
黑马程序员™
www.itheima.com
传智播客旗下
高端IT教育品牌
好客租房移动Web（中）

目录 Contents

- 地图找房模块
- 列表找房模块
- 房屋详情模块

1. 地图找房模块

1.1 功能分析

业务：使用百度地图 API 实现地图找房。

功能：

- 展示当前定位城市
- 展示该城市所有区的房源数据
- 展示某区下所有镇的房源数据
- 展示某镇下所有小区的房源数据
- 展示某小区下的房源数据列表

难点： 百度地图标注、缩放级别、缩放事件等的使用。

1. 地图找房模块

1.2 顶部导航栏

步骤

1. 封装 NavHeader 组件实现城市选择、地图找房页面的复用。
2. 在 components 目录中创建组件 NavHeader/index.js。
3. 在该组件中封装 antd-mobile 组件库中的 NavBar 组件。
4. 在地图找房页面使用封装好的 NavHeader 组件实现顶部导航栏功能。
5. 使用 NavHeader 组件，替换城市选择页面的 NavBar 组件。

1. 地图找房模块

1.2 顶部导航栏

添加 props 校验

1. 安装：yarn add prop-types。
2. 导入 PropTypes。
3. 给 NavHeader 组件的 children 和 onLeftClick 属性添加 props 校验。

```
NavHeader.propTypes = {
  children: PropTypes.string.isRequired,
  onLeftClick: PropTypes.func
}
```

1.3 组件间样式覆盖问题

1. 概述

① 问题：CityList 组件的样式，会影响 Map 组件的样式。

② 原因：在配置路由时，CityList 和 Map 组件都被导入到项目中，那么组件的样式也就被导入到项目中了。如果组件之间样式名称相同，那么一个组件中的样式就会在另一个组件中也生效，从而造成组件之间样式相互覆盖的问题。

③ 结论：默认，只要导入了组件，不管组件有没有显示在页面中，组件的样式就会生效。

④ 如何解决?

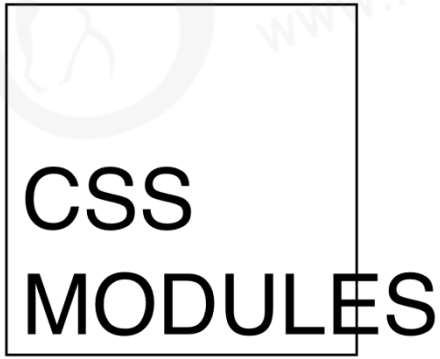
- 手动处理 （起不同的类名）
- CSS IN JS

1. 地图找房模块

1.3 组件间样式覆盖问题

2. CSS IN JS

- CSS IN JS：是使用 JavaScript 编写 CSS 的统称，用来解决 CSS 样式冲突、覆盖等问题
- CSS IN JS 的具体实现有 50 多种，比如：CSS Modules、styled-components 等
- 推荐使用：CSS Modules （React脚手架已集成，可直接使用）



1. 地图找房模块

1.3 组件间样式覆盖问题

3. CSS Modules 的说明

- CSS Modules 通过对 CSS 类名重命名，保证每个类名的唯一性，从而避免样式冲突的问题
- 换句话说：所有类名都具有“局部作用域”，只在当前组件内部生效
- 实现方式：webpack的 css-loader 插件
- 命名采用：BEM（Block 块、Element 元素、Modifier 三部分组成）命名规范，比如：.list__item_active
- 在 React 脚手架中演化成：文件名、类名、hash（随机）三部分，只需要指定类名即可

```
/* 自动生成的类名，我们只需要提供 classname 即可 */
[filename]_[classname]__[hash]
```

```
// 类名
.error {}
// 生成的类名为：
.Button_error__ax7yz
```

1. 地图找房模块

1.3 组件间样式覆盖问题

4. CSS Modules 在项目中的使用

1. 创建名为 [name].module.css 的样式文件（React脚手架中的约定，与普通 CSS 作区分）

```
// 在 CityList 组件中创建的样式文件名称：
index.module.css
```

2. 组件中导入该样式文件（注意语法）

```
// 在 CityList 组件中导入样式文件：
import styles from './index.module.css'
```

3. 通过 styles 对象访问对象中的样式名来设置样式

```
<div className={styles.test}></div>
```

1.3 组件间样式覆盖问题

5. 使用 CSS Modules 修改 NavHeader 样式

1. 在 NavHeader 目录中创建名为 index.module.css 的样式文件。
2. 在样式文件中修改当前组件的样式（使用单个类名设置样式，不使用嵌套样式）。
3. 对于组件库中已经有的全局样式（比如：.am-navbar-title），需要使用 :global() 来指定。

```
// OK
.navBar {}
// 不推荐嵌套
.navbar .test {}
```

```
:global(.am-navbar-title) { color: #333; }
```

```
.root :global(.am-navbar-title) {}
```

1. 地图找房模块

1.4 根据定位展示当前城市

步骤

1. 获取当前定位城市。
2. 使用地址解析器解析当前城市坐标。
3. 调用 centerAndZoom() 方法在地图中展示当前城市，并设置缩放级别为11。
4. 在地图中添加比例尺和平移缩放控件。

1. 地图找房模块

1.5 创建文本覆盖物

1. 实现步骤

① 打开百度地图添加文字标签DEMO

② 创建 Label 实例对象。

③ 调用 setStyle() 方法设置样式。

④ 在 map 对象上调用 addOverlay() 方法，将文本覆盖物添加到地图中。

```
const opts = {
  position : point,     // 指定文本标注所在的地理位置
  offset   : new BMap.Size(30, -30)     //设置文本偏移量
}
const label = new BMap.Label("文本信息", opts)  // 创建文本标注对象
label.setStyle({  // 设置样式
   color : "red",
})
map.addOverlay(label)  // 将创建好的文本标注添加为地图的覆盖物
```

1. 地图找房模块

1.5 创建文本覆盖物

2. 绘制房源覆盖物

1. 调用 Label 的 setContent() 方法，传入 HTML 结构，修改 HTML 内容的样式。

```
// 设置HTML内容
label.setContent(`<div class="${styles.bubble}">...其他结构</div>`)
```

2. 调用 setStyle() 修改覆盖物样式。

```
// 设置覆盖物样式
label.setStyle()
```

3. 给文本覆盖物添加单击事件。

```
// 添加单击事件
label.addEventListener('click', e => {})
```

1. 地图找房模块

1.6 地图找房

1. 功能分析

① 获取房源数据，渲染覆盖物。

② 单击覆盖物后：1 放大地图 2 获取数据，渲染下一级覆盖物（重复第一步）。

③ 区、镇：单击事件中，清除现有覆盖物，创建新的覆盖物。

④ 小区：不清除覆盖物。移动地图，展示该小区下面的房源列表。

1.6 地图找房

2. 渲染所有区的房源覆盖物

① 获取房源数据。

② 遍历数据，创建覆盖物，给每个覆盖物添加唯一标识（后面要用）。

③ 给覆盖物添加单击事件。

④ 在单击事件中，获取到当前单击项的唯一标识。

⑤ 放大地图（级别为13），调用 clearOverlays() 方法清除当前覆盖物。

1.6 地图找房

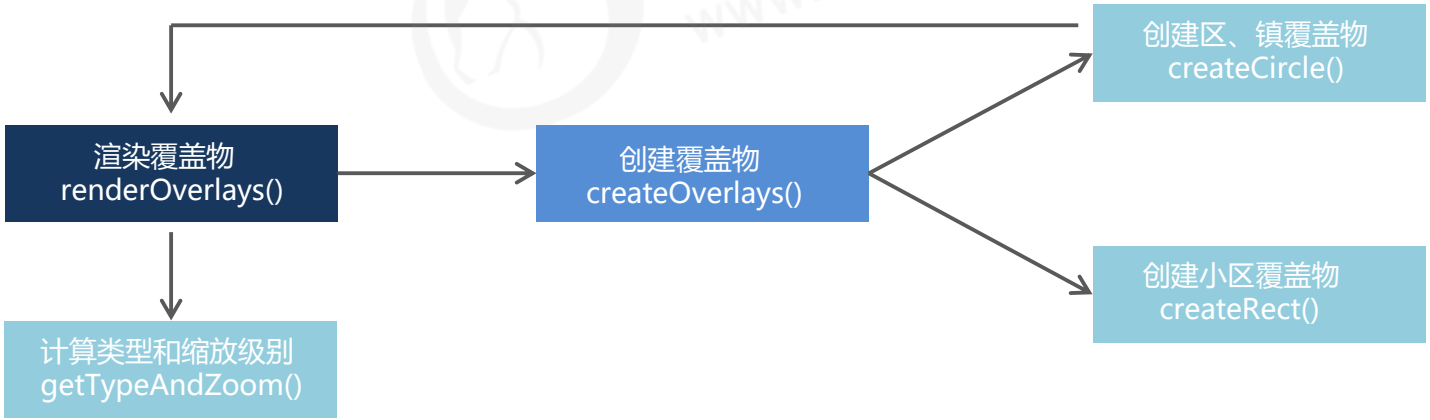
3. 封装流程

① renderOverlays() 作为入口：（1）接收区域 id 参数，获取该区域下的房源数据。（2）获覆盖物类型以及下级地图缩放级别。

② createOverlays() 方法：根据传入的类型，调用对应方法，创建覆盖物。

③ createCircle() 方法：根据传入的数据创建覆盖物，绑定事件（放大地图、清除覆盖物、渲染下一级房源数据）。

④ createRect() 方法：根据传入的数据创建覆盖物，绑定事件（移动地图、渲染房源列表）。



1.6 地图找房

4. renderOverlays()

1. 接收区域 id 参数，根据该参数获取房源数据

```
async renderOverlays(id) {
  const res = await axios.get(`/area/map?id=${id}`)
  const data = res.data.body

  data.forEach(item => {
    // this.createOverlays(item)
  })
}
```

1. 地图找房模块

1.6 地图找房

4. renderOverlays()

1. 接收区域 id 参数，根据该参数获取房源数据
2. 调用 getTypeAndZoom 方法获取地图缩放级别、覆盖物类别（根据缩放级别来得到）

```
getTypeAndZoom() {
  const zoom = this.map.getZoom()
  let nextZoom, type
  if (zoom >= 10 && zoom < 12) {
    // 11 是默认缩放级别，此时展示所有区的覆盖物
    type = 'circle'
    nextZoom = 13
  }
  return { type, nextZoom }
}
```

1.6 地图找房

5. createOverlays()

- 根据传入的类型等数据，调用相应的创建覆盖物，并提供参数

```
createOverlays(data, nextZoom, type) {
  if (type === 'rect') {
    // 小区
    this.createRect(参数1, 参数2...)
  } else {
    // 区和镇
    this.createCircle(参数1, 参数2...)
  }
}
```

1.6 地图找房

6. createCircle()

① 复用之前创建覆盖物的代码逻辑。

② 在覆盖物的单击事件中，调用 renderOverlays(id) 方法，重新渲染该区域的房屋数据。

```
createCircle(point, name, count, id, zoom) {
  // ... 省略其他代码

  label.addEventListener('click', e => {
    // 获取下一级房源数据
    this.renderOverlays(id)
  })
}
```

1.6 地图找房

7. createRect()

① 创建 Label 、设置样式、设置 HTML 内容，绑定单击事件。
② 在单击事件中，获取该小区的房源数据。
③ 展示房源列表。
④ 渲染获取到的房源数据。
⑤ 调用地图 panBy() 方法，移动地图到中间位置。
⑥ 监听地图 movestart 事件，在地图移动时隐藏房源列表。

```
this.map.panBy(x, y)
```

```
map.addEventListener('movestart', () => {
  // 隐藏房源列表
})
```

1.7 axios 优化和环境变量

问题分析

1. 在使用 axios 发送请求的时候，接口地址每次都要写 http://localhost:8080 ，太繁琐。

```
// 解决方式：通过 axios 配置，统一处理 baseURL
baseURL: 'http://localhost:8080/'
```

2. 接口域名、图片域名，分为开发环境和生产环境，直接写在代码中，项目发布时，很难替换。

```
// 解决方式：通过脚手架提供的 环境变量 来解决
在开发环境变量文件 .env.development 中，
配置 REACT_APP_URL=http://localhost:8080/
```

```
// 解决方式：通过脚手架提供的 环境变量 来解决
在生产环境变量文件 .env.production 中，
配置 REACT_APP_URL=线上接口地址
```

1.7 axios 优化和环境变量

1. 使用环境变量

1. 在项目根目录创建文件 .env.development。
2. 在该文件中添加环境变量 REACT_APP_URL（注意：环境变量约定以 REACT_APP_ 开头）。
3. 设置 REACT_APP_URL 的值为：http://localhost:8080。
4. 重新启动脚手架。
5. 在 utils/url.js 文件中，创建 BASE_URL 变量，设置其值为 process.env.REACT_APP_URL。
6. 导出 BASE_URL。
7. 使用 BASE_URL 修改图片地址。

```
REACT_APP_URL=http://localhost:8080
```

```
// 代码中获取环境变量的值
process.env.REACT_APP_URL
```

1.7 axios 优化和环境变量

2. axios 优化

1. 在 utils/api.js 文件中，导入 axios 和 BASE_URL。
2. 调用 axios.create() 方法创建一个 axios 实例。
3. 给 create() 方法，添加配置 baseURL，值为：BASE_URL。
4. 导出 API 对象。
5. 导入 API，使用 API 代替 axios，去掉接口地址中的 http://localhost:8080。

```
const API = axios.create({
  baseURL: BASE_URL
})
export { API }
```

目录 Contents

- 地图找房模块
- 列表找房模块
- 房屋详情模块

2. 列表找房模块

2.1 功能分析

业务：根据查询条件筛选房源列表。

功能：

- 搜索导航栏组件封装
- 条件筛选栏组件封装
- 条件筛选栏吸顶功能
- 房屋列表

难点：条件筛选组件的封装、房屋列表处理。

2. 列表找房模块

2.2 顶部搜索导航栏

1. 封装首页搜索导航栏

1. 在 components 目录中创建组件 SearchHeader/index.js。
2. 在该组件中，复用首页中已经实现的结构、样式来封装组件。

2. 列表找房模块

2.2 顶部搜索导航栏

2. 实现找房页面搜索导航栏

1. 在找房页面 SearchHeader 组件基础上，调整结构（添加返回icon等）。
2. 给 SearchHeader 组件传递 className 属性，来调整组件样式，让其适应找房页面效果。

2. 列表找房模块

2.3 条件筛选栏

1. 组件结构分析

- 父组件：Filter。
- 子组件：FilterTitle 标题菜单组件。
- 子组件：FilterPicker 前三个菜单对应的内容组件。
- 子组件：FilterMore 最后一个菜单对应的内容组件。
- FilterPicker 内容组件中，使用 antd-mobile 组件库的 PickerView 选择器组件。
- <></> 语法是<React.Fragment> 的简化语法，作用：不添加额外元素，返回多个节点。

```
<React.Fragment>
  <span>Some text. </span> <h2>A heading</h2>
</React.Fragment>
// 或者简化语法：
<>
  <span>Some text. </span> <h2>A heading</h2>
</>
```

2.3 条件筛选栏

2. 功能分析

① 点击 FilterTitle 组件菜单，展开该条件筛选对话框，被点击的标题高亮。

② 点击取消按钮或空白区域，隐藏对话框，取消标题高亮。

③ 选择筛选条件后，点击确定按钮，隐藏对话框，当前标题高亮。

④ 打开对话框时，如果已经选择了筛选条件，就默认选中已选择的条件。

⑤ 对话框的展示和隐藏都有动画效果。

⑥ 吸顶功能。

⑦ 注意：Filter组件不仅要实现自身功能，还要提供获取房源列表数据的筛选条件！

2.4 FilterTitle 组件

实现思路

- 功能一：根据标题菜单数据，渲染标题列表。
- 功能二：标题可点击（绑定事件）。
- 功能三：标题高亮。
- 标题高亮：1 点击时 2 有筛选条件选中时。
- 标题高亮状态：提升至父组件 Filter 中（状态提升），由父组件提供高亮状态，子组件通过 props 接收状态来实现高亮。
- 原则：单一数据源！也就是说，状态只应该有一个组件提供并且提供操作状态的方法，其他组件直接使用该组件中的状态和操作态的方法即可。

2.4 FilterTitle 组件

实现步骤

① 通过 props 接收，高亮状态对象 titleSelectedStatus。

② 遍历 titleList 数组，渲染标题列表。

③ 判断高亮对象中当前标题是否高亮，如果是，就添加高亮类。

④ 给标题项绑定单击事件，在事件中调用父组件传过来的方法 onClick。

⑤ 将当前标题 type，通过 onClick 的参数，传递给父组件。

⑥ 父组件中接收到当前 type，修改该标题的选中状态为true。

```
const titleSelectedStatus = {
  area: false,
  mode: false,
  price: false,
  more: false
}
```

2.5 FilterPicker 组件

实现思路

- 功能一：点击前三个标题展示该组件，点击取消按钮或空白区域隐藏该组件。
- 功能二：使用 PickerView 组件展示筛选条件数据。
- 功能三：获取到 PickerView 组件中，选中的筛选条件值。
- 功能四：点击确定按钮，隐藏该组件，将获取到的筛选条件值传递给父组件。
- 展示或隐藏对话框的状态：由父组件提供（状态提升），通过 props 传递给子组件。
- 筛选条件数据：由父组件提供（因为所有筛选条件是通过一个接口来获取的），通过 props 传递给子组件。

2.5 FilterPicker 组件

1. 控制组件的展示和隐藏

① 在 Filter 组件中，提供组件展示或隐藏的状态： openType（表示展示的对话框类型）。

② 在 render 中判断 openType 值为 area/mode/price 时，就展示 FilterPicker 组件，以及遮罩层。

③ 在 onTitleClick 方法中，修改状态 openType 为当前 type，展示对话框。

④ 在 Filter 组件中，提供 onCancel 方法（作为取消按钮和遮罩层的事件处理程序）。

⑤ 在 onCancel 方法中，修改状态 openType 为空，隐藏对话框。

⑥ 将 onCancel 通过 props 传递给 FilterPicker 组件，在取消按钮的单击事件中调用该方法。

⑦ 在 Filter 组件中，提供 onSave 方法，作为确定按钮的事件处理程序，逻辑同上。

2.5 FilterPicker 组件

2. 获取当前筛选条件的数据

① 在 Filter 组件中，发送请求，获取所有筛选条件数据。

② 将数据保存为状态：filtersData。

③ 封装方法 renderFilterPicker 来渲染 FilterPicker 组件。

④ 在方法中，根据 openType 的类型，从 filtersData 中获取到需要的数据。

⑤ 将数据通过 props 传递给 FilterPicker 组件。

⑥ FilterPicker 组件接收到数据后，将其作为 PickerView 组件的 data （数据源）。

2.5 FilterPicker 组件

3. 获取选中值

① 在 FilterPicker 组件中，添加状态 value（用于获取 PickerView 组件的选中值）。

② 给 PickerView 组件添加配置项 onChange，通过参数获取到选中值，并更新状态 value。

③ 在确定按钮的事件处理程序中，将 type 和 value 作为参数传递给父组件。

2.5 FilterPicker 组件

4. 设置默认选中值

① 在 Filter 组件中，提供 FilterPicker 组件的选中值状态对象：selectedValues。

② 根据 openType 获取到当前类型的选中值（defaultValue），通过 props 传递给 FilterPicker 组件。

③ 在 FilterPicker 组件中，将 defaultValue 设置为状态 value 的默认值。

④ 父组件中更新当前 type 对应的 selectedValues 状态值。

```
const selectedValues = {
  area: ['area','null'],
  mode: ['null'],
  price: ['null'],
  more: []
}
```

2.6 完善 FilterTitle 高亮功能

1. 实现思路

- 点击标题时，遍历标题高亮数据。
- 如果是当前标题，直接设置为高亮。
- 分别判断每个标题对应的筛选条件有没有选中值（判断每个筛选条件的选中值与默认值是否相同，相同表示没有选中值；不同，表示选中了值）。
- 如果有，就让该标题保持高亮。
- 如果没有，就让该标题去掉高亮。

2.6 完善 FilterTitle 高亮功能

2. 实现步骤

① 在标题点击事件 onTitleClick 方法中，获取到两个状态：标题选中状态对象和筛选条件的选中值对象。

② 根据当前标题选中状态对象，获取到一个新的标题选中状态对象（newTitleSelectedStatus）。

③ 使用 Object.keys() 方法，遍历标题选中状态对象。

④ 先判断是否为当前标题，如果是，直接让该标题选中状态为 true（高亮）。

⑤ 否则，分别判断每个标题的选中值是否与默认值相同。

⑥ 如果不同，则设置该标题的选中状态为 true。

⑦ 如果相同，则设置该标题的选中状态为 false。

⑧ 更新状态 titleSelectedStatus 的值为：newTitleSelectedStatus。

2.7 FilterMore 组件

1. 渲染组件数据

① 封装 renderFilterMore 方法，渲染 FilterMore 组件。

② 从 filtersData 中，获取数据（roomType, oriented, floor, characteristic），通过 props 传递给 FilterMore 组件。

③ FilterMore 组件中，通过 props 获取到数据，分别将数据传递给 renderFilters 方法。

④ 在 renderFilters 方法中，通过参数接收数据，遍历数据，渲染标签。

2.7 FilterMore 组件

2. 获取选中值以及设置高亮

① 在 state 中添加状态 selectedValues（表示选中项的值）。

② 给标签绑定单击事件，通过参数获取到当前项的 value。

③ 判断 selectedValues 中是否包含当前项的 value 值。

④ 如果不包含，就将当前项的 value 添加到 selectedValues 数组中。

⑤ 如果包含，就从 selectedValues 数组中移除（使用数组的 splice 方法，根据索引号删除）。

⑥ 在渲染标签时，判断 selectedValues 数组中，是否包含当前项的 value，包含，就添加高亮类。

2.7 FilterMore 组件

3. 清除和确定按钮的逻辑处理

① 设置 FilterFooter 组件的取消按钮文字为：清除。

② 点击取消按钮时，清空所有选中项的值（selectedValues: []）。

③ 点击确定按钮时，将当前选中项的值和 type，传递给 Filter 父组件。

④ 在 Filter 组件中的 onSave 方法中，接收传递过来的选中值，更新状态 selectedValues。

2.7 FilterMore 组件

4. 设置默认选中值

① 在渲染 FilterMore 组件时，从 selectedValues 中，获取到当前选中值 more。

② 通过 props 将选中值传递给 FilterMore 组件。

③ 在 FilterMore 组件中，将获取到的选中值，设置为子组件状态 selectedValues 的默认值。

④ 给遮罩层绑定单击事件。

⑤ 在单击事件中，调用父组件的方法 onCancel 关闭 FilterMore 组件。

2.8 完成 FilterTitle 高亮功能

步骤

① 在 Filter 组件的 onTitleClick 方法中，添加 type 为 more 的判断条件。

② 当选中值数组长度不为 0 时，表示 FilterMore 组件中有选中项，此时，设置选中状态高亮。

③ 在点击确定按钮时，根据参数 type 和 value，判断当前菜单是否高亮。

④ 在关闭对话框时（onCancel），根据 type 和当前type的选中值，判断当前菜单是否高亮。

2.9 根据筛选条件获取房屋列表数据

1. 组装筛选条件

① 在 Filter 组件的 onSave 方法中，根据最新 selectedValues 组装筛选条件数据 filters。

② 获取区域数据的参数名： area 或 subway（选中值数组的第一个元素）。

③ 获取区域数据的值（以最后一个 value 为准）。

④ 获取方式和租金的值（选中值的第一个元素）。

⑤ 获取筛选（more）的值（将选中值数组转化为以逗号分隔的字符串）。

```
{
  area: 'AREA|67fad918-f2f8-59df', // 或 subway: '...'
  mode: 'true', // 或 'null'
  price: 'PRICE|2000',
  more: 'ORIEN|80795f1a-e32f-feb9,ROOM|d4a692e4-a177-37fd'
}
```

2.9 根据筛选条件获取房屋列表数据

2. 获取房屋列表数据

① 将筛选条件数据 filters 传递给父组件 HouseList。

② HouseList 组件中，创建方法 onFilter，通过参数接收 filters 数据，并存储到 this 中。

③ 创建方法 searchHouseList（用来获取房屋列表数据）。

④ 根据接口，获取当前定位城市 id 参数。

⑤ 将筛选条件数据与分页数据合并后，作为接口的参数，发送请求，获取房屋数据。

2.9 根据筛选条件获取房屋列表数据

3. 进入页面时获取数据

① 在 componentDidMount 钩子函数中，调用 searchHouseList，来获取房屋列表数据。

② 给 HouseList 组件添加属性 filters，值为对象。

③ 添加两个状态：list 和 count（存储房屋列表数据和总条数）。

④ 将获取到的房屋数据，存储到 state 中。

2. 列表找房模块

2.10 渲染房屋列表数据

1. 使用 List 组件渲染数据

① 封装 HouseItem 组件，实现 Map 和 HouseList 页面中，房屋列表项的复用。

② 使用 HouseItem 组件改造 Map 组件的房屋列表项。

③ 使用 react-virtualized 的 List 组件渲染房屋列表（参考 CityList 组件的使用）。

2.10 渲染房屋列表数据

2. 使用 WindowScroller 跟随页面滚动

- 默认，List 组件只让组件自身出现滚动条，无法让整个页面滚动，也就无法实现标题栏吸顶功能。
- 解决方式：使用 WindowScroller 高阶组件，让 List 组件跟随页面滚动（为 List 组件提供状态，同时还需设置 List 组件的 autoHeight 属性）。
- 注意：WindowScroller 高阶组件只能提供 height，无法提供 width。
- 解决方式：在 WindowScroller 组件中使用 AutoSizer 高阶组件来为 List 组件提供 width。

```
// height：视口高度
// isScrolling：表示是否滚动中，用来覆盖List组件自身的滚动状态
// scrollTop：页面滚动的距离，用来同步 List 组件的滚动距离
<WindowScroller>
  {({ height, isScrolling, scrollTop }) => {}}
</WindowScroller>
```

2.10 渲染房屋列表数据

3. InfiniteLoader 组件

- 需求：滚动房屋列表时，动态加载更多房屋数据。
- 解决方式：使用 InfiniteLoader 组件，来实现无限滚动列表，从而加载更多房屋数据。
- 根据 InfiniteLoader 组件文档 示例，在项目使用该组件。

```
// isRowLoaded 表示每一行数据是否加载完成
// loadMoreRows 加载更多数据的方法，在需要加载更多数据时，会调用该方法
// rowCount 列表数据总条数
<InfiniteLoader
  isRowLoaded={isRowLoaded} loadMoreRows={loadMoreRows}
  rowCount={remoteRowCount}
>
{({ onRowsRendered, registerChild }) =>{}}
</InfiniteLoader>
```

2.10 渲染房屋列表数据

4. 加载更多房屋列表数据

1. 在 loadMoreRows 方法中，根据起始索引和结束索引，发送请求，获取更多房屋数据。
2. 获取到最新的数据后，与当前 list 中的数据合并，再更新 state，并调用 Promise 的 resolve()。
3. 在 renderHouseList 方法中，判断 house 是否存在。
4. 不存在的时候，就渲染一个 loading 元素（防止拿不到数据时报错）。
5. 存在的时候，再渲染 HouseItem 组件。

2. 列表找房模块

2.11 条件筛选栏吸顶功能

1. 实现思路

- 在页面滚动的时候，判断筛选栏上边是否还在可视区域内。
- 如果在，不需要吸顶；
- 如果不在，就吸顶。
- 问题：吸顶后，元素脱标，房屋列表会突然往上跳动筛选栏的高度，如何解决？
- 解决方式：使用跟筛选栏高度相同的占位元素，在筛选栏脱标后，代替它撑起高度。

2.11 条件筛选栏吸顶功能

2. 实现步骤

① 封装 Sticky 组件。
② 在 HouseList 页面中，导入 Sticky 组件。
③ 使用 Sticky 组件包裹要实现吸顶功能的 Filter 组件。
④ 在 Sticky 组件中，创建两个 ref 对象（placeholder、content），分别指向占位元素和内容元素。
⑤ 组件中，监听浏览器的 scroll 事件（注意销毁事件）。
⑥ 在 scroll 事件中，通过 getBoundingClientRect() 方法得到筛选栏占位元素当前位置（top）。
⑦ 判断 top 是否小于 0（是否在可视区内）。
⑧ 如果小于，就添加需要吸顶样式（fixed），同时设置占位元素高度（与条件筛选栏高度相同）。
⑨ 否则，就移除吸顶样式，同时让占位元素高度为 0。

```
<Sticky>
  <Filter />
</Sticky>
```

2.11 条件筛选栏吸顶功能

3. 通用性优化

- 问题：如果 Filter 组件的高度为 100 像素，此时，应该怎么处理？。
- 处理方式：修改 Sticky 组件中，占位元素的高度为 100 像素。
- 如果其他地方也用到了 Sticky 组件，高度为 88 像素，该如何处理?
- 解决方式：哪个地方用到了，将当前高度通过 props 传给组件，组件内部通过 props 设置高度值。
- 封装原则：对变化点封装，变化点作为 props 动态设置。

```
<Sticky height={40}>
  <Filter />
</Sticky>
```

2.12 列表找房功能优化

1. 加载提示

1. 实现加载房源数据时：加载中、加载完成的提示（需要解决：没有房源数据时，不弹框提示）。
2. 找不到房源数据时的提示（需要解决：进入页面就展示该提示的问题）。

2.12 列表找房功能优化

2. 条件筛选栏优化

1. 点击条件筛选栏，展示 FilterPicker 组件时，样式错乱问题（需要解决：样式问题）。
2. 使用条件筛选查询数据时，页面没有回到顶部（需要解决：每次重新回到页面顶部）。

```
// 回到页面顶部
window.scrollTo(0, 0)
```

3. 点击条件筛选栏，展示对话框后，页面还会滚动（需要解决：展示对话框后页面不滚动）。

2. 列表找房模块

2.12 列表找房功能优化

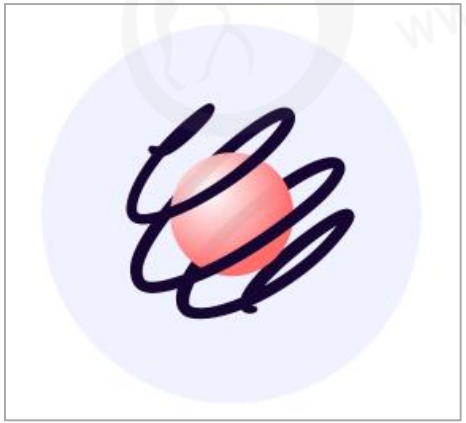
3. 切换城市显示房源

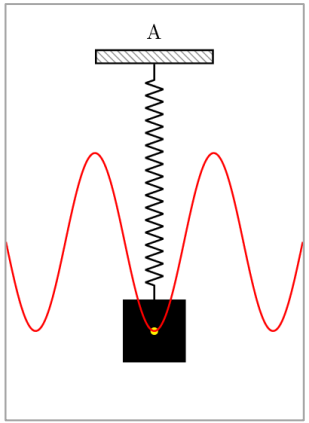
1. 切换城市后，该页面无法展示当前定位城市名称和当前城市房屋数据，刷新页面后才会生效（需要解决：切换城市后立即生效）。

2.13 react-spring 动画库

1. 概述

- 场景：展示筛选对话框的时候，实现动画效果，增强用户体验。
- react-spring 是基于 spring-physics（弹簧物理）的 react 动画库，动画效果更加流畅、自然。
- Github地址： react-spring （借鉴了 react-motion）。
- 优势1：几乎可以实现任意 UI 动画效果。
- 优势2：组件式使用方式（render-props 模式），简单易用、符合 react 的声明式特性、性能高。





2.13 react-spring 动画库

2. 基本使用

① 安装：yarn add react-spring 。

② 打开 Spring 组件文档（Spring 组件用来将数据从一个状态移动到另一个状态）。

③ 导入 Spring 组件，使用 Spring 组件包裹要实现动画效果的遮罩层 div。

④ 通过 render-props 模式，将参数 props（样式） 设置为遮罩层 div 的 style。

⑤ 给 Spring 组件添加 from 属性，指定：组件第一次渲染时的动画状态。

⑥ 给 Spring 组件添加 to 属性，指定：组件要更新的新动画状态。

```
<Spring
  from={{ opacity: 0 }}
  to={{ opacity: 1 }}>
  {props => <div style={props}>要实现动画的内容</div>}
</Spring>
```

2.13 react-spring 动画库

3. 实现遮罩层动画

① 创建方法 renderMask 来渲染遮罩层 div。

② 修改渲染遮罩层的逻辑，保证 Spring 组件一直都被渲染（Spring 组件都被销毁了，就无法实现动画效果）。

③ 修改 to 属性的值，在遮罩层隐藏时为 0，在遮罩层展示时为 1。

④ 在 render-props 的函数内部，判断 props.opacity 是否等于 0。

⑤ 如果等于 0，就返回 null（不渲染遮罩层），解决遮罩层遮挡页面导致顶部导航失效问题。

⑥ 如果不等于 0，渲染遮罩层 div。

目录 Contents

- 地图找房模块
- 列表找房模块
- 房屋详情模块

3. 房屋详情模块

3.1 功能分析

业务：根据房源 id，展示房源详情信息

功能：

- 使用路由参数获取房源 id
- 根据房源 id 获取房源详情数据并展示
- 使用地图展示所在小区位置信息
- 房源收藏（需要登录）

重点：路由参数。

3. 房屋详情模块

3.2 页面模板说明

1. 创建房屋详情页面 HouseDetail。
2. 修改 NavHeader 组件（添加了 className 和 rightContent 两个props）。
3. 创建了 HousePackage 组件（房屋配套）。

3. 房屋详情模块

3.3 路由参数

1. 概述

- 问题：房源有很多个，URL 路径也就很多个，需要多少个路由规则来匹配呢？ 一个还是多个？
- 答案：一个。
- 如何使用一个路由规则匹配不同的 URL 路径，同时获取到 URL 中的不同内容呢？
- 解决方式：路由参数。
- 作用：让一个路由规则，可以同时匹配多个符合该规则（格式）的 URL 路径。
- 语法：/detail/:id，其中 :id 就表示路由参数。

```
<Link to="/detail/1">房源1</Link>
<Link to="/detail/2">房源2</Link>
```

```
<Route path="/detail/:id" component=... />
能够匹配 /detail/1 或 /detail/2 等符合该规则的pathname
```

3. 房屋详情模块

3.3 路由参数

2. 获取路有参数

- 如何获取路由参数?
- 获取方式： props.match.params。
- params 是一个对象，对象中的属性名与路由规则中 /:id 名字相同。

```
class HouseDetail extends Component {
  componentDidMount() {
    // 获取路由参数
    const { params } = this.props.match
    // <Route paht="/detail/:id" component=… />
    const { id } = params
  }
}
```

3. 房屋详情模块

3.4 展示房屋详情

步骤

1. 在找房页面中，给每一个房源列表项添加单击事件，在点击时跳转到房屋详情页面。
2. 在单击事件中，获取到当前房屋 id。
3. 根据房屋详情的路由地址，调用 history.push() 实现路由跳转。
4. 封装 getHouseDetail 方法，在 componentDidMount 中调用该方法。
5. 在方法中，通过路由参数获取到当前房屋 id。
6. 使用 API 发送请求，获取房屋数据，保存到 state 中。
7. 使用房屋数据，渲染房屋详情。

好客租房移动Web（中）

1. 地图找房模块：百度地图API，地图覆盖物，CSS Modules 解决样式覆盖问题，脚手架环境变量，axios 公共 URL 配置。
2. 列表找房模块：条件筛选栏组件封装（变化点），房源列表，react-virtualized（InfiniteLoader、WindowScroller）、react-spring 动画库。
3. 房屋详情模块：路由参数（/:id 和 props.match.params），展示房屋详情。

传智播客旗下高端IT教育品牌