react-native-router-flux使用技巧（API篇）

可能很多人之前就用过这个三方，也可能有些人听过没用过。在这系列文章中，我将重点推荐使用该三方，有这样几方面原因：

> 1. 流畅度：在最新的V4版本中是基于`react-navigation`实现的，如果使用过`react-navigation`那么对它的流畅度应该很有信心；
> 2. 功能性：`react-navigation`虽然是官方推荐的导航库，但其库内部提供的，可以直接使用的功能很简单，有些还需要配合redux来实现需要的功能。而`react-native-router-flux`中基于`react-navigation`实现了`popTo`，`refresh`，`replace`等经常用到的功能，在下面的翻译中有详细说明；
> 3. 更新维护：这点上我很佩服这个库的作者，这个库从V1更新到V4，一直未背离作者的初衷，从未停止更新，一直再优化`react-native`的导航，而且作者最让我佩服的一点是，他好像将`react-navigation`的`Issues`全都翻看过，如果有机会，可以看一下CHANGELOG.md，hui

Available imports

- `Router`
- `Scene`
- `Tabs`
- `Tabbed Scene`
- `Drawer`
- `Modal`
- `Lightbox`
- `Actions`
- `ActionConst`

Router:

Property	Type	Default	Descript
children		required	页面根组件

`wrapBy`	`Function`		允许集成诸如Redux（connect）和Mob
`sceneStyle`	`Style`		适用于所有场景的Style（可选）
`backAndroidHandler`	`Function`		允许在Android中自定义控制返回按钮（ ckHandler

Scene:

此路由器的最重要的组件， 所有 `<Scene>` 组件必须要有一个唯一的 `key` 。父节点 `<Scene>` 不能将 `component` 作为 `prop` ，因为它将作为其子节点的组件。

Property	Type	Default	
`key`	`string`	`required`	将用于标识
`component`	`React.Component`	`semi-required`	要显示的组
`back`	`boolean`	`false`	如果是 `tru`
`backButtonImage`	`string`		设置返回按
`backButtonTintColor`	`string`		自定义后退
`init`	`boolean`	`false`	如果是 `tru`
`clone`	`boolean`	`false`	标有 `clone`
`contentComponent`	`React.Component`		用于呈现抽
`drawer`	`boolean`	`false`	载入Drawe
`failure`	`Function`		如果 `on` 返
`backTitle`	`string`		指定场景的
`headerMode`	`string`	`float`	指定标题应
`hideNavBar`	`boolean`	`false`	隐藏导航栏
`hideTabBar`	`boolean`	`false`	隐藏标签栏
`hideBackImage`	`boolean`	`false`	隐藏返回图
`initial`	`boolean`	`false`	设置为 `tru`

`leftButtonImage`	`Image`		替换左侧按
`leftButtonTextStyle`	`Style`		左侧按钮的
`modal`	`boolean`	`false`	将场景容器
`navBar`	`React.Component`		可以使用自
`navBarButtonColor`	`string`		设置导航栏
`navigationBarStyle`	`Style`		导航栏的样
`navigationBarTitleImage`	`Object`		导航栏中的
`navigationBarTitleImageStyle`	`object`		`navigatio`
`navTransparent`	`boolean`	`false`	导航栏是否
`on`	`Function`		又名 `onEn`
`onEnter`	`Function`		当 `Scene` 要
`onExit`	`Function`		当 `Scene` 要
`onLeft`	`Function`		当导航栏左
`onRight`	`Function`		当导航栏右
`renderTitle`	`React.Component`		使用React
`renderLeftButton`	`React.Component`		使用React
`renderRightButton`	`React.Component`		使用React
`renderBackButton`	`React.Component`		使用React
`rightButtonImage`	`Image`		设置右侧按
`rightButtonTextStyle`	`Style`		右侧按钮文
`success`	`Function`		如果 `on` 返
`tabs`	`boolean`	`false`	将子场景加
`title`	`string`		要显示在导
`titleStyle`	`Style`		title的样式
`type`	`string`	`push`	可选的导航
all other props			此处未列出

Tabs (`<Tabs>` or `<Scene tabs>`)

标签栏组件。
你可以使用 `<Scene>` 中的所有 `props` 来作为 `<Tabs>` 的属性。 如果要使用该组件需要设置 `<Scene tabs={true}>` 。

Property	Type	Default	
`wrap`	`boolean`	`true`	自动使用自己的导航栏包装
`activeBackgroundColor`	`string`		指定焦点的选项卡的选中背
`activeTintColor`	`string`		指定标签栏图标的选中色调
`inactiveBackgroundColor`	`string`		指定非焦点的选项卡的未选
`inactiveTintColor`	`string`		指定标签栏图标的未选中色
`labelStyle`	`object`		设置tabbar上文字的样式
`lazy`	`boolean`	`false`	在选项卡处于活动状态之前
`tabBarComponent`	`React.Component`		使用React组件以自定义标
`tabBarPosition`	`string`		指定标签栏位置。iOS上默
`tabBarStyle`	`object`		标签栏演示
`tabStyle`	`object`		单个选项卡的样式
`showLabel`	`boolean`	`true`	是否显示标签栏文字
`swipeEnabled`	`boolean`	`true`	是否可以滑动选项卡。
`tabBarOnPress`	`function`		自定义tabbar点击事件
`backToInitial`	`boolean`	`false`	如果选项卡图标被点击，返

Stack (`<Stack>`)

将场景组合在一起的组件，用于自己的基于堆栈实现的导航。使用它将为此堆栈创建一个单独的 navigator，因此，除非您添加 `hideNavBar` ，否则将会出现两个导航条。

Tab Scene (child `<Scene>` within `Tabs`)

用于实现`Tabs`的效果展示，可以自定义icon和label。

Property	Type	Default	Description
`icon`	`component`	`undefined`	作为选项卡图标放置的RN组件
`tabBarLabel`	`string`		tabbar上的文字

Drawer (`<Drawer>` or `<Scene drawer>`)

用于实现抽屉的效果，如果要使用该组件需要设置`<drawer tabs={true}>`。

Property	Type	Default	De
`drawerImage`	`Image`		替换抽屉'hamburger'图标，你必须
`drawerIcon`	`React.Component`		用于抽屉'hamburger'图标的任意组
`hideDrawerButton`	`boolean`	`false`	是否显示`drawerImage`或者`drawe`
`drawerPosition`	`string`		抽屉是在右边还是左边。可选属性
`drawerWidth`	`number`		抽屉的宽度（以像素为单位）（可

Modals (`<Modal>` or `<Scene modal>`)

想要实现模态，您必须将其`<Modal>`作为您`Router`的根场景。在`Modal`将正常呈现第一个场景（应该是你真正的根场景），它将渲染第一个元素作为正常场景，其他所有元素作为弹出窗口（当它们 被push）。

示例：在下面的示例中，`root`场景嵌套在`<Modal>`中，因为它是第一个嵌套`Scene`，所以它将正常呈现。如果要`push`到`statusModal`，`errorModal`或者`loginModal`，他们将呈现为`Modal`，默认情况下会从屏幕底部向上弹出。重要的是要注意，目前`Modal`不允许透明的背景。

```
//... import components
<Router>
  <Modal>
    <Scene key="root">
```

```
      <Scene key="screen1" initial={true} component={Screen1} />
      <Scene key="screen2" component={Screen2} />
    </Scene>
    <Scene key="statusModal" component={StatusModal} />
    <Scene key="errorModal" component={ErrorModal} />
    <Scene key="loginModal" component={LoginModal} />
  </Modal>
</Router>
```

Lightbox (`<Lightbox>`)

`Lightbox`是用于将组件渲染在当前组件上`Scene`的组件 。与`Modal`不同，它将允许调整大小和背景的透明度。

示例：
在下面的示例中， `root`场景嵌套在中，因为它是第一个嵌套`Scene`，所以它将正常呈现。如果要`push`到`loginLightbox`，他们将呈现为`Lightbox`，默认情况下将放置在当前场景的顶部，允许透明的背景。

```
//... import components
<Router>
  <Lightbox>
    <Scene key="root">
      <Scene key="screen1" initial={true} component={Screen1} />
      <Scene key="screen2" component={Screen2} />
    </Scene>

    {/* Lightbox components will lay over the screen, allowing transparency*/}
    <Scene key="loginLightbox" component={loginLightbox} />
  </Lightbox>
</Router>
```

Actions

该对象的主要工具是为您的应用程序提供导航功能。 假设您的`Router`和`Scenes`配置正确，请使用下列属性在场景之间导航。 有些提供添加的功能，将React道具传递到导航场景。

这些可以直接使用，例如， `Actions.pop()`将在源代码中实现的操作，或者，您可以在场景类型中设置这些常量，当您执行`Actions.main()`时，它将根据您的场景类型或默认值来执行动作。

Property	Type	Parameters	
`[key]`	`Function`	`Object`	`Actions` 将'自动'使用路由器中的场景 `key`
`currentScene`	`String`		返回当前活动的场景
`jump`	`Function`	`(sceneKey: String, props: Object)`	用于切换到新选项卡. For `Tabs` only.
`pop`	`Function`		回到上一个页面
`popTo`	`Function`	`(sceneKey: String, props: Object)`	返回到指定的页面
`push`	`Function`	`(sceneKey: String, props: Object)`	跳转到新页面
`refresh`	`Function`	`(props: Object)`	重新加载当前页面
`replace`	`Function`	`(sceneKey: String, props: Object)`	从堆栈中弹出当前场景，并将新场景推送到
`reset`	`Function`	`(sceneKey: String, props: Object)`	清除路由堆栈并将场景推入第一个索引. 没
`drawerOpen`	`Function`		如果可用，打开 `Drawer`
`drawerClose`	`Function`		如果可用，关闭 `Drawer`

ActionConst

键入常量以确定Scene转换，这些是**优先**于手动键入其值，因为项目更新时可能会发生更改。

Property	Type	Value
`ActionConst.JUMP`	`string`	'REACT_NATIVE_ROUTER_FLUX_JUMP'
`ActionConst.PUSH`	`string`	'REACT_NATIVE_ROUTER_FLUX_PUSH'
`ActionConst.PUSH_OR_POP`	`string`	'REACT_NATIVE_ROUTER_FLUX_PUSH_OR_P
`ActionConst.REPLACE`	`string`	'REACT_NATIVE_ROUTER_FLUX_REPLACE'
`ActionConst.BACK`	`string`	'REACT_NATIVE_ROUTER_FLUX_BACK'
`ActionConst.BACK_ACTION`	`string`	'REACT_NATIVE_ROUTER_FLUX_BACK_ACTI
`ActionConst.POP_TO`	`string`	'REACT_NATIVE_ROUTER_FLUX_POP_TO'
`ActionConst.REFRESH`	`string`	'REACT_NATIVE_ROUTER_FLUX_REFRESH'
`ActionConst.RESET`	`string`	'REACT_NATIVE_ROUTER_FLUX_RESET'
`ActionConst.FOCUS`	`string`	'REACT_NATIVE_ROUTER_FLUX_FOCUS'
`ActionConst.BLUR`	`string`	'REACT_NATIVE_ROUTER_FLUX_BLUR'
`ActionConst.ANDROID_BACK`	`string`	'REACT_NATIVE_ROUTER_FLUX_ANDROID_B