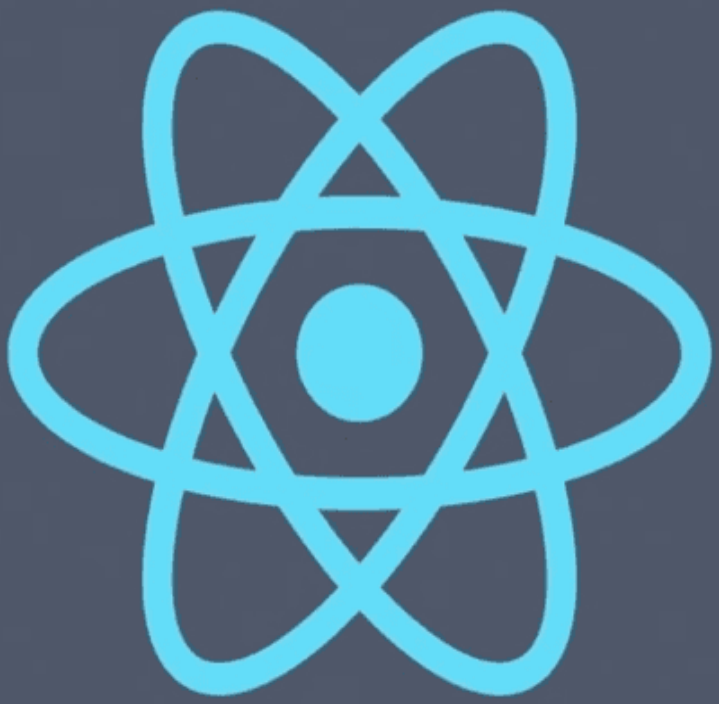


React Navtive 框架教程

前言

> 原文链接 : React Native Tutorial: Building Apps with JavaScript
> 原文作者 : ColinEberhardt
> 译文出自 : 开发技术前线 www.devtf.cn
> 译者 : kmyhy
> 校对者 : lastdays
> 状态 : 完成几个月前Facebook推出了React Native框架，允许开发者用JavaScript编写本地iOS App——今天，官方代码库的beta版本终于放出！

早在几年前，开发者就已经在使用JavaScript和HTML5加上PhoneGap编译器来编写iOS App了，因此React Native框架是不是有点多余？

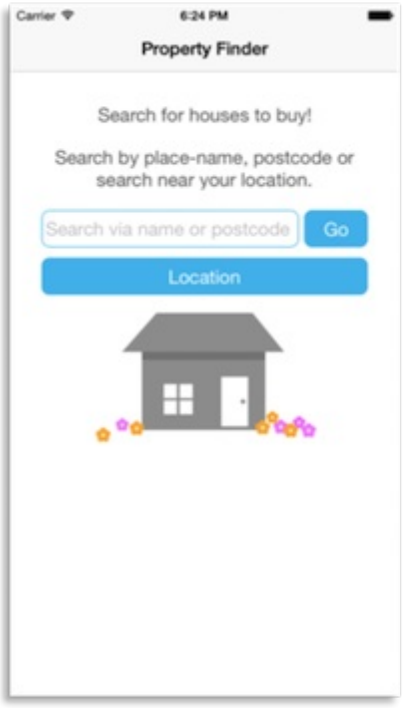
但React Native确实是一个很了不起的东东，开发者为之欢欣鼓舞，这是因为：
1. 通过React Native框架，你可以用JavaScript来编写和运行应用程序逻辑，而UI却可以是真正的本地代码编写的；因此，你完全不需要一个HTML5编写的UI。
2. React框架采用了一种新颖的、激进的和高度函数式的方式来构建UI。简单说，应用程序的UI可以简单地用一个函数来表示应用程序当前的状态。

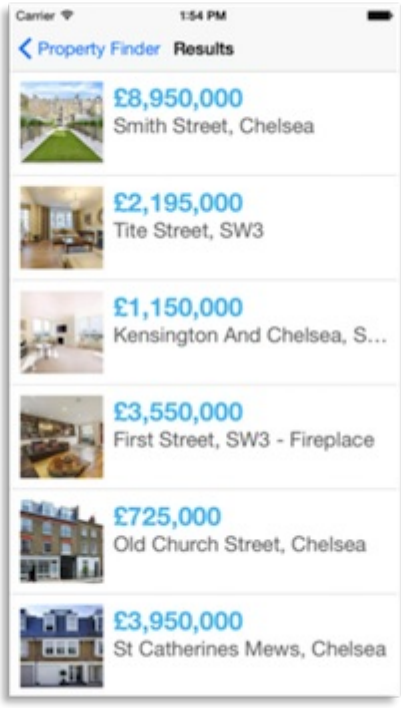
React Native的重点是把React编程模型引进到移动App的开发中去。它的目的并不是跨平台，一次编写到处运行。它真正的目标是“一次学习多处编写”。这是一个重大的区别。本教程只涉及iOS，但一旦你学会了它的思想，你就可以快速将同样的知识应用到Android App的编写上。

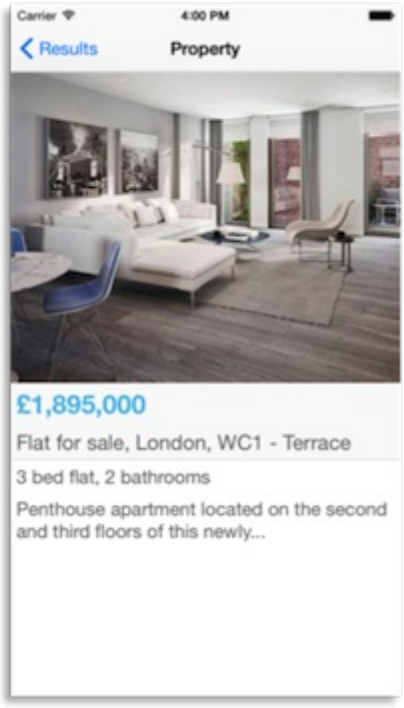
如果你过去只使用O-C或Swift编写过iOS应用程序，你可能不会因为可以用JavaScript编写iOS App而激动不已。但是，作为一个Swift开发者，上面所说的第二点原因肯定会激起你的兴趣！

通过Swift，你毫无疑问已经学习到那些新颖的、函数式的编写代码的方法；以及一些和过去不同或相同的技术。但是，你构建UI的方式仍然和O-C时代没有太大的不同：仍然要离不开UIKit。

通过一些有趣的概念，比如虚拟DOM和reconciliation，React直接将函数式编程的理念用到了UI层面。这个教程带你一起构建一个搜索英国房产信息的应用：







如果你之前从未写过任何 JavaScript ，别担心；这篇教程带着你一点一点编写代码。React 使用 CSS 属性来定义样式，这些样式通常都很易于阅读和理解，但是如果你想进一步了解，可以参考 Mozilla Developer Network reference。

要想学习更多内容，请往下看！

开始

React Native 框架托管在GitHub。要获得这个框架，你可以使用git命令克隆项目到本地，或者直接下载zip包。 如果你不想使用源代码，也可以用命令行界面（CLI）创建React Native项目，本文将使用这种方式。

React Native 使用了 Node.js，如果你的机器上没有安装Node.js，请先安装它。
首先需要安装 Homebrew，安装指南请参考Homebrew网站。然后用brew命令来安装Node.js:

```
brew install node
```

然后安装 watchman（Facebook推出的文件改动监听器）:

```
brew install watchman
```

React Native通过watchman来监视代码文件的改动并适时进行编译。这就好比Xcode，它会在每次文件被保存时对文件进行编译。
然后用npm命令安装React Native 的CLI工具：

```
npm install  -g  react-native-cli
```

这个命令通过Node Package Manager来下载和安装CLI工具，npm是一个类似CocoPods或Carthage工具。
定位到要创建React Native 项目的文件夹，使用CLI工具创建一个新的React Native项目：

```
react-native  init PropertyFinder
```

这将创建一个默认的React Native项目，其中包含有能够让React Native项目编译运行的必要内容。
在React Native项目文件夹中，有一个node_modules文件夹，它包含React Native 框架文件。此外还有一个 index.ios.js 文件，这是CLI创建的脚手架代码。最后，还有一个Xcode项目文件及一个iOS文件夹，后者会有一些iOS代码用于引导React Navtive App。
打开Xcode项目文件，build&run。模拟器启动并显示一句问候语：

与此同时Xcode还会打开一个终端窗口，并显示如下信息：

```
================================================================
| Running packager on port 8081.
| Keep this packager running while developing on any JS
| projects. Feel free to close this tab and run your own
| packager instance if you prefer.
|
| https://github.com/facebook/react-native
|
================================================================

Looking for JS files in /Users/colineberhardt/Temp/TestProject
React packager ready.
```

这是React Navtive Packager，它在node容器中运行。你待会就会发现它的用处。

千万不要关闭这个窗口，让它一直运行在后面。如果你意外关闭它，可以在Xcode中先停止程序，再重新运行程序。

> 注意:
> 在开始接触具体的代码之前（在本教程中，主要是js代码），我们将推荐 Sublime Text这个文本编辑工具，因为Xcode并不适合用于编写js代码的。当然，你也可以使用 atom, brackets 等其他轻量级的工具来替代。

你好， React Native

在开始编写这个房产搜索App之前，我们先来创建一个简单的Hello World项目。我们将通过这个例子来演示React Native的各个组件和概念。
用你喜欢的文本编辑器（例如Sublime Text）打开index.ios.js ，删除所有内容。然后加入以下语句：

`'use strict';`

这将开启严谨模式，这会改进错误的处理并禁用某些js语法特性，这将让JavaScript表现得更好。

> 注意: 关于严谨模式，读者可以参考 Jon Resig的文章： “ECMAScript 5 Strict Mode, JSON, and More” 。

然后加入这一句：

`var React = require('react-native');`

这将加载 react-native 模块，并将其保存为React变量。React Native 使用和Node.js 一样的 require 函数来加载模块，类似于Swift中的import语句。

> 注意:
> 关于JavaScript 模块的概念，请参考 Addy Osmani的这篇文章。

然后加入如下语句：

```
var styles = React.StyleSheet.create({
  text: {
    color: 'black',
    backgroundColor: 'white',
    fontSize: 30,
    margin: 80
  }
});
```

这将定义一个css样式，我们将在显示“Hello World”字符串时应用这个样式。
在React Native 中，我们可以使用 Cascading Style Sheets (CSS) 语法来格式化UI样式。
接下来敲入如下代码:

```
class PropertyFinderApp extends React.Component {
  render() {
    return React.createElement(React.Text, {style: styles.text}, "Hello World!");
  }
}
```

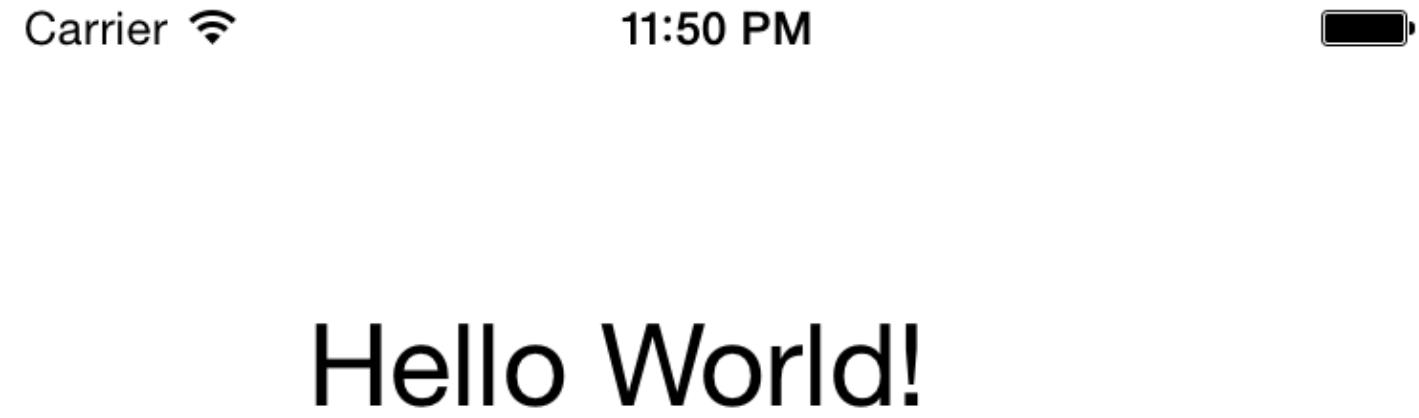
这里我们定义了一个JavaScript 类。JavaScript类的概念出现自ECMAScript 6。由于JavaScript是一门不断演变的语言，因此web开发者必须保持与浏览器的向下兼容。由于React Native基于JavaScriptCore，因此我们完全可以放心使用它的现代语法特性，而不需要操心与老版本浏览器兼容的问题。

> 注意:如果你是Web开发人员，我建议你使用新的JavaScript语法。有一些工具比如 Babel，可以将现代JavaScript语法转变为传统JavaScript语法，这样就能和老式浏览器进行兼容。

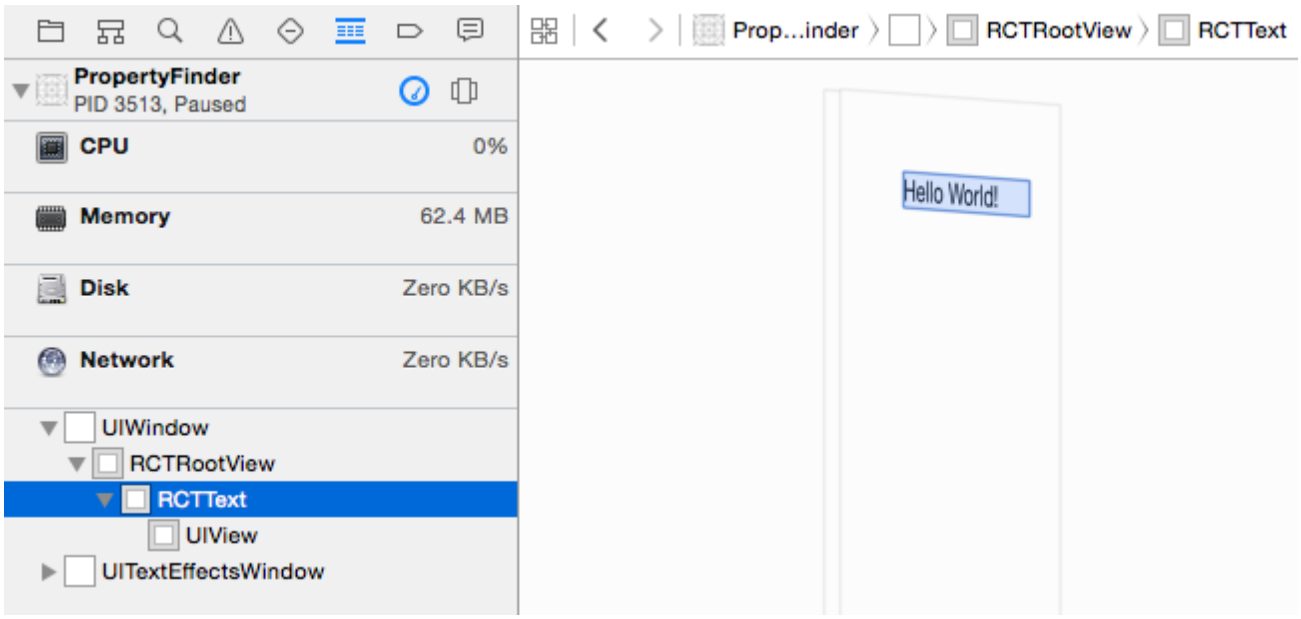
PropertyFinderApp 类继承自 React.Componen，后者是React UI中的基础。Components包含了一些不可变属性、可变属性和一些渲染方法。当然，这个简单App中，我们就用到一个render方法。
React Native 的Components不同于UIKit 类，它们是轻量级的对象。框架会将React Components转换为对应的本地UI对象。
最后敲入如下代码：

```
React.AppRegistry.registerComponent('PropertyFinder', function() { return PropertyFinderApp });
```

AppRegistry 代表了App的入口以及根组件。保存文件，返回Xcode。确保当前Scheme为PropertyFinder ，然后在模拟器运行App。你将看到如下效果：



看到了吧，模拟器将JavaScript代码渲染为本地UI组件，你不会看到任何浏览器的痕迹。
你可以这样来确认一下：
在Xcode中，选中 Debug\View Debugging\Capture View Hierarchy，查看本地视图树。你将找不到任何UIWebView实例。



你一定会惊奇这一切是怎么发生的。在 Xcode 中打开 AppDelegate.m，找到 application:didFinishLaunchingWithOptions:方法。
在这个方法中，创建了一个RCTRootView，该对象负责加载JavaScript App并渲染相关视图。
App一启动，RCTRootView会加载如下URL的内容:

http://localhost:8081/index.ios.bundle

还记得App启动时弹出的终端窗口吗？终端窗口中运行的packager和server会处理上述请求。
你可以用Safari来打开上述URL，你将会看到一些JavaScript代码。在React Native 框架代码中你会找到“Hello World”相关代码。

当App打开时，这些代码会被加载并执行。以我们的App来说，PropertyFinderApp组件会被加载，然后创建相应的本地UI组件。

你好， JSX

前面我们用React.createElement构建了一个简单的UI ，React 会将之转换为对应的本地对象。但对于复杂UI来说（比如那些组件嵌套的UI），代码会变得非常难看。
确保App保持运行，回到文本编辑器，修改index.ios.js中的return语句为：

```
return <React.Text style={styles.text}>Hello World (Again)</React.Text>;
```

这里使用了JSX语法，即JavaScript 语法扩展，它基本上是将JavaScript代码混合了HTML风格。如果你是一个web开发人员，对此你应该不会陌生。 在本文中，JSX随处可见。

保存 index.ios.js回到iPhone模拟器，按下快捷键 Cmd+R，你会看到App的显示变成了 “Hello World (Again)” 。

重新运行React Navtive App如同刷新Web页面一样简单。

因为你实际上是在和JavaScript打交道，所以只需修改并保存index.ios.js，即可让App内容得到更新，同时不中断App的运行。

> 注意:
> 如果你还有疑问，你可以用浏览器在看一下你的“Bundle”内容，它应该也发生了变化。

好了，“Hello World” 的演示就到此为止；接下来我们要编写一个真正的React App了！

实现导航

这个demo使用了标准的UIKit中的导航控制器来提供”栈式导航体验“。接下来我们就来实现这个功能。

在 index.ios.js, 将 PropertyFinderApp 类修改为 HelloWorld:

```
class HelloWorld extends React.Component {
```

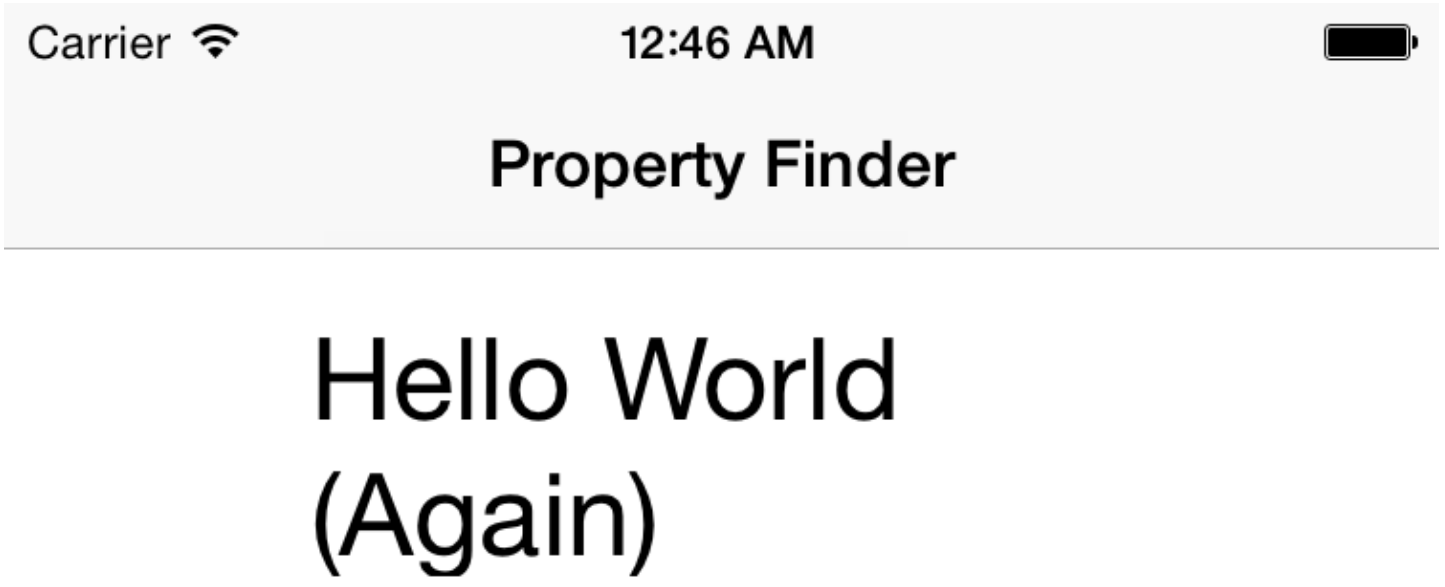
我们仍然要显示“Hello World”字样，但不再将它作为App的根视图。
然后为HelloWorld加入以下代码：

```
class PropertyFinderApp extends React.Component {
  render() {
    return (
      <React.NavigatorIOS
        style={styles.container}
        initialRoute={{
          title: 'Property Finder',
          component: HelloWorld,
        }}/>
    );
  }
}
```

这将创建一个导航控制器，并指定了它的外观样式和初始route（相对于HelloWorld视图）。在web开发中，routing是一种技术，用于表示应用程序的导航方式，即哪个一页面（或route）对应哪一个URL。
然后修改css样式定义，在其中增加一个container样式：

```
var styles = React.StyleSheet.create({
  text: {
    color: 'black',
    backgroundColor: 'white',
    fontSize: 30,
    margin: 80
  },
  container: {
    flex: 1
  }
});
```

flex: 1的意思稍后解释。
回到模拟器，按 Cmd+R 查看效果:



这个导航控制器有一个根视图，即图中显示的”Hello World“文本。非常好——我们的App已经具备了基本的导航功能。是时候显示一些”真正的“UI了！

实现查找

新建一个 SearchPage.js 文件，保存在 index.ios.js同一目录。在这个文件中加入代码：

```
'use strict';

var React = require('react-native');
var {
  StyleSheet,
  Text,
  TextInput,
  View,
  TouchableHighlight,
  ActivityIndicatorIOS,
  Image,
  Component
} = React;
```

这里使用了一个解构赋值（destructuring assignment），可以将多个对象属性一次性赋给多个变量。这样，在后面的代码中，我们就可以省略掉React前缀，比如用StyleSheet 来代替 React.StyleSheet。解构赋值对于数组操作来说尤其方便，请参考well worth learning more about.
然后定义如下Css样式:

```
var styles = StyleSheet.create({
description: {
marginBottom: 20,
fontSize: 18,
textAlign:  ‘center’ ,
color:  ‘#656565’
},
container: {
padding: 30,
marginTop: 65,
alignItems:  ‘center’
}
});
```

这里，再次使用了标准的 CSS 属性。虽然用CSS设置样式在可视化方面比起在IB中要差一些，但总比在viewDidLoad()方法中用代码写要好一些。
然后加入以下代码：

```
class SearchPage extends Component {
  render() {
    return (
      <View style={styles.container}>
        <Text style={styles.description}>
          Search for houses to buy!
        </Text>
        <Text style={styles.description}>
          Search by place-name, postcode or search near your location.
        </Text>
      </View>
    );
  }
}
```

在render方法中使用了大量的JSX语法来构造UI组件。通过这种方式，你可以非常容易地构造出如下组件：在一个Container View中包含了两个label。
在源文件的最后加入这一句：

```
module.exports = SearchPage;
```

这一句将使 SearchPage 类可被其他js文件引用。

然后需要修改App的导航。

打开 index.ios.js 在文件头部、现有的require 语句后加入 require 语句:

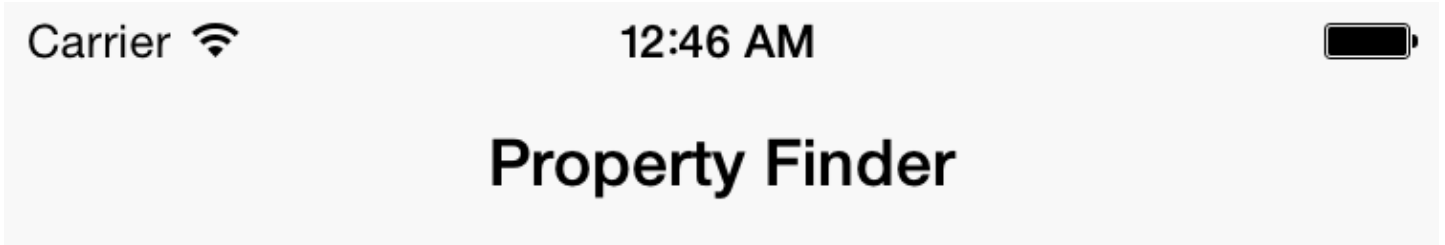
```
var SearchPage = require('./SearchPage');
```

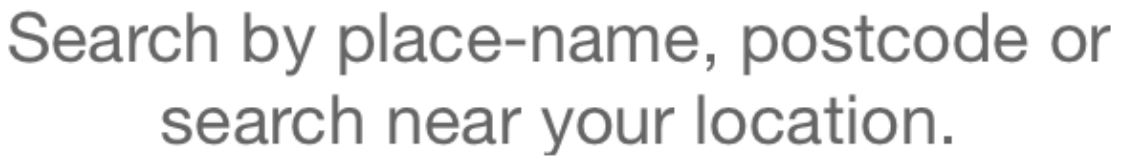
在 PropertyFinderApp 类的 render 函数中，修改 initialRoute 为:

```
component: SearchPage
```

这里我们可以将HelloWorld类和它对应的样式移除了，我们不再需要它。

回到模拟器，按下 Cmd+R 查看效果:




Search by place-name, postcode or
search near your location.

你新创建的组件SearchPage显示在屏幕中。

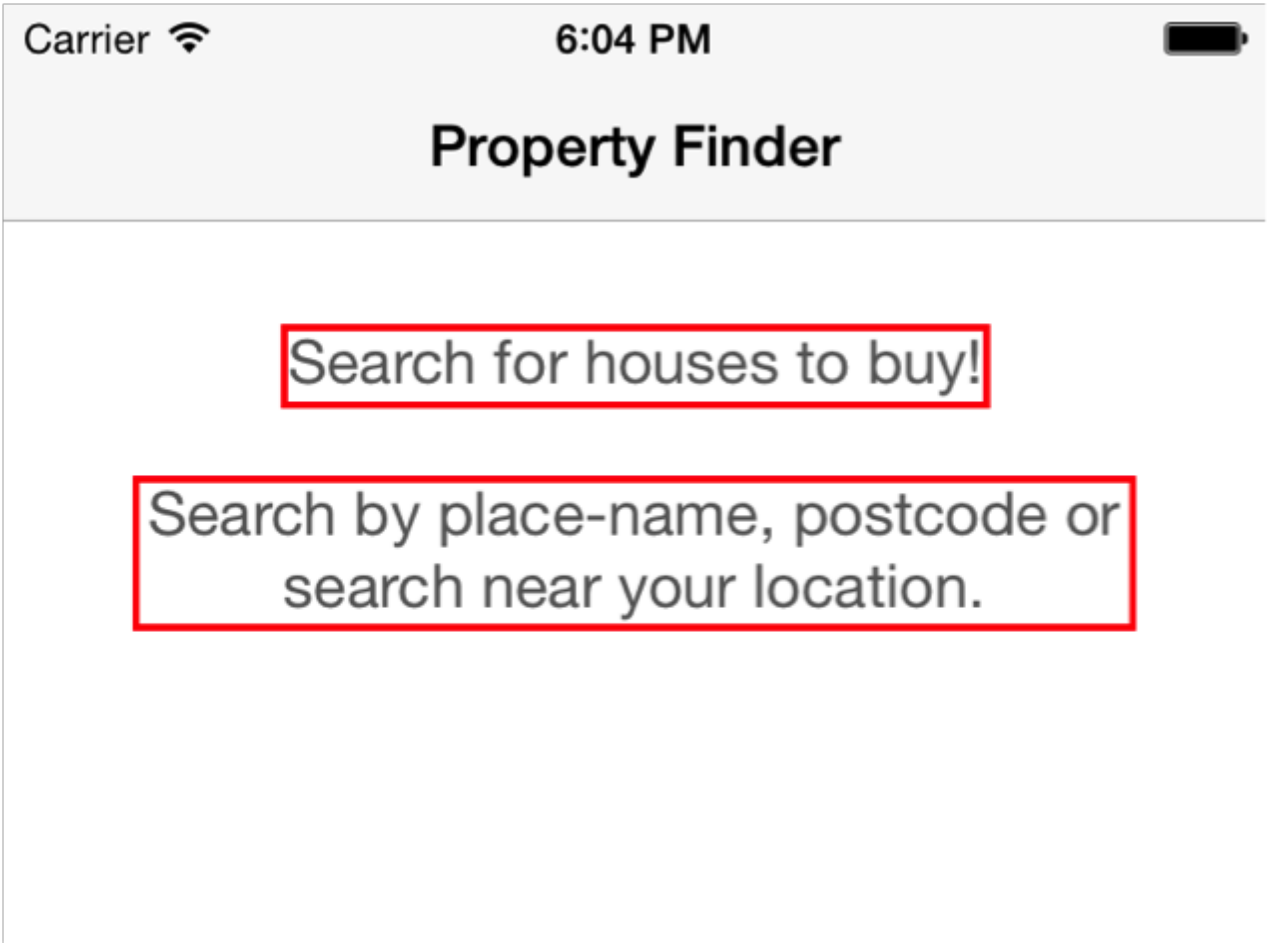
弹性盒子模型

一直以来，我们都在用原始的CSS属性来设置外边距、内边距和颜色。但是，最新的CSS规范中增加了弹性盒子的概念，非常利于我们对App的UI进行布局，虽然你可能还不太熟悉它。

React Native 使用了 css-layout 库，在这个库中实现了弹性盒子，而这种模型无论对iOS还是Android来说都很好理解。

更幸运的是，Facebook针对许多语言单独实现了这个项目，这就引申出了许多新颖的用法，比如在SVG中应用弹性盒子布局（是的，这篇文章也是我写的，为此我不得不熬到深夜）。

在这个App中，采用了默认的垂直流式布局，即容器中的子元素按照从上到下的顺序进行布局。比如：



这被称作主轴, 主轴可能是水平方向，也可能是垂直方向。

每个子元素的纵向位置由它们的边距（margin）、间距（padding）和高决定。容器的alignItems属性会被设置为居中（center），这决定了子元素在交叉轴上的位置。在本例里，将导致子元素水平居中对齐。

接下来我们添加一些文本输入框和按钮。打开SearchPage.js 在第二个 Text 元素后添加:

```
<View style={styles.flowRight}>
  <TextInput
    style={styles.searchInput}
    placeholder='Search via name or postcode'/>
  <TouchableHighlight style={styles.button}
      underlayColor='#99d9f4'>
    <Text style={styles.buttonText}>Go</Text>
  </TouchableHighlight>
</View>
<TouchableHighlight style={styles.button}
    underlayColor='#99d9f4'>
  <Text style={styles.buttonText}>Location</Text>
</TouchableHighlight>
```

这段代码添加了两个顶级的视图：一个文本输入框外加一个按钮，以及一个单独的按钮。它们所使用的样式待会我们再介绍。

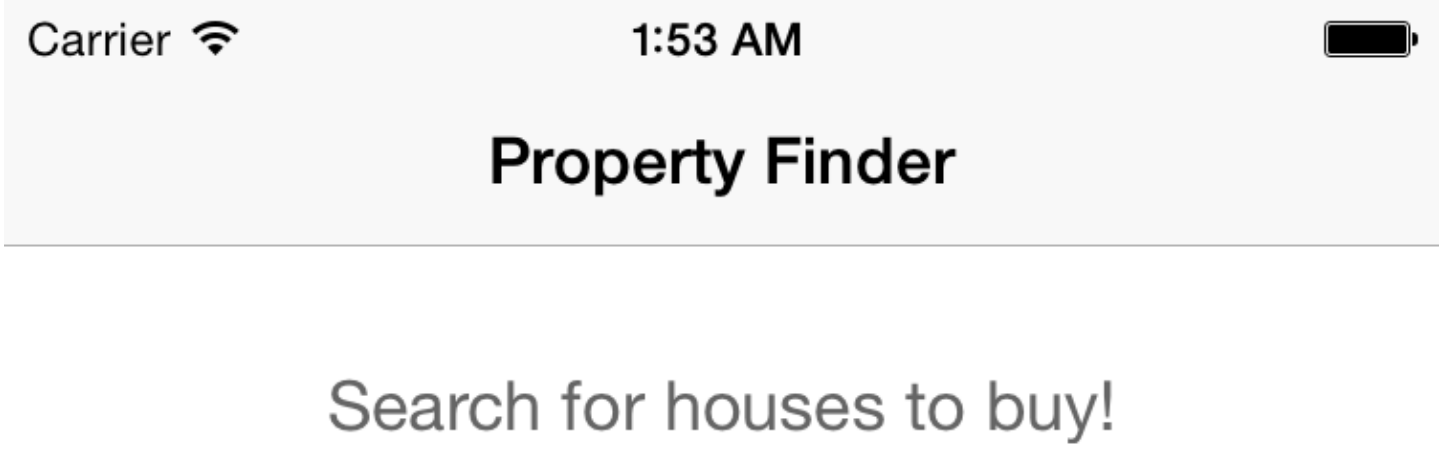
接着，在styles中增加如下样式：

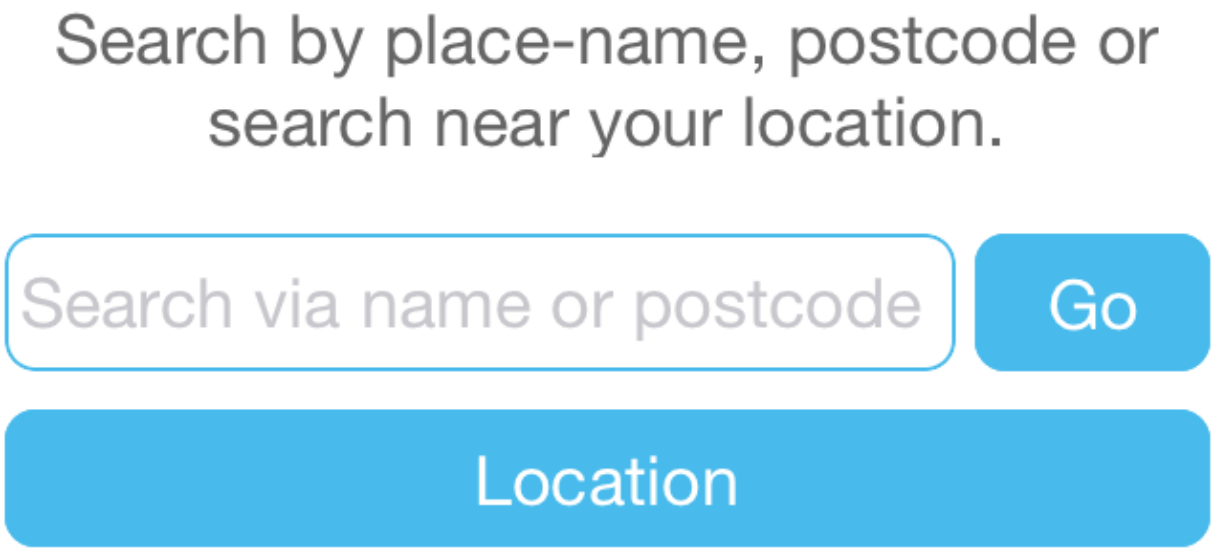
```
flowRight: {
  flexDirection: 'row',
  alignItems: 'center',
  alignSelf: 'stretch'
},
buttonText: {
  fontSize: 18,
  color: 'white',
  alignSelf: 'center'
},
button: {
  height: 36,
  flex: 1,
  flexDirection: 'row',
  backgroundColor: '#48BBEC',
  borderColor: '#48BBEC',
  borderWidth: 1,
  borderRadius: 8,
  marginBottom: 10,
  alignSelf: 'stretch',
  justifyContent: 'center'
},
searchInput: {
  height: 36,
  padding: 4,
  marginRight: 5,
  flex: 4,
  fontSize: 18,
  borderWidth: 1,
  borderColor: '#48BBEC',
  borderRadius: 8,
  color: '#48BBEC'
}
```

不同样式属性间以逗号分隔，这样你在container选择器后必须以一个逗号结尾。

这些样式将被文本输入框和按钮所用。

回到模拟器，按下Cmd+R ，你将看到如下效果：



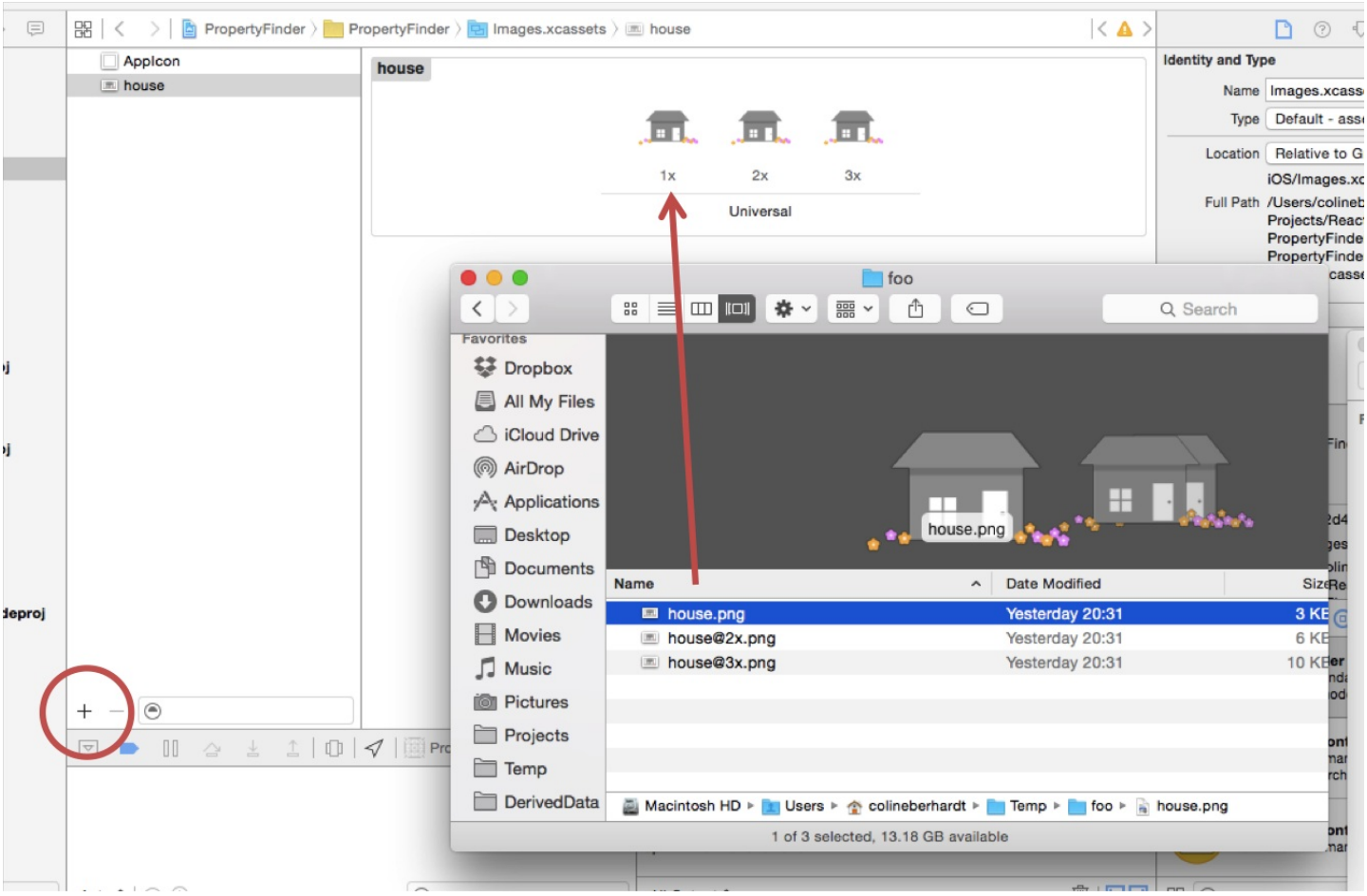

Search by place-name, postcode or search near your location.
Search via name or postcode
Go
Location

Go按钮和其紧随的文本框在同一行，因此我们将它们用一个容器装在一起，同时容器的flexDirection: 样式属性设置为’ row’ 。我们没有显式指定文本框和按钮的宽度，而是分别指定它们的flex样式属性为4和1。也就是说，它们的宽度在整个宽度（屏幕宽度）中所占的份额分别为4和1。

而且，视图中的两个按钮都不是真正的按钮。对于UIKit，按钮不过是可以点击的标签而已，因此React Native开发团队能够用JavaScript以一种简单的方式构建按钮：TouchableHighlight是一种React Native组件，当它被点击时，它的前景会变得透明，从而显示其隐藏在底部的背景色。

最后我们还要在视图中添加一张图片。这些图片可以在此处下载。下载后解压缩zip文件。

在 Xcode 打开 Images.xcassets 文件，点击加号按钮，添加一个新的image set。然后将需要用到的图片拖到image set右边窗口对应的位置。



要让这些图片显示，必须停止你的 React Native App并重新启动。

在location按钮对应的 TouchableHighlight 组件下加入：

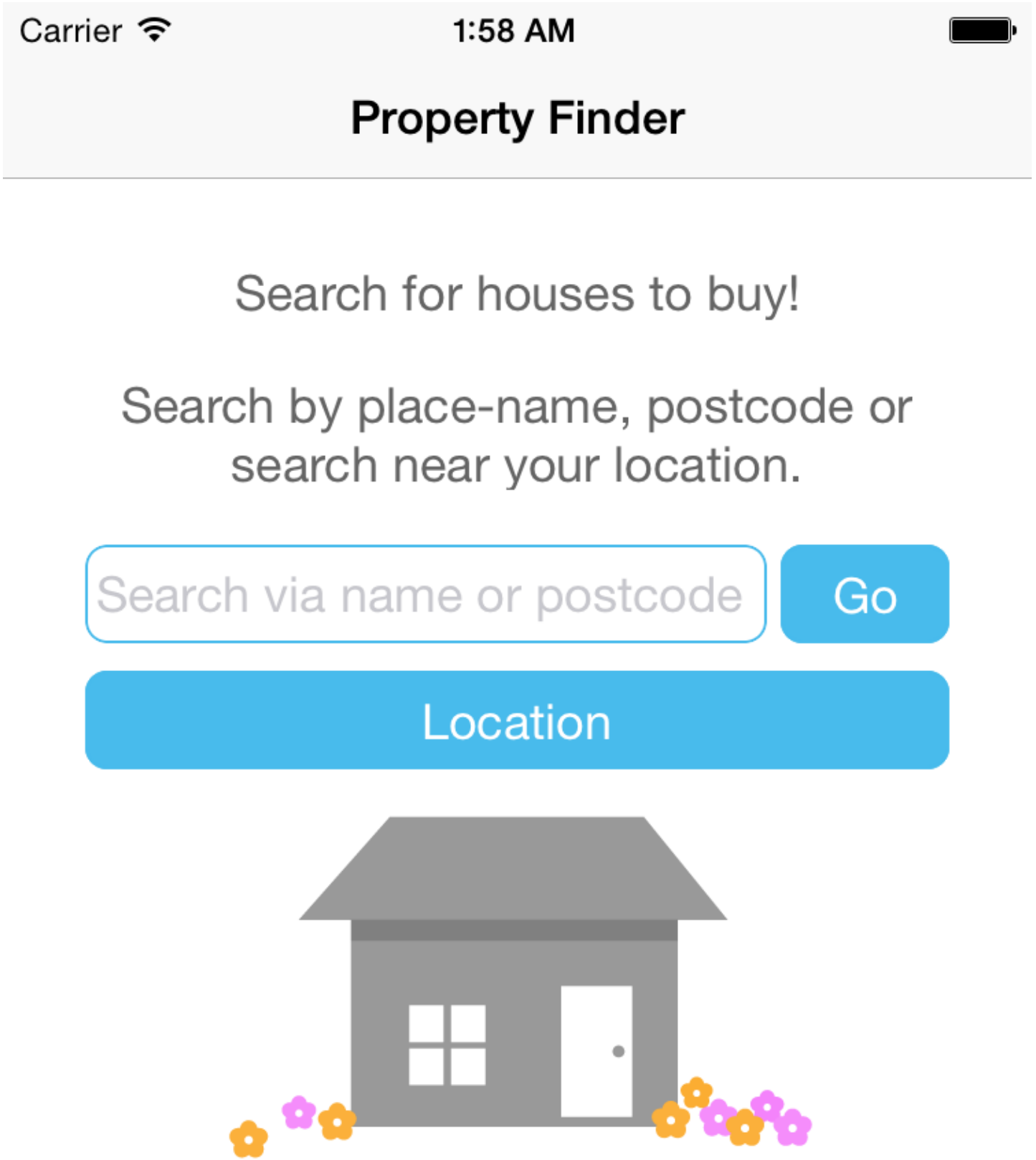
```
<Image source={require('image!house')} style={styles.image}/>
```

然后，为图片添加适当的样式定义，记得在上一个样式之后添加一个逗号结尾：

```
image: {
  width: 217,
  height: 138
}
```

由于我们将图片添加到了Images.xcasset资源包中，我们需要用require(‘image!house’)语句获得图片在App包中的正确路径。在Xcode中，打开Images.xcassets ，你可以找到名为house的image set。

回到模拟器，按下Cmd+R 查看运行效果:



> 注意: 如果图片没有显示，却看到一个“ “image!house” cannot be found”的提示，则可以重启packager(在终端中输入npm start命令)。

到目前为止，我们的App看上去有模有样，但它还缺少很多实际的功能。接下来的任务就是为App增加一些状态，执行一些动作。

组件状态

每个React组件都有一个state对象，它是一个键值存储对象。在组件被渲染之前，我们可以设置组件的state对象。

打开SearchPage.js,在 SearchPage 类的 render()方法前，加入以下代码：

```
constructor(props) {
  super(props);
  this.state = {
    searchString: 'london'
  };
}
```

现在组件就有一个state变量了，同时我们在state中存放了一个 searchString:’ london’ 的键值对象。

然后我们来使用这个state变量。在render方法中，修改TextInput元素为：

```
<TextInput
  style={styles.searchInput}
  value={this.state.searchString}
  placeholder='Search via name or postcode'/>
```

这将改变 TextInput 的value 属性，即在TextInput中显示一个london的文本——即state变量中的searchString。这个值在我们初始化state时指定的，但如果用户修改了文本框中文本，那又怎么办？

首先创建一个事件处理方法。在 SearchPage 类中增加一个方法：

```
onSearchTextChanged(event) {
console.log( ‘onSearchTextChanged’ );
this.setState({ searchString: event.nativeEvent.text });
console.log(this.state.searchString);
}
```

首先从事件参数event中获得text属性，然后将它保存到组件的state中，并用控制台输出一些感兴趣的内容。

为了让文本内容改变时这个方法能被调用，我们需要回到TextInput的onChange事件属性中，绑定这个方法，即新加一个onChange属性，如以下代码所示：

```
<TextInput
style={styles.searchInput}
value={this.state.searchString}
onChange={this.onSearchTextChanged.bind(this)}
placeholder='Search via name or postcode'/>
```

一旦用户改变了文本框中的文本，这个函数立即就会被调用。

> 注意: bind(this) 的使用有点特殊。JavaScript 中 this 关键字的含义其实和大部分语言都不相同，它就好比Swift语言中的self。bind方法的调用使得 onSearchTextChanged 方法中能够引用到this，并通过this引用到组件实例。更多内容请参考 MDN page on this。

然后，我们在render方法顶部、return语句之前加一条Log语句：

```
console.log('SearchPage.render');
```

通过这些log语句，你应该能明白大致发生了什么事情！

返回模拟器，按下Cmd+R，我们将看到文本框中一开始就有了一个london的字样，当你编辑这段文本后，控制台中的内容将显示：

```
RCTJSLog> "SearchPage.render"
RCTJSLog> "onSearchTextChanged"
RCTJSLog> "SearchPage.render"
RCTJSLog> "london "
```

查看上面的截屏，log语句输出的顺序似乎有点问题：
1. 组件初始化后调用 render() 方法
2. 当文本被改变， onSearchTextChanged() 被调用
3. 我们在代码中改变了组件的state 属性，因此render()方法会被调用
4. onSearchTextChanged() 打印新的search string.

当React 组件的状态被改变时，都会导致整个UI被重新渲染——所有组件的render方法都会被调用。这样做的目的，是为了将渲染逻辑和组件状态的改变完全进行分离。

在其他所有的UI框架中，要么程序员在状态改变时自己手动刷新UI，要么使用一种绑定机制在程序状态和UI之间进行联系。就像我另一篇文章 MVVM pattern with ReactiveCocoa 所讲。

而在React中，我们不再操心状态的改变会导致那一部分UI需要刷新，因为当状态改变所有的UI都会刷新。

当然，你也许会担心性能问题。

难道每次状态改变时，整个UI都会被舍弃然后重新创建吗？
这就是React真正智能的地方。每当UI要进行渲染时，它会遍历整个视图树并计算render方法，对比与当前UIKit视图是否一致，并将需要改变的地方列出一张列表，然后在此基础上刷新视图。也就是说，只有真

正发生变化的东西才会被重新渲染。

ReactJS将一些新奇的概念应用到了iOS App中，比如虚拟DOM（Document Object Modal，web文档可视树）和一致性。这些概念我们可以稍后再讨论，先来看下这个App接下来要做的工作。删除上面添加的Log语句。

开始搜索

为了实现搜索功能，我们需要处理Go按钮点击事件，创建对应的API请求，显示网络请求的状态。
打开SearchPage.js, 在constructor方法中修改state的初始化代码：

```
this.state = {
  searchString: 'london',
  isLoading: false
};
```

isLoading 属性用于表示查询是否正在进行。
在render方法最上面增加：

```
var spinner = this.state.isLoading ?
  ( <ActivityIndicatorIOS
      hidden='true'
      size='large'/> ) :
  ( <View/>);
```

这里用了一个三目运算，这是一个if语句的简化形式。如果isLoading为true，显示一个网络指示器，否则显示一个空的view。
在return语句中，在Image下增加：

```
{spinner}
```

在Go按钮对应的 TouchableHighlight 标签中增加如下属性：

```
onPress={this.onSearchPressed.bind(this)}
```

在 SearchPage 类中新增如下方法:

```
_executeQuery(query) {
  console.log(query);
  this.setState({ isLoading: true });
}

onSearchPressed() {
  var query = urlForQueryAndPage('place_name', this.state.searchString, 1);
  this._executeQuery(query);
}
```

_executeQuery() 目前仅仅是在控制台中输出一些信息，同时设置isLoading属性为true，剩下的功能我们留到后面完成。

> 注意: JavaScript 类没有访问器，因此也就没有私有的概念。因此我们会在方法名前加一个下划线，以表示该方法视同为私有方法。

当Go按钮被点击， onSearchPressed() 即被调用。
然后，在 SearchPage 类声明之前，声明如下实用函数：

```
function urlForQueryAndPage(key, value, pageNumber) {
  var data = {
      country: 'uk',
      pretty: '1',
      encoding: 'json',
      listing_type: 'buy',
      action: 'search_listings',
      page: pageNumber
  };
  data[key] = value;

  var querystring = Object.keys(data)
    .map(key => key + '=' + encodeURIComponent(data[key]))
    .join('&');

  return 'http://api.nestoria.co.uk/api?' + querystring;
};
```

这个函数不依赖SearchPage类，因此被定义为函数而不是方法。它首先将key\value参数以键值对形式放到了data集合中，然后将data集合转换成以&符分隔的“键=值”形式。=>语法是箭头函数的写法，一种创建匿名函数简洁写法，具体请参考recent addition to the JavaScript language 。

回到模拟器，按下 Cmd+R 重启App，然后点击 ‘Go’ 按钮。你将看到网络指示器开始转动。同时控制台将输出：

网络指示器显示，同时要请求的URL也打印出来了。拷贝并粘贴URL到Safari，查看搜索结果。你将看到一堆JSON对象。我们将用代码解析这些JSON对象。

> 注意: 这个App使用 Nestoria API 来查找房子。查找结果以JSON格式返回。官方文档中列出了所有请求的URL规范及响应格式。

发送请求

打开 SearchPage.js，在初始化状态过程中增加一个message属性：

```
this.state = {
  searchString: 'london',
  isLoading: false,
  message: ''
};
```

在render方法中，在UI元素的最后加入：

```
<Text style={styles.description}>{this.state.message}</Text>
```

这个Text用于向用户显示一些文本。

在 SearchPage 类中，在 _executeQuery()方法最后加入:

```
fetch(query)
  .then(response => response.json())
  .then(json => this._handleResponse(json.response))
  .catch(error =>
     this.setState({
      isLoading: false,
      message: 'Something bad happened ' + error
   }));
```

fetch 函数在 Fetch API中定义，这个新的JavaScript规范被Firefox 39（Nightly版）以及Chrome 42（开发版）支持，它在XMLHttpRequest的基础上进行了极大的改进。结果是异步返回的，同时使用了 promise规范，如果response中包含有效的JSON对象则将JSON对象的response成员（另一个JSON）传到_handleResponse方法（后面实现）。

然后在 SearchPage类中增加方法：

```
_handleResponse(response) {
  this.setState({ isLoading: false , message: '' });
  if (response.application_response_code.substr(0, 1) === '1') {
    console.log('Properties found: ' + response.listings.length);
  } else {
    this.setState({ message: 'Location not recognized; please try again.'});
  }
}
```

如果查询结果成功返回，我们重置 isLoading 属性为false，然后打印结果集的总行数。

> 注意: Nestoria 有 不以1开头的响应码， 这些代码都非常有用。例如202 和 200表示返回一个推荐位置的列表。当完成这个实例后，你可以尝试处理这些返回码，并将列表提供给用户选择。

保存，返回模拟器，按下Cmd+R ，然后搜索 ‘london’ 你将在控制台看到一条消息，表示搜索到20条房子信息。尝试输入一个不存在的地名，比如 ‘narnia’ 你将看到如下信息：

接下来我们在伦敦或者别的什么城市搜索20座房子。

显示搜索结果

新建一个文件： SearchResults.js, 编写如下代码：

```
'use strict';

var React = require('react-native');
var {
  StyleSheet,
  Image,
  View,
  TouchableHighlight,
  ListView,
  Text,
  Component
} = React;
```

你注意到了吗？一切都是老样子，一条requires语句和一个结构赋值。

然后定义一个Componet子类：

```
class SearchResults extends Component {

  constructor(props) {
    super(props);
    var dataSource = new ListView.DataSource(
      {rowHasChanged: (r1, r2) => r1.guid !== r2.guid});
    this.state = {
      dataSource: dataSource.cloneWithRows(this.props.listings)
    };
  }

  renderRow(rowData, sectionID, rowID) {
    return (
      <TouchableHighlight
        underlayColor='#dddddd'>
        <View>
          <Text>{rowData.title}</Text>
        </View>
      </TouchableHighlight>
    );
  }

  render() {
    return (
      <ListView
        dataSource={this.state.dataSource}
        renderRow={this.renderRow.bind(this)}/>
    );
  }

}
```

上述代码中使用了一个专门的组件——ListView ——该组件非常像ITableView。通过ListView.DataSource, 我们可以向ListView提供数据。renderRow函数则用于为每个行提供UI。

在构建数据源的时候，我们使用箭头函数对不同的行进行识别。这个函数在ListView进行“一致化”的时候被调用，以便判断列表中的数据是否被改变。在本例中，Nestoria API有一个guid属性，刚好可以用来作为判断的标准。

最后，加入一条模块输出语句：

```
module.exports = SearchResults;
```

在SearchPage.js 头部，require 下方加入:

```
var SearchResults = require('./SearchResults');
```

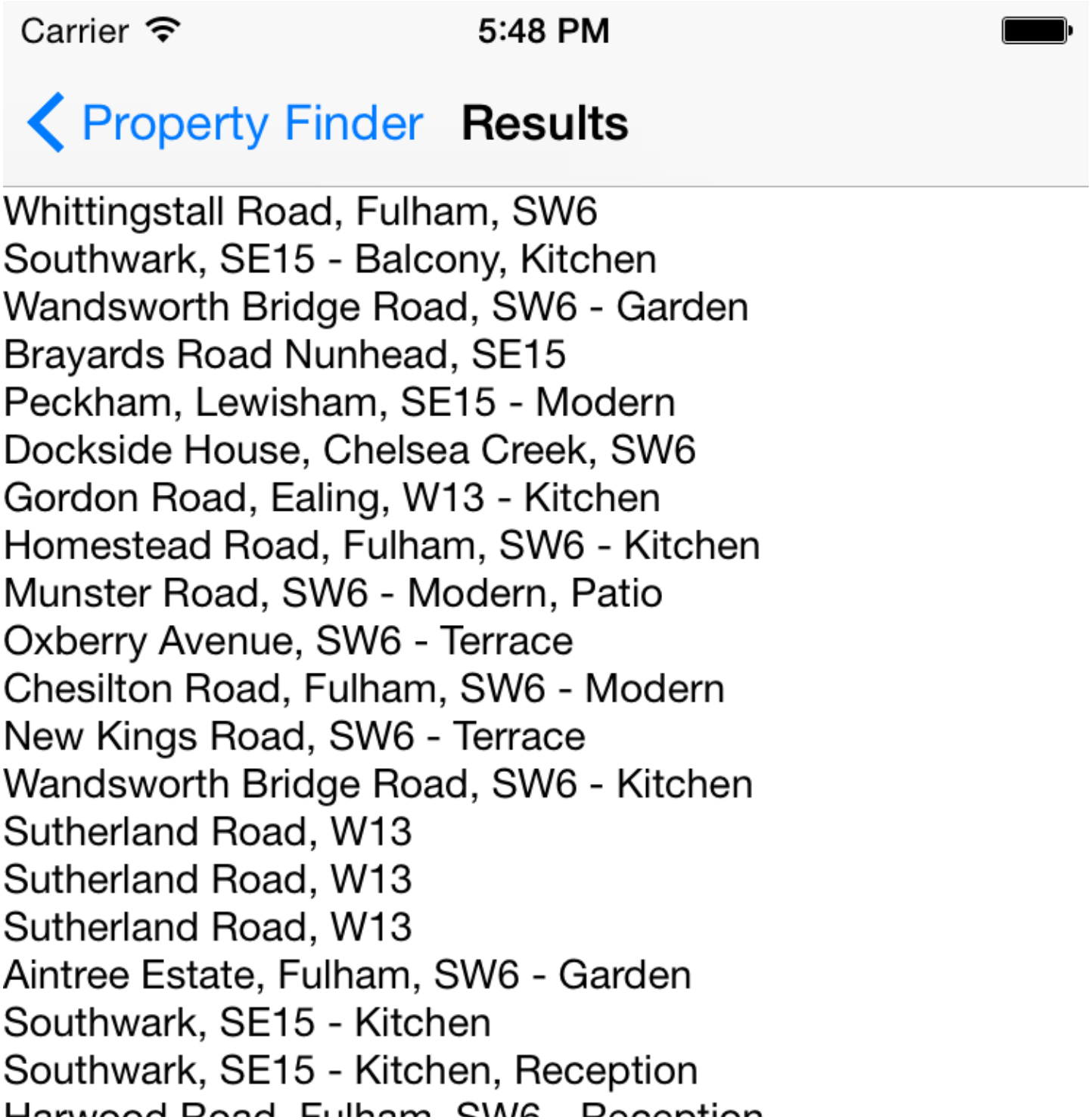
这样我们就可以 SearchPage 类中使用SearchResults类了。

在_handleResponse 方法，将console.log 一句替换为:

```
this.props.navigator.push({
  title: 'Results',
  component: SearchResults,
  passProps: {listings: response.listings}
});
```

上述代码将导航至SearchResults 页面，并将请求到的列表数据传递给它。Push方法可以将页面添加到导航控制器的ViewController堆栈中，同时你的导航栏上将出现一个Back按钮，点击它可以返回到上一页面。

回到模拟器, 按下Cmd+R ，进行一个查找动作。你将看到搜索结果如下：



好了，房子清单已经列出来了，不过列表有一点丑陋。接下来我们会让它变得漂亮一点。

表格样式

现在，React Native的代码对我们来说已经不陌生了，接下来我们的教程可以稍微加快一点节奏了。

在 SearchResults.js文件的解构赋值语句之后，添加样式定义：

```
var styles = StyleSheet.create({
  thumb: {
    width: 80,
    height: 80,
    marginRight: 10
  },
  textContainer: {
    flex: 1
  },
  separator: {
    height: 1,
    backgroundColor: '#dddddd'
  },
  price: {
    fontSize: 25,
    fontWeight: 'bold',
    color: '#48BBEC'
  },
  title: {
    fontSize: 20,
    color: '#656565'
  },
  rowContainer: {
    flexDirection: 'row',
    padding: 10
  }
});
```

这些代码中的样式将在渲染单元格时用到。

修改renderRow() 方法如下:

```
renderRow(rowData, sectionID, rowID) {
  var price = rowData.price_formatted.split(' ')[0];

  return (
    <TouchableHighlight onPress={() => this.rowPressed(rowData.guid)}
        underlayColor='#dddddd'>
      <View>
        <View style={styles.rowContainer}>
          <Image style={styles.thumb} source={{ uri: rowData.img_url }} />
          <View  style={styles.textContainer}>
            <Text style={styles.price}>£{price}</Text>
            <Text style={styles.title}
                 numberOfLines={1}>{rowData.title}</Text>
          </View>
        </View>
        <View style={styles.separator}/>
      </View>
    </TouchableHighlight>
  );
}
```

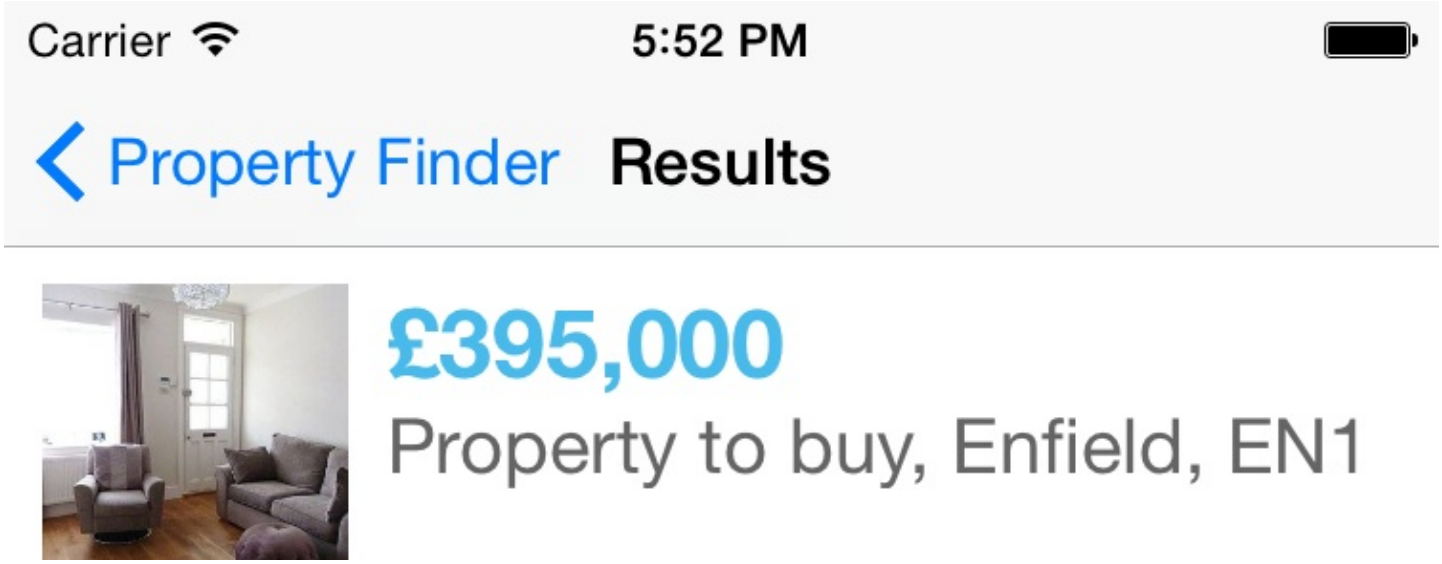
其中价格将以‘300,000 GBP’的格式显示，记得将GBP 后缀删除。上述代码用你已经很熟悉的方式来渲染单元格UI。缩略图以URL方式提供，React Native 自动将其解码（主线程中）。

在TouchableHightlight组件的onPress属性中再次使用了箭头函数，并将该行数据的guid作为传递的参数。

最后一个方法，用于处理点击事件

```
rowPressed(propertyGuid) {
  var property = this.props.listings.filter(prop => prop.guid === propertyGuid)[0];
}
```

这里，当用户点击某行时，通过guid去房产列表中找到对应的房屋信息。
回到模拟器，按下 Cmd+R ，观察运行结果：



£580,000

Property to buy, Lewisham,...

£574,950

Arbuthnot Road, SE14 - M...

£344,999

Billington Road, New Cros...

£1,295,000

Vanilla And Sesame Court,...

£349,950

Lucey Way, SE16 - Garden...

这下看起来好多了——只不过，那些住在London的人居然住得起这么贵房子？真是令人难以置信！

接下来，我们就来实现App的最后一个界面了。

查看房屋详情

新建一个 PropertyView.js 文件到项目中，编辑如下内容：

```
'use strict';

var React = require('react-native');
var {
  StyleSheet,
  Image,
  View,
  Text,
  Component
} = React;
```

确保进行到这一步的时候，你还没有睡着！:]

继续添加如下样式：

```
var styles = StyleSheet.create({
  container: {
    marginTop: 65
  },
  heading: {
    backgroundColor: '#F8F8F8',
  },
  separator: {
    height: 1,
    backgroundColor: '#DDDDDD'
  },
  image: {
    width: 400,
    height: 300
  },
  price: {
    fontSize: 25,
    fontWeight: 'bold',
    margin: 5,
    color: '#48BBEC'
  },
  title: {
    fontSize: 20,
    margin: 5,
    color: '#656565'
  },
  description: {
    fontSize: 18,
    margin: 5,
    color: '#656565'
  }
});
```

然后将组件加入视图：

```
class PropertyView extends Component {

  render() {
    var property = this.props.property;
    var stats = property.bedroom_number + ' bed ' + property.property_type;
    if (property.bathroom_number) {
      stats += ', ' + property.bathroom_number + ' ' + (property.bathroom_number > 1
        ? 'bathrooms' : 'bathroom');
    }

    var price = property.price_formatted.split(' ')[0];

    return (
      <View style={styles.container}>
        <Image style={styles.image}
           source={{uri: property.img_url}} />
        <View style={styles.heading}>
          <Text style={styles.price}>£{price}</Text>
          <Text style={styles.title}>{property.title}</Text>
          <View style={styles.separator}/>
        </View>
        <Text style={styles.description}>{stats}</Text>
        <Text style={styles.description}>{property.summary}</Text>
      </View>
    );
  }
}
```

render() 方法的第一步，是封装数据。因为从API获得的数据经常不太规范而且某些字段不全。代码采用简单手段让数据变得更便于展示一些。

剩下来的事情就非常简单了，填充组件的状态到UI上。

在文件最后加入export语句：

```
module.exports = PropertyView;
```

回到SearchResults.js 在文件头部加入 require 语句：

```
var PropertyView = require('./PropertyView');
```

修改 rowPressed() 方法，调用 PropertyView类：

```
  rowPressed(propertyGuid) {
    var property = this.props.listings.filter(prop => prop.guid === propertyGuid)[0];

    this.props.navigator.push({
      title: "Property",
      component: PropertyView,
      passProps: {property: property}
    });
  }
```

老规矩：返回模拟器，按下 Cmd+R, 点击搜索结果列表中的某行：

能住得起的房子才是最好的房子——在Pad上看到的这个房子确实很有吸引力！

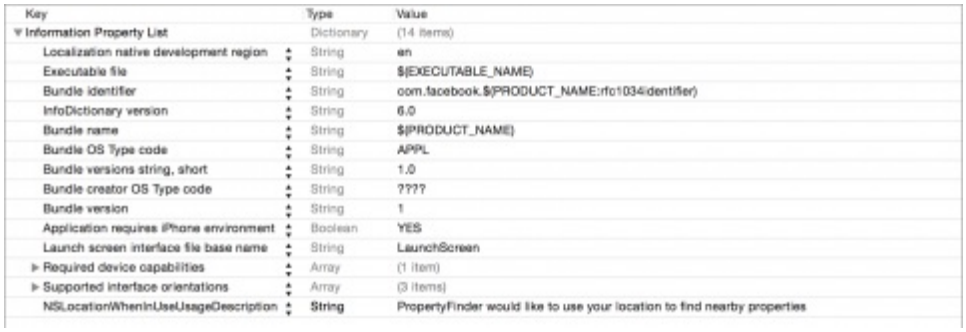
你的App快接近完成了，最后一步是让用户能够查找距离他们最近的房子。

根据位置查找

在Xcode中打开 Info.plist ，右键，Add Row，增加一个key。 使用 NSLocationWhenInUseUsageDescription 作为键名，使用
下列字符串作为键值：

> PropertyFinder would like to use your location to find nearby properties

添加完新值后，你的Plist文件将如下图所示：



Key	Type	Value
▼ Information Property List	Dictionary	(14 items)
Localization native development region	String	en
Executable file	String	$(EXECUTABLE_NAME)
Bundle identifier	String	com.facebook.$(PRODUCT_NAME:rfc1034identifier)
InfoDictionary version	String	6.0
Bundle name	String	$(PRODUCT_NAME)
Bundle OS Type code	String	APPL
Bundle versions string, short	String	1.0
Bundle creator OS Type code	String	????
Bundle version	String	1
Application requires iPhone environment	Boolean	YES
Launch screen interface file base name	String	LaunchScreen
▸ Required device capabilities	Array	(1 item)
▸ Supported interface orientations	Array	(3 items)
NSLocationWhenInUseUsageDescription	String	PropertyFinder would like to use your location to find nearby properties

这将在询问用户是否允许App使用他们的当前位置时，以这串文本作为提示信息。

打开 SearchPage.js, 找到TouchableHighlight 中渲染 ‘Location’ 按钮的代码，加入下列属性值：

```
onPress={this.onLocationPressed.bind(this)}
```

这样，当点击Location按钮，会调用 onLocationPressed 方法。

在SearchPage 类中，增加方法：

```
onLocationPressed() {
  navigator.geolocation.getCurrentPosition(
    location => {
      var search = location.coords.latitude + ',' + location.coords.longitude;
      this.setState({ searchString: search });
      var query = urlForQueryAndPage('centre_point', search, 1);
      this._executeQuery(query);
    },
    error => {
      this.setState({
        message: 'There was a problem with obtaining your location: ' + error
      });
    });
}
```

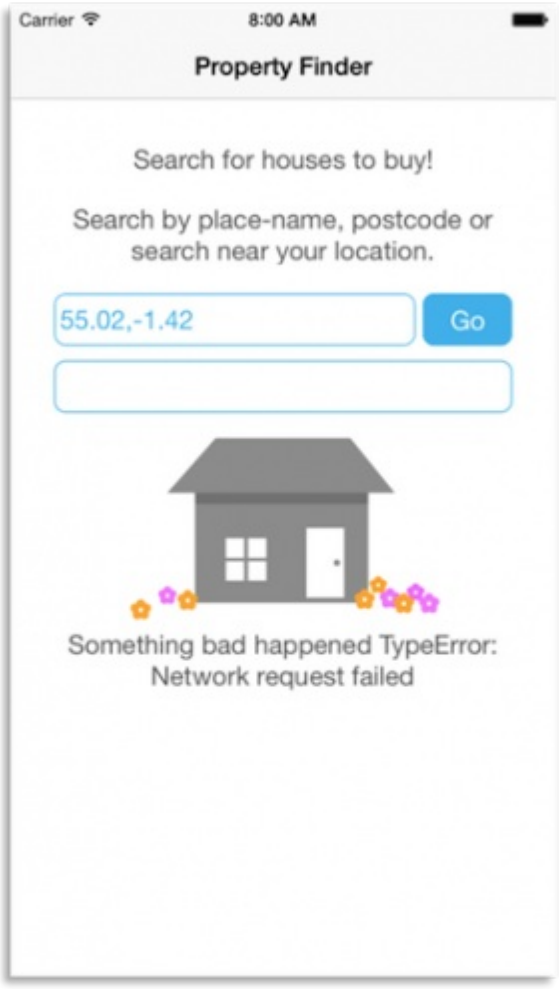
navigator.geolocation可获取当前位置。这个方法定义在 Web API中，这对于曾经在浏览器中使用过位

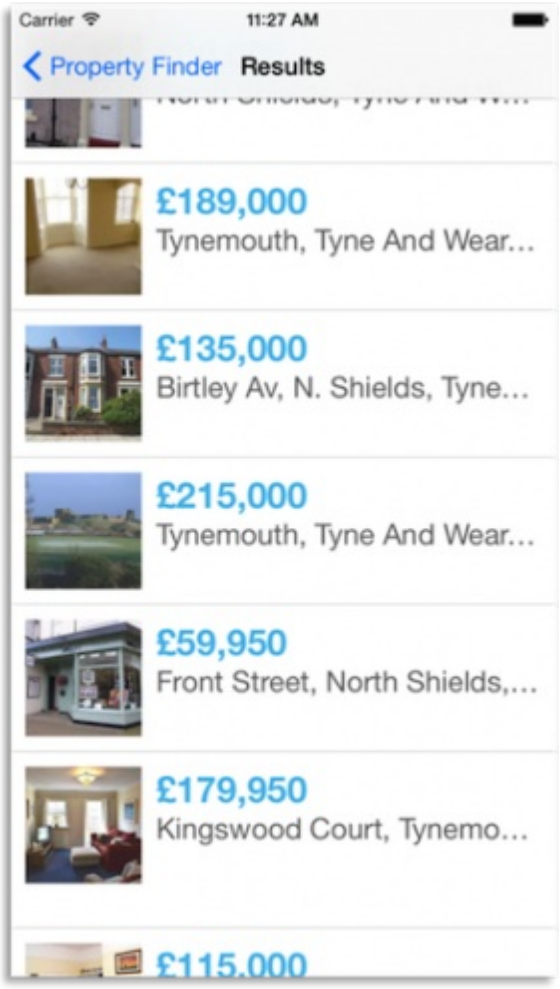
置服务的人来说并不陌生。React Native 框架用本地iOS服务重新实现了这个API。

如果当前位置获取成功，我们将调用第一个箭头函数，否则调用第二个箭头函数简单显示一下错误信息。

因为我们修改了plist文件，因此我们需要重新启动App，而不能仅仅是Cmd+R了。请在Xcode中终止App，然后重新编译运行App。

在使用基于地理定位的搜索之前，我们需要指定一个Nestoria数据库中的默认位置。在模拟器菜单中，选择Debug\Location\Custom Location ... 然后输入 55.02的纬度和-1.42的经度。这是一个位于英格兰北部的非常优美的海边小镇，我的家。





它远没有伦敦那么繁华——但住起来真的很便宜！:]

接下来做什么

恭喜你，你的第一个React Native App终于完成了！你可以在GitHub上找到每一个”可运行的“步骤的项目源文件，如果你搞不定的时候它们会非常有用的 :]

如果你来自Web领域，你可能觉得在代码中用JS和React框架建立基于本地化UI的App的界面并实现导航不过是小菜一碟。但如果你主要开发的是本地App，我希望你能从中感受到React Native的优点：快速的App迭代，现代JavaScript语法的支持和清晰的CSS样式规则。

在你的下一个App中，你是会使用这个框架，还是会继续顽固不化地使用Swift和O-C呢？

无论你怎么选择，我都希望你能从本文的介绍中学习到一些有趣的新东西，并把其中一些原理应用到你的下一个项目中。

如果你有任何问题及建议，请参与到下面的讨论中来！