Selected}
                    badge={1}
                    onPress={()=>{
                        this.setState({
                            item1Selected:true,
                            item2Selected:false
                            });
                    }}>
                        <View>
                            <Text style={{top:100,textAlign:'center',
fontSize:30}}>历史界面</Text>
                        </View>
                    </TabBarIOS.Item>
                    <TabBarIOS.Item
                    title="数学"
                    systemIcon="bookmarks"
                    selected={this.state.item2Selected}
                    onPress={()=>{
                        this.setState({
                            item1Selected:false,
                            item2Selected:true
                            });
                    }}>
                        <View>
                            <Text style={{top:100,textAlign:'center',
fontSize:30}}>数学界面</Text>
                        </View>
                    </TabBarIOS.Item>
                </TabBarIOS>
            );
        }
    }
```

TabBarIOS 组件中的属性用来配置导航栏，是一个双标签，其中需要嵌套 TabBarIOS.Item 组件。Item 组件用来配置标签栏中每个具体的标签，也是一个双标签，在 Item 组件中进行对应界面视图的编写。在 iOS 平台运行工程，效果如图 7-10 所示。

TabBarIOS 组件十分简洁，扩展于 View 组件，默认可以使用 View 组件中包含的属性，其他用于配置标签栏的属性如表 7-13 所示。

图 7-10　TabBarIOS 组件效果

表 7-13　TabBarIOS 组件属性

属性名	解释	平台	值类型或可选值
unselectedTintColor	设置未选中状态下的标签 Item 文字颜色	iOS	颜色字符串
tintColor	设置选中状态下的标签 Item 图标颜色	iOS	颜色字符串
unselectedItemTintColor	设置未选中状态下的标签 Item 图标颜色	iOS	颜色字符串
barTintColor	设置标签栏背景色	iOS	颜色字符串
translucent	设置标签栏是否半透明	iOS	布尔值
itemPositioning	设置标签栏中标签 Item 图标显示模式	iOS	枚举字符串 • fill：充满 • center：居中 • auto：自动

TabBarIOS.Item 实际上是 TabBarIOSItem 组件，其中提供的常用属性如表 7-14 所示。

表 7-14　TabBarIOS.Item 的常用属性

属性名	解释	平台	值类型或可选值
badge	设置此标签头标气泡显示的文案	iOS	字符串或数值
badgeColor	设置此标签头标气泡的颜色	iOS	颜色字符串
systemIcon	设置此标签显示为系统图标	iOS	枚举字符串 • bookmarks：书库图标 • contacts：联系人图标 • downloads：下载图标 • favorites：偏好图片 • history：历史图标 • more：更多图标 • most-recent：最近图标

（续表）

属性名	解释	平台	值类型或可选值
systemIcon	设置此标签显示为系统图标	iOS	• most-viewed：浏览图标 • recents：最近图标 • search：搜索图标 • top-rated：排行图标
icon	为此标签设置一个自定义的图标	iOS	本地图片素材
selectedIcon	为此标签设置一个选中状态的图标	iOS	本地图片素材
onPress	当单击标签时触发的回调	iOS	函数
renderAsOriginal	图标的渲染是否保持原始状态	iOS	布尔值
selected	设置此标签是否选中	iOS	布尔值
style	设置标签的风格属性	iOS	style 对象
title	设置标签的标题	iOS	字符串

需要注意，TabBarIOSItem 组件是一个受控组件，也就是说除非手动修改 selected 属性的值，否则标签的选中状态不会随用户的操作进行修改。通常情况下，开发者会通过属性来确定标签栏的选中状态，当 onPress 方法被调用时来进行属性的刷新。

博 闻 强 识

在 iOS 原生开发中，一种常用的界面搭建框架便是在标签栏中嵌套导航，标签栏并列展示几个功能模块，导航来控制控能模块中子级界面的切换。