前言

> *本小册是《千锋大前端小册》系列之 React Navigation 部分。通过本小册，可以系统学习 React Native 路由和导航基础知识。*
>
> *—— 作者：千锋教育·古艺散人*

React Navigation 导航系统

React Navigation，是专门为 React Native 的应用打造的路由和导航系统。

特点

使用简单

开箱即用，体验超好的内置导航器。

为iOS和Android构建的组件

兼容iOS 和 Android 平台，根据特定的平台，打造外观和感觉与平滑的动画和手势。

完全可定制

如果你知道如何使用JavaScript编写应用程序，则可以自定义React导航的任何部分。

平台扩展特性

React Navigation 在每一层都是可扩展的——你可以编写自己的导航器，甚至可以替换面向用户的API。

起步

如果您已经熟悉React Native，那么你将能够快速使用React导航！如果没有，您可能需要先阅读React Native Express的第1到4节（包括第1到4节），完成后再回到这里。

本文档的基础知识部分将介绍React导航的最重要方面。它应该足以让您了解如何构建典型的小型移动应用程序，并为你提供深入了解React导航更高级部分所需的基础知识。

安装

安装

在React Native项目中安装所需的软件包：

```
1.  npm install @react-navigation/native
```

React Navigation由一些核心实用程序组成，应用在RN中，可以实现路由导航功能。

我们要安装的模块有 react-native-gesture-handler, react-native-reanimated, react-native-screens,react-native-safe-area-context

在 expo 项目里安装

```
1.  npm install react-native-gesture-handler react-native-reanimated react-native-screens react-native-safe-area-context @react-native-community/masked-view
```

在纯 React Native 项目中安装

在项目根目录下，运行：

```
1.  npm install react-native-reanimated react-native-gesture-handler react-native-screens react-native-safe-area-context @react-native-community/masked-view
```

> *在安装依赖包时可能抛出一些警告，这通常是模块的版本匹配问题。你可以暂时忽略。*

From React Native 0.60 and higher, linking is automatic. So you don’t need to run react-native link.

自从 React Native 0.60 或更高版本后，会自动做link。因此你不需要运行 react-native link。

If you’re on a Mac and developing for iOS, you need to install pods to complete the linking. Make sure you have Cocoapods installed. Then run:

如果你开发环境是Mac的iOS系统，你需要安装pods来完成link。确保安装了 `Cocoapods` 。然后运行：

```
1.  cd ios; pod install; cd ..
```

要完成react native手势处理程序的安装，请入口文件（例如index.js或App.js）的顶部添加以下内容（确保它位于文件内容的顶部，并且前面没有其他内容）：

```
1.  import 'react-native-gesture-handler';
```

> *注意：如果你跳过这一步，你的应用程序可能会在生产中崩溃，即使它在开发中运行良好。*

Now, we need to wrap the whole app in NavigationContainer. Usually you’d do this in your entry file, such as index.js or App.js:

现在，我们需要将整个组件包裹在 NavigationContainer 组件中。通常入口文件（如index.js或App.js）

现在，我们需要将整个组件包装在 NavigationContainer 组件中。通常我们会在（如index.js或App.js）中执行此操作：

```
import * as React from 'react';
import { NavigationContainer } from '@react-navigation/native';

export default function App() {
 return (
  <NavigationContainer>
  {/* 其他应用代码 */}
  </NavigationContainer>
  );
}
```

> *注意：当你使用 navigator（比如 navigator 路由栈），你需要安装路由系统其他的依赖。 如果系统出现 “Unable to resolve module” 错误，那么就需要在系统中安装这些模块。*

现在，你可以运行你的程序在手机或模拟器上了。

初探 React Navigation

在web浏览器中，可以使用 `<a>` 标签链接到不同的页面。当用户单击一个链接时，URL被推送到浏览器历史堆栈中。当用户按下后退按钮时，浏览器会从历史堆栈的顶部弹出项目，因此活动页面现在是以前访问过的页面。React Native并不像web浏览器那样有一个内置的全局历史堆栈的概念。

React Navigation 的 stack navigator 为你的应用程序提供了在屏幕之间转换和管理导航历史的方法。如果您的应用程序只使用一个堆栈导航器，那么它在概念上与web浏览器处理导航状态的方式类似-当用户与它交互时，应用程序从导航堆栈中推送和弹出项目，这会导致用户看到不同的屏幕。这在web浏览器和React导航中的一个关键区别，React导航的stack navigator提供了在Android和iOS上在堆栈中的路由之间导航时所期望的手势和动画。

让我们从演示最常用的导航器 createStackNavigator 开始。

安装 stack navigator library

到目前为止，我们安装的库是导航器的构建块和共享基础，每个导航器都在自己的React Navigation 库中导航。要使用堆栈导航器，我们需要安装 @react navigation/stack：

```
npm install @react-navigation/stack
```

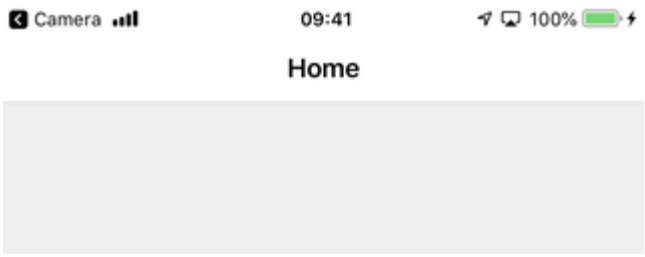
> *@react navigation/stack 依赖于 @react native community/masked view和我们在《起步》中安装的其他库。如果您还没有安装，请转到《起步》并按照安装说明进行操作。*

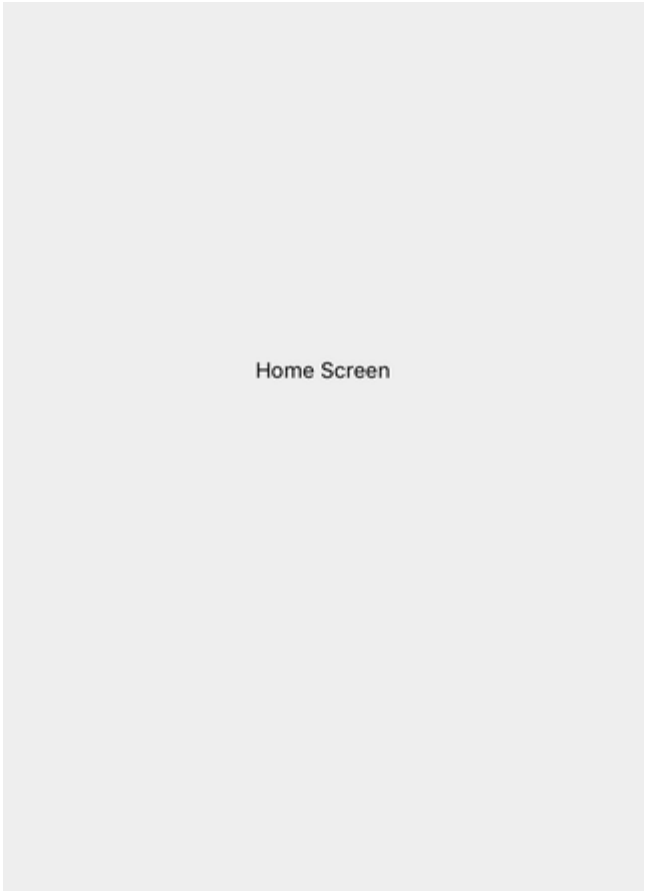
创建导航栈

createStackNavigator 是返回包含两个属性的对象的函数：Screen和Navigator。它们都是用于配置导航器的React组件。导航器应包含屏幕元素作为其子元素，以定义路由的配置。

NavigationContainer 是一个组件，它管理我们的导航树并包含导航状态。此组件必须包装所有导航器结构。通常，我们会在应用程序的根目录中定义这个组件，它通常是从app.js导出的组件。

```
// 如 App.js
import * as React from 'react'
import { View, Text } from 'react-native'
import { NavigationContainer } from '@react-navigation/native'
import { createStackNavigator } from '@react-navigation/stack'

function HomeScreen() {
 return (
 <View style={{ flex: 1, alignItems: 'center', justifyContent: 'center' }}>
 <Text>Home Screen</Text>
 </View>
 );
}

const Stack = createStackNavigator()

function App() {
 return (
 <NavigationContainer>
 <Stack.Navigator>
 <Stack.Screen name="Home" component={HomeScreen} />
 </Stack.Navigator>
 </NavigationContainer>
 )
}

export default App
```





如果运行此代码，您将看到一个带有空导航栏和包含主屏幕组件的灰色内容区域的屏幕（如上图所示）。您在导航栏和内容区域中看到的样式是堆栈导航器的默认配置，我们稍后将学习如何配置这些样式。

> *路由名称的大小写无关紧要。你可以使用小写的 home 或大写的 Home。更推荐大家用大写的路由名称。*
>
> *screen 唯一需要的配置是name 和 props。*

导航配置

所有路由配置都被指定为导航器的 props。我们没有向导航器传递任何 props，所以它只是使用默认配置。

让我们将第二个 screen 添加到堆栈导航器，并将主屏幕配置为首屏显示：

```
function DetailsScreen() {
  return (
    <View style={{ flex: 1, alignItems: 'center', justifyContent: 'center' }}>
      <Text>Details Screen</Text>
    </View>
  )
```

```
}

const Stack = createStackNavigator()

function App() {
  return (
    <NavigationContainer>
      <Stack.Navigator initialRouteName="Home">
        <Stack.Screen name="Home" component={HomeScreen} />
        <Stack.Screen name="Details" component={DetailsScreen} />
      </Stack.Navigator>
    </NavigationContainer>
  )
}
```

现在我们的路由栈有两条路由，一条主页路由和一条详情页路由。可以使用 Screen 组件指定路由。Screen 组件接受一个 name 属性，该属性对应于我们将用于导航的路由的名称，以及一个组件属性，该属性对应于它将显示的组件。

这里，Home route对应于HomeScreen组件，Details route对应于DetailsScreen组件。路由栈的初始路由是主路由。尝试将其更改为详细信息并重新加载应用程序（React Native的快速刷新不会更新来自initialRouteName的更改，如您所料），请注意，您现在将看到详细信息屏幕。然后把它改回原位，重新装填。

> *注意：component 属性值只能是组件，不是渲染函数（如不能赋值：component={() => <HomeScreen />}）。*

制定 options 选项

导航器中的每个 Screen 都可以为导航器指定一些选项，例如要在标题中显示的标题。这些选项可以在每个 Screen 组件的选项 options 中传递：

```
<Stack.Screen
  name="Home"
  component={HomeScreen}
  options={{ title: 'Overview' }}
/>
```

有时我们想为导航器所有的 Screen 指定相同的 options，为此我们可以将 `screenOptions` 属性传递给导航器。

传递附加的属性

有时我们可能想把附加的属性传给Screen。我们可以通过两种方法做到这一点：

1. 使用React context，用上下文提供组件包装导航器来将数据传递到Screen（推荐）。
2. 给 Screen 传递渲染函数(render functions)，而不是指定 component 属性：

```
<Stack.Screen name="Home">
{props => <HomeScreen {...props} extraData={someData} />}
</Stack.Screen>
```

> *注意：默认情况下，React Navigation 对 Screen 组件应用优化以防止不必要的渲染。使用渲染函数将删除这些优化。因此，如果使用渲染函数，则需要确保对屏幕组件使用React.memo 或 React.PureComponent，以避免性能问题。*

总结

- React Native没有像web浏览器那样的内置导航API。React Navigation为您提供了这一功能，同时还提供了iOS和Android手势和动画，以便在屏幕之间切换。
- Stack.Navigator 是一个组件，它将路由配置作为它的子级，并使用附加的配置 props 来渲染我们的内容。
- 每个Stack.Screen组件都有一个name属性，它引用路由的名称，而component属性指定要为路由渲染的组件。这是两个必需的属性。
- 要指定路由栈中的初始路由，请使用initialRouteName属性配置。
- 要指定特定于屏幕的选项，我们可以将选项属性传递给Stack.screen，对于常用选项，我们可以将screenOptions 传递给Stack.Navigator。

页面间跳转

在上一节 Hello React Navigation中，我们定义了一个具有两个路由（Home和Details）的导航器，但是我们没有学习如何让用户从Home导航到Details。

如果是浏览器，我们可以写这样的东西：

```
<a href="details.html">Go to Details</a>
```

或者：

```
<a
onClick={() => {
window.location.href = 'details.html';
}}
>
Go to Details
```

```
</a>
```

我们将实现类似第二种操作，但我们将使用传递到屏幕组件的导航属性，而不是使用window.location 全局选项。

导航到新窗口

```
import * as React from 'react';
import { Button, View, Text } from 'react-native';
import { NavigationContainer } from '@react-navigation/native';
import { createStackNavigator } from '@react-navigation/stack';

function HomeScreen({ navigation }) {
return (
<View style={{ flex: 1, alignItems: 'center', justifyContent: 'center' }}>
<Text>Home Screen</Text>
<Button
title="Go to Details"
onPress={() => navigation.navigate('Details')}
/>
</View>
);
}

// ... 其他代码
```

解释一下：

- navigation：在路由栈的定义的组件(component属性定义的组件)，默认会传给定义的每个组件一个 navigation 属性。
- navigate(‘Details’)：我们调用 navigate 函数，函数参数是在导航属性定义的名字（注意这里用字符串直接引用路由名字）

> *如果调用 navigation.navigate 时使用的路由名称不是在堆栈导航器上定义的，则不会发生任何事情。换句话说，我们只能导航到堆栈导航器上定义的路由，我们不能导航到任意组件。*

现在我们有了一个包含两条路由的堆栈：1）主路由 2）详细页路由。如果我们从详情页再次导航到详情页，会发生什么情况？

多次导航同一路由

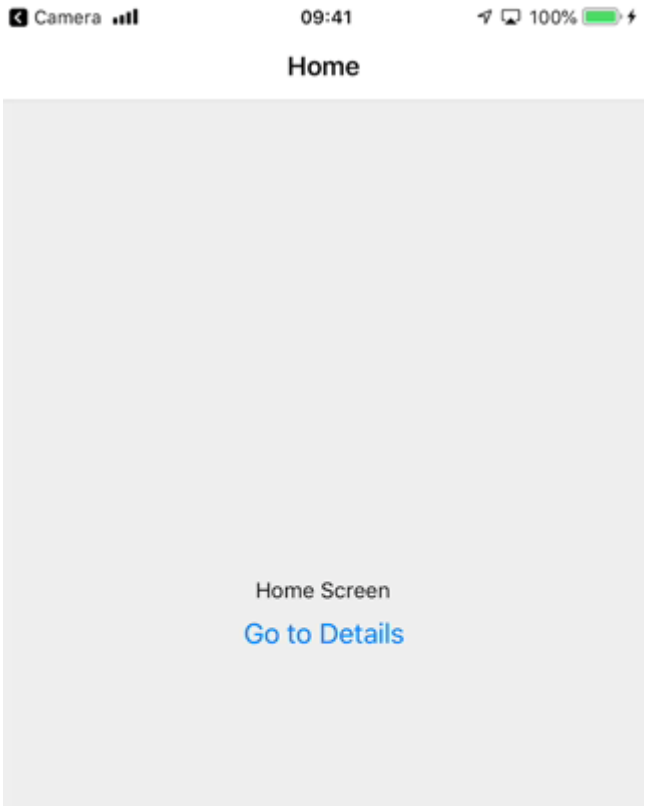
```
function DetailsScreen({ navigation }) {
```

```
                return (
    <View style={{ flex: 1, alignItems: 'center', justifyContent: 'center' }}>
                <Text>Details Screen</Text>
                <Button
                title="Go to Details... again"
                onPress={() => navigation.navigate('Details')}
                />
                </View>
                );
                }
```

如果您运行此代码，您将注意到当您点击 “转到详细信息…”，你会发现它什么也做不了！这是因为我们已经在详情页路由上。如果你已经在那个屏幕上，那么它将什么也不会做。

假设我们真的想导航到另一个详细页。这在向每个路由传递一些唯一数据的情况下非常常见（稍后我们讨论params时会详细介绍）。为此，我们可以将“导航”更改为“推送”。这使我们能够表达添加另一条路线的意图，而不管现有的导航历史。

```
                <Button
                title="Go to Details... again"
                onPress={() => navigation.push('Details')}
                />
```



每次调用push时，我们都会向导航堆栈中添加一条新路由。当您调用导航时，它首先试图找到具有该名称的现有路由，如果栈上还没有一个路由，则只推一条新路由。

返回

当从活动页面返回时，stack navigator 提供的标题将自动包含一个后退按钮（如果导航堆栈中只有一个屏幕，则没有可返回的内容，因此没有后退按钮）。

有时您希望能够以编程方式触发此行为，为此您可以使用 `navigation.goBack();`

```
function DetailsScreen({ navigation }) {
  return (
    <View style={{ flex: 1, alignItems: 'center', justifyContent: 'center' }}>
      <Text>Details Screen</Text>
      <Button
        title="Go to Details... again"
        onPress={() => navigation.push('Details')}
      />
      <Button title="Go to Home" onPress={() => navigation.navigate('Home')} />
      <Button title="Go back" onPress={() => navigation.goBack()} />
    </View>
  );
}
```

> *在 Android 设备上，React Navigation 会勾住硬件“后退”按钮，当用户按下“后退”按钮，它会为您触发goBack()函数，因此它的行为与用户期望的一样。*

另一个常见的需求是能够跨页面返回：例如，如果您在一个堆栈中有多个页面，并且希望关闭所有这些页面以返回到第一个页面。在这种情况下，我们就可以使用navigate(‘Home’)（而不是push！试试看区别）。另一种选择是 `navigation.popToTop()` ，它返回到堆栈中的第一个页面。

```
function DetailsScreen({ navigation }) {
  return (
    <View style={{ flex: 1, alignItems: 'center', justifyContent: 'center' }}>
```

```
      <Text>Details Screen</Text>
      <Button
        title="Go to Details... again"
        onPress={() => navigation.push('Details')}
      />
      <Button title="Go to Home" onPress={() => navigation.navigate('Home')} />
      <Button title="Go back" onPress={() => navigation.goBack()} />
      <Button
        title="Go back to first screen in stack"
        onPress={() => navigation.popToTop()}
      />
    </View>
  );
}
```

总结

- navigation.navigate(‘RouteName’) 如果新路由不在堆栈中，则将其推送到堆栈导航器，否则将跳到该页面。
- 我们可以随意多次调用 `navigation.push('RouteName')` ，它将继续推送路由。
- 标题栏将自动显示后退按钮，但您可以通过调用 `navigation.goBack()` 以编程方式返回。在 Android 上，硬件后退按钮正如预期的那样工作。
- 你可以在导航中返回到堆栈中的现有页面。navigation(‘RoutNeMy’)，你可以返回到堆栈中的第一个页面。
- 导航 props 可用于所有 Screen 定义的组件。

路由传参

现在我们知道了如何创建带有某些路由的堆栈导航器，以及如何在这些路由之间导航，下面让我们看看如何在导航到路由时将数据传递给它们。

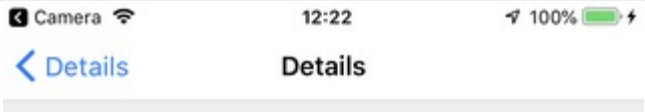
这里有两个部分：

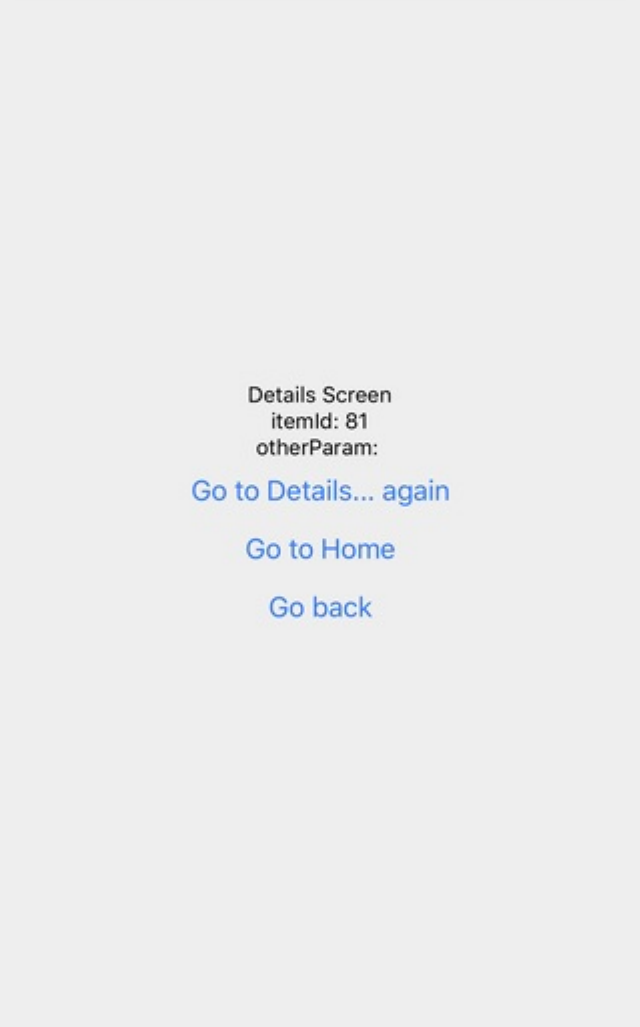
通过将参数作为 navigation.navigate 函数的第二个参数放入到一个对象中，将参数传递给路由：navigation.navigate(‘RouteName’, {/*参数*/})

读取Screen组件route.params中的参数。

```
function HomeScreen({ navigation }) {
  return (
```

```
      <View style={{ flex: 1, alignItems: 'center', justifyContent: 'center' }}>
        <Text>Home Screen</Text>
        <Button
          title="Go to Details"
          onPress={() => {
            /* 1. 携带参数后导航到Details页面 */
            navigation.navigate('Details', {
              itemId: 86,
              otherParam: 'anything you want here',
            })
          }}
        />
      </View>
    );
  }

function DetailsScreen({ route, navigation }) {
  /* 2. 在DetailsScreen组件里获取参数 */
  const { itemId } = route.params
  const { otherParam } = route.params
  return (
    <View style={{ flex: 1, alignItems: 'center', justifyContent: 'center' }}>
      <Text>Details Screen</Text>
      <Text>itemId: {JSON.stringify(itemId)}</Text>
      <Text>otherParam: {JSON.stringify(otherParam)}</Text>
      <Button
        title="Go to Details... again"
        onPress={() =>
          navigation.push('Details', {
            itemId: Math.floor(Math.random() * 100),
          })
        }
      />
      <Button title="Go to Home" onPress={() => navigation.navigate('Home')} />
      <Button title="Go back" onPress={() => navigation.goBack()} />
    </View>
  )
}
```





组件内部还可以更新其参数，就像它们可以更新其状态一样。navigation.setParams 方法允许您更新页面的参数。有关更多详细信息，请参阅setParams的API参考。

还可以向页面传递一些初始参数。如果导航到此页面时未指定任何参数，则将使用初始参数。它们还与你传递的任何参数浅合并。可以使用Initial params属性指定初始参数：

```
<Stack.Screen
name="Details"
component={DetailsScreen}
initialParams={{ itemId: 42 }}
/>
```

参数不仅对于将某些数据传递到新页面有用，而且对于将数据传递到上一个页面也很有用。例如，假设你有一个带有create post按钮的页面，create post按钮打开一个新的页面来创建post。创建post后，需要将post的数据传递回上一个页面。

为了实现这一点，可以使用导航方法，如果屏幕已经存在，它会像 goBack 一样。可以使用 navigate 传递参数以将数据传递回：

```
function HomeScreen({ navigation, route }) {
React.useEffect(() => {
if (route.params?.post) {
// 提交信息更新，利用 `route.params.post` 参数做些事情
// 比如，传递 post 数据给服务器
}
}, [route.params?.post]);

return (
<View style={{ flex: 1, alignItems: 'center', justifyContent: 'center' }}>
<Button
title="Create post"
onPress={() => navigation.navigate('CreatePost')}
/>
<Text style={{ margin: 10 }}>Post: {route.params?.post}</Text>
</View>
);
}

function CreatePostScreen({ navigation, route }) {
const [postText, setPostText] = React.useState('');

return (
<>
<TextInput
multiline
placeholder="What's on your mind?"
style={{ height: 200, padding: 10, backgroundColor: 'white' }}
value={postText}
onChangeText={setPostText}
/>
<Button
title="Done"
onPress={() => {
// 返回首页同时携带post参数
navigation.navigate('Home', { post: postText });
}}
/>
</>
);
`
```

```
41.                                                    }
```

在这里，点击“完成”按钮后，主页的 route.params 将更新，以反映在 navigate 中传递的 post 文本。

总结

- navigate 和 push 接受可选的第二个参数，以便将参数传递到要导航到的路由。例如：navigation.navigate(‘RouteName’，{paramName:’value’})。
- 可以通过页面内的 route.params 读取参数
- 可以使用navigation.setParams更新页面的参数
- 初始参数可以通过 Screen 上的 Initial params 属性传递