React细节知识点 - 实战1

1. React中使用styled-components

1. **React中使用css文件**

```
//1.根目录下新建style.css
//2.在根目录的index.js中导入style.css文件，因为index.js是项目入口，那么全局(其他组件)都可以使用style.css中的样式，也就是说组件的样式都可以写在style.css中

import React from 'react';
import ReactDOM from 'react-dom';
import './style.css';
import App from './App';

ReactDOM.render(<App />, document.getElementById('root'));
```

2. **使用styled-components**

```
//1.安装yarn add styled-components

//2.根目录创建style.js文件 -->为了在里面用reset.css，兼容更多浏览器
import { createGlobalStyle } from 'styled-components';

export const Globalstyle = createGlobalStyle`
html, body, div, span, applet, object, iframe,
h1, h2, h3, h4, h5, h6, p, blockquote, pre,
a, abbr, acronym, address, big, cite, code,
del, dfn, em, img, ins, kbd, q, s, samp,
small, strike, strong, sub, sup, tt, var,
b, u, i, center,
dl, dt, dd, ol, ul, li,
fieldset, form, label, legend,
table, caption, tbody, tfoot, thead, tr, th, td,
article, aside, canvas, details, embed,
figure, figcaption, footer, header, hgroup,
menu, nav, output, ruby, section, summary,
time, mark, audio, video {
  margin: 0;
  padding: 0;
  border: 0;
```

```
  font-size: 100%;
  font: inherit;
  vertical-align: baseline;
}
/* HTML5 display-role reset for older browsers */
article, aside, details, figcaption, figure,
footer, header, hgroup, menu, nav, section {
  display: block;
}
body {
  line-height: 1;
}
ol, ul {
  list-style: none;
}
blockquote, q {
  quotes: none;
}
blockquote:before, blockquote:after,
q:before, q:after {
  content: '';
  content: none;
}
table {
  border-collapse: collapse;
  border-spacing: 0;
}
`;

//3.在主组件里(其他子组件都导入在这个里面)App.js中
import React, { Component, Fragment } from 'react';
import { Globalstyle } from './style';
import Header from './common/header';

class App extends Component {
  render() {
    return (
      <Fragment>
        <Globalstyle />       //为了兼容更多浏览器
        <Header />            //头部子组件
      </Fragment>
    );
  }
}
export default App;
```

```
//4.在src目录下新建common文件夹，里面新建header文件夹，在header文件夹中新建index.js和style.js。这个index中就是写头部子组件的地方，style.js是写头部组件样式的地方
/*common/header/index.js*/
import React, { Component } from 'react';
import { HeaderWrapper, Logo, Nav } from './style';

class Header extends Component {
  render() {
    return (
      <HeaderWrapper>
        <Logo />
        <Nav />
      </HeaderWrapper>
    );
  }
}
export default Header;


/*common/header/style.js*/
import styled from 'styled-components';
import logoPic from '../../static/logo.png';

export const HeaderWrapper = styled.div`
  position: relative;
  height: 56px;
  border-bottom: 1px solid #f0f0f0;
`;

export const Logo = styled.a.attrs({
  href: '/'    //想要点击图片跳到根路径：第一种方法 <Logo *href*="/" />
})`           //第二种方法：export const Logo = styled.a.attrs({ href: '/' })`
  position: absolute;
  top: 0;
  left: 0;
  display: block; //a标签是内联标签，所以给它加width没有效果想改变a标签宽度怎么操作呢?
  height: 56px;   //想改变a标签宽度怎么操作=>把内敛标签变成块标签 => display: block
  width: 100px;
  background: url(${logoPic});  //直接使用图片的话webpack打包找不到路径
  background-size: contain;  //import logoPic from '../../static/logo.png';

`;                    //background: url(${logoPic}); 然后图片太大，不能完全显示->
                      //background-size: contain;

export const Nav = styled.div`
  width: 960px;
```

```
  height: 100%;
  padding-right: 70px;
  box-sizing: border-box;
  margin: 0 auto;
`;
```

实现左右浮动

```
/*header/index.js中*/
<Nav>
  <NavItem className="left">首页</NavItem>
  <NavItem className="left">下载App</NavItem>
  <NavItem className="right">登陆</NavItem>
  <NavItem className="right">Aa</NavItem>
</Nav>

/*header/style.js中*/
export const NavItem = styled.div`
  line-height: 56px;  //Navbar高是56px,我们把文字撑起来喝navbar一样
  padding: 0 15px;
  font-size: 17px;
  color: #333;
  &.left {            //当在NavItem组件中 同时(&) className="left", 则左浮动
    float: left;
  }
  &.right {
    float: right;
    color: #969696;
  }
  &.active {
    color: #ea6f5a;
  }
`;
```

搜索框

```
export const NavSearch = styled.input.attrs({
  placeholder: '搜索'
})`
  width: 160px;
  height: 38px;
  padding: 0 30px 0 20px;  //按正常的话 左右的20px padding会把width撑成200px
```

```
  margin-top: 9px;          //如果不想撑开怎么解决：box-sizing:border-box
  margin-left: 20px;
  box-sizing: border-box;
  border: none;
  outline: none;
  border-radius: 19px;
  background: #eee;
  font-size: 14px;
  color: #666;
  &::placeholder {
    color: #999;
  }
`;
```

包裹右侧按钮们

```
export const Addition = styled.div`
  position: absolute;
  right: 0;
  top: 0;
  height: 56px;
`;
```

按钮

```
export const Button = styled.div`
  float: right;
  margin-top: 9px;
  margin-right: 20px;
  padding: 0 20px;
  line-height: 38px;
  border-radius: 19px;
  border: 1px solid #ec6149;
  font-size: 14px;
  &.reg {
    color: #ec6149;
  }
  &.writting {
    color: #fff;
    background: #ec6149;
  }
`;
```

---

2. React中使用iconfont图标

1. **操作**

```
//1.先去iconfont.cn上选需要用的图标，然后下载至本地

//2.把我们需要的
iconfont.eot,iconfont.js,iconfont.svg,iconfont.ttf,iconfont.woff,woff2放入项目
的src/static目录下

//3.修改iconfont.css文件名 -> iconfont.js
//在其中，把src: url()括号里前面加./，删除最下面那些class，然后写成全局的形式
import { createGlobalStyle } from 'styled-components';

export const Globalstyle = createGlobalStyle`
...
...
`;

//4.在根目录的index.js中，导入这个全局样式;之前那个reset.css也在这里导入下
import './static/iconfont/iconfont';
```

但是当我们在组件里使用图标的时候发现页面上显示不出来，只能显示出小正方形。所以还是采用不把.css改成js的形式，直接在根目录index.js中导入，然后发现图标可以使用了

```
/*iconfont.css*/
@font-face {
  font-family: 'iconfont';
  src: url('./iconfont.eot?t=1560067477018'); /* IE9 */
  src: url('./iconfont.eot?t=1560067477018#iefix') format('embedded-
opentype'),
    /* IE6-IE8 */
      url('data:....省略)
      format('woff2'),
    url('./iconfont.woff?t=1560067477018') format('woff'),
    url('./iconfont.ttf?t=1560067477018') format('truetype'),
    /* chrome, firefox, opera, Safari, Android, iOS 4.2+ */
      url('./iconfont.svg?t=1560067477018#iconfont') format('svg'); /* iOS
4.1- */
}

.iconfont {
```

```
  font-family: 'iconfont' !important;
  font-size: 16px;
  font-style: normal;
  -webkit-font-smoothing: antialiased;
  -moz-osx-font-smoothing: grayscale;
}

/*跟目录index.js*/
import './static/iconfont/iconfont.css';
```

2. **Aa图标**

```
<NavItem className="right">
  <i className="iconfont">&#xe636;</i>
</NavItem>
```

3. **搜索框图标**

    1. **父级的相对定位：是要把子DIV的绝对定位的起点以父DIV左上角为起点，如果没有就默认屏幕左上角了。**
    2. **子DIV的：是无视父DIV下别的元素。然后就是left:0; top:0; 这个是用来定位的**
    3. **BFC 全称为块格式化上下文 (Block Formatting Context)**

```
//1.给<NavSearch>外面加个<SearchWrapper>，我们是想先为后面动画做个搜索框图标的背景(那个小圆)
/*header/index.js*/
<SearchWrapper>
  <NavSearch />
  <i className="iconfont">&#xe62d;</i>
</SearchWrapper>

/*header/style.js*/
export const SearchWrapper = styled.div`
  position: relative;      //SearchWrapper父div相对定位，iconfont子div绝对定位
  float: left;             //因为前面的Nav是浮动的，实际没有占位的，如果这边不写float，那么
  .iconfont {              //设置个Background,我们发现SearchWrapper实际占了整个<Nav>
    position: absolute;
    right: 5px;
    bottom: 5px;
    width: 30px;
    line-height: 30px;
    border-radius: 15px;
```

```
    background: green;
    text-align: center;
  }
`;
```

BFC 全称为 块格式化上下文 (Block Formatting Context) 。

一个块格式化上下文（block formatting context） 是Web页面的可视化CSS渲染出的一部分。它是块级盒布局出现的区域，也是浮动层元素进行交互的区域。
一个块格式化上下文由以下之一创建：

- 根元素或其它包含它的元素
- 浮动元素 (元素的 float 不是 none)
- 绝对定位元素 (元素具有 position 为 absolute 或 fixed)
- 内联块 (元素具有 display: inline-block)
- 表格单元格 (元素具有 display: table-cell，HTML表格单元格默认属性)
- 表格标题 (元素具有 display: table-caption，HTML表格标题默认属性)
- 具有overflow 且值不是 visible 的块元素，
- display: flow-root
- column-span: all 应当总是会创建一个新的格式化上下文，即便具有 column-span: all 的元素并不被包裹在一个多列容器中。
- 一个块格式化上下文包括创建它的元素内部所有内容，除了被包含于创建新的块级格式化上下文的后代元素内的元素。

块格式化上下文对于定位 (参见 float) 与清除浮动 (参见 clear) 很重要。定位和清除浮动的样式规则只适用于处于同一块格式化上下文内的元素。浮动不会影响其它块格式化上下文中元素的布局，并且清除浮动只能清除同一块格式化上下文中在它前面的元素的浮动。

//BFC特性
1. 使BFC内部浮动元素不会到处乱跑
  //(1)在正常的文档流中，块级元素是按照从上自下，内联元素从左到右的顺序排列的。如果我给里面的元素一个 float 或者绝对定位，它就会脱离普通文档流中。
  //(2)此时如果我们还想让外层元素包裹住内层元素该如果去做？？让外层元素产生一个 BFC 。(产生 BFC 的方法 MDN 文档里有写)
2. 和浮动元素产生边界

——

3. 搜索框动画效果实现

1. **操作**

```
//1. 通过this.state.focused中的一个变量来控制搜索框展开状态
//当focused=false，没展开，当focused=true，展开

//2.给<NavSearch>加className，然后加个样式，如果有className有focused，则宽度变为
240px
<NavSearch
  className={this.state.focused ? 'focused' : ''}
  onFocus={this.handleInputFocus}
  onBlur={this.handleInputBlur}
/>
handleInputFocus() {
  this.setState({
    focused: true
  });
}
handleInputBlur() {
  this.setState({
    focused: false
  });
}

NavSearch样式下面加：
  &.focused {
    width: 240px;
  }

//3.当搜索框展开时，搜索图标背景颜色也要变
<i className={this.state.focused ? 'focused iconfont':'iconfont'}>&#xe62d;
</i>
在SearchWrapper已有的.iconfont下添加样式：
&.focused {
  background: #777;
  color: #fff;
}

//4.上面已经实现了当点击搜索框，展开和图标背景变色效果，现在我们想添加动画，让展开缩回效果更
好看
//先安装yarn add react-transition-group
<SearchWrapper>
  <CSSTransition
    in={this.state.focused}      //控制什么时候进场
    timeout={200}                //动画时长
    classNames="slide"    //样式文件里要以这个classNames作为开头,注意是classNames有
个s
    >
    <NavSearch       //要动画的是这个标签，CSSTransition写在它的外面
```

```
        className={this.state.focused ? 'focused' : ''}
        onFocus={this.handleInputFocus}
        onBlur={this.handleInputBlur}
      />
    </CSSTransition>

在NavSearch样式里添加：.classNames-enter, .xx-enter-active, .xx-exit, .xx-exit-active这四个状态,这几个是和input同级的，所以需要加&
export const NavSearch = styled.input`
  &.slide-enter {
    transition: all 0.2s ease-out;
  }
  &.slide-enter-active {
    width: 240px;
  }
  &.slide-exit {
    transition: all 0.2s ease-out;
  }
  &.slide-exit-active {
    width: 160px;
  }
`
```

——

4. 使用react-redux进行数据管理

1. **操作**

```
//1.安装yarn add redux; yarn add react-redux

//2.根目录下新建store文件夹，在其中新建index.js和reducer.js文件
/*store/index.js*/
import { createStore } from 'redux';
import reducer from './reducer';

const store = createStore(reducer);

export default store;
/*store/reducer.js*/
const defaultState = {};

export default (state = defaultState, action) => {
```

```
  return state;
};

//3.根组件App.js中，加入Provider提供器包裹全部组件
import { Provider } from 'react-redux';

<Provider store={store}>
  <Fragment>
    <Globalstyle />
    <Header />
  </Fragment>
</Provider>

//4.组件中需要使用connect方法连接到store
/*header/index.js中*/
import { connect } from 'react-redux';

const mapStateToProps = state => {
  return {};
};

const mapDispatchToProps = dispatch => {
  return {};
};

export default connect(mapStateToProps, mapDispatchToProps)(Header);
```

2. **将通过constructor函数管理state 转换到 通过连接store获取数据**

```
//5.删除constructor函数，将focused: false放入defaultState
const defaultState = {
  focused: false
};
/*header/index.js中*/
//获取store里的focused，并在组件里使用，通过this.props.focused方式
const mapStateToProps = state => {
  return {
    focused: state.focused
  };
};

//6.将handleInputToProps和handleInputBlur方法放入mapDispatchToProps中
const mapDispatchToProps = dispatch => {
  return {
    handleInputFocus() {
      const action = {
```

```
        type: 'search_focus'
      };
      dispatch(action);
    },

    handleInputBlur() {
      const action = {
        type: 'search_blur'
      };
      dispatch(action);
    }
  };
};

//7.在reducer.js中进行基于action类型的对应操作处理
export default (state = defaultState, action) => {
  if (action.type === 'search_focus') {
    return {
      focused: true
    };
  }
  if (action.type === 'search_blur') {
    return {
      focused: false
    };
  }
  return state;
};
```

3. **使用redux-devtools-extension**

```
//在根目录store/index.js中
import { createStore, compose } from 'redux';
import reducer from './reducer';

const composeEnhancers = window.__REDUX_DEVTOOLS_EXTENSION_COMPOSE__ ||
compose;
const store = createStore(reducer, composeEnhancers());

export default store;
```

4. **使用combineReducers完成对数据的拆分**

```
//1.在header/store目录下新建reducer.js，这个专门从来管理header组件的store，将原本写在根目录store/reducer.js中的内容拿过来
/*header/store/reducer.js*/
const defaultState = {
  focused: false
};

export default (state = defaultState, action) => {
  if (action.type === 'search_focus') {
    return {
      focused: true
    };
  }
  if (action.type === 'search_blur') {
    return {
      focused: false
    };
  }
  return state;
};

//2.在header/store目录下新建index.js，用来导出store目录下的东西
/*header/store/index.js*/
import reducer from './reducer';

export { reducer };

//3.在根目录的store/reducer.js中导入header的reducer，合并到总的reducer中
/*store/reducer.js*/
import { combineReducers } from 'redux';
import { reducer as headerReducer } from '../common/header/store';  //写上面那个index.js的作用

const reducer = combineReducers({
  header: headerReducer           //组件中取数据的时候要注意：state.header.focused
});

export default reducer;

//3.在header/index.js中,接收store中的数据有些变化，因为这个数据是在header这个名字下的
const mapStateToProps = state => {
  return {
    focused: state.header.focused   //之前是focused: state.focused
  };
};
```

---

5. actionCreators与actionTypes的拆分

1. **操作**

```
//1.在store目录下新建actionCreators.js
import * as constants from './actionTypes';

export const searchFocus = () => ({
  type: constants.SEARCH_FOCUS
});

export const searchBlur = () => ({
  type: constants.SEARCH_BLUR
});


//2.在store目录下新建actionTypes.js
export const SEARCH_FOCUS = 'header/search_focus';
export const SEARCH_BLUR = 'header/search_blur';

//3.header/index.js中，创建action写法就需要改下
import { actionCreators } from './store';    //store目录下的文件全都在
store/index.js中导出，统一的出口

const mapDispatchToProps = dispatch => {
  return {
    handleInputFocus() {
      //之前写法
      const action = {
        type: 'search_focus'
      }
      dispatch(action)

      //现在使用了actionCreators.js的写法
      dispatch(actionCreators.searchFocus());
    },

    handleInputBlur() {
      dispatch(actionCreators.searchBlur());
    }
```

```
  };
};

//4.在header/store/reducer.js中，也需要对应的改下
import * as constants from './actionTypes';

export default (state = defaultState, action) => {
  if (action.type === constants.SEARCH_FOCUS) {
    return {
      focused: true
    };
  }
  if (action.type === constants.SEARCH_BLUR) {
    return {
      focused: false
    };
  }
  return state;
};

//5.我们把header/store目录下的所有文件都放在header/store/index.js中导出，统一的到出口
import reducer from './reducer';
import * as actionCreators from './actionCreators';
import * as constants from './actionTypes';

export { reducer, actionCreators, constants };
```

——

6. 使用immutable.js管理store中的数据

1. **操作**

```
//1.安装yarn add immutable

//2.在header/store/reducer.js中给数据使用immutable.js
import { fromJS } from 'immutable';

const defaultState = fromJS({
  focused: false
});

//3.在header/store/index.js中获取到store中的数据时的写法需要改下
```

```
const mapStateToProps = state => {
  return {
    //原本写法:
    focused: state.header.focused;
    //使用immutable之后获取store中数据的写法
    focused: state.header.get('focused');
  };
};

//4.在header/store/reducer.js中,进行操作时
export default (state = defaultState, action) => {
  if (action.type === constants.SEARCH_FOCUS) {
    //immutable对象的set方法，会结合之前immutable对象的值和设置的值，返回一个全新的对象
    //原本写法:
    return {
      focused: true;
    }
    //使用immutable之后写法:
    return state.set('focused', true);
  }
  if (action.type === constants.SEARCH_BLUR) {
    return state.set('focused', false);
  }
  return state;
};
```

2. **header/store/index.js中获取store中数据的写法*focused: state.get('header').get('focused')，前面的state是JSX，后面的focused是immutable写法，很乱。所以我们想能不能把state也变成immutable对象***

```
//1.安装yarn add redux-immutable

//2.在根目录的store/reducer.js中使用redux-immutable
import { combineReducers } from 'redux';
改成:
import { combineReducers } from 'redux-immutable';
这样我们的store就变成了immutable对象

//3.在header/index.js中获取store中的数据时写法需要改下
const mapStateToProps = state => {
  return {
    //原本写法:
    focused: state.header.get('focused');
    //state变成immutable后的写法(下面这两种都行)
```

```
    focused: state.getIn(['header', 'focused'])
    // focused: state.get('header').get('focused')
  };
};
```

---

7. 热门搜索样式布局

1. **操作**

```
//1.<SearchInfo>
export const SearchInfo = styled.div`
  position: absolute;
  left: 0;
  top: 56px;
  width: 240px;
  padding: 0 20px;
  box-shadow: 0 0 8px rgba(0, 0, 0, 0.2);
`;

//2.<SearchInfoTitle>
<SearchInfoTitle>
  热门搜索
  <SearchInfoSwitch>换一批</SearchInfoSwitch>
</SearchInfoTitle>

export const SearchInfoTitle = styled.div`
  margin-top: 20px;
  margin-bottom: 15px;
  line-height: 20px;
  font-size: 14px;
  color: #969696;
`;

//3.<SearchInfoSwitch>
export const SearchInfoSwitch = styled.span`
  float: right;
  font-size: 13px;
`;

//4.<SearchInfoList>
<SearchInfoList>
```

```
  <SearchInfoItem>教育</SearchInfoItem>
  <SearchInfoItem>简书</SearchInfoItem>
  <SearchInfoItem>投稿</SearchInfoItem>
</SearchInfoList>

/*触发bfc，我们发现SearchItem会超出最外面的框，所以我们把包裹SearchItem的div改成
SearchInfoList,在这个样式里触发BFC，那么外面的框会自动扩张包裹住所有的SearchItem，触发
方式就是overflow:hidden*/
export const SearchInfoList = styled.div`
  overflow: hidden;
`;

//5.<SearchInfoItem>
export const SearchInfoItem = styled.a`
  display: block;
  float: left;
  line-height: 20px;
  padding: 0 5px;
  margin-right: 10px;
  margin-bottom: 15px;
  font-size: 12px;
  border: 1px solid #ddd;
  border-radius: 3px;
  color: #787878;
`;

//6.当点击搜索框时，热门搜索布局框弹出来，如何写呢?
//我们用一个变量show的true/false来控制是否显示热门搜索布局框
//很明显这个show实际就用focused这个值来控制
class Header extends Component {
getListArea = show => {
  if (show) {
    return (
      <SearchInfo>
        <SearchInfoTitle>
          热门搜索
          <SearchInfoSwitch>换一批</SearchInfoSwitch>
        </SearchInfoTitle>
        <SearchInfoList>
          <SearchInfoItem>教育</SearchInfoItem>
          <SearchInfoItem>简书</SearchInfoItem>
          <SearchInfoItem>投稿</SearchInfoItem>
          <SearchInfoItem>教育</SearchInfoItem>
          <SearchInfoItem>教育</SearchInfoItem>
          <SearchInfoItem>教育</SearchInfoItem>
        </SearchInfoList>
```

```
      </SearchInfo>
    );
  } else {
    return null;
  }
};
...
...
  {this.getListArea(this.props.focused)}
</SearchWrapper>
```

2. **Ajax获取推荐数据**

```
//1.安装yarn add redux-thunk

//2.在根目录store/index.js中配置redux-thunk，因为中间件其实就是action和store的中间
import { createStore, compose, applyMiddleware } from 'redux';
import thunk from 'redux-thunk';
import reducer from './reducer';

const composeEnhancers = window.__REDUX_DEVTOOLS_EXTENSION_COMPOSE__ ||
compose;
const store = createStore(reducer, composeEnhancers(applyMiddleware(thunk)));

export default store;

//3.发送异步请求需要用到第三方模块
(1)yarn add axios

(2)/*public/api/headerList.json中放入假数据*/
{
  "success": true,
  "data": [
    "高考",
    "勇士vs猛龙",
    "React",
    "Vue",
    "三生三世",
    "小程序",
    "区块链",
    "厦门吃货",
    "鼓浪屿",
    "Facebook",
    "Flutter",
```

```
    "知乎",
    "新闻联播",
    "华为芯片",
    "高校招生"
  ]
}

(3)handleInputFocus() {
  dispatch(actionCreators.getList());     //发送异步请求action
  dispatch(actionCreators.searchFocus());
},

在/*actionCreators.js中*/
const changeList = data => ({   //getList异步请求获取到数据放入store中派发的action
  type: constants.CHANGE_LIST,
  data: fromJS(data)     //注意这边的data要是immutable类型，因为defaultState中的
list
});                      //是immutable类型，更新list的值同时类型不一致的话，会报错

export const getList = () => {
  return dispatch => {
    axios
      .get('/api/headerList.json')     //为什么异步请求函数中也要派发一个action?
      .then(res => {      //这是因为这边异步请求获取到数据，我们要把数据放入store中
        const data = res.data; //修改store中数据必须是通过派发action，reducer接收处
理
        dispatch(changeList(data.data));
      })
      .catch(() => {
        console.log('error');
      });
  };
};

(4)/*actionTypes.js中定义常量*/
export const CHANGE_LIST = 'header/change_list';

(5)/*header/store/reducer.js中进行处理*/
if (action.type === constants.CHANGE_LIST) {
  return state.set('list', action.data);
}

//4.header/index.js中获取到store中更新的list
const mapStateToProps = state => {
  return {
    focused: state.getIn(['header', 'focused']),
```

```
    list: state.getIn(['header', 'list'])    //这里!
  };
};

JSX模板中遍历list输出:
<SearchInfoList>
 {this.props.list.map(item => {
   return <SearchInfoItem key={item}>{item}</SearchInfoItem>;
 })}
</SearchInfoList>


getListArea()这个函数原本是通过show这个变量，传focused进去，现在直接用
this.props.focused就行了，不用写show
getListArea() {
  if(this.props.focused) {
    return (
      ...
    )
  } else {
    return null;
  }
}
```

---

8. 代码优化

1. 操作

```
//1.getListArea(){
  const { focused, list } = this.props;
  //下面this.props.focused就可以直接写focused
  //list同上
}

//2.JSX模板中
  const { focused, handleInputFocus, handleInputBlur } = this.props;

//3.在header/store/reducer.js中，把之前if形式改成switch形式
export default (state = defaultState, action) => {
  switch (action.type) {
    case constants.SEARCH_FOCUS:
```

```
      return state.set('focused', true);
    case constants.SEARCH_BLUR:
      return state.set('focused', false);
    case constants.CHANGE_LIST:
      return state.set('list', action.data);
    default:
      return state;
  }
};
```

---

9. 热门搜索换页功能

1. 热门搜索框展示

```
//1.header/store/reducer.js中
const defaultState = fromJS({
  focused: false,
  list: [],
  page: 1,          //新增
  totalPage: 1      //新增
});

//2.header/store/actionCreators.js中
//请求获取到数据后，派发action修改store中数据时候，把totalPage也传过去，这样我们就能在reducer.js中操作更新store中的totalPage了
const changeList = data => ({
  type: constants.CHANGE_LIST,
  data: fromJS(data),
  totalPage: Math.ceil(data.length / 10)
});

//3.在header/store/reducer.js中
case constants.CHANGE_LIST:
  return state.set('list', action.data).set('totalPage', action.totalPage);

//4.header/index.js中展示内容
const mapStateToProps = state => {
  return {
    focused: state.getIn(['header', 'focused']),
    list: state.getIn(['header', 'list']),
    page: state.getIn(['header', 'page'])        //新增
```

```
  };
};

class Header extends Component {
  getListArea() {
    const { focused, list, page } = this.props;
    //因为list是一个immutable类型，不能直接用list[i]，先把它转成js类型
    const newList = list.toJS();
    const pageList = [];

    for (var i = (page - 1) * 10; i < page * 10; i++) {
      pageList.push(
        <SearchInfoItem key={newList[i]}>{newList[i]}</SearchInfoItem>
      );
    }
```

2. **搜索框缩回时热门搜索框扔显示功能**

```
//1.之前控制搜索框变化和热门搜索框弹出的是focused这个变量，但是仅仅只有它是不够的，因为我们发现当我想点击换一批的时候，框消失了，说明focused变为false了。我们的解决方法是再添加一个控制变量，那就是我鼠标还是不是在这个热门搜索框内mouseIn:false
先在reducer.js中defaultState中添加一个新变量：mouseIn:false

//2.给<SearchInfo>添加onMouseEnter和onMouseLeave方法
现在reducer.js中defaultState中添加一个新变量：mouseIn:false
const { focused, list, page, handleMouseEnter, handleMouseLeave } =
this.props;
    ...
<SearchInfo onMouseEnter={handleMouseEnter} onMouseLeave={handleMouseLeave}
/>

  handleMouseEnter() {              //onMouseEnter实行后,store中mouseIn修改为true
    dispatch(actionCreators.mouseEnter());
  },

  handleMouseLeave() {              //onMouseLeave实行后,store中mouseIn修改为
false
    dispatch(actionCreators.mouseLeave());
  }

//3.在actionCreators.js中写action
export const mouseEnter = () => ({
  type: constants.MOUSE_ENTER
});
```

```
export const mouseLeave = () => ({
  type: constants.MOUSE_LEAVE
});
/*actionTypes中写常量*/
export const MOUSE_ENTER = 'header/mouse_enter';
export const MOUSE_LEAVE = 'header/mouse_leave';

//4.在reducer.js中进行操作
case constants.MOUSE_ENTER:
  return state.set('mouseIn', true);
case constants.MOUSE_LEAVE:
  return state.set('mouseIn', false);

//5.在header/index.js中使用mouseIn来和focused一起控制热门搜索框的显示
const mapStateToProps = state => {
  return {
    focused: state.getIn(['header', 'focused']),
    list: state.getIn(['header', 'list']),
    page: state.getIn(['header', 'page']),
    mouseIn: state.getIn(['header', 'mouseIn'])     //新增
  };
};

const { focused, list, page, mouseIn, handleMouseEnter,
handleMouseLeave}=this.props;

if (focused || mouseIn) {       //这里
   return (
      <SearchInfo
        onMouseEnter={handleMouseEnter}
        onMouseLeave={handleMouseLeave}
      >
        <SearchInfoTitle>
        热门搜索
          <SearchInfoSwitch>换一批</SearchInfoSwitch>
        </SearchInfoTitle>
        <SearchInfoList>{pageList}</SearchInfoList>
      </SearchInfo>
    );
  } else {
    return null;
}
```

3. **换一批功能实现**

```
//1.给换一批标签添加事件,我们需要把当前页和总页数一起传给传给reducer，在reducer中更新page为下一页(我们在这个函数里就先判断如果当前页<总页数，那么page+1,否则应该跳到第一页，page=1)
<SearchInfoSwitch onClick={() => handleChangePage(page, totalPage)} />
//因为我们需要用到总页数，所以从store中把totalPage拿过来
const mapStateToProps = state => {
  return {
    focused: state.getIn(['header', 'focused']),
    list: state.getIn(['header', 'list']),
    page: state.getIn(['header', 'page']),
    totalPage: state.getIn(['header', 'totalPage']),      //新增
    mouseIn: state.getIn(['header', 'mouseIn'])
  };
};

//把下一页传个reducer，reducer更新store里的page
  handleChangePage(page, totalPage) {
    if (page < totalPage) {
      dispatch(actionCreators.changePage(page + 1));
    } else {
      dispatch(actionCreators.changePage(1));
    }
  }

/*actionCreators.js*/
export const changePage = page => ({
  type: constants.CHANGE_PAGE,
  page
});

/*actionTypes.js*/
export const CHANGE_PAGE = 'header/change_page';

/*reducer.js*/
case constants.CHANGE_PAGE:
  return state.set('page', action.page);

//2.修改个warning，header/index.js中
for (var i = (page - 1) * 10; i < page * 10; i++) {
  pageList.push(
    <SearchInfoItem key={newList[i]}>{newList[i]}</SearchInfoItem>
  );
}
```

这边为什么key会报警告呢？如果我们打印下console.log(newList[i]),发现控制台有10个undefined，这是为什么呢？这是因为一开始的时候，newList是用defaultState中的list,是空数组，那么再执行上面这段代码，就肯定会报警告了。我们是希望异步获取到数据后，再执行上面这个代码，初始化还没获取异步数据的时候，不要执行。其实加个if判断就行了

```
if (newList.length) {
  for (var i = (page - 1) * 10; i < page * 10; i++) {
    pageList.push(
      <SearchInfoItem key={newList[i]}>{newList[i]}</SearchInfoItem>
    );
  }
}

//3.代码优化，在header/store/reducer.js中
return state.set('list', action.data).set('totalPage', action.totalPage);
可以写成：
return state.merge({list: action.data, totalPage: action.totalPage});
```

——

10. 换页旋转动画效果的实现

1. **操作**

```
//1.iconfont.cn上找一个旋转的图标，下载下来

//2.在换一批文字上面加入旋转图标，发现旋转图标位置在热门搜索框右下角，这是因为之前的搜索图标的设置是显示在右下角，所以我们给之前搜索图标多加一个class zoom，并且把style.js中搜索图标的部分改为.zoom.
<i className="iconfont spin">&#xe7e9;</i>
export const SearchInfoSwitch = styled.span`
  float: right;
  font-size: 13px;
  .spin {                    //旋转图标
    display: block;
    float: left;
    font-size: 12px;
    margin-right: 4px;
    transition: all 0.5s ease-in;
    transform-origin: center center;  //以自己为中心旋转
  }
`;

//3.通过ref获取到dom节点
```

```
<i ref={icon => { this.spinIcon = icon; }} className="iconfont spin">&#xe7e9;
</i>
```

换一批有一个onClick函数，我们在这个函数里直接把这个dom节点传过去

```
<SearchInfoSwitch
  onClick={() => handleChangePage(page, totalPage, this.spinIcon)}
>
  handleChangePage(page, totalPage, spin) {
  let originAngle = spin.style.transform.replace(/[^0-9]/gi, '');
  if (originAngle) {
    originAngle = parseInt(originAngle, 10);
  } else {
    originAngle = 0;
  }
  spin.style.transform = 'rotate(' + (originAngle + 360) + 'deg)';

  if (page < totalPage) {
    dispatch(actionCreators.changePage(page + 1));
  } else {
    dispatch(actionCreators.changePage(1));
  }
}
```

——

11. 避免无意义的ajax请求

1. **操作**

```
//1.我们发现每次点击搜索框时都会发送一次ajax请求，其实只需要第一次发送就行了。如何改呢？发ajax请求的操作是在handleInputFocus函数里进行的，所以我们给它传个list,下面函数里接收list,console.log(list)看，发现每次点击第一次点击搜索框size:0，第二次点击搜索框size:有值，所以我们可以根据size的值来判断要不要发ajax请求了
handleInputFocus(list) {
  if (list.size == 0) {    //==0就是list中没数据的时候，才去ajax请求数据
    dispatch(actionCreators.getList());    //ajax请求
  }
  dispatch(actionCreators.searchFocus());
},

//2.当我们鼠标放在换一批上时候，想让鼠标显示手指形状
/*header/style.js中*/SearchInfoSwitch中加入：
cursor: pointer;
```