前言

> *本小册是《千锋大前端小册》系列之 React Native 部分。内容包含RN基础和项目实战两个部分。通过本小册，可以系统学习 RN 基础知识和在项目中的应用。*
>
> *—— 作者：千锋教育·古艺散人*

起步

本节将帮助您安装和构建第一个 React Native 应用程序。如果您已经安装了 React Native，那么可以跳到本教程。

如果你是移动开发新手，最简单的入门方法是使用Expo CLI。Expo是一套围绕React Native构建的工具，虽然它有很多功能，最基础的功能是它可以让你在几分钟内编写一个React Native应用程序。你只需要Node.js的最新版本和一个手机或模拟器。如果您想在安装任何工具之前直接在web浏览器中试用React Native，可以试用Snack。

如果您已经熟悉移动开发，那么可能需要使用React Native CLI。它需要Xcode或Android Studio才能启动。如果你已经安装了其中一个工具，您应该能够在几分钟内启动并运行。如果没有安装，您应该花大约一个小时来安装和配置它们。

使用 Expo CLI

假设已安装 Node.js 10 LTS或更高版本，则可以使用npm安装Expo CLI命令行实用程序：

```
npm install -g expo-cli
```

然后运行以下命令，创建一个名为“rn-basics”本地项目：

```
expo init rn-basics

cd rn-basics
npm start # 也可以使用命令: expo start
```

此时会启动一个开发服务器。

使用 React Native CLI

待完善

运行 React Native 应用程序

在iOS或Android手机上安装Expo客户端应用程序，并连接到与计算机相同的无线网络（Wifi热点）。在Android上，使用Expo应用程序从终端扫描二维码以打开项目。在iOS上，按照屏幕上的说明（一般为使用相机扫描）获取链接。

修改你的程序

现在你已经成功运行了应用程序，让我们修改一下代码试试。在文本编辑器中打开 App.js 并编辑一些行。保存更改后，应用程序会自动重新加载。

基础知识

React Native 与 React类似，但它使用原生(native)组件而不是基于浏览器(web)组件作为构建块。因此，要了解 React Ntive 应用程序的基本结构，您需要了解一些基本的 React 概念，如JSX、组件、状态和属性。如果你已经了解 React，那么你仍然需要学习一些 React Native 特定的东西，比如 原生(Native) 组件。本教程面向所有人群，无论你是否有 React 经验。

Hello World

编程界的老习惯，先来个 Hello World 尝尝鲜：

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

export default class HelloWorldApp extends Component {
 render() {
 return (
 <View style={{ flex: 1, justifyContent: "center", alignItems: "center" }}>
 <Text>Hello, world!</Text>
 </View>
 );
 }
}
```

如果你感到好奇，这不就是React程序吗？是的，可以直接在web模拟器中运行这段代码。也可以将其粘贴到App.js文件中，以便在本地计算机上创建真正的原生应用程序。

奇葩的语法

这里的一些内容看来可能不像 JavaScript。别慌。这就是未来。

首先，ES2015（也称为ES6）是对JavaScript的一系列改进，ECMAScript 现在是官方标准的一部分，但还没有得到所有浏览器的支持。React Native ships 支持 ES2015，因此你可以使用这些内容而不必担心兼容性。上述示例中的import、from、class 和 extends 都是ES2015的特性。如果你不熟悉ES2015，你也可以通过阅读本教程中的示例代码来了解它。

在这个代码示例中，另一个不寻常的事情是 `<View><Text>Hello world! </Text></View>` 。这是JSX——一种在JavaScript中嵌入XML的语法。许多框架使用一种专门的模板语言，允许您在标记语言中嵌入代码。在React中，没有使用模板。JSX允许您在代码中编写标记语言。它看起来像web上的HTML，但这里使用的是React组件，而不是像 `<div>` 或 `<span>` 这样的 HTML 标签。在本例中，<Text>是一个内置组件，它显示一些文本，类似于 `<div>` 或 `<span>` 。

组件

这段代码定义了HelloWorldApp，这是一个新组件。当你在构建一个 React 本地应用程序时，你将大量地生成新组件。你在屏幕上看到的任何东西都是某种组件。

Props

大多数组件在创建时都可以使用不同的参数进行自定义。这些创建参数称为props，是properties的缩写。例如，一个基本的React Native 组件 Image。创建图像时，可以使用名为source的属性来控制它显示的图像。

```
1.  import React, { Component } from 'react'
2.  import { Image } from 'react-native'
3.
4.  export default class Bananas extends Component {
5.   render() {
6.   let pic = {
7.   uri: 'https://upload.wikimedia.org/wikipedia/commons/d/de/Bananavarieties.jpg'
8.   }
9.   return (
10.  <Image source={pic} style={{width: 193, height: 110}}/>
11.  )
12.  }
13.  }
```

注意 {pic} 周围的大括号-它们将变量pic嵌入到JSX中。您可以将任何JavaScript表达式放在JSX中的大括号中。

你自己的组件也可以使用 props。这允许你创建一个在应用程序中的许多不同位置使用的组件，每个组件的属性可以略有不同，获取值可以在渲染函数中引用This.props。下面是一个例子：

```
1.  import React, { Component } from 'react';
2.  import { Text, View } from 'react-native';
3.
4.  class Greeting extends Component {
5.   render() {
6.   return (
7.   <View style={{alignItems: 'center'}}>
8.   <Text>Hello {this.props.name}!</Text>
9.   </View>
10.  );
11.  }
12. }
13.
14. export default class LotsOfGreetings extends Component {
15.  render() {
16.  return (
17.  <View style={{alignItems: 'center', top: 50}}>
18.  <Greeting name='张三' />
19.  <Greeting name='李四' />
20.  <Greeting name='王五' />
21.  </View>
22.  );
23.  }
24. }
```

我们在Greeting组件中将name作为一个属性来定制，这样可以复用这一组件来制作各种不同的“问候语”。上面的例子把Greeting组件写在 JSX 语句中，用法和内置组件并无二致——这正是 React 体系的魅力所在——如果你想搭建一套自己的基础 UI 框架，那就放手做吧！

上面的例子出现了一个新的名为 `View` 的组件。 `View` 常用作其他组件的容器，来帮助控制布局和样式。

仅仅使用 `props` 和基础的 `Text` 、 `Image` 以及 `View` 组件，你就已经足以编写各式各样的 UI 组件了。要学习如何动态修改你的界面，那就需要进一步学习 State（状态）的概念。

State

我们使用两种数据来控制一个组件：props 和 state。props是在父组件中指定，而且一经指定，在被指定

的组件的生命周期中则不再改变。对于需要改变的数据，我们需要使用state。

一般来说，你需要在class中声明一个state对象，然后在需要修改时调用setState方法。

假如我们需要制作一段不停闪烁的文字。文字内容本身在组件创建时就已经指定好了，所以文字内容应该是一个prop。而文字的显示或隐藏的状态（快速的显隐切换就产生了闪烁的效果）则是随着时间变化的，因此这一状态应该写到state中。

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

class Blink extends Component {
 // 声明state对象
 state = { isShowingText: true };

 componentDidMount() {
 // 每1000毫秒对showText状态做一次取反操作
 setInterval(() => {
 this.setState({
 isShowingText: !this.state.isShowingText
 });
 }, 1000);
 }

 render() {
 // 根据当前showText的值决定是否显示text内容
 if (!this.state.isShowingText) {
 return null;
 }

 return (
 <Text>{this.props.text}</Text>
 );
 }
}

export default class BlinkApp extends Component {
 render() {
 return (
 <View>
 <Blink text='I love to blink' />
 <Blink text='Yes blinking is so great' />
```

```
<Blink text='Yes Blinking is so great' />
<Blink text='Why did they ever take this out of HTML' />
<Blink text='Look at me look at me look at me' />
</View>
);
}
}
```

实际开发中，我们一般不会在定时器函数（setInterval、setTimeout 等）中来操作 state。典型的场景是在接收到服务器返回的新数据，或者在用户输入数据之后。你也可以使用一些“状态容器”比如 `Redux` 来统一管理数据流。

每次调用setState时，BlinkApp 都会重新执行 render 方法重新渲染。这里我们使用定时器来不停调用 setState，于是组件就会随着时间变化不停地重新渲染。

State 的工作原理和 React.js 完全一致，所以对于处理 state 的一些更深入的细节，你可以参阅 React.Component API。

提示一些初学者应该牢记的要点：
一切界面变化都是状态state变化
state的修改必须通过setState()方法
this.state.likes = 100; // 这样的直接赋值修改无效！
setState 是一个 merge 合并操作，只修改指定属性，不影响其他属性
setState 是异步操作，修改不会马上生效

样式

在 React Native 中，你并不需要学习什么特殊的语法来定义样式。我们仍然是使用 JavaScript 来写样式。所有的核心组件都接受名为 style 的属性。这些样式名基本上是遵循了 web 上的 CSS 的命名，只是按照 JS 的语法要求使用了驼峰命名法，例如将 background-color 改为 backgroundColor。

style属性可以是一个普通的 JavaScript 对象。这是最简单的用法，因而在示例代码中很常见。你还可以传入一个数组——在数组中位置居后的样式对象比居前的优先级更高，这样你可以间接实现样式的继承。

实际开发中组件的样式会越来越复杂，我们建议使用 `StyleSheet.create` 来集中定义组件的样式。比如像下面这样：

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

const styles = StyleSheet.create({
  bigBlue: {
  color: 'blue'
```

```
  color: 'blue',
  fontWeight: 'bold',
  fontSize: 30,
  },
  red: {
  color: 'red',
  },
 });

 export default class LotsOfStyles extends Component {
  render() {
  return (
  <View>
  <Text style={styles.red}>just red</Text>
  <Text style={styles.bigBlue}>just bigBlue</Text>
  <Text style={[styles.bigBlue, styles.red]}>bigBlue, then red</Text>
  <Text style={[styles.red, styles.bigBlue]}>red, then bigBlue</Text>
  </View>
  );
  }
 }
```

常见的做法是按顺序声明和使用style属性，以借鉴 CSS 中的“层叠”做法（即后声明的属性会覆盖先声明的同名属性）。

宽度(Width) 和 高度(Height)

组件的高度和宽度决定了其在屏幕上显示的尺寸。

指定宽高

最简单的给组件设定尺寸的方式就是在样式中指定固定的 `width` 和 `height` 。React Native 中的尺寸都是无单位的，表示的是与设备像素密度无关的逻辑像素点。

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

export default class FixedDimensionsBasics extends Component {
 render() {
 return (
  <View>
```

```
  <View style={{width: 50, height: 50, backgroundColor: 'powderblue'}} />
  <View style={{width: 100, height: 100, backgroundColor: 'skyblue'}} />
  <View style={{width: 150, height: 150, backgroundColor: 'steelblue'}} />
  </View>
  )
  }
}
```

这样给组件设置尺寸也是一种常见的模式，比如要求在不同尺寸的屏幕上都显示成一样的大小。

弹性（Flex）宽高

在组件样式中使用flex可以使其在可利用的空间中动态地扩张或收缩。一般而言我们会使用flex:1来指定某个组件扩张以撑满所有剩余的空间。如果有多个并列的子组件使用了flex:1，则这些子组件会平分父容器中剩余的空间。如果这些并列的子组件的flex值不一样，则谁的值更大，谁占据剩余空间的比例就更大（即占据剩余空间的比等于并列组件间flex值的比）。

> *组件能够撑满剩余空间的前提是其父容器的尺寸不为零。如果父容器既没有固定的width和height，也没有设定flex，则父容器的尺寸为零。其子组件如果使用了flex，也是无法显示的。*

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

export default class FlexDimensionsBasics extends Component {
 render() {
 return (
 // 试试去掉父 View 中的`flex: 1`。
 // 则父View不再具有尺寸，因此子组件也无法再撑开。
 // 然后再用`height: 300` 来代替父 View 的 `flex: 1` 试试看？
 <View style={{flex: 1}}>
 <View style={{flex: 1, backgroundColor: 'powderblue'}} />
 <View style={{flex: 2, backgroundColor: 'skyblue'}} />
 <View style={{flex: 3, backgroundColor: 'steelblue'}} />
 </View>
 )
 }
}
```

使用 flexbox 布局

我们在 React Native 中使用 flexbox 规则来指定某个组件的子元素的布局。Flexbox 可以在不同屏幕尺寸上提供一致的布局结构。

一般来说，使用 `flexDirection` 、 `alignItems` 和 `justifyContent` 三个样式属性就已经能满足大多数布局需求。

> *React Native 中的 Flexbox 的工作原理和 web 上的 CSS 基本一致，当然也存在少许差异。首先是默认值不同：flexDirection的默认值是column而不是row，而flex也只能指定一个数字值。*

Flex

flex 属性决定元素在主轴上如何填满可用区域。整个区域会根据每个元素设置的flex属性值被分割成多个部分。

在下面的例子中，在设置了 `flex: 1` 的容器view中，有红色，黄色和绿色三个子 `view` 。红色 `view` 设置了 `flex: 1` ，黄色 `view` 设置了 `flex: 2` ，绿色 `view` 设置了 `flex: 3` 。 `1+2+3 = 6` ，这意味着红色 `view` 占据整个区域的 `1/6` ，黄色 `view` 占据整个区域的 `2/6` ，绿色 `view` 占据整个区域的 `3/6` 。

Flex Direction

在组件的 style 中指定 `flexDirection` 可以决定布局的主轴。子元素是应该沿着水平轴 (row) 方向排列，还是沿着竖直轴 (column) 方向排列呢？默认值是竖直轴 (column) 方向。

```
1.  import React, { Component } from 'react'
2.  import { View } from 'react-native'
3.
4.  export default class FlexDirectionBasics extends Component {
5.   render() {
6.   return (
7.   // 尝试把`flexDirection`改为`column`看看
8.   <View style={{flex: 1, flexDirection: 'row'}}>
9.   <View style={{width: 50, height: 50, backgroundColor: 'powderblue'}} />
10.  <View style={{width: 50, height: 50, backgroundColor: 'skyblue'}} />
11.  <View style={{width: 50, height: 50, backgroundColor: 'steelblue'}} />
12.  </View>
13.  )
14.  }
15.  }
```

Layout Direction

Layout Direction

布局方向指定层次结构中的子项和文本的布局方向。布局方向也会影响边起点和终点所指的对象。默认情况下，React Native布局使用LTR布局方向。在这种模式下，开始是指左边，结束是指右边。

- LTR（默认值）文本和子级，并从左到右排列。应用的边距和填充元素的开头应用于左侧。
- 从右到左排列的RTL文本和子项。应用的边距和填充元素的开头应用于右侧。

Justify Content

在组件的 style 中指定 justifyContent 可以决定其子元素沿着主轴的排列方式。子元素是应该靠近主轴的起始端还是末尾段分布呢？亦或应该均匀分布？对应的这些可选项有：flex-start、center、flex-end、space-around、space-between 以及 space-evenly。

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

export default class JustifyContentBasics extends Component {
 render() {
 return (
 // 尝试把`justifyContent`改为`center`看看
 // 尝试把`flexDirection`改为`row`看看
 <View style={{
 flex: 1,
 flexDirection: 'column',
 justifyContent: 'space-between',
 }}>
 <View style={{width: 50, height: 50, backgroundColor: 'powderblue'}} />
 <View style={{width: 50, height: 50, backgroundColor: 'skyblue'}} />
 <View style={{width: 50, height: 50, backgroundColor: 'steelblue'}} />
 </View>
 )
 }
}
```

使用长列表

React Native 提供了几个适用于展示长列表数据的组件，一般而言我们会选用FlatList或是SectionList。

FlatList组件用于显示一个垂直的滚动列表，其中的元素之间结构近似而仅数据不同

FlatList组件用于显示一个垂直的滚动列表，其中的元素之间结构近似而仅数据不同。

FlatList更适于长列表数据，且元素个数可以增删。和ScrollView不同的是，FlatList并不立即渲染所有元素，而是优先渲染屏幕上可见的元素。

FlatList组件必须的两个属性是data和renderItem。data是列表的数据源，而renderItem则从数据源中逐个解析数据，然后返回一个设定好格式的组件来渲染。

下面的例子创建了一个简单的FlatList，并预设了一些模拟数据。首先是初始化FlatList所需的data，其中的每一项（行）数据之后都在renderItem中被渲染成了Text组件，最后构成整个FlatList。

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

export default class FlatListBasics extends Component {
 render() {
 return (
 <View style={styles.container}>
 <FlatList
 data={[
 {key: 'Devin'},
 {key: 'Dan'},
 {key: 'Dominic'},
 {key: 'Jackson'},
 {key: 'James'},
 {key: 'Joel'},
 {key: 'John'},
 {key: 'Jillian'},
 {key: 'Jimmy'},
 {key: 'Julie'},
 ]}
 renderItem={({item}) => <Text style={styles.item}>{item.key}</Text>}
 />
 </View>
 );
 }
}

const styles = StyleSheet.create({
 container: {
 flex: 1,
 paddingTop: 22
```

```
  },
  item: {
  padding: 10,
  fontSize: 18,
  height: 44,
  },
 })
```

如果要渲染的是一组需要分组的数据，也许还带有分组标签的，那么SectionList将是个不错的选择。

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

 export default class SectionListBasics extends Component {
  render() {
  return (
  <View style={styles.container}>
  <SectionList
  sections={[
  {title: 'D', data: ['Devin', 'Dan', 'Dominic']},
  {title: 'J', data: ['Jackson', 'James', 'Jillian', 'Jimmy', 'Joel', 'John', 'Julie']},
  ]}
  renderItem={({item}) => <Text style={styles.item}>{item}</Text>}
  renderSectionHeader={({section}) => <Text style={styles.sectionHeader}>{section.title}
</Text>}
  keyExtractor={(item, index) => index}
  />
  </View>
  );
  }
 }

 const styles = StyleSheet.create({
  container: {
  flex: 1,
  paddingTop: 22
  },
  sectionHeader: {
  paddingTop: 2,
  paddingLeft: 10,
  paddingRight: 10,
  paddingBottom: 2,
```

```
  fontSize: 14,
  fontWeight: 'bold',
  backgroundColor: 'rgba(247,247,247,1.0)',
  },
  item: {
  padding: 10,
  fontSize: 18,
  height: 44,
  },
 })
```

网络

很多移动应用都需要从远程地址中获取数据或资源。你可能需要给某个 REST API 发起 POST 请求以提交用户数据，又或者可能仅仅需要从某个服务器上获取一些静态内容——以下就是你会用到的东西。

使用 Fetch

React Native 提供了和 web 标准一致的 Fetch API，用于满足开发者访问网络的需求。如果你之前使用过 XMLHttpRequest(即俗称的 ajax)或是其他的网络 API，那么 Fetch 用起来将会相当容易上手。这篇文档只会列出 Fetch 的基本用法，并不会讲述太多细节，你可以使用你喜欢的搜索引擎去搜索fetch api关键字以了解更多信息。

发起请求

要从任意地址获取内容的话，只需简单地将网址作为参数传递给 fetch 方法即可（fetch 这个词本身也就是获取的意思）

```
fetch('https://mywebsite.com/mydata.json');
```

Fetch 还有可选的第二个参数，可以用来定制 HTTP 请求一些参数。你可以指定 header 参数，或是指定使用 POST 方法，又或是提交数据等等：

```
fetch('https://mywebsite.com/endpoint/', {
 method: 'POST',
 headers: {
 Accept: 'application/json',
 'Content-Type': 'application/json',
 },
 body: JSON.stringify({
 firstParam: 'yourValue',
```

```
    secondParam: 'yourOtherValue',
  }),
 })
```

提交数据的格式关键取决于 headers 中的Content-Type。Content-Type有很多种，对应 body 的格式也有区别。到底应该采用什么样的Content-Type取决于服务器端，所以请和服务器端的开发人员沟通确定清楚。常用的’Content-Type’除了上面的’application/json’，还有传统的网页表单形式，示例如下：

```
fetch('https://mywebsite.com/endpoint/', {
 method: 'POST',
 headers: {
 'Content-Type': 'application/x-www-form-urlencoded',
 },
 body: 'key1=value1&key2=value2',
})
```

可以参考Fetch 请求文档来查看所有可用的参数。

> *注意：使用 Chrome 调试目前无法观测到 React Native 中的网络请求，你可以使用第三方的react-native-debugger来进行观测。*

处理服务器的响应数据

上面的例子演示了如何发起请求。很多情况下，你还需要处理服务器回复的数据。

网络请求天然是一种异步操作。Fetch 方法会返回一个Promise，这种模式可以简化异步风格的代码：

```
function getMoviesFromApiAsync() {
 return fetch('https://facebook.github.io/react-native/movies.json')
 .then((response) => response.json())
 .then((responseJson) => {
 return responseJson.movies;
 })
 .catch((error) => {
 console.error(error);
 });
}
```

你也可以在 React Native 应用中使用 ES2017 标准中的async/await 语法：

```
// 注意这个方法前面有async关键字
async function getMoviesFromApi() {
 try {
 // 注意这里的await语句，其所在的函数必须有async关键字声明
 let response = await fetch(
```

```
    'https://facebook.github.io/react-native/movies.json',
    );
    let responseJson = await response.json();
    return responseJson.movies;
    } catch (error) {
    console.error(error);
    }
  }
```

别忘了 catch 住fetch可能抛出的异常，否则出错时你可能看不到任何提示。

```
import React from 'react';
import { FlatList, ActivityIndicator, Text, View } from 'react-native';

export default class FetchExample extends React.Component {

  constructor(props){
  super(props);
  this.state ={ isLoading: true}
  }

  componentDidMount(){
  return fetch('https://facebook.github.io/react-native/movies.json')
  .then((response) => response.json())
  .then((responseJson) => {
  this.setState({
  isLoading: false,
  dataSource: responseJson.movies,
  }, function(){

  });

  })
  .catch((error) =>{
  console.error(error);
  })
  }

  render(){
  if(this.state.isLoading){
  return(
  <View style={{flex: 1, padding: 20}}>
```

```
  <ActivityIndicator/>
  </View>
  )
  }

  return(
  <View style={{flex: 1, paddingTop:20}}>
  <FlatList
  data={this.state.dataSource}
  renderItem={({item}) => <Text>{item.title}, {item.releaseYear}</Text>}
  keyExtractor={(item, index) => item.id}
  />
  </View>
  )
  }
 }
```

> *默认情况下，iOS 会阻止所有 http 的请求，以督促开发者使用 https。如果你仍然需要使用 http 协议，那么首先需要添加一个 App Transport Security 的例外，详细可参考这篇帖子。*
>
> *从 Android9 开始，也会默认阻止 http 请求，请参考相关配置*

使用其他的网络库

React Native 中已经内置了XMLHttpRequest API(也就是俗称的 ajax)。一些基于 XMLHttpRequest 封装的第三方库也可以使用，例如frisbee或是axios等。但注意不能使用 jQuery，因为 jQuery 中还使用了很多浏览器中才有而 RN 中没有的东西（所以也不是所有 web 中的 ajax 库都可以直接使用）。

```
 const request = new XMLHttpRequest();
request.onreadystatechange = (e) => {
 if (request.readyState !== 4) {
 return
 }

 if (request.status === 200) {
 console.log('success', request.responseText);
 } else {
 console.warn('error')
 }
};

request.open('GET', 'https://mywebsite.com/endpoint/')
```

```
request.send()
```

axios 用例：

```
import React, { Component } from 'react'
import { Text, View } from 'react-native'
import axios from 'axios'

export default class App extends Component {
 async componentDidMount() {
 let result = await axios({
 url: 'https://m.lagou.com/listmore.json?pageNo=2&pageSize=15'
 })

 console.log(result)
 }

 render() {
 return (
 <View>
 <Text> textInComponent </Text>
 </View>
 )
 }
}
```

> 需要注意的是，安全机制与网页环境有所不同：在应用中你可以访问任何网站，没有跨域的限制。

WebSocket 支持

React Native 还支持WebSocket，这种协议可以在单个 TCP 连接上提供全双工的通信信道。

```
const ws = new WebSocket('ws://localhost:8081');

ws.onopen = () => {
 // connection opened
 ws.send('something'); // send a message
}

ws.onmessage = (e) => {
 // a message was received
 console.log(e.data);
}
```

```
}

ws.onerror = (e) => {
 // an error occurred
 console.log(e.message);
}

ws.onclose = (e) => {
 // connection closed
 console.log(e.code, e.reason);
}
```

websocket 服务端：

```
const WebSocket = require('ws')
const ws = new WebSocket.Server({ port: 8081 })

let clients = {}
let clientName = 0

ws.on('connection', (client) => {
 client.name = ++clientName
 clients[client.name] = client

 client.on('message', (msg) => {
 broadcast(client, msg)
 })

 client.on('close', () => {
 delete clients[client.name]
 console.log(client.name + ' 离开了~')
 })
})

function broadcast(client, msg) {
 for (var key in clients) {
 clients[key].send(client.name + ' 说: ' + msg)
 }
}
```

本文档由千锋大前端互联网标准化研究院出品

搭建项目环境

本项目是应用 ReactNative，TypeScript，Mobx等技术开发的一个“美食大全”的项目，基本的环境搭建，大家参照本文基础部分。

```
expo init rn-cookbooks
```

然后选择 blank (TypeScript)：

```
? Choose a template:
 ----- Managed workflow -----
  blank a minimal app as clean as an empty canvas
❯ blank (TypeScript) same as blank but with TypeScript configuration
  tabs several example screens and tabs using react-navigation
 ----- Bare workflow -----
  minimal bare and minimal, just the essentials to get you started
  minimal (TypeScript) same as minimal but with TypeScript configuration
```

启动项目：

```
cd rn-cookbooks
yarn start
```

编写 Index组件 基本结构

在根目录下创建 `pages/index` 文件夹，在里面创建一个 `Index.tsx` 文件，编辑内容：

```
// pages/index/Index.tsx
import React, { Component } from 'react'

import {
  View,
  Text,
  StyleSheet
} from 'react-native'

interface Props {

}

interface State {

}
```

```
17.
18. export default class Index extends Component<Props, State> {
19.   constructor(props) {
20.   super(props)
21.   }
22.
23.   state: State = {
24.
25.   }
26.
27.   componentDidMount() {
28.
29.   }
30.
31.   render() {
32.   return (
33.   <View style={styles.container}>
34.   <Text>
35.   Index 组件内容
36.   </Text>
37.   </View>
38.   )
39.   }
40. }
41.
42. const styles = StyleSheet.create({
43.   container: {
44.   flex: 1,
45.   backgroundColor: '#fff',
46.   alignItems: 'center',
47.   justifyContent: 'center',
48.   },
49.  })
```

修改根目录下的 `App.tsx` ：

```
1.  import React from 'react'
2.  import Index from './pages/index/Index'
3.
4.  export default function App() {
5.   return (
6.   <Index></Index>
```

```
  )
}
```

引入 react-native-tab-navigator

在项目环境命令行里安装 tabbar 导航器，详细内容可参见 react-native-tab-navigator 官网

```
yarn add react-native-tab-navigator -S
```

修改 index.tsx, 引入 tab-navigator 代码：

```
import React, { Component } from 'react'
import TabNavigator from 'react-native-tab-navigator'

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

import {
 Img
} from './styled_index'
import styles from './style_index'

import cookbook from '../../assets/images/cookbook.png'
import cookbookActive from '../../assets/images/cookbook-active.png'
import category from '../../assets/images/menu.png'
import categoryActive from '../../assets/images/menu-active.png'
import map from '../../assets/images/location.png'
import mapActive from '../../assets/images/location-active.png'
import more from '../../assets/images/more.png'
import moreActive from '../../assets/images/more-active.png'

interface Props {

}

interface State {
 selectedTab: string
```

```
}

class Index extends Component<Props, State> {
 constructor(props: Props) {
 super(props)
 }

 state: State = {
 selectedTab: 'home'
 }

 componentDidMount() {

 }

 render() {
 return (
 <TabNavigator
 tabBarStyle={styles.tabBarStyle}
 >
 <TabNavigator.Item
 selected={this.state.selectedTab === 'home'}
 title="美食大全"
 titleStyle={styles.titleStyle}
 selectedTitleStyle={styles.selectedTitleStyle}
 renderIcon={() => <Img source={cookbook} />}
 renderSelectedIcon={() => <Img source={cookbookActive} />}
 onPress={() => this.setState({ selectedTab: 'home' })}
 >
 {<View><Text>美食大全</Text></View>}
 </TabNavigator.Item>
 <TabNavigator.Item
 selected={this.state.selectedTab === 'category'}
 title="分类"
 titleStyle={styles.titleStyle}
 selectedTitleStyle={styles.selectedTitleStyle}
 renderIcon={() => <Img source={category} />}
 renderSelectedIcon={() => <Img source={categoryActive} />}
 onPress={() => this.setState({ selectedTab: 'category' })}
 >
 {<View><Text>分类</Text></View>}
```

```
70.   </TabNavigator.Item>
71.   <TabNavigator.Item
72.   selected={this.state.selectedTab === 'map'}
73.   title="地图"
74.   titleStyle={styles.titleStyle}
75.   selectedTitleStyle={styles.selectedTitleStyle}
76.   renderIcon={() => <Img source={map} />}
77.   renderSelectedIcon={() => <Img source={mapActive} />}
78.   onPress={() => this.setState({ selectedTab: 'map' })}
79.   >
80.   {<View><Text>地图</Text></View>}
81.   </TabNavigator.Item>
82.   <TabNavigator.Item
83.   selected={this.state.selectedTab === 'more'}
84.   title="更多"
85.   titleStyle={styles.titleStyle}
86.   selectedTitleStyle={styles.selectedTitleStyle}
87.   renderIcon={() => <Img source={more} />}
88.   renderSelectedIcon={() => <Img source={moreActive} />}
89.   onPress={() => this.setState({ selectedTab: 'more' })}
90.   >
91.   {<View><Text>更多</Text></View>}
92.   </TabNavigator.Item>
93.   </TabNavigator>
94.   )
95.   }
96. }
97.
98. export default Index
```

问题：

- ts 提示引入的 png 不能识别，飘红了。解决方案是在项目跟目录下创建 `images.d.ts` 文件，内容如下：

```
1.  declare module '*.svg'
2.  declare module '*.png'
3.  declare module '*.jpg'
4.  declare module '*.jpeg'
5.  declare module '*.gif'
6.  declare module '*.bmp'
7.  declare module '*.tiff'
```

在 `pages/index` 下创建样式文件：

- 安装 styled-components 模块，创建 styled_index.js，内容如下：

```
import styled from 'styled-components'

const Img = styled.Image `
 width: 25px;
 height: 25px;
`

export {
 Img
}
```

再创建一个样式文件 style_index.js，内容如下：

```
import { StyleSheet } from 'react-native'

export default StyleSheet.create({
 titleStyle: {
 color: '#666'
 },

 tabBarStyle: {
 paddingBottom: 34,
 height: 80
 },

 selectedTitleStyle: {
 color: '#000'
 }
})
```

tabbar 的兼容处理

安装 expo-device

```
npm i expo-device -S
```

修改 index.ts，根据您设备情况引入不同的样式，此处只是测试性代码，只做了iphone XR 和 其他非“齐刘海” iPhone 手机：

```
// 载入模块
import * as Device from 'expo-device'

// 在 TabNavigator 上修改 tabBarStyle
<TabNavigator
 tabBarStyle={
 Device.deviceName === 'iPhone XR' ? styles.tabBarStyle : null
 }
 >
```

ant-design-mobile-rn 快速上手

> *在开始之前，推荐先学习 React 和 ES2015。我们使用了 `babel`，试试用 ES2015 的写法来提升编码的愉悦感。*
>
> *确认 Node.js 已经升级到 v4.x 或以上。*

1. 创建一个项目

可以是已有项目，或者是使用 create-react-native-app 新创建的空项目，你也可以从 官方示例 脚手架里拷贝并修改

> *参考更多官方示例集*
> *或者你可以利用 React 生态圈中的 各种脚手架*

完整步骤请查看此处文档： antd-mobile-sample/create-react-native-app

2. 安装

```
npm install @ant-design/react-native --save
```

or

```
yarn add @ant-design/react-native
```

链接字体图标

```
react-native link @ant-design/icons-react-native
```

3. 使用

按需加载

下面两种方式都可以**只加载**用到的组件，选择其中一种方式即可。

- 使用 babel-plugin-import（推荐）。

```
// .babelrc or babel-loader option
{
  "plugins": [
  ["import", { libraryName: "@ant-design/react-native" }] // 与 Web 平台的区别是不需要设置 style
  ]
}
```

然后改变从 @ant-design/react-native 引入模块方式即可。

```
import { Button } from '@ant-design/react-native';
```

Home.tsx 构建

在项目根目录下创建 pages/home 文件夹，在这个文件夹下创建 Home.tsx 文件，内容如下：

```
import React, { Component } from 'react'
import Swiper from './Swiper'
import HotCate from './HotCate'

interface Props {

}

interface State {

}

class Home extends Component<Props, State> {
 render() {
 return (
 <>
```

```
    <Swiper></Swiper>
    <HotCate></HotCate>
    </>
    )
    }
}

export default Home
```

此时在 Home.tsx 中引入 Swiper 和 HotCate 两个组价。

Swiper.tsx 组价构建

在根目录下创建 utils 文件夹，在这个文件夹里创建 http.js 文件，内容如下：

```
// utils/http.js
export const get = (url) => {
 return fetch(url, {
 method: 'get'
 })
 .then(response => response.json())
 .then(result => {
 return result.data
 })
}
```

在 pages/home 文件夹里再创建一个 Swiper.tsx 组件，内容如下：

```
import React, { Component } from 'react'
import { Carousel } from '@ant-design/react-native'
import { get } from '../../utils/http'

import {
 View,
 Image
} from 'react-native'

import styles from './style_home'

interface Props {

}
```

```
15.
16. interface State {
17.  list: Array<any>
18. }
19.
20. class Swiper extends Component<Props, State> {
21.  state = {
22.  list: []
23.  }
24.  async componentDidMount() {
25.  let list = await get('http://localhost:9000/api/swiper')
26.  this.setState({
27.  list
28.  })
29.  }
30.
31.  render() {
32.  return (
33.  <Carousel
34.  style={styles.wrapper}
35.  selectedIndex={0}
36.  autoplay
37.  infinite
38.  >
39.  {
40.  this.state.list.slice(0, 5).map((value, index) => {
41.  return (
42.  <View
43.  style={styles.containerHorizontal}
44.  key={value.id}
45.  >
46.  <Image
47.  source={{uri: value.img}}
48.  style={styles.slideImg}
49.  resizeMode='cover'
50.  ></Image>
51.  </View>
52.  )
53.  })
54.  }
55.  </Carousel>
```

```
  )
  }
}

export default Swiper
```

在 page/home 文件里创建 style_home.js 文件，编辑样式如下：

```
import { StyleSheet } from 'react-native'

export default StyleSheet.create({
 // swiper
 wrapper: {
 height: 170
 },

 containerHorizontal: {
 flex: 1,
 alignItems: 'center',
 justifyContent: 'center',
 height: 170
 },

 slideImg: {
 height: 170,
 width: '100%'
 },
})
```

构建 HotCate.tsx 文件

在 pages/home 文件夹里构建 HotCate.tsx 文件，内容为：

```
import React, { Component } from 'react'
import { Grid } from '@ant-design/react-native'
import { get } from '../../utils/http'

import styles from './style_home'

import {
 View,
```

```
  Text,
  Image,
  StyleSheet
} from 'react-native'

interface Props {

}
interface State {
 hotCate: Array<object>
}

export default class HotCate extends Component<Props, State> {
 state = {
 hotCate: []
 }

 _renderItem(el, index) {
 return (
 <View
 style={styles.container}
 >
 {el.img ? <Image source={{uri: el.img}} style={styles.gridImg}></Image> : null}
 <Text style={styles.gridText}>{el.title}</Text>
 </View>
 )
 }

 async componentDidMount() {
 let hotCate = await get('http://localhost:9000/api/hotcate')

 // 补全最后一项数据
 hotCate.push({
 img: '',
 title: '更多...'
 })

 this.setState({
 hotCate
 })
 }
```

```
50.
51.   render() {
52.   return (
53.   <View>
54.   <Grid
55.   data={this.state.hotCate}
56.   renderItem={this._renderItem}
57.   hasLine={false}
58.   ></Grid>
59.   </View>
60.   )
61.   }
62.  }
```

修改 pages/home/style_home.js 文件，样式如下：

```
1.  import { StyleSheet } from 'react-native'
2.
3.  export default StyleSheet.create({
4.   // hotcate
5.   container: {
6.   paddingTop: 20,
7.   paddingBottom: 10
8.   },
9.
10.   gridContainer: {
11.   flex: 1,
12.   justifyContent: 'center',
13.   alignItems: 'center'
14.   },
15.
16.   gridText: {
17.   fontSize: 16,
18.   margin: 6
19.   },
20.
21.   gridImg: {
22.   width: 70,
23.   height: 70,
24.   borderRadius: 5
25.   },
26.  })
```

构建 Top10.tsc 组件

Top10组件渲染的数据和Swiper组件可以使用同一个接口的数据，因此我们决定应用Mobx来管理这个数据。

安装 Mobx 相关模块

```
npm i mobx mobx-react -S
```

构建 store

在项目根目录下创建 store 文件夹，在这个文件下创建 index.js 文件：

```
// store/index.js
import {
 observable,
 action,
 computed
} from 'mobx'

class Store {
 // swiper 与 top10 共享的数据
 @observable
 list = []

 // swiper 数据过滤
 @computed
 get swiper() {
 return this.list.slice(0, 5).map((value, index) => {
 return {
 img: value.img
 }
 })
 }

 // top10 数据过滤
 @computed
 get top10() {
 return this.list.slice(0, 10).map((value, index) => {
```

```
    return this.list.slice(0, 10).map((value, index) => {
    return {
    img: value.img,
    all_click: value.all_click,
    favorites: value.favorites,
    name: value.name
    }
    })
    }

    // 装载 list 数据
    @action.bound
    setList(data) {
    this.list = data
    }
}

export default new Store()
```

开始构建 Top.tsx 组件

在 pages/home 下创建 Top.tsx 文件：

```
// pages/home/Top.tsx
import React, { Component } from 'react'
import { Grid } from '@ant-design/react-native'
import { observer, inject } from 'mobx-react'

import {
  View,
  Text,
  Image
} from 'react-native'

import styles from './style_home.js'

interface Props {
  // store 作为组件的 props
  store?: any
}

```

```
interface State {

}

// 注入 store 与 将类变为可观察的对象
@inject('store')
@observer
class Top10 extends Component<Props, State> {

 renderTop10(el, index) {
 return (
 <View style={styles.top10ItemContainer}>
 <View style={styles.top10ImgContainer}>
 <Image style={styles.Top10Img} source={{uri: el.img}}></Image>
 </View>
 <View style={styles.top10DesContainter}>
 <Text style={styles.top10Titie}>{el.name}</Text>
 <Text style={styles.Top10Desc}>{el.all_click} {el.favorites}</Text>
 </View>
 </View>
 )
 }

 render() {
 return (
 <View style={styles.top10Container}>
 <View style={styles.top10Head}>
 <Text style={styles.top10HeadText}>精品好菜</Text>
 </View>
 <View style={styles.gridContainer}>
 <Grid
 data={this.props.store.top10}
 columnNum={2}
 hasLine={false}
 renderItem={this.renderTop10}
 />
 </View>
 </View>
 )
 }
}
```

```
60.
61.  export default Top10
```

注意：expo-cli 构建的项目，默认 ts 配置不支持装饰器，会给出如下警告：

```
1.  Experimental support for decorators is a feature that is subject to change in a future release. Set the 'experimentalDecorators' option in your 'tsconfig' or 'jsconfig' to remove this warning.
```

需要修改项目根目录下的 tsconfig.json，添加：

```
1.  "experimentalDecorators": true
```

如果不能起作用，重新启动VSCode即可。

添加 top10 样式

```
1.  // pages/home/style_home.js
2.  import { StyleSheet } from 'react-native'
3.  export default StyleSheet.create({
4.   // top10
5.   top10Container: {
6.   paddingBottom: 44,
7.   backgroundColor: '#eee'
8.   },
9.
10.  top10Head: {
11.  height: 50,
12.  paddingLeft: 10,
13.  justifyContent: 'flex-end',
14.  },
15.
16.  top10HeadText: {
17.  fontSize: 18
18.  },
19.
20.  top10ItemContainer: {
21.  flex: 1,
22.  paddingRight: 10
23.  },
24.
25.  top10DesContainter: {
26.  marginLeft: 10,
27.  paddingTop: 10
```

```
    paddingTop: 10,
    paddingBottom: 10,
    justifyContent: 'center',
    alignItems: 'center',
    backgroundColor: '#fff'
  },

  top10ImgContainer: {
    paddingLeft: 10,
    paddingTop: 10,
    flex: 1
  },

  Top10Img: {
    width: '100%',
    height: '100%',
  },

  top10Titie: {
    fontSize: 20
  },

  Top10Desc: {
    color: '#666'
  }
})
```

改造 Swiper.tsx 组件

Swiper 组件和 Top10 组件共享了数据，因此在 store 构建好后，需要改造一下：

```
// pages/home/Swiper.tsx
import React, { Component } from 'react'
import { Carousel } from '@ant-design/react-native'
import { get } from '../../utils/http'

import { observer, inject } from 'mobx-react'

import {
  View,
```

```
  Image
} from 'react-native'

import styles from './style_home'

interface Props {
  // store 作为组件的 props
  store?: any
}

interface State {

}

// 注入 store 与 将类变为可观察的对象
@inject('store')
@observer
class Swiper extends Component<Props, State> {
  state = {
  list: []
  }
  async componentDidMount() {
  let list = await get('http://localhost:9000/api/swiper')
  this.props.store.setList(list)
  }

  render() {
  return (
  <Carousel
  style={styles.wrapper}
  selectedIndex={0}
  autoplay
  infinite
  >
  {
  this.props.store.swiper.map((value, index) => {
  return (
  <View
  style={styles.containerHorizontal}
  key={value.id}
  >
```

```
  <Image
  source={{uri: value.img}}
  style={styles.slideImg}
  resizeMode='cover'
  ></Image>
  </View>
  )
  })
  }
  </Carousel>
  )
  }
}

export default Swiper
```

改造 Home.tsx 组件

在 Home.tsx 组件引入 Top10 组件，同时添加 ScrollView 组件，实现页面滚动效果。

```
// page/home/Home.tsx

import React, { Component } from 'react'
import { ScrollView } from 'react-native'

import Swiper from './Swiper'
import HotCate from './HotCate'
import Top10 from './Top10'

interface Props {

}

interface State {

}

class Home extends Component<Props, State> {
 render() {
 return (
```

```
  <ScrollView>
  <Swiper></Swiper>
  <HotCate></HotCate>
  <Top10></Top10>
  </ScrollView>
  )
  }
}

export default Home
```

构建 List.tsx 组件

接下来构建另一个页面，首先在 pages 目录下创建 list 文件夹，在此文件夹里创建 List.tsx 组件文件和 style_list.js 样式文件。

List.tsx

```
// pages/list/List
import React, { Component, createRef } from 'react'

import {
 inject,
 observer
} from 'mobx-react'

import {
 View,
 Text,
 Image,
 FlatList
} from 'react-native'

import styles from './style_list'

interface Props {
 store?: any
}

```

```
interface State {
 // 记录上拉加载更多的当前页码
 curPage: number,

 // 页面显示的数据
 datalist: Array<object>,

 // 控制下拉刷新的开关
 refresh: boolean
}

let pageSize = 10

@inject('store')
@observer
export default class List extends Component<Props, State> {
 constructor (
 public props: Props,
 public flatlist,
 ) {
 super(props)
 this.flatlist = createRef()
 }

 state = {
 curPage: 1,
 datalist: [],
 refresh: false
 }

 // 渲染 Flatlist 组件数据
 _renderItem(item) {
 let {img, name, burdens, all_click, favorites} = item.item.data
 return (
 <View style={styles.listWrap}>
 <View style={styles.imgWrap}>
 <Image style={styles.image} source={{uri: img}}></Image>
 </View>
 <View style={styles.descWrap}>
 <Text style={styles.title}>{name}</Text>
 <Text style={styles.subtitle} numberOfLines={1}>{burdens}</Text>
```

```
62.   <Text style={styles.subtitle} numberOfLines={1}>{burdens}</Text>
63.   <Text style={styles.desc}>{all_click} {favorites}</Text>
64.   </View>
65.   </View>
66.   )
67.   }
68.
69.   // 处理用户拉到底端的响应函数
70.   _handleReachEnd() {
71.   // 如果还有数据，一直加载
72.   if (this.state.curPage < Math.ceil(this.props.store.list.length / pageSize)) {
73.   this.setState((state) => {
74.   return {
75.   curPage: state.curPage + 1
76.   }
77.   }, () => {
78.   this._loadData()
79.   })
80.   }
81.   }
82.
83.   // 下拉刷新的响应函数
84.   _handleRefresh() {
85.   this.setState({
86.   refresh: true
87.   })
88.
89.   // 此处可以异步获取后端接口数据，具体实现思路见上拉加载。
90.   setTimeout(() => {
91.   this.setState({
92.   refresh: false
93.   })
94.   }, 2000)
95.   }
96.
97.   // 加载数据
98.   // 注：这里的 key: value.id 由于模拟接口会出现重复的情况
99.   _loadData() {
100.  let data = this.props.store.list.slice(0, this.state.curPage * pageSize)
101.  let flatListData = data.map((value, index) => ({
102.  data: value,
```

```
103.   key: value.id
104.   })
105.   )
106.   this.setState({
107.   datalist: flatListData
108.   })
109.   }
110.
111.   // 执行第一次数据加载
112.   componentDidMount() {
113.   setTimeout((params) => {
114.   this._loadData()
115.   }, 0)
116.   }
117.
118.   render() {
119.   return (
120.   <FlatList
121.   ref={this.flatlist}
122.   renderItem={this._renderItem.bind(this)}
123.   data={this.state.datalist}
124.   refreshing={this.state.refresh}
125.   onEndReached={this._handleReachEnd.bind(this)}
126.   onEndReachedThreshold={1}
127.   onRefresh={this._handleRefresh.bind(this)}
128.   ></FlatList>
129.   )
130.   }
131.  }
```

style_list.js 样式

```
1.  import { StyleSheet } from 'react-native'
2.
3.  export default StyleSheet.create({
4.   listWrap: {
5.   flexDirection: 'row',
6.   padding: 10,
7.   borderBottomWidth: 1,
8.   borderStyle: 'solid',
9.   borderBottomColor: '#ccc'
```

```
    borderBottomColor: '#eee'
  },

  imgWrap: {
  width: 135,
  paddingRight: 10
  },

  image: {
  width: 115,
  height: 75
  },

  descWrap: {
  flex: 1
  },

  title: {
  fontSize: 20,
  lineHeight: 30
  },

  subtitle: {
  fontSize: 16,
  color: '#666',
  lineHeight: 30,
  overflow: 'hidden'
  },

  desc: {
  fontSize: 12,
  color: '#666'
  }
})
```

配置导航

本项目应用 React Navigation 构建路由系统。有关 React Navigation 请参阅〈千锋大前端小册〉系列之 React Navigation 部分》。

安装 React Navigation 环境

```
npm install @react-navigation/native

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

npm install @react-navigation/stack
```

给App.tsx配置路由

```
import React from 'react'
import { NavigationContainer } from '@react-navigation/native'
import { createStackNavigator } from '@react-navigation/stack'

import { Provider } from 'mobx-react'
import store from './store/'

import Index from './pages/index/Index'
import List from './pages/list/List'
import Detail from './pages/detail/Detail'

const Stack = createStackNavigator()

export default function App() {

 // 这里配置了三个页面
 return (
 <NavigationContainer>
 <Provider store={store}>
 <Stack.Navigator
 screenOptions={{
 headerStyle: {
 backgroundColor: '#ee7530'
 },
 headerTintColor: '#fff',
 headerTitleStyle: {
 fontWeight: 'bold',
 fontSize: 20
 }
```

```
    }}
    >
    <Stack.Screen
    name="Index"
    component={Index}
    options={{
    title: '首页'
    }}
    />
    <Stack.Screen
    name="List"
    component={List}
    options={{
    title: '热门'
    }}
    />
    <Stack.Screen
    name="Detail"
    component={Detail}
    options={{
    title: '详情'
    }}
    />
    </Stack.Navigator>
    </Provider>
    </NavigationContainer>
    )
  }
```

创建 Context

为了让组件能收到路由的信息，这里我们自己构建了一个 Context。

在根目录下创建一个context目录，在此目录下创建一个 `navigation.js` 文件，内容如下：

```
// context/navigations.js

import { createContext } from 'react'

const navigationContext = createContext()
```

```
6.
7.  let { Provider, Consumer } = navigationContext
8.
9.  export {
10.  navigationContext,
11.  Provider,
12.  Consumer
13.  }
```

修改 index.tsx

1.解构出Provider

```
1.  import { Provider } from '../../context/navigation'
```

2.通过Context 的Provider，将props递交给后代组件

```
1.  <Provider value={{...this.props}}>
2.   <Home></Home>
3.  </Provider>
4.
5.  <Provider value={{...this.props}}>
6.   <List></List>
7.  </Provider>
```

3.全部内容

```
1.  // pages/index/Index.tsx
2.
3.  import React, { Component, ContextType } from 'react'
4.  import TabNavigator from 'react-native-tab-navigator'
5.  import * as Device from 'expo-device'
6.
7.  // 解构出 Provider
8.  import { Provider } from '../../context/navigation'
9.
10.  import {
11.   View,
12.   Text
13.  } from 'react-native'
14.
15.  import {
16.   Img
```

```
  img
} from './styled_index'
import styles from './style_index'

import cookbook from '../../assets/images/cookbook.png'
import cookbookActive from '../../assets/images/cookbook-active.png'
import category from '../../assets/images/menu.png'
import categoryActive from '../../assets/images/menu-active.png'
import map from '../../assets/images/location.png'
import mapActive from '../../assets/images/location-active.png'
import more from '../../assets/images/more.png'
import moreActive from '../../assets/images/more-active.png'

import Home from '../home/Home'
import List from '../list/List'
import Detail from '../detail/Detail'

interface Props {
  navigation?: any
}

interface State {
  selectedTab: string
}

class Index extends Component<Props, State> {
  constructor(props: Props) {
  super(props)
  }

  state: State = {
  selectedTab: 'home'
  }


  componentDidMount() {

  }

  render() {
  return (
```

```
57.   <>
58.   <TabNavigator
59.   tabBarStyle={Device.deviceName === 'iPhone XR' ? styles.tabBarStyle : null}
60.   >
61.   <TabNavigator.Item
62.   selected={this.state.selectedTab === 'home'}
63.   title="美食大全"
64.   titleStyle={styles.titleStyle}
65.   selectedTitleStyle={styles.selectedTitleStyle}
66.   renderIcon={() => <Img source={cookbook} />}
67.   renderSelectedIcon={() => <Img source={cookbookActive} />}
68.   onPress={() => {
69.   this.setState({ selectedTab: 'home' })
70.   this.props.navigation.setOptions({ title: '美食大全' })
71.   }}
72.   >
73.   {/* 通过Context 的Provider，将props递交给后代组件 */}
74.   <Provider value={{...this.props}}>
75.   <Home></Home>
76.   </Provider>
77.   </TabNavigator.Item>
78.   <TabNavigator.Item
79.   selected={this.state.selectedTab === 'category'}
80.   title="热门"
81.   titleStyle={styles.titleStyle}
82.   selectedTitleStyle={styles.selectedTitleStyle}
83.   renderIcon={() => <Img source={category} />}
84.   renderSelectedIcon={() => <Img source={categoryActive} />}
85.   onPress={
86.   () => {
87.   this.setState({ selectedTab: 'category' })
88.   this.props.navigation.setOptions({ title: '热门' })
89.   }
90.   }
91.   >
92.   {/* 通过Context 的Provider，将props递交给后代组件 */}
93.   <Provider value={{...this.props}}>
94.   <List></List>
95.   </Provider>
96.   </TabNavigator.Item>
97.   <TabNavigator.Item
```

```
  selected={this.state.selectedTab === 'map'}
  title="地图"
  titleStyle={styles.titleStyle}
  selectedTitleStyle={styles.selectedTitleStyle}
  renderIcon={() => <Img source={map} />}
  renderSelectedIcon={() => <Img source={mapActive} />}
  onPress={() => {
  this.setState({ selectedTab: 'map' })
  this.props.navigation.setOptions({ title: '地图' })
  }}
  >
  {<View><Text>地图</Text></View>}
  </TabNavigator.Item>
  <TabNavigator.Item
  selected={this.state.selectedTab === 'more'}
  title="更多"
  titleStyle={styles.titleStyle}
  selectedTitleStyle={styles.selectedTitleStyle}
  renderIcon={() => <Img source={more} />}
  renderSelectedIcon={() => <Img source={moreActive} />}
  onPress={() => {
  this.setState({ selectedTab: 'more' })
  this.props.navigation.setOptions({ title: '更多' })
  }}
  >
  {<View><Text>更多</Text></View>}
  </TabNavigator.Item>
  </TabNavigator>
  </>
  )
  }
 }

 export default Index
```

修改 home.tsx

1. 将路由信息传给HotCate

```
<HotCate { ...this.props }></HotCate>
```

2. 定义Props

```
interface Props {
 navigation?: any
}
```

3. 全部内容

```
// pages/home/Home.tsx

import React, { Component } from 'react'
import { ScrollView, StatusBar } from 'react-native'

import Swiper from './Swiper'
import HotCate from './HotCate'
import Top10 from './Top10'

interface Props {
 navigation?: any
}

interface State {

}

class Home extends Component<Props, State> {
 render() {
 return (
 <ScrollView>
 <StatusBar backgroundColor="blue" barStyle="light-content" />
 <Swiper></Swiper>
 {/* 将路由信息传给HotCate */}
 <HotCate { ...this.props }></HotCate>
 <Top10></Top10>
 </ScrollView>
 )
 }
}

export default Home
```

修改 HotCate.tsx

修改 HotCate.tsx

1. 导入

```
import { Consumer } from '../../context/navigation'
```

2. 路由到“热门”页面

```
_onPress = (navigation) => {
 return () => {
 navigation.push('List')
 }
}

<View style={styles.container}>
 <Consumer>
 {
 ({navigation}) => {
 return (
 <Grid
 data={this.state.hotCate}
 renderItem={this._renderItem}
 hasLine={false}
 onPress={this._onPress(navigation)}
 ></Grid>
 )
 }
 }
 </Consumer>
</View>
```

3. 全部代码

```
// pages/home/HotCate.tsx

import React, { Component } from 'react'
import { Grid } from '@ant-design/react-native'
import { get } from '../../utils/http'
import { Consumer } from '../../context/navigation'

import styles from './style_home'

import {
```

```
  View,
  Text,
  Image
} from 'react-native'

interface Props {

}
interface State {
 hotCate: Array<object>
}

export default class HotCate extends Component<Props, State> {
 state = {
 hotCate: []
 }

 _renderItem(el, index) {
 return (
 <View
 style={styles.gridContainer}
 >
 {el.img ? <Image source={{uri: el.img}} style={styles.gridImg}></Image> : null}
 <Text style={styles.gridText}>{el.title}</Text>
 </View>
 )
 }

 _onPress = (navigation) => {
 return () => {
 navigation.push('List')
 }
 }

 async componentDidMount() {
 let hotCate = await get('http://localhost:9000/api/hotcate')

 // 补全最后一项数据
 hotCate.push({
 img: '',
 title: '更多...'
```

```
52.   })
53.
54.   this.setState({
55.   hotCate
56.   })
57.   }
58.
59.   render() {
60.   return (
61.   <View style={styles.container}>
62.   <Consumer>
63.   {
64.   ({navigation}) => {
65.   return (
66.   <Grid
67.   data={this.state.hotCate}
68.   renderItem={this._renderItem}
69.   hasLine={false}
70.   onPress={this._onPress(navigation)}
71.   ></Grid>
72.   )
73.   }
74.   }
75.   </Consumer>
76.   </View>
77.   )
78.   }
79.  }
```

修改 Top10.tsx

1. 通过 contextType 定义 context

```
1.   import { navigationContext } from '../../context/navigation'
```

2. 导航到详情页，并传参

```
1.   import { navigationContext } from '../../context/navigation'
2.
3.   static contextType = navigationContext
4.
5.   onPress = (e) => {
```

```
_onPress = (e) => {
 this.context.navigation.push('Detail', { name: e.name })
}

<Grid
 data={this.props.store.top10}
 columnNum={2}
 hasLine={false}
 renderItem={this._renderTop10}
 onPress={this._onPress}
/>
```

3. 全部代码

```
// pages/home/Top10.tsx

import React, { Component } from 'react'
import { Grid } from '@ant-design/react-native'
import { observer, inject } from 'mobx-react'
import { navigationContext } from '../../context/navigation'

import {
 View,
 Text,
 Image
} from 'react-native'

import styles from './style_home.js'

interface Props {
 // store 作为组件的 props
 store?: any
}

interface State {

}

// 注入 store 与 将类变为可观察的对象
@inject('store')
@observer
class Top10 extends Component<Props, State> {
```

```
29.
30.   static contextType = navigationContext
31.
32.   _renderTop10(el, index) {
33.   return (
34.   <View style={styles.top10ItemContainer}>
35.   <View style={styles.top10ImgContainer}>
36.   <Image style={styles.Top10Img} source={{uri: el.img}}></Image>
37.   </View>
38.   <View style={styles.top10DesContainter}>
39.   <Text style={styles.top10Titie}>{el.name}</Text>
40.   <Text style={styles.Top10Desc}>{el.all_click} {el.favorites}</Text>
41.   </View>
42.   </View>
43.   )
44.   }
45.
46.   _onPress = (e) => {
47.   this.context.navigation.push('Detail', { name: e.name })
48.   }
49.
50.   render() {
51.   return (
52.   <View style={styles.top10Container}>
53.   <View style={styles.top10Head}>
54.   <Text style={styles.top10HeadText}>精品好菜</Text>
55.   </View>
56.   <View style={styles.gridContainer}>
57.   <Grid
58.   data={this.props.store.top10}
59.   columnNum={2}
60.   hasLine={false}
61.   renderItem={this._renderTop10}
62.   onPress={this._onPress}
63.   />
64.   </View>
65.   </View>
66.   )
67.   }
68. }
69.
```

```
export default Top10
```

修改 List.tsx

1. 载入路由相关模块，实现路由到详情页的功能，主要代码：

```
// 1. 载入Context
import { navigationContext } from '../../context/navigation'

// 2. 在 Props 里定义 navigation
interface Props {
 store?: any,
 navigation?: any
}

// 3. 在类里定义 contextType 静态变量
static contextType = navigationContext

// 4. 在组件类里定义路由跳转响应方法
_onPress = (name: string) => {
 return () => {
 // 鉴于此页面从 TabBar 和 首页两个入口进入
 // 路由跳转的方式也不同
 if (this.context) {
 // 从Tabbar进入
 this.context.navigation.push('Detail', {name})
 } else {
 // 从首页进入
 this.props.navigation.push('Detail', {name})
 }
 }
}

// 5. 应用 TouchableOpacity 组件绑定路由跳转事件
<TouchableOpacity
 onPress={this._onPress(name)}
>
 <View style={styles.listWrap}>
 <View style={styles.imgWrap}>
 <Image style={styles.image} source={{uri: img}}></Image>
```

```
    </View>
    <View style={styles.descWrap}>
    <Text style={styles.title}>{name}</Text>
    <Text style={styles.subtitle} numberOfLines={1}>{burdens}</Text>
    <Text style={styles.desc}>{all_click} {favorites}</Text>
    </View>
    </View>
</TouchableOpacity>
```

2. 全部代码

```
import React, { Component, createRef } from 'react'
import { navigationContext } from '../../context/navigation'

import {
  inject,
  observer
} from 'mobx-react'

import {
  View,
  Text,
  Image,
  FlatList,
  TouchableOpacity
} from 'react-native'

import styles from './style_list'

interface Props {
  store?: any,
  navigation?: any
}

interface State {
  // 记录上拉加载更多的当前页码
  curPage: number,

  // 页面显示的数据
  datalist: Array<object>,

  // 控制下拉刷新的开关
```

```
  refresh: boolean
}

let pageSize = 10

@inject('store')
@observer
export default class List extends Component<Props, State> {
  constructor (
  public props: Props,
  public flatlist,
  ) {
  super(props)
  this.flatlist = createRef()
  }

  state = {
  curPage: 1,
  datalist: [],
  refresh: false
  }

  static contextType = navigationContext

  _onPress = (name: string) => {
  return () => {
  if (this.context) {
  this.context.navigation.push('Detail', {name})
  } else {
  this.props.navigation.push('Detail', {name})
  }
  }
  }

  // 渲染 Flatlist 组件数据
  _renderItem(item) {
  let {img, name, burdens, all_click, favorites} = item.item.data
  return (
  <TouchableOpacity
  onPress={this._onPress(name)}
  >
```

```
<View style={styles.listWrap}>
<View style={styles.imgWrap}>
<Image style={styles.image} source={{uri: img}}></Image>
</View>
<View style={styles.descWrap}>
<Text style={styles.title}>{name}</Text>
<Text style={styles.subtitle} numberOfLines={1}>{burdens}</Text>
<Text style={styles.desc}>{all_click} {favorites}</Text>
</View>
</View>
</TouchableOpacity>
)
}

// 处理用户拉到底端的响应函数
_handleReachEnd() {
// 如果还有数据，一直加载
// 加载更多，由于Mock数据问题，有ID重复问题
// if (this.state.curPage < Math.ceil(this.props.store.list.length / pageSize)) {
// console.log(this.state.curPage)
// this.setState((state) => {
// return {
// curPage: state.curPage + 1
// }
// }, () => {
// this._loadData()
// })
// }
}

// 下拉刷新的响应函数
_handleRefresh() {
this.setState({
refresh: true
})

// 此处可以异步获取后端接口数据，具体实现思路见上拉加载。
setTimeout(() => {
this.setState({
refresh: false
})
```

```
});
}, 2000)
}

// 加载数据
_loadData() {
let data = this.props.store.list.slice(0, this.state.curPage * pageSize)
let flatListData = data.map((value, index) => ({
data: value,
key: value.id
})
)
this.setState({
datalist: flatListData
})
}

// 执行第一次数据加载
componentDidMount() {
setTimeout((params) => {
this._loadData()
}, 0)
}

render() {
return (
<View style={styles.container}>
<FlatList
ref={this.flatlist}
renderItem={this._renderItem.bind(this)}
data={this.state.datalist}
refreshing={this.state.refresh}
onEndReached={this._handleReachEnd.bind(this)}
onEndReachedThreshold={1}
onRefresh={this._handleRefresh.bind(this)}
></FlatList>
</View>
)
}
}
```

构建 Detail 页面

在路由信息定义好后，就可以构建详情页了。

Detail.tsx

```
// pages/detail/Detail.tsx

import React, { Component } from 'react'
import { get } from '../../utils/http'
import {
 View,
 ScrollView,
 Text,
 Image,
 StatusBar,
 TouchableOpacity,
 Alert
} from 'react-native'

import styles from './style_detail'

interface Props {
 navigation?: any,
 route?: any
}
interface State {
 detail: {}
}

export default class Detail extends Component<Props, State> {
 state = {
 detail: null
 }

 async componentDidMount() {
 let result = await get('http://localhost:9000/api/detail')
 this.setState({
 detail: result
```

```
34.   })
35.   // 根据路由传递过来参数，修改本页的 title
36.   this.props.navigation.setOptions({ title: this.props.route.params.name })
37.   }
38.
39.   render() {
40.   let detail = this.state.detail
41.   return (
42.   <>
43.   {
44.   detail && (
45.   <ScrollView>
46.   <View style={styles.container}>
47.   <StatusBar backgroundColor="blue" barStyle="light-content" />
48.   <Image
49.   source={{uri: detail.img}}
50.   style={styles.mainImg}
51.   ></Image>
52.   <View style={styles.mainInfo}>
53.   <View>
54.   <Text style={styles.mainInfoName}>{detail.name}</Text>
55.   </View>
56.   <View>
57.   <Text style={styles.mainInfoSubTitle}>{detail.all_click}浏览/{detail.favorites}收藏
</Text>
58.   </View>
59.   <TouchableOpacity
60.   onPress={() => Alert.alert('已经收藏.')}
61.   >
62.   <View style={styles.mainInfoButtonWrap}>
63.   <Text style={styles.mainInfoButton}>收藏</Text>
64.   </View>
65.   </TouchableOpacity>
66.   </View>
67.   <View style={styles.infoWrap}>
68.   <View>
69.   <Text style={styles.infoTitle}>心得</Text>
70.   </View>
71.   <View>
72.   <Text style={styles.infoText}>
73.   {detail.info}
```

```
74.   </Text>
75.   </View>
76.
77.   <View>
78.   <Text style={[styles.infoTitle, {marginTop: 20}]}>做法</Text>
79.   </View>
80.   <View>
81.   {
82.   detail.makes.map((value) => {
83.   return (
84.   <View
85.   key={value.num}
86.   >
87.   <View>
88.   <Text style={styles.makesTitle}>{value.num} {value.info}</Text>
89.   </View>
90.   <View>
91.   <Image
92.   source={{uri: value.img}}
93.   style={styles.makesImg}
94.   ></Image>
95.   </View>
96.   </View>
97.   )
98.   })
99.   }
100.  </View>
101.  </View>
102.  </View>
103.  </ScrollView>
104.  )
105.  }
106.  </>
107.  )
108.  }
109. }
```

style_detail.js 页面样式

```
1.  // pages/detail/style_detail.js
```

```
2.
3.  import { StyleSheet } from 'react-native'
4.
5.  export default StyleSheet.create({
6.   container: {
7.   flex: 1,
8.   backgroundColor: '#eee',
9.   paddingBottom: 34
10.  },
11.
12.  // main
13.  mainImg: {
14.  width: '100%',
15.  height: 250
16.  },
17.
18.  mainInfo: {
19.  height: 170,
20.  justifyContent: 'center',
21.  alignItems: 'center',
22.  backgroundColor: '#fff'
23.  },
24.
25.  mainInfoName: {
26.  fontSize: 24
27.  },
28.
29.  mainInfoSubTitle: {
30.  marginTop: 10,
31.  fontSize: 12,
32.  color: '#666'
33.  },
34.
35.  // button
36.  mainInfoButtonWrap: {
37.  width: 140,
38.  height: 40,
39.  backgroundColor: '#df7b42',
40.  marginTop: 20,
41.  borderRadius: 6
42.  },
```

```
43.
44.   mainInfoButton: {
45.   textAlign: 'center',
46.   lineHeight: 40,
47.   color: '#fff',
48.   fontSize: 16
49.   },
50.
51.   // info
52.   infoWrap: {
53.   marginTop: 15,
54.   backgroundColor: '#fff',
55.   paddingTop: 25,
56.   paddingLeft: 15,
57.   paddingBottom: 25
58.   },
59.
60.   infoTitle: {
61.   fontSize: 20,
62.   fontWeight: 'bold',
63.   marginBottom: 20
64.   },
65.
66.   infoText: {
67.   fontSize: 16,
68.   lineHeight: 20,
69.   paddingRight: 20
70.   },
71.
72.   // makes
73.   makesTitle: {
74.   fontSize: 16,
75.   paddingRight: 30
76.   },
77.
78.   makesImg: {
79.   width: 300,
80.   height: 210,
81.   margin: 20
82.   }
83.  })
```

构建 Map.tsx 美食地图页面

“美食地图” 主要应用的知识点是 WebView，根据官网推荐，使用 react-native-webview 项目来实现在RN里嵌入Web页面。

模块安装

```
npm install --save react-native-webview

react-native link react-native-webview
```

Map.tsx 文件构建

在项目根目录 pages 下创建目录 map，在 map 目录下创建 Map.tsx 文件，文件内容如下：

```
import React, { Component } from 'react'
import { View } from 'react-native'
import { WebView } from 'react-native-webview'

interface Props {

}
interface State {

}

export default class Map extends Component<Props, State> {
 state = {}

 render() {
 return (
 <View
 style={{
 width: '100%',
 flex: 1
 }}
 >
```

```
  <WebView
  source={{ uri:
'https://map.baidu.com/search/%E7%BE%8E%E9%A3%9F/@12959238.56,4825347.47,12z?querytype=s&da_src=shareurl&wd=%E7%BE%8E%E9%A3%9F&c=131&src=0&pn=0&sug=0&l=12&b=(12905478.56,4795011.47;13012998.56,4855683.47)&from=webmap&biz_forward=%7B%22scaler%22:2,%22styles%22:%22pl%22%7D&device_ratio=2' }}
  style={{
  width: '100%',
  height: '100%'
  }}
  />
  </View>
  )
  }
 }
```

构建 More.tsx 页面

更多页面实现了两个功能：

1、是否显示地图页签

2、拍照功能

是否显示地图页签

更新 store

添加了 isShow 属性，和 setVisible 方法。

```
// store/index.js

import {
 observable,
 action,
 computed
} from 'mobx'

class Store {
 // swiper 与 top10 共享的数据
```

```
  @observable
  list = []

  // 定义是否显示地图按钮
  @observable
  isShow = true

  // swiper 数据过滤
  @computed
  get swiper() {
  return this.list.slice(0, 5).map((value, index) => {
  return {
  img: value.img
  }
  })
  }

  // top10 数据过滤
  @computed
  get top10() {
  return this.list.slice(0, 10).map((value, index) => {
  return {
  img: value.img,
  all_click: value.all_click,
  favorites: value.favorites,
  name: value.name
  }
  })
  }

  // 装载 list 数据
  @action.bound
  setList(data) {
  this.list = data
  }

  // 修改是否显示地图按钮
  @action.bound
  setVisible(status) {
  this.isShow = status
  }
```

```
}

export default new Store()
```

添加 More.tsx 文件

在根目录pages下创建 more 文件夹，再创建 More.tsx 文件，内容如下

```
// pages/more/More.tsx

import React, { Component } from 'react'
import { View, Text, Switch, AsyncStorage } from 'react-native'
import { observer, inject } from 'mobx-react'

interface Props {
 store?: any
}

interface State {

}

@inject('store')
@observer
export default class Profile extends Component<Props, State> {
 state = {

 }

 async componentDidMount() {

 }

 render() {
 return (
 <View>
 <View style={{
 flexDirection: 'row',
 justifyContent: 'flex-start',
 alignItems: 'flex-start',
 padding: 20
```

```
34.   }}>
35.   <View style={{
36.   height: 30,
37.   justifyContent: 'center',
38.   alignItems: 'center'
39.   }}>
40.   <Text>是否显示地图：</Text>
41.   </View>
42.   <Switch
43.   value={this.props.store.isShow}
44.   onValueChange={(value) => {
45.   this.props.store.setVisible(value)
46.   AsyncStorage.setItem('isShow', value.toString())
47.   }}
48.   ></Switch>
49.   </View>
50.   </View>
51.   )
52.   }
53.  }
```

修改 pages/index/Index.tsx 文件

1、修改代码的要点

```
1.   // 定义 store
2.   interface Props {
3.   navigation?: any
4.   store?: any
5.   }
6.
7.   // 记录用户缓存
8.   async componentDidMount() {
9.   let isShow = await AsyncStorage.getItem('isShow')
10.   this.props.store.setVisible(JSON.parse(isShow))
11.   }
12.
13.   // 在 tabbar 里修改
14.   {
15.   this.props.store.isShow
16.   ? (
17.   <TabNavigator.Item
```

```
  selected={this.state.selectedTab === 'map'}
  title="地图"
  titleStyle={styles.titleStyle}
  selectedTitleStyle={styles.selectedTitleStyle}
  renderIcon={() => <Img source={map} />}
  renderSelectedIcon={() => <Img source={mapActive} />}
  onPress={() => {
  this.setState({ selectedTab: 'map' })
  this.props.navigation.setOptions({ title: '地图' })
  }}
  >
  <Map></Map>
  </TabNavigator.Item>
  )
  : null
  }
```

2. 全部代码

```
// pages/index/Index.tsx

import React, { Component, ContextType } from 'react'
import TabNavigator from 'react-native-tab-navigator'
import * as Device from 'expo-device'
import { observer, inject } from 'mobx-react'

import { Provider } from '../../context/navigation'

import {
 View,
 Text,
 AsyncStorage
} from 'react-native'

import {
 Img
} from './styled_index'
import styles from './style_index'

import cookbook from '../../assets/images/cookbook.png'
import cookbookActive from '../../assets/images/cookbook-active.png'
```

```
import category from '../../assets/images/menu.png'
import categoryActive from '../../assets/images/menu-active.png'
import map from '../../assets/images/location.png'
import mapActive from '../../assets/images/location-active.png'
import more from '../../assets/images/more.png'
import moreActive from '../../assets/images/more-active.png'

import Home from '../home/Home'
import List from '../list/List'
import Map from '../map/Map'
import More from '../more/More'

interface Props {
 navigation?: any
 store?: any
}

interface State {
 selectedTab: string
}

@inject('store')
@observer
class Index extends Component<Props, State> {
 constructor(props: Props) {
 super(props)
 }

 state: State = {
 selectedTab: 'home'
 }

 async componentDidMount() {
 let isShow = await AsyncStorage.getItem('isShow')
 this.props.store.setVisible(JSON.parse(isShow))
 }

 render() {
 return (
 <>
 <TabNavigator
```

```
64.   tabBarStyle={Device.deviceName === 'iPhone XR' ? styles.tabBarStyle : null}
65.   >
66.   <TabNavigator.Item
67.   selected={this.state.selectedTab === 'home'}
68.   title="美食大全"
69.   titleStyle={styles.titleStyle}
70.   selectedTitleStyle={styles.selectedTitleStyle}
71.   renderIcon={() => <Img source={cookbook} />}
72.   renderSelectedIcon={() => <Img source={cookbookActive} />}
73.   onPress={() => {
74.   this.setState({ selectedTab: 'home' })
75.   this.props.navigation.setOptions({ title: '美食大全' })
76.   }}
77.   >
78.   <Provider value={{...this.props}}>
79.   <Home></Home>
80.   </Provider>
81.   </TabNavigator.Item>
82.   <TabNavigator.Item
83.   selected={this.state.selectedTab === 'category'}
84.   title="热门"
85.   titleStyle={styles.titleStyle}
86.   selectedTitleStyle={styles.selectedTitleStyle}
87.   renderIcon={() => <Img source={category} />}
88.   renderSelectedIcon={() => <Img source={categoryActive} />}
89.   onPress={
90.   () => {
91.   this.setState({ selectedTab: 'category' })
92.   this.props.navigation.setOptions({ title: '热门' })
93.   }
94.   }
95.   >
96.   <Provider value={{...this.props}}>
97.   <List></List>
98.   </Provider>
99.   </TabNavigator.Item>
100.  {
101.  this.props.store.isShow
102.  ? (
103.  <TabNavigator.Item
104.  selected={this.state.selectedTab === 'map'}
```

```
105.   title="地图"
106.   titleStyle={styles.titleStyle}
107.   selectedTitleStyle={styles.selectedTitleStyle}
108.   renderIcon={() => <Img source={map} />}
109.   renderSelectedIcon={() => <Img source={mapActive} />}
110.   onPress={() => {
111.   this.setState({ selectedTab: 'map' })
112.   this.props.navigation.setOptions({ title: '地图' })
113.   }}
114.   >
115.   <Map></Map>
116.   </TabNavigator.Item>
117.   )
118.   : null
119.   }
120.   <TabNavigator.Item
121.   selected={this.state.selectedTab === 'more'}
122.   title="更多"
123.   titleStyle={styles.titleStyle}
124.   selectedTitleStyle={styles.selectedTitleStyle}
125.   renderIcon={() => <Img source={more} />}
126.   renderSelectedIcon={() => <Img source={moreActive} />}
127.   onPress={() => {
128.   this.setState({ selectedTab: 'more' })
129.   this.props.navigation.setOptions({ title: '更多' })
130.   }}
131.   >
132.   <More></More>
133.   </TabNavigator.Item>
134.   </TabNavigator>
135.   </>
136.   )
137.   }
138. }
139.
140. export default Index
```

构建 More.tsx 页面

更多页面实现了两个功能

更多页面实现了两个功能：

1、是否显示地图页签

2、拍照功能

拍照功能

安装模块

```
npm install expo-camera -S
```

改写 More.tsx 代码

以下代码是 拍照 和 切换显示地图按钮 的全部代码。

```
import React, { Component } from 'react'
import { View, Text, Switch, AsyncStorage, TouchableOpacity, Image } from 'react-native'
import { observer, inject } from 'mobx-react'

import * as Permissions from 'expo-permissions'
import { Camera } from 'expo-camera'

interface Props {
  store?: any
}

interface State {
  hasCameraPermission: boolean
  type: boolean
  isTakePic: boolean,
  picUri: string
}

@inject('store')
@observer
export default class Profile extends Component<Props, State> {
  camera = null
```

```
23.
24.   state = {
25.   hasCameraPermission: null,
26.   type: Camera.Constants.Type.back,
27.   isTakePic: false,
28.   picUri: 'http://placehold.it/240x180'
29.   }
30.
31.   async componentDidMount() {
32.   const { status } = await Permissions.askAsync(Permissions.CAMERA);
33.   this.setState({
34.   hasCameraPermission: status === 'granted'
35.   })
36.   }
37.
38.   showTakePicScene() {
39.   this.setState({
40.   isTakePic: true
41.   })
42.   }
43.
44.   async takePicture() {
45.   let result = await this.camera.takePictureAsync()
46.   this.setState({
47.   isTakePic: false,
48.   picUri: result.uri
49.   })
50.   }
51.
52.   render() {
53.   return (
54.   <>
55.   {
56.   this.state.isTakePic
57.   ? (
58.   <Camera
59.   style={{ flex: 1 }}
60.   type={this.state.type}
61.   ref={ref => {
62.   this.camera = ref
63.   }}
```

```
  >
  <TouchableOpacity
  onPress={this.takePicture.bind(this)}
  >
  <View style={{
  marginLeft: 20,
  width: 80,
  height: 40,
  backgroundColor: '#f9efd4',
  justifyContent: 'center',
  alignItems: 'center'
  }}>
  <Text>拍照</Text>
  </View>
  </TouchableOpacity>
  </Camera>
  )
  : (
  <View>
  <View style={{
  flexDirection: 'row',
  justifyContent: 'flex-start',
  alignItems: 'flex-start',
  padding: 20
  }}>
  <View style={{
  height: 30,
  justifyContent: 'center',
  alignItems: 'center'
  }}>
  <Text>是否显示地图：</Text>
  </View>
  <Switch
  value={this.props.store.isShow}
  onValueChange={(value) => {
  this.props.store.setVisible(value)
  AsyncStorage.setItem('isShow', value.toString())
  }}
  ></Switch>
  </View>
  <TouchableOpacity
```

```
  onPress={this.showTakePicScene.bind(this)}
  >
  <View style={{
  marginLeft: 20,
  width: 80,
  height: 40,
  backgroundColor: '#df7b42',
  justifyContent: 'center',
  alignItems: 'center'
  }}>
  <Text style={{color: '#fff'}}>拍照</Text>
  </View>
  </TouchableOpacity>
  <View>
  <Image style={{marginLeft: 20, marginTop: 20, width: 240, height: 180}} source={{uri: this.state.picUri}}></Image>
  </View>
  </View>
  )
  }
  </>
  )
  }
 }
```

项目发布

本项目发布利用expo发布功能，详细可参考 构建独立的应用程序

安装 Expo CLI

此步骤已经完成。

配置 app.json

```
 {
  "expo": {
  "name": "rn-cookbooks",
```

```
  "slug": "rn-cookbooks",
  "privacy": "public",
  "sdkVersion": "36.0.0",
  "platforms": [
  "ios",
  "android",
  "web"
  ],
  "version": "1.0.0",
  "orientation": "portrait",
  "icon": "./assets/icon.png",
  "splash": {
  "image": "./assets/splash.png",
  "resizeMode": "contain",
  "backgroundColor": "#ffffff"
  },
  "updates": {
  "fallbackToCacheTimeout": 0
  },
  "assetBundlePatterns": [
  "**/*"
  ],
  "ios": {
  "bundleIdentifier": "com.qianfeng.felixlu",
  "buildNumber": "1.0.0"
  },
  "android": {
  "package": "com.qianfeng.felixlu",
  "versionCode": 1
  }
  }
}
```

开始Build

```
expo build:android
```

或:

```
expo build:ios
```

build 过程监测

build 过程监测

输入以上命令后，会在控制台看到下边信息：

```
Build started, it may take a few minutes to complete.
You can check the queue length at https://expo.io/turtle-status

You can monitor the build at

https://expo.io/dashboard/felixlurt/builds/15b2ae11-c98d-48dc-879e-9ff05fb0b9f1
```

可以通过访问 `https://expo.io/dashboard/felixlurt/builds/15b2ae11-c98d-48dc-879e-9ff05fb0b9f1` 来监控build过程。

build 成功后，点击 “Download” 按钮即可下载打完的APP安装包了。

> *注：iOS 需要有开发者账号，没有账号的同学建议运行 `expo build:android` 进行试验*