React.js

基本语法 / 功能

1. 用 react.js 实现类似 vue.js 的 v-if / v-for 的指令

```
    const tag = <div>real content </div>;
    const tag2 = <p>false content</p>;
    const data = {
    bool: true,
    content: ['你好', '你是谁', '外国人']
    }
    const methods = {

        vIf: (bool, tag1, tag2) => {
            return bool ? tag1 : tag2
        },
        vFor: array => {
            return array.map((item, index) => {
                return <p key={index}>{item}</p>
            })
        }
    }
```

2. `toggle` 功能

```
    import React from 'react';//引进react

    class Header extends React.Component {   //构建组件
        constructor(props) {
            super(props)
            this.state = { isToggleOn: true }
            this.handleClick = this.handleClick.bind(this)//修正点击事件的
this指向
        }

        handleClick() {
            this.setState(prevState => ({
                isToggleOn: !prevState.isToggleOn
            }))
        }

        render() {
            const tag1 = <div>我是第一条内容</div>
            const tag2 = <div>我是第二条内容</div>
            return (
                <div onClick={this.handleClick}>
```

```
                    {this.state.isToggleOn ? tag1 :tag2}
                </div>
            )
        }
    }

export default Header; //暴露Header组件
```

3. 主件 render()

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

4. 父子通信

- 子拿父的内容通过 props, 父拿子的内容通过回调。
- Father.js

```
import React from 'react';
import SubBtn from './children'
export default class bigBtn extends React.Component {
    constructor(props) {
        super(props)
        this.state = {
            hideContent: '点击让我消失',
            btnContent: '点击显示子组件',
            isShow: true,
        }
        // this.callback = this.callback.bind(this);
    };
    callback(childState) {
        // 记录下子组件传递过来的值
        this.childState = childState;
        // console.log(this.childState)
        this.setState({
            isShow: this.childState
        })
    };
    render() {
        console.log(this)
        return <div>
            <div
                style={{
                    width: '200px',
                    height: '50px',
                    cursor: 'pointer',
                    background: 'tomato',
                    lineHeight: '50px',
                    textAlign: 'center'
                }}
```

```
                onClick={
                    () => {
                        this.setState({
                            isShow: !this.state.isShow
                        })
                    }
                }
            >
                {this.state.btnContent}
            </div>
            {this.state.isShow ? (<SubBtn
                hideContent={this.state.hideContent}
                isShow={this.state.isShow}
                callback={this.callback.bind(this)}
            />) : ''}
        </div>
    };
}
```

- Children.js

```
import React from 'react';

export default class subBtn extends React.Component {
    constructor(props) {
        super(props)
        this.state = {
            show: this.props.isShow
        }

    };
    showTrigger() {
        const {
            callback
        } = this.props;
        var show = !this.state.show;

        this.setState({
            show: show
        });

        if (callback) {
            // 将子组件改变后的状态值传给父级
            callback(show);
        }
    }
    render() {
        return <div
            style={{
                width: '200px',
                height: '50px',
```

```
                cursor: 'pointer',
                background: 'tomato',
                lineHeight: '50px',
                textAlign: 'center'
            }}
            onClick={this.showTrigger.bind(this)}
        >
            {this.props.hideContent}
        </div>
    };
}
```

5. 语法

    1. 暴露一个类组件

    ```
    import React from 'react';
    export default class bigBtn extends React.Component
    ```

    2. 在标签内写样式 style

    ```
    <div style={{width: '200px',height: '50px',}}></div>
    ```

    3. 在标签内写事件

    ```
    <div onClick={this.showTrigger.bind(this)}></div> //注意this指向
    ```

    4.

    ```
    ```
    class A {}
    class B extends A {
    constructor() {
    super(); // ES6 要求，子类的构造函数必须执行一次 super 函数，否则会报错。
    }
    }
    ```
    ```

    - 注：在 constructor 中必须调用 super 方法，因为子类没有自己的 this 对象，而是继承父类的 this 对象，然后对其进行加工，而 super 就代表了父类的构造函数。super 虽然代表了父类 A 的构造函数，但是返回的是子类 B 的实例，即 super 内部的 this 指的是 B，因此 super() 在这里相当于

```
    A.prototype.constructor.call(this, props)
```

路由

> 需要的依赖：`react-router-dom`

- **index.js** demo

```
/* 引进分页面自定义模块  */
import Home from './pages/Home';
import Details from './pages/Details';
import TogglePlus from './components/father';

/* 引进路由依赖*/
import { HashRouter as Router, Route, Link, Redirect, Switch } from
"react-router-dom";

ReactDOM.render(
    <Provider store={store}>
        <Router>
            <Switch>
                <Route exact path="/home" component={Home} />
                <Route path="/detail" component={Details} />
                <Route path="/togg" component={TogglePlus} />
            </Switch>
            <Redirect to="/home" />
        </Router>
    </Provider>, document.getElementById('root'));
```

- **Home.js** demo

```
import React, { Component } from 'react'
import Search from '../components/Search';
import Panel from '../components/Panel';
import {Link} from "react-router-dom";

export default class Home extends Component {
    render() {
        return (
            <div>
                <Link
                    style={{
                        width: '100%',
                        display: 'block',
                        height: '50px',
                        background: 'tomato',
                        textAlign: 'center',
                        lineHeight: '50px'
                    }}
                    to={{
```

```
                        pathname: "/detail",
                        search: "?sort=name",
                        hash: "#the-hash",
                        state: { fromDashboard: true }
                    }}>detail</Link>
                <Search />
                <Panel />
            </div>
        )
    }
}
```

搜索关键字渲染相应内容

> 解决方案：filter(),map()

```
filterNews(arr, searchInputText) {
        return arr.filter((item) => {
            if (item.title.indexOf(searchInputText) > -1) {
                return item
            }
        }).map((item, index) => {
            return (//要渲染的内容)
        })

render(){
    return <div
            {this.filterNews(this.state.news, this.props.searchInputText)}
            </div>
```

获取滚动条到底部的高度

```
componentDidMount(){
    window.addEventListener('scroll', this.handleScroll);
  }

  handleScroll(){
    console.log(window.scrollY)
  }
```

轮播图 Swiper 插件

> 依赖：swiper

```
import Swiper from 'swiper/dist/js/swiper.js';

componentDidMount () {

        const mySwiper = new Swiper('.swiper-container', {
          autoplay: true,
          auto:5000,
          loop: true,
          pagination : {
              el: '.slidesjs-pagination-item',//焦点类名
              clickable: true,

          }
        });
  }
//结构demo
   <div className="swiper-container sroll-banner swiper-container-
horizontal swiper-container-autoheight">
            <div className="swiper-wrapper"
            >
                <div className="swiper-slide swiper-slide-duplicate
swiper-slide-next swiper-slide-duplicate-prev"
                    data-swiper-slide-index="0" ><img

src="http://images.cmbchina.com/cmbadv/201906/3d8bdd98-5438-41c5-9397-
8bf3a342ae12.jpg"
                        width="100%" height="100%" alt="" keys="0"
className="J_ScrollImgs"
                     />

                        </div>
                <div className="swiper-slide swiper-slide-duplicate-active
swiper-slide-prev swiper-slide-duplicate-next"
                    data-swiper-slide-index="0" ><img

src="http://images.cmbchina.com/cmbadv/201906/3d8bdd98-5438-41c5-9397-
8bf3a342ae12.jpg"
                        width="100%" height="100%" alt="" keys="0"
className="J_ScrollImgs"
                       />

                        </div>
                <div className="swiper-slide swiper-slide-duplicate
swiper-slide-active swiper-slide-duplicate-prev"
                    data-swiper-slide-index="0" ><img

src="http://images.cmbchina.com/cmbadv/201906/3d8bdd98-5438-41c5-9397-
8bf3a342ae12.jpg"
                        width="100%" height="100%" alt="" keys="0"
className="J_ScrollImgs"
                     />

                        </div>
```

```
            </div>
            <ul className="slidesjs-pagination swiper-pagination-custom">
                <li className="slidesjs-pagination-item"><a
className="active"></a></li>
            </ul>
        </div>
```

路由的跳转

```
//前往某一页
   toExDetail(index){
        this.props.history.push('/exdetail/'+index)
    };
//返回上一页
    goBack(){
        window.history.back()
    }
```