React Navigation 簡介(v4)

> v4 版發佈於 2019/9，與v3版本的差異，主要是分離出不同的包針對新的版本的需求，詳情參考：官網blog

安裝

在你的 React Native 專案中安裝 react-navigation 這個包

```
yarn add react-navigation
# or with npm
# npm install --save react-navigation
```

以下針對不同的導航器(Navigator)來安裝：

```
# stack
yarn add react-navigation-stack @react-native-community/masked-view

# tab
yarn add react-navigation-tabs
yarn add react-navigation-material-bottom-tabs react-native-paper

# Drawer
yarn add react-navigation-drawer
```

AppContainer

管理你的應用程式狀態與連結最上層 navigator 到應用程式環境的容器。 使用 createAppContainer 方法來建立。

/App.js

```
import { createAppContainer } from 'react-navigation'
import { createStackNavigator } from 'react-navigation-stack'

// 建立Stack堆疊
const AppNavigator = createStackNavigator(...)

// 建立應用程式容器
const AppContainer = createAppContainer(AppNavigator)

// 直接導出
export default AppContainer
```

SwitchNavigator

> 註：SwitchNavigator為內建在 `react-navigation` 的模組之中

SwitchNavigator 的目的在於，一次只顯示"切換"到的一個視窗(show one screen at a time)，因此並不會處理返回(back)的動作，而且當你進行切換時，會重置路由到預設狀態。也就是在"切換"時，切到的視窗會進行 mount，而其它的視窗將會進行 unmount。

SwitchNavigator 用於認証用的(最上層的 Navigator，次於 AppContainer)的切換，程式碼如下:

```
import { createAppContainer, createSwitchNavigator } from 'react-navigation';
import { createStackNavigator } from 'react-navigation-stack';

// Implementation of HomeScreen, OtherScreen, SignInScreen, AuthLoadingScreen
// goes here.

const AppStack = createStackNavigator({ Home: HomeScreen, Other: OtherScreen })
const AuthStack = createStackNavigator({ SignIn: SignInScreen })
```

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

完整的程式碼參考: Authentication flows

Navigator 種類

除了上述用於特殊目的的 SwitchNavigator 外，共有三種經常使用的 Navigator

- StackNavigator: 當使用者觸碰一個連結，一個新的視窗會被移到舊的視窗上面。只有這個 Navigator 的動作可以加上切換的動畫。
- TabNavigator: 使用者使用畫面最上方或最下方的資訊標籤(tab)來移動到不同的視窗。
- DrawerNavigator: 用一個會滑出的區域(抽屜，drawer)，其中帶有可以移動到不同視窗的連結。

StackNavigator

StackNavigator 依靠 createStackNavigator 方法建立，一個最簡單的範例如下:

```
import React from 'react';
import { Button, View, Text } from 'react-native';
import { createAppContainer } from 'react-navigation';
import { createStackNavigator } from 'react-navigation-stack';

class HomeScreen extends React.Component {
  render() {
    return (
      <View style={{ flex: 1, alignItems: 'center', justifyContent: 'center' }}>
        <Text>Home Screen</Text>
        <Button
          title="Go to Details"
          onPress={() => this.props.navigation.navigate('Details')}
        />
      </View>
    )
  }
}

class DetailsScreen extends React.Component {
  render() {
    return (
      <View style={{ flex: 1, alignItems: 'center', justifyContent: 'center' }}>
        <Text>Details Screen</Text>
      <Button
          title="Go to Details"
          onPress={() => this.props.navigation.goBack()}
        />
      </View>
    )
  }
}

const RootStack = createStackNavigator(
  {
    Home: HomeScreen,
    Details: DetailsScreen,
  },
  {
```

```
    initialRouteName: 'Home',
  }
)

const AppContainer = createAppContainer(RootStack)

export default class App extends React.Component {
  render() {
    return <AppContainer />
  }
}
```

標頭(Header)

StackNavigator 可以產生標頭(Header)，標頭的定義是在每個視窗(screen)的元件定義中，例如以下的例子:

```
class HomeScreen extends React.Component {
  static navigationOptions = ({ navigation }) => {
    return {
      headerTitle: () => <LogoTitle />,
      headerRight: () => (
        <Button
          onPress={navigation.getParam('increaseCount')}
          title="+1"
          color="#fff"
        />
      ),
    };
  };

  componentDidMount() {
    this.props.navigation.setParams({ increaseCount: this._increaseCount });
  }

  state = {
    count: 0,
  };

  _increaseCount = () => {
    this.setState({ count: this.state.count + 1 });
  };

  /* later in the render function we display the count */
}
```

上面的 `navigationOptions` ，可以在 createStackNavigator 方法中進行覆蓋，例如:

```
createStackNavigator({
  A: {
    screen: AScreen,
    navigationOptions: () => ({
      title: `A`,
      headerBackTitle: null,
    }),
  },
  B: {
    screen: BScreen,
    navigationOptions: () => ({
      title: `B`,
    }),
  },
})
```

返回(back)按鈕(或連結)，createStackNavigator 會自動產生，唯一能自行定義的是返回(back)按鈕(或連結)的字詞，使用的是 headerBackTitle 與 headerTruncatedBackTitle 兩個屬性值。如下的範例:

```
createStackNavigator({
  A: {
```

```
    screen: AScreen,
    navigationOptions: () => ({
      title: `A`,
      headerBackTitle: 'A much too long text for back button from B to A',
      headerTruncatedBackTitle: `to A`,
    }),
  },
  B: {
    screen: BScreen,
    navigationOptions: () => ({
      title: `B`,
    }),
  },
})
```

> 註: iOS 與 Android 的返回樣式會不太一樣這是正常的，React Navigation 使用的是各平台原本的返回功能。

defaultNavigationOptions 可以讓所有的視窗共享同樣的設定，通常用於定義絲一的樣式風格使用。範例如下:

```
const RootStack = createStackNavigator(
  {
    Home: HomeScreen,
    Details: DetailsScreen,
  },
  {
    initialRouteName: 'Home',
    /* The header config from HomeScreen is now here */
    defaultNavigationOptions: {
      headerStyle: {
        backgroundColor: '#f4511e',
      },
      headerTintColor: '#fff',
      headerTitleStyle: {
        fontWeight: 'bold',
      },
    },
  }
)
```

navigationOptions 的覆蓋方式

以下依次序覆蓋，最上面的是預設的設定值，下面覆蓋上面的。

- defaultNavigationOptions
- 每個視窗中定義的 navigationOptions
- 在 createStackNavigator 方法中針對每個視窗定義的 navigationOptions

在不同的視窗中移動

> 參考 navigation 屬性章節

在不同的視窗傳遞參數

傳遞參數主要要使用以下兩個部份:

- 傳送方 - this.props.navigation.navigate('要傳遞過去的視窗名稱', {參數物件值})
- 接受方 - this.props.navigation.getParam('參數名稱', '預設值')

範例如下:

```
class HomeScreen extends React.Component {
  render() {
    return (
      <View style={{ flex: 1, alignItems: 'center', justifyContent: 'center' }}>
        <Text>Home Screen</Text>
        <Button
```

```
          title="Go to Details"
          onPress={() => {
            /* 1. Navigate to the Details route with params */
            this.props.navigation.navigate('Details', {
              itemId: 86,
              otherParam: 'anything you want here',
            })
          }}
        />
      </View>
    )
  }
}

class DetailsScreen extends React.Component {
  render() {
    /* 2. Get the param, provide a fallback value if not available */
    const { navigation } = this.props
    const itemId = navigation.getParam('itemId', 'NO-ID')
    const otherParam = navigation.getParam('otherParam', 'some default value')

    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={() =>
            this.props.navigation.push('Details', {
              itemId: Math.floor(Math.random() * 100),
            })
          }
        />
        <Button
          title="Go to Home"
          onPress={() => this.props.navigation.navigate('Home')}
        />
        <Button
          title="Go back"
          onPress={() => this.props.navigation.goBack()}
        />
      </View>
    )
  }
}
```

> 註: navigation.navigate 方法有很多細部的用法，可以參考API 頁面中的更多說明

TabNavigator

react-navigation 提供了 createBottomTabNavigator 與 createMaterialTopTabNavigator 可以建立 TabNavigator。

另外 createMaterialBottomTabNavigator 需要額外安裝以下的模組套件:

```
npm install react-navigation-material-bottom-tabs react-native-paper
```

一個最簡單的範例如下:

```
import React from 'react'
import { Text, View } from 'react-native'
import { createAppContainer } from 'react-navigation'
import { createBottomTabNavigator } from 'react-navigation-tabs';

class HomeScreen extends React.Component {
  render() {
    return (
      <View style={{ flex: 1, justifyContent: 'center', alignItems: 'center' }}>
        <Text>Home!</Text>
```

```
      </View>
    )
  }
}

class SettingsScreen extends React.Component {
  render() {
    return (
      <View style={{ flex: 1, justifyContent: 'center', alignItems: 'center' }}>
        <Text>Settings!</Text>
      </View>
    )
  }
}

const TabNavigator = createBottomTabNavigator({
  Home: HomeScreen,
  Settings: SettingsScreen,
})

export default createAppContainer(TabNavigator)
```

每個 Tab 的圖示(Icon)、標記文字(label)、顏色，設置的方式與 StackNavigator 類似(也是可以覆蓋，不過通常會設定在一處)，顏色等樣式則會統一設定一處而已。如下面的範例:

```
const TabNavigator = createMaterialBottomTabNavigator(
  {
    Home: {
      screen: HomeStack,
      navigationOptions: {
        tabBarLabel: '首頁',
        tabBarIcon: ({ tintColor, focused }) => (
          <Icon size={24} name="md-home" style={{ color: tintColor }} />
        ),
        gesturesEnabled: false,
      },
    },
    Map: {
      screen: Map,
      navigationOptions: {
        tabBarLabel: '地圖',
        tabBarIcon: ({ tintColor, focused }) => (
          <Icon size={24} name="md-pin" style={{ color: tintColor }} />
        ),
        gesturesEnabled: false,
      },
    },
  },
  {
    shifting: true, //控制圖示文字特效(點到時要不要出現文字)
    initialRouteName: 'Home',
    activeColor: customColor.activeColor,
    inactiveColor: customColor.inactiveColor,
    barStyle: { backgroundColor: customColor.backgroundColor },
  }
)
```

在不同的 Tab 視窗中切換，則是使用 `this.props.navigation.navigate` 方法，如下範例:

```
import { Button, Text, View } from 'react-native'

class HomeScreen extends React.Component {
  render() {
    return (
      <View style={{ flex: 1, justifyContent: 'center', alignItems: 'center' }}>
        <Text>Home!</Text>
        <Button
          title="Go to Settings"
          onPress={() => this.props.navigation.navigate('Settings')}
        />
      </View>
```

```
    )
  }
}

class SettingsScreen extends React.Component {
  render() {
    return (
      <View style={{ flex: 1, justifyContent: 'center', alignItems: 'center' }}>
        <Text>Settings!</Text>
        <Button
          title="Go to Home"
          onPress={() => this.props.navigation.navigate('Home')}
        />
      </View>
    )
  }
}
```

在一個 Tab 中還有許多 Stack 視窗

原理就是先用 createStackNavigator 組合好，然後再放到 createBottomTabNavigator(或其它 Tab 建立方法)建立而已，範例如下:

```
import {
  createAppContainer,
} from 'react-navigation';
import { createBottomTabNavigator } from 'react-navigation-tabs';
import { createStackNavigator } from 'react-navigation-stack';


class DetailsScreen extends React.Component {
  render() {
    return (
      <View style={{ flex: 1, justifyContent: 'center', alignItems: 'center' }}>
        <Text>Details!</Text>
      </View>
    )
  }
}

class HomeScreen extends React.Component {
  render() {
    return (
      <View style={{ flex: 1, justifyContent: 'center', alignItems: 'center' }}>
        {/* other code from before here */}
        <Button
          title="Go to Details"
          onPress={() => this.props.navigation.navigate('Details')}
        />
      </View>
    )
  }
}

class SettingsScreen extends React.Component {
  render() {
    return (
      <View style={{ flex: 1, justifyContent: 'center', alignItems: 'center' }}>
        {/* other code from before here */}
        <Button
          title="Go to Details"
          onPress={() => this.props.navigation.navigate('Details')}
        />
      </View>
    )
  }
}

const HomeStack = createStackNavigator({
  Home: HomeScreen,
  Details: DetailsScreen,
})
```

```
const SettingsStack = createStackNavigator({
  Settings: SettingsScreen,
  Details: DetailsScreen,
})

export default createAppContainer(
  createBottomTabNavigator(
    {
      Home: HomeStack,
      Settings: SettingsStack,
    },
    {
      /* Other configuration remains unchanged */
    }
  )
)
```

如果需要在某個 Tab 視窗中的子 Stack 視窗隱藏 Tabbar 的話，可以用以下的程式碼範例:

```
const FeedStack = createStackNavigator({
  FeedHome: FeedScreen,
  Details: DetailsScreen,
})

const TabNavigator = createBottomTabNavigator({
  Feed: FeedStack,
  Profile: ProfileScreen,
})

const AppNavigator = createSwitchNavigator({
  Auth: AuthScreen,
  Home: TabNavigator,
})

FeedStack.navigationOptions = ({ navigation }) => {
  let tabBarVisible = true
  if (navigation.state.index > 0) {
    tabBarVisible = false
  }

  return {
    tabBarVisible,
  }
}
```

另一種作法是把 Tab 視窗組合到某個上層的 Stack 視窗中:

```
const FeedStack = createStackNavigator({
  FeedHome: FeedScreen,
  /* any other route you want to render under the tab bar */
});

const TabNavigator = createBottomTabNavigator({
  Feed: FeedStack,
  Profile: ProfileScreen,
});

const HomeStack = createStackNavigator({
  Tabs: TabNavigator,
  Details: DetailsScreen,
  /* any other route you want to render above the tab bar */
});

const AppNavigator = createSwitchNavigator({
  Auth: AuthScreen,
  Home: HomeStack,
});)
```

DrawerNavigator

使用 createDrawerNavigator 來建立 DrawerNavigator，範例如下:

```
import { createDrawerNavigator } from 'react-navigation-drawer';

class MyHomeScreen extends React.Component {
  static navigationOptions = {
    drawerLabel: 'Home',
    drawerIcon: ({ tintColor }) => (
      <Image
        source={require('./chats-icon.png')}
        style={[styles.icon, { tintColor: tintColor }]}
      />
    ),
  }

  render() {
    return (
      <Button
        onPress={() => this.props.navigation.navigate('Notifications')}
        title="Go to notifications"
      />
    )
  }
}

class MyNotificationsScreen extends React.Component {
  static navigationOptions = {
    drawerLabel: 'Notifications',
    drawerIcon: ({ tintColor }) => (
      <Image
        source={require('./notif-icon.png')}
        style={[styles.icon, { tintColor: tintColor }]}
      />
    ),
  }

  render() {
    return (
      <Button
        onPress={() => this.props.navigation.goBack()}
        title="Go back home"
      />
    )
  }
}

const styles = StyleSheet.create({
  icon: {
    width: 24,
    height: 24,
  },
})

const MyDrawerNavigator = createDrawerNavigator({
  Home: {
    screen: MyHomeScreen,
  },
  Notifications: {
    screen: MyNotificationsScreen,
  },
})

const MyApp = createAppContainer(MyDrawerNavigator)
```

開啟或關閉 drawer 的語法:

```
this.props.navigation.openDrawer()
this.props.navigation.closeDrawer()
```

切換(開變關、關變開)drawer 的語法:

```
this.props.navigation.toggleDrawer()
```

取得目前的 drawer 是開或關的狀態:

```
const parent = this.props.navigation.dangerouslyGetParent()
const isDrawerOpen = parent && parent.state && parent.state.isDrawerOpen
```

Drawer 的樣式、動畫均可以自訂，請參考這裡的API 頁面說明。

補充 - navigation 屬性

props.navigation 是在所有包含在應用程式中的視窗(screen)都會自動得到的屬性值，提供了各種方便的方法與API，可以移動，在不同的視窗中傳遞參數等等。

- navigate - 移到某個視窗
- goBack - 關閉目前的視窗，返回移動
- addListener - 訂閱導覽生命周期的更新
- isFocused - 回傳true如果是被聚焦狀態時
- state - 目前的狀態/路由
- setParams - 設定(更動)路由器的參數
- getParam - 獲取特定的路由器的參數
- dispatch - 送一個動作到路由器
- dangerouslyGetParent - 回傳父導航器的函式

以下為堆疊(stack)導航專有的函式：

- `push` - 推入(push)一個新路由到堆疊(stack)
- `pop` - 在堆疊中返回
- `popToTop` - 到堆疊的最上層視窗
- `replace` - 以新的視窗取到目前的路由
- `reset` - 重設導航器
- `dismiss` - 清除目前的堆疊

以下為抽屜(drawer)導航專有的函式：

- openDrawer - 開啟
- closeDrawer - 關閉
- toggleDrawer - 切換(開到關，關到開)

參考：navigation-prop