React Naitve Day3

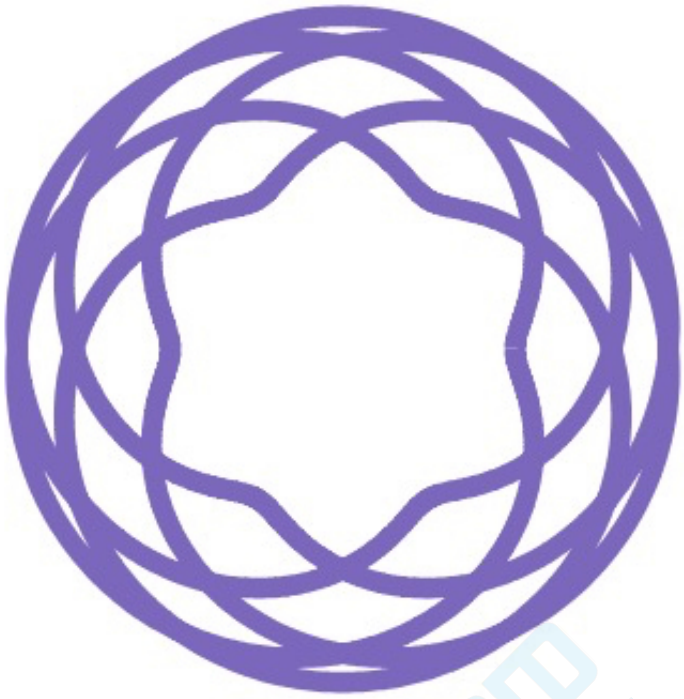
课前回顾

- 环境搭建
- 项目启动与调试
- React Native Flex布局与样式
- 核心组件&&API

课堂目标

- 掌握react navigation3.x的安装流程
- 了解react navigation 相关属性，和API
- 掌握createStackNavigator普通导航使用
- 熟练使用react-native-vector-icons图标库

react navigation介绍



源于React Native社区对基于Javascript的可扩展且使用简单的导航解决方案的需求

react-natvigation自开源以来。在短短不到3个月的时间，github上星数已达4000+。Fb推荐使用库，并且在React Native当前最新版本0.44中将Navigator删除。react-navigation据称有原生般的性能体验效果。可能会成为未来React Native导航组件的主流军

导航器

导航器也可以看成是一个普通的React组件，你可以通过导航器来定义你的APP中的导航结构。导航器还可以渲染通用元素，例如可以配置的标题栏和选项卡栏。

在react-navigation中有以下类型的导航器：

- createStackNavigator:类似普通的Navigator，导航上方导航栏
- createTabNavigator:已弃用，使用createBottomTabNavigator、createMaterialTopTabNavigator替代
- createBottomTabNavigator:相当于IOS里面的UITabBarController，屏幕下方的标签栏
- createMaterialTopTabNavigator:屏幕顶部的材料设计主题标签栏
- createDrawerNavigator:抽屉效果，侧边滑出
- createSwitchNavigator:SwitchNavigator的用途是一次只显示一个页面

你可以通过以上几种导航器来创建你的APP，可以是其中一个也可以多个组合，这个可以根据具体的应用场景并结合每一个导航器的特性进行选择。

相关概念

在开始学习导航器之前，我们需要了解两个和导航有关的概念：

- Screen navigation prop(屏幕导航属性)：通过navigation可以完成屏幕之间的调度操作，例如打开另一个屏幕
- Screen navigationOptions(屏幕导航选项)：通过navigationOptions可以定制导航器显示屏幕的方式(例如：头部标题，选项卡标签等)

导航器所支持的Props

```
const SomeNav =
createStackNavigator/createBottomTabNavigator/createMaterialTopTabNavigator({
  // config
});
<SomeNav
  screenProps = {xxx} //向子屏幕传递额外的数据，子屏幕可以通过this.props.screenProps获取到该数据
  ref = {nav => {navigation = nav;}} //可以通过ref属性获取到navigation;
  onNavigationStateChange=(prevState,newState,action)=>{
    //onNavigationStateChange:每次当导航器所管理的state发生改变时，都会回调该方法
    //prevState:变化之前的state
    //newState:新的state
    //action:导致state变化的action
  }
/>
```

Screen Navigation Prop(屏幕的navigation props)

当导航器中的屏幕被打开时，它会接收到一个navigation prop，它是整个导航环节的关键一员，接下来详细的讲解一下navigation prop的作用

navigation:包含以下功能

- navigate：跳转到其他界面
- state：屏幕的当前state
- setParams：改变路由的params
- goBack：关闭当前屏幕
- disPatch：向路由发送一个action
- addListener：订阅导航生命周期的更新
- isFocused：true标识屏幕获取了焦点
- getParam：获取具有回退的特定参数
- dangerouslyGetParent：返回父导航器

注意：一个navigation有可能没有navigate、setParams以及goBack，只有state与dispatch，所以在使用navigate时要进行判断，如果没有navigate可以使用navigation去dispatch一个新的action。如：

```
const {navigation,theme,selectedTab} = this.props;
const resetAction = StackActions.reset({
  index:0,
  action:[
    NavigationActions.navigate({
      routeName: 'HomePage',
      params:{
        theme:theme,
        selectedTab:selectedTab
      }
    })
  ]
})
navigation.dispatch(resetAction)
```

提示：这里的reset在2.0及以后的版本中，从NavigationActions中移到了StackActions中了，使用时记得留意。

StackNavigator的navigation的额外功能：

当且仅当当前navigator是stackNavigator时，this.props.navigation上有一些附加功能。这些函数是navigate和goBack的替代方法，你可以使用任何你喜欢的方法。这些功能是：

- this.props.navigation
  - Push ： 导航到堆栈中的一个新路由
  - pop：返回堆栈中的上一个页面
  - popToTop：跳转到堆栈中最顶层的页面
  - replace：用新路由替换当前路由
  - reset：擦除导航器状态并将其替换为多个操作的结果
  - dismiss：关闭当前栈

使用navigate进行界面之间的跳转

- navigation.navigate({routeName,params,action,key})或者 navigation.navigate(routeName,params,action)
  - routeName：要跳转到的界面路由名称，也就是导航其中配置的路由名
  - params：要传递给下一个界面的参数
  - action：如果该界面是一个navigator的话，将运行这个sub-action
  - Key：要导航到的路由的可选标识符，如果已存在，将后退到此路由

```
export const AppStackNavigator = createStackNavigator({
  HomeScreen: {
    screen: HomeScreen
  },
  Page1: {
    screen: Page1
  }
```

```
})

class HomeScreen extends React.Component {
  render(){
    const {navigate} = this.props.navigation;
    return (
      <View>
        <Text>This is HomeScreen</Text>
        <Button
          onPress={()=> navigate('Page1',{
            name: 'David'
          })}
          title = "Go to Page1"
        />
      </View>
    )
  }
}
```

使用state的params

可以通过this.props.state.params来获取通过setParams()，或navigation.navigate()传递的参数。

```
<Button
  title={params.mode === 'edit' ? '保存' : '编辑'}
  onPress={()=>
    setParams({
      mode: params.mode === 'edit'? '' : 'edit'
    })
  }
/>
<Button
  title="Go to Page1"
  onPress={()=>{
    navigation.navigate('Page1',{name:'David'})
  }}
/>
const {navigation} = this.props
const {state,setParams} = navigation
const {params} = state
const showText = params.mode === 'edit'? '正在编辑':'编辑完成'
```

使用setParams改变route Params

- setParams：function setParams(params)：可以借助setParams来改变route Params，比如，

通过setParams来更新页面顶部的标题，返回按钮等

```
class ProfileScreen extends React.Component {
  render(){
    const {setParams} = this.props.navigation
    return(
      <Button
        onPress={()=>setParams({name: 'Lucy'})}
        title="set title name to 'Lucy'"
      />
    )
  }
}
```

注意navigation.setParams改变的是当前页面的Params，如果要改变其他页面的Params可以通过NavigationActions.setParams完成，下文会讲到

使用goBack返回上一页面或指定页面

- goBack：function goBack(key)：我们可以借助goBack返回到上一页或者路由栈的指定页面。
  - 之中key标识你要返回到页面的页面标识符，如：id-1517035332238-4，不是routeName
  - 可以通过指定页面的navigation.state.key来获取页面的标识
  - key非必传，也可传null

navigation.state {params: {...},key:"id-1517035332238-4",routeName:"Page1"}

```
export default class Page1 extends React.ComPonent {
  render(){
    const {navigation} = this.props;
    return <View>
      <Text>欢迎来到Page1</Text>
      <Button
        title="Go Back"
        onPress={()=>{
          navigation.goBack()
        }}
      />
    </View>
  }
}
```

通过dispatch 发送一个action

- dispatch：function dispatch(action)：给当前界面设置action，会替换原来的跳转，回退等事件

```
const resetAction = StackActions.reset({
  index:0,
  actions:[
    NavigationActions.navigate({
      routeName: 'HomePage',
      params:{
        theme:theme,
        selectedTab:selectedTab
      }
    })
  ]
})
navigation.dispatch(resetAction)
```

NavigationActions

- Navigate：导航到其他页面
- Back：返回到上一个页面
- Set Params：设置指定页面的Params
- Init：初始化一个state，如果state是undefined

Navigate

navigate action 会使用 navigate action的结果来更新当前的state

```
navigate({routeName,params,action,key})
```

- routeName: 字符串，必选项，在app的router里注册的导航目的地的routeName
- params：对象，可选项，融合进目的地route的参数
- actions：对象，可选项(高级)，如果screen也是一个navigator，次级action可以在子route中运行，在文档中描述的任何actions都可以作为次级action.
- key：string or null 可选，要导航到的路由的标识符，如果已存在，则导航回此路由

```
import {NavigationActions} from 'react-navigation'
const navigateAction = NavigationActions.navigate({
  routeName: 'Profile',
  params: {},
  action: NavigationActions.navigate({
    routeName: 'SubProfileRoute'
  })
})
this.props.navigation.dispatch(navigationAction)
```

Back

返回到前一个screen并且关闭当前screen.backaction creator 接受一个可选的参数：

back(key)

- Key String可选，这个可以和上文中讲到的goBack的key是一个概念；

```
import { NavigationActions } from 'react-navigation'
const backAction = NavigationActions.back();
this.props.navigation.dispacth(backAction)
```

SetParams

通过SetParams我们可以修改指定页面的Params

- Params:对象，必选参数，将会被合并到已经存在页面的Params中
- key：字符串，必选参数，页面的key

```
import {NavigationActions} from 'react-navigation'
const setParamsAction = NavigationActions.setParams({
  params:{title:'HomePage'},
  key:'id-1517035332238-4'
})
```

有很多小伙伴可能会问：navigation中有setParams为什么还要有NavigationActions.setParams?

从两方面来回答一下这个问题:

- 在上文中讲到过navigation中有可能只有stata和dispatch，这个时候如果要修改页面的Params，则只能通过NavigationActions.setParams了
- 另外，navigation.setParams只能修改当前页面的params,而navigationActions.setParams可以修改所有页面的Params

StackActions

- Reset：重置当前state到一个新的state
- replace：使用另一个路由替换指定的路由
- Push：在堆栈顶部添加一个页面，然后跳转到该页面
- Pop：跳转到上一个页面
- PopToTop：跳转到堆栈最顶层的页面，并销毁其他所有页面

Reset

Reset action删掉所有的navigation state并且使用这个actions的结果来代替

- index：数组，必选，navigation state中route数组中激活route的index
- actions：数组，必选，navigation actions数组，将会替代navigation state
- key：string or null 可选，如果设置，具有给定key的导航器将重置。如果为null,则根导航器将重置。

createStackNavigator-普通导航

`createStackNavigator`提供APP屏幕之间切换的能力，它是以栈的形式还管理屏幕之间的切换，新切换到的屏幕会放在栈的顶部。

createStackNavigator API

```
createStackNavigator(RouteConfigs, StackNavigatorConfig):
```

- `RouteConfigs`(必选)：路由配置对象是从路由名称到路由配置的映射，告诉导航器该路由呈现什么。
- `StackNavigatorConfig`(可选)：配置导航器的路由(如：默认首屏，navigationOptions，paths等)样式(如，转场模式mode、头部模式等)。

RouteConfigs

RouteConfigs支持三个参数`screen`、`path`以及`navigationOptions`；

- `screen`(必选)：指定一个 React 组件作为屏幕的主要显示内容，当这个组件被createStackNavigator加载时，它会被分配一个`navigation` prop。
- `path`(可选)：用来设置支持schema跳转时使用，具体使用会在下文的有关`Schema`章节中讲到；
- `navigationOptions`(可选)：用以配置全局的屏幕导航选项如：title、headerRight、headerLeft等；

StackNavigatorConfig

从`react-navigation`源码中可以看出StackNavigatorConfig支持配置的参数有10个。

```
function createStackNavigator(routeConfigMap, stackConfig = {}) {
  const {
    initialRouteKey,
    initialRouteName,
    initialRouteParams,
    paths,
    defaultNavigationOptions,
    disableKeyboardHandling,
    getCustomActionCreators
  } = stackConfig;
  ...
```

这7个参数可以根据作用不同分为路由配置、视图样式配置两类，首先看用于路由配置的参数：

> 用于路由配置的参数：

- initialRouteName: 设置默认的页面组件，必须是上面已注册的页面组件。
- initialRouteParams: 初始路由的参数。
- **defaultNavigationOptions**: 屏幕导航的默认选项，下文会详细讲解。
- initialRouteKey - 初始路由的可选标识符。
- paths: 用来设置支持schema跳转时使用，具体使用会在下文的有关`Schema`章节中讲到。

> 用于导航样式配置的参数：

- mode: 页面切换模式: 左右是card(相当于iOS中的push效果), 上下是modal(相当于iOS中的modal效果)
  - card: 普通app常用的左右切换。
  - modal: 上下切换。
- headerMode: 导航栏的显示模式: screen: 有渐变透明效果, float: 无透明效果, none: 隐藏导航栏。
  - float: 无透明效果, 默认。
  - screen: 有渐变透明效果, 如微信QQ的一样。
  - none: 隐藏导航栏。
- headerBackTitleVisible : 提供合理的默认值以确定后退按钮标题是否可见，但如果要覆盖它，则可以使用true或` false 在此选项中。
  - fade-in-place: 标题组件交叉淡入淡出而不移动，类似于iOS的Twitter，Instagram和Facebook应用程序。 这是默认值。
  - uikit: iOS的默认行为的近似值。 headerTransitionPreset: 指定在启用headerMode：float时header应如何从一个屏幕转换到另一个屏幕。
- cardStyle: 样式（iOS上页面切换会有白色渐变蒙层，想去掉则可以这样设置，cardStyle: { opacity: null },切换页面时的页面边框也在这里可以设置）。
- onTransitionStart: 页面切换开始时的回调函数 (我们可以在这里注册一些通知，告知我们切面切换的状态，方便后面处理页面切换事件)。
- onTransitionEnd: 页面切换结束时的回调函数。

navigationOptions（屏幕导航选项）

支持一下参数：

- title: 可以作为headerTitle的备选字段(当没设置headerTitle时会用该字段作为标题)，也可以作为TabNavigator的tabBarLabel以及DrawerNavigator的drawerLabel。
- header: 自定义导航条，可以通过设置null来隐藏导航条；
- headerTitle: 标题；
- headerTitleAllowFontScaling: 标题是否允许缩放，默认true；
- headerBackTitle: 定义在iOS上当前页面进入到下一页面的回退标题，可以通过设置null来禁用它；
- headerTruncatedBackTitle: 当回退标题不能显示的时候显示此属性的标题，比如回退标题太长了；
- headerBackImage：React 元素或组件在标题的后退按钮中显示自定义图片。 当组件被调用时，它会在渲染时收到许多 props 如：（tintColor，title）。 默认为带有 react-navigation/views/assets/back-icon.png 这张图片的组件，后者是平台的默认后图标图像（iOS上为向左的符号，Android上为箭头）。
- headerRight: 定义导航栏右边视图；
- headerLeft: 定义导航栏左边视图；
- headerStyle: 定义导航栏的样式，比如背景色等；
- headerTitleStyle: 定义标题的样式；
- headerLeftContainerStyle：自定义 headerLeft 组件容器的样式，例如，增加 padding。
- headerRightContainerStyle：自定义 headerRight 组件容器的样式,，例如，增加 padding。
- headerTitleContainerStyle：自定义 headerTitle 组件容器的样式, 例如，增加 padding。
- headerBackTitleStyle: 定义返回标题的样式；
- headerPressColorAndroid：颜色为材料波纹 (Android >= 5.0)；
- headerTintColor: 定义导航条的tintColor，会覆盖headerTitleStyle中的颜色；
- headerTransparent：默认为 false。如果 true, 则标头将不会有背景, 除非您显式提供 headerStyle 或 headerBackground。
- headerBackground：与headerTransparent一起使用，以提供在标题后台呈现的组件。 例如，您可以使用模糊视图来创建半透明标题。
- gesturesEnabled: 定义是否能侧滑返回，iOS默认true，Android默认false；
- gestureResponseDistance: 定义滑动返回的有效距离，水平状态下默认：25，垂直状态默认135；
- gestureDirection: 设置关闭手势的方向。默认从左向右，可以设置从右到左的滑动操作。

创建一个StackNavigator类型的导航器

```
export const AppStackNavigator = createStackNavigator({
    HomePage: {
        screen: HomePage
    },
    Page1: {
        screen: Page1,
```

```
        navigationOptions: ({navigation}) => ({
            title: `${navigation.state.params.name}页面名`//动态设置
navigationOptions
        })
    },
    Page2: {
        screen: Page2,
        navigationOptions: {//在这里定义每个页面的导航属性，静态配置
            title: "This is Page2.",
        }
    },
    Page3: {
        screen: Page3,
        navigationOptions: (props) => {//在这里定义每个页面的导航属性，动态配置
            const {navigation} = props;
            const {state, setParams} = navigation;
            const {params} = state;
            return {
                title: params.title ? params.title : 'This is Page3',
                headerRight: (
                    <Button
                        title={params.mode === 'edit' ? '保存' : '编辑'}
                        onPress={() =>
                            setParams({mode: params.mode === 'edit' ? '' :
'edit'})}
                    />
                ),
            }
        }
    },
}, {
    defaultNavigationOptions: {
        // header: null,// 可以通过将header设为null 来禁用StackNavigator的Navigation
Bar
    }
});
```

配置navigationOptions：

步骤一的代码中通过两种方式配值了navigationOptions：

静态配置：

对Page2的navigationOptions配置是通过静态配置完成的：

```
Page2: {
    screen: Page2,
    navigationOptions: {//在这里定义每个页面的导航属性，静态配置
        title: "This is Page2.",
    }
},
```

这种方式被称为静态配置，因为navigationOptions中的参数是直接Hard Code的不依赖于变量。

动态配置：

对Page3的navigationOptions配置是通过动态配置完成的：

```
Page3: {
    screen: Page3,
    navigationOptions: (props) => {//在这里定义每个页面的导航属性，动态配置
        const {navigation} = props;
        const {state, setParams} = navigation;
        const {params} = state;
        return {
            title: params.title ? params.title : 'This is Page3',
            headerRight: (
                <Button
                    title={params.mode === 'edit' ? '保存' : '编辑'}
                    onPress={() =>
                        setParams({mode: params.mode === 'edit' ? '' : 'edit'})}
                />
            ),
        }
    }
},
```

从上述代码中可以看出Page3的navigationOptions依赖于props这个变量所以是动态的，当props中的内容发生变化时，navigationOptions也会跟着变化；

> 提示：除了在创建createStackNavigator时配置navigationOptions外，在StackNavigator之外也可以配置navigationOptions；

createStackNavigator之外也可以配置navigationOptions

> 方式一：

```
Page2.navigationOptions = {
     title: "This is Page2.",
};
```

> 方式二：

```
export default class Page1 extends React.Component {
    //也可在这里定义每个页面的导航属性，这里的定义会覆盖掉别处的定义
    static navigationOptions = {
        title: 'Page1',
    };
    ...
```

> 第三步：界面跳转

```
export default class HomePage extends React.Component {
    //在这里定义每个页面的导航属性
    static navigationOptions = {
        title: 'Home',
        headerBackTitle:'返回哈哈',//设置返回此页面的返回按钮文案，有长度限制
    }

    render() {
        const {navigation} = this.props;
        return <View style=>
            <Text style={styles.text}>欢迎来到HomePage</Text>
            <Button
                title="Go To Page1"
                onPress={() => {
                    navigation.navigate('Page1', {name: '动态的'});
                }}
            />
            <Button
                title="Go To Page2"
                onPress={() => {
                    navigation.navigate('Page2');
                }}
            />
            <Button
                title="Go To Page3"
                onPress={() => {
                    navigation.navigate('Page3',{ name: 'Devio' });
                }}
            />
        </View>
    }
}
```

> 代码解析：

页面跳转可分为两步：

- 1. 获取navigation：

```
const {navigation} = this.props;
```

- 1. 通过`navigate(routeName, params, action)`进行页面跳转：

```
navigation.navigate('Page2');
navigation.navigate('Page3',{ name: 'Devio' });
```

这里在跳转到`Page3`的时候传递了参数`{ name: 'Devio' }`；

> 第四步：更新页面Params与返回

```
export default class Page3 extends React.Component {
    render() {
        const {navigation} = this.props;
        const {state, setParams} = navigation;
        const {params} = state;
        const showText = params.mode === 'edit' ? '正在编辑' : '编辑完成';
        return <View style=>
            <Text style={styles.text}>欢迎来到Page3</Text>
            <Text style={styles.showText}>{showText}</Text>
            <TextInput
                style={styles.input}
                onChangeText={text=>{
                    setParams({title:text})
                }}
            />
            <Button
                title="Go Back"
                onPress={() => {
                    navigation.goBack();
                }}
            />
        </View>
    }
}
```

> 代码解析：

在上述代码中通过：

```
 <TextInput
    style={styles.input}
    onChangeText={text=>{
        setParams({title:text})
    }}
/>
```

将输入框中内容的变化，通过`setParams({title:text})`更新到页面的标题上，你会看到当输入框中内容发生变化时，标题也会跟着变。

当用户单击`Go Back`按钮时，通过：

```
navigation.goBack();
```

实现了返回上一页；

createBottomTabNavigator-底部导航

相当于iOS里面的TabBarController

createBottomTabNavigator API

> `createBottomTabNavigator(RouteConfigs, BottomTabNavigatorConfig):`

- `RouteConfigs`(必选)：路由配置对象是从路由名称到路由配置的映射，告诉导航器该路由呈现什么。
- `BottomTabNavigatorConfig`(可选)：配置导航器的路由(如：默认首屏，navigationOptions，paths等)样式(如，转场模式mode、头部模式等)。

从createBottomTabNavigator API上可以看出`createBottomTabNavigator`支持通过`RouteConfigs`和`BottomTabNavigatorConfig`两个参数来创建createBottomTabNavigator导航器。

RouteConfigs

RouteConfigs支持三个参数`screen`、`path`以及`navigationOptions`；

- `screen`(必选)：指定一个 React 组件作为屏幕的主要显示内容，当这个组件被TabNavigator加载时，它会被分配一个`navigation` prop。
- `path`(可选)：用来设置支持schema跳转时使用，具体使用会在下文的有关`Schema`章节中讲到；
- `navigationOptions`(可选)：用以配置全局的屏幕导航选项如：title、headerRight、headerLeft等；

BottomTabNavigatorConfig

- tabBarComponent：指定createBottomTabNavigator的TabBar组件，如果不指定在iOS上默认使用TabBarBottom，在Android平台上默认使用TabBarTop。
    - `TabBarBottom`与`TabBarTop`都是`react-navigation`所支持的组件，要自定义TabBar可以重写这两个组件也可以根据需要自己实现一个；
- tabBarOptions: 配置TaBar下文会详细讲解；

- initialRouteName : 默认页面组件，createBottomTabNavigator显示的第一个页面；
- order: 定义tab顺序的routeNames数组。
- paths: 提供routeName到path config的映射，它覆盖routeConfigs中设置的路径。
- backBehavior: 后退按钮是否会导致标签切换到初始tab？ 如果是，则设切换到初始tab，否则什么也不做。 默认为切换到初始tab。

tabBarOptions（tab配置）

- activeTintColor: 设置TabBar选中状态下的标签和图标的颜色；
- inactiveTintColor: 设置TabBar非选中状态下的标签和图标的颜色；
- showIcon: 是否展示图标，默认是false；
- showLabel: 是否展示标签，默认是true；
- upperCaseLabel - 是否使标签大写，默认为true。
- tabStyle: 设置单个tab的样式；
- indicatorStyle: 设置 indicator(tab下面的那条线)的样式；
- labelStyle: 设置TabBar标签的样式；
- iconStyle: 设置图标的样式；
- style: 设置整个TabBar的样式；
- allowFontScaling: 设置TabBar标签是否支持缩放，默认支持；
- safeAreaInset：覆盖的forceInset prop，默认是{ bottom: 'always', top: 'never' }，可选值：top | bottom | left | right ('always' | 'never')；

navigationOptions（屏幕导航选项）

createBottomTabNavigator支持的屏幕导航选项的参数有：

- title: 可以用作headerTitle和tabBarLabel的备选的通用标题。
- tabBarVisible: 显示或隐藏TabBar，默认显示；
- tabBarIcon: 设置TabBar的图标；
- tabBarLabel: 设置TabBar的标签；
- tabBarOnPress: Tab被点击的回调函数，它的参数是一保函一下变量的对象：
  - navigation: navigation prop ；
  - defaultHandler: tab按下的默认处理程序；
- tabBarButtonComponent：React组件，它包装图标和标签并实现onPress。 默认情况下是TouchableWithoutFeedback的一个封装，使其其表现与其它可点击组件相同，tabBarButtonComponent: TouchableOpacity 将使用 TouchableOpacity 来替代；
- tabBarAccessibilityLabel：选项卡按钮的辅助功能标签。 当用户点击标签时，屏幕阅读器会读取这些信息。 如果您没有选项卡的标签，建议设置此项；
- tabBarTestID：用于在测试中找到该选项卡按钮的 ID；

案例

```
export const AppTabNavigator = createBottomTabNavigator({
    Page1: {
        screen: Page1,
        navigationOptions: {
            tabBarLabel: 'Page1',
            tabBarIcon: ({tintColor, focused}) => (
                <Ionicons
                    name={focused ? 'ios-home' : 'ios-home-outline'}
                    size={26}
                    style=
                />
            ),
        }
    },
    Page2: {
        screen: Page2,
        navigationOptions: {
            tabBarLabel: 'Page2',
            tabBarIcon: ({tintColor, focused}) => (
                <Ionicons
                    name={focused ? 'ios-people' : 'ios-people-outline'}
                    size={26}
                    style=
                />
            ),
        }
    },
    Page3: {
        screen: Page3,
        navigationOptions: {
            tabBarLabel: 'Page3',
            tabBarIcon: ({tintColor, focused}) => (
                <Ionicons
                    name={focused ? 'ios-chatboxes' : 'ios-chatboxes-outline'}
                    size={26}
                    style=
                />
            ),
        }
    },
}, {
    tabBarComponent: TabBarComponent,
    tabBarOptions: {
        activeTintColor: Platform.OS === 'ios' ? '#e91e63' : '#fff',
    }
});
```

在上述代码中使用了`react-native-vector-icons`的矢量图标作为Tab的显示图标，tabBarIcon接收一个React 组件，大家可以根据需要进行定制：

- tintColor: 当前状态下Tab的颜色；
- focused: Tab是否被选中；

react-native-vector-icons

安装

```
yarn add react-native-vector-icons

react-native link react-native-vector-icons
```

图标库地址：

https://oblador.github.io/react-native-vector-icons/

createMaterialTopTabNavigator-顶部导航

createMaterialTopTabNavigator API

> `createMaterialTopTabNavigator(RouteConfigs, TabNavigatorConfig):`

- `RouteConfigs`(必选)：路由配置对象是从路由名称到路由配置的映射，告诉导航器该路由呈现什么。
- `TabNavigatorConfig`(可选)：配置导航器的路由(如：默认首屏，navigationOptions，paths等)样式(如，转场模式mode、头部模式等)。

从createMaterialTopTabNavigator API上可以看出`createMaterialTopTabNavigator`支持通过`RouteConfigs`和 `TabNavigatorConfig`两个参数来创建createMaterialTopTabNavigator导航器。

RouteConfigs

RouteConfigs支持三个参数`screen`、`path`以及`navigationOptions`；

- `screen`(必选)：指定一个 React 组件作为屏幕的主要显示内容，当这个组件被TabNavigator加载时，它会被分配一个`navigation` prop。
- `path`(可选)：用来设置支持schema跳转时使用，具体使用会在下文的有关`Schema`章节中讲到；
- `navigationOptions`(可选)：用以配置全局的屏幕导航选项如：title、headerRight、headerLeft等；

TabNavigatorConfig

- tabBarComponent：指定TabNavigator的TabBar组件；
- tabBarPosition: 用于指定TabBar的显示位置，支持'top' 与 'bottom'两种方式；
- swipeEnabled : 是否可以左右滑动切换tab；
- lazy - 默认值是 false。如果是true，Tab 页只会在被选中或滑动到该页时被渲染。当为 false 时，所有的 Tab 页都将直接被渲染；（可以轻松实现多Tab 页面的懒加载）；
- optimizationsEnabled -是否将 Tab 页嵌套在到 中。如果是，一旦该 Tab 页失去焦点，将被移出当前页面, 从而提高内存使用率。
- animationEnabled : 切换页面时是否有动画效果。
- initialLayout : 包含初始高度和宽度的可选对象可以被传递以防止react-native-tab-view呈现中的一个帧延迟；
- tabBarOptions: 配置TaBar下文会详细讲解；
- initialRouteName : 默认页面组件，TabNavigator显示的第一个页面；
- order: 定义tab顺序的routeNames数组。
- paths: 提供routeName到path config的映射，它覆盖routeConfigs中设置的路径。
- backBehavior: 后退按钮是否会导致标签切换到初始tab？ 如果是，则设切换到初始tab，否则什么也不做。 默认为切换到初始tab。
-

tabBarOptions（tab配置）

- activeTintColor: 设置TabBar选中状态下的标签和图标的颜色；
- inactiveTintColor: 设置TabBar非选中状态下的标签和图标的颜色；
- showIcon: 是否展示图标，默认是false；
- showLabel: 是否展示标签，默认是true；
- upperCaseLabel - 是否使标签大写，默认为true。
- tabStyle: 设置单个tab的样式；
- indicatorStyle: 设置 indicator(tab下面的那条线)的样式；
- labelStyle: 设置TabBar标签的样式；
- iconStyle: 设置图标的样式；
- style: 设置整个TabBar的样式；
- allowFontScaling: 设置TabBar标签是否支持缩放，默认支持；
- pressColor -Color for material ripple（仅支持 Android >= 5.0；
- pressOpacity -按下标签时的不透明度（支持 iOS 和 Android < 5.0）；
- scrollEnabled -是否支持 选项卡滚动

navigationOptions（屏幕导航选项）

createMaterialTopTabNavigator支持的屏幕导航选项的参数有：

- title: 可以用作headerTitle和tabBarLabel的备选的通用标题。
- swipeEnabled：是否允许tab之间的滑动切换，默认允许；
- tabBarIcon: 设置TabBar的图标；
- tabBarLabel: 设置TabBar的标签；

- tabBarOnPress: Tab被点击的回调函数，它的参数是一保函一下变量的对象：
  - navigation：页面的 navigation props
  - defaultHandler: tab press 的默认 handler
- tabBarAccessibilityLabel：选项卡按钮的辅助功能标签。 当用户点击标签时，屏幕阅读器会读取这些信息。 如果您没有选项卡的标签，建议设置此项；
- tabBarTestID：用于在测试中找到该选项卡按钮的 ID；

createDrawerNavigator-抽屉导航

createDrawerNavigator API

```
createDrawerNavigator(RouteConfigs, DrawerNavigatorConfig):
```

- `RouteConfigs`(必选)：路由配置对象是从路由名称到路由配置的映射，告诉导航器该路由呈现什么。
- `DrawerNavigatorConfig`(可选)：配置导航器的路由(如：默认首屏，navigationOptions，paths等)样式(如，转场模式mode、头部模式等)。

从createDrawerNavigator API上可以看出`createDrawerNavigator`支持通过`RouteConfigs`和`DrawerNavigatorConfig`两个参数来创建createDrawerNavigator导航器。

RouteConfigs

RouteConfigs支持三个参数`screen`、`path`以及`navigationOptions`；

- `screen`(必选)：指定一个 React 组件作为屏幕的主要显示内容，当这个组件被DrawerNavigator加载时，它会被分配一个`navigation` prop。
- `path`(可选)：用来设置支持schema跳转时使用，具体使用会在下文的有关`Schema`章节中讲到；
- `navigationOptions`(可选)：用以配置全局的屏幕导航选项如：title、headerRight、headerLeft等；

DrawerNavigatorConfig

- drawerWidth: 设置侧边菜单的宽度；
- drawerPosition: 设置侧边菜单的位置，支持'left'、 'right'，默认是'left'；
- contentComponent: 用于呈现抽屉导航器内容的组件，例如导航项。接收抽屉导航器的navigation 属性 。默认为DrawerItems。有关详细信息，请参阅下文；
- contentOptions: 配置抽屉导航器内容，见下文;
- useNativeAnimations: 是否启用Native动画，默认启用；
- drawerBackgroundColor: 侧边菜单的背景；
- initialRouteName: 初始化哪个界面为根界面，如果不配置，默认使用RouteConfigs中的第一个页面当做根界面；
- order: drawer排序，默认使用配置路由的顺序；
- paths: 提供routeName到path config的映射，它覆盖routeConfigs中设置的路径。

- backBehavior: 后退按钮是否会导致标签切换到初始drawer？ 如果是，则设切换到初始drawer，否则什么也不做。 默认为切换到初始drawer。

```
import React,{Component} from 'react';
import {
  AppRegistry,
  StyleSheet,
  Text,
  View,
  Button
} from 'react-native';

import { createDrawerNavigator } from 'react-navigation';

import Ionicons from 'react-native-vector-icons/Ionicons';


const HomeScreen=({navigation})=>(
        <View style={{ flex: 1, alignItems: 'center', justifyContent: 'center'
}}>
            <Text>HomeScreen</Text>
            <Button
                 onPress={()=>(navigation.toggleDrawer())}
                 title="Open Drawer"
            ></Button>
        </View>
    );

const ProfileScreen=()=>(
        <View style={{flex: 1, alignItems: 'center', justifyContent: 'center'}}>
<Text>ProfileScreen</Text></View>
    );

const RootDrawer =createDrawerNavigator({
    Home:{
        screen:HomeScreen,
        navigationOptions:{
            drawerLabel:'MyHome',
            drawerIcon:({tintColor, focused })=>(
                <Ionicons
                      name={'ios-home'}
                      size={20}
                      style={{ color: tintColor }}
                ></Ionicons>
                ),
        },
    },
```

```
    Profile:{
        screen:ProfileScreen,
        navigationOptions:{
            drawerLabel:'MyProfile',
            drawerIcon:({tintColor})=>(
                <Ionicons
                    name={'ios-person'}
                    size={20}
                    style={{color:tintColor}}
                ></Ionicons>
                ),
        },
    }
});

export default RootDrawer;
```

自定义侧边栏(contentComponent)

DrawerNavigator有个默认的带滚动的侧边栏，你也可以通过重写这个侧边栏组件来自定义侧边栏：

```
contentComponent:(props) => (
    <ScrollView style=>
        <SafeAreaView forceInset=>
            <DrawerItems {...props} />
        </SafeAreaView>
    </ScrollView>
)
```

DrawerItems的contentOptions

contentOptions主要配置侧滑栏item的属性，只对DrawerItems，例如我们刚才写的例子，就可以通过这个属性来配置颜色，背景色等。其主要属性有：

- items: 路由数组，如果要修改路由可以可以修改或覆盖它；
- activeItemKey: 定义当前选中的页面的key；
- activeTintColor: 选中item状态的文字颜色；
- activeBackgroundColor: 选中item的背景色；
- inactiveTintColor: 未选中item状态的文字颜色；
- inactiveBackgroundColor: 未选中item的背景色；
- onItemPress: 选中item的回调，这个参数属性为函数，会将当前路由回调过去；
- itemsContainerStyle: 定义itemitem容器的样式；
- itemStyle: 定义item的样式；
- labelStyle: 定义item文字的样式；
- iconContainerStyle: 定义item图标容器的样式；
- activeLabelStyle：选中状态下文本样式；
- inactiveLabelStyle：非选中状态下文本样式；

- iconContainerStyle： 用于设置图标容器的样式。

> eg:

```
contentOptions: {
  activeTintColor: '#e91e63',
  itemsContainerStyle: {
    marginVertical: 0,
  },
  iconContainerStyle: {
    opacity: 1
  }
}
```

navigationOptions（屏幕导航选项）

DrawerNavigator支持的屏幕导航选项的参数有：

- title: 可以用作headerTitle和drawerLabel的备选的通用标题。
- drawerLabel：侧滑标题；
- drawerIcon：侧滑的标题图标，这里会回传两个参数：
    - `{focused: boolean, tintColor: string}`
        - focused: 表示是否是选中状态；
        - tintColor: 表示选中的颜色；
- drawerLockMode：指定抽屉的锁定模式。 这也可以通过在顶级路由器上使用 screenProps.drawerLockMode 动态更新。

侧边栏操作(打开、关闭、切换)

侧边栏支持以下几种操作方式：

```
navigation.openDrawer();
navigation.closeDrawer();
navigation.toggleDrawer();
//或
navigation.dispatch(DrawerActions.openDrawer());
navigation.dispatch(DrawerActions.closeDrawer());
navigation.dispatch(DrawerActions.toggleDrawer());
```

其中路由名`openDrawer`对应这打开侧边栏的操作，`DrawerClose`对应关闭侧边栏的操作，`toggleDrawer`对应切换侧边栏操作，要进行这些操作我么还需要一个`navigation`，`navigation`可以从props中获取；

- 打开侧边栏：`navigation.openDrawer();`
- 关闭侧边栏：`navigation.closeDrawer();`
- 切换侧边栏：`navigation.toggleDrawer();`

createSwitchNavigator-开关导航

SwitchNavigator 的用途是一次只显示一个页面。 默认情况下，它不处理返回操作，并在你切换时将路由重置为默认状态。

createSwitchNavigator API

> `createSwitchNavigator(RouteConfigs, SwitchNavigatorConfig);`

- `RouteConfigs`(必选，同createStackNavigator的RouteConfigs)：路由配置对象是从路由名称到路由配置的映射，告诉导航器该路由呈现什么。
- `SwitchNavigatorConfig`(可选)：配置导航器的路由;

SwitchNavigatorConfig

几个被传递到底层路由以修改导航逻辑的选项：

- initialRouteName -第一次加载时初始选项卡路由的 routeName。
- resetOnBlur - 切换离开屏幕时，重置所有嵌套导航器的状态。 默认为true。
- paths - 提供 routeName 到 path 配置的映射, 它重写 routeConfigs 中设置的路径。
- backBehavior - 控制 “返回” 按钮是否会导致 Tab 页切换到初始 Tab 页? 如果是, 设置为 initialRoute, 否则 none。 默认为none行为。

```
const AppStack = createStackNavigator({
    Home: {
        screen: HomePage
    },
    Page1: {
        screen: Page1
    }
});
const AuthStack = createStackNavigator({
    Login: {
        screen: Login
    },
},{
```

```
    navigationOptions: {
        // header: null,// 可以通过将header设为null 来禁用StackNavigator的Navigation
Bar
    }
});

export default createSwitchNavigator(
    {
        Auth: AuthStack,
        App: AppStack,
    },
    {
        initialRouteName: 'Auth',
    }
);
```

课程回顾

- React Navigation介绍
- React Navigation概念与属性介绍
- 核心导航器的学习与使用