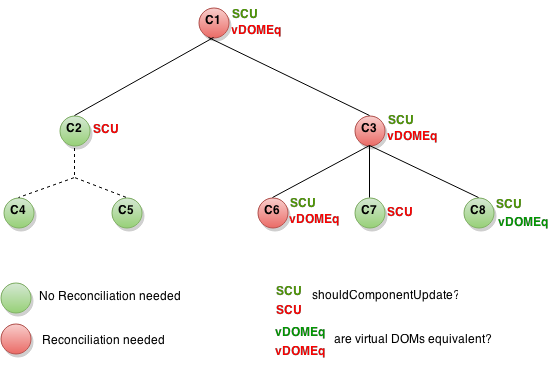
**1. React 的渲染机制————Reconciliation**

如果某个props或者state发生了变化，引起相应元素发生变化，则该元素以及该元素的所有父元素的shouldComponentUpdate 都会返回true，都会被标记为需要进行diff比对，并且父而子一个个寻找真正需要重新渲染的元素，确认不是所需寻找的对象以后直接跳过，如果是需要重新渲染的元素，则删除原DOM树该元素以及该元素的所有子节点，并复制新DOM树上该元素以及该元素的所有子节点，重新渲染。



React 为函数组件提供了一个 memo 方法，它和 PureComponent 在数据比对上唯一的区别就在于 只进行了 props 的浅比较，因为函数组件是没有 state 的。而且它的用法很简单，直接将函数传入 memo 中导出即可。形如:

**function** **Home** () {

//xxx

} **export** **default** React.memo (Home);

**划重点：**

**Diff性能优化分为两步：**

1. **使用React.memo（组件名），使之只进行浅对比**
2. **使用immutable，在更新节点的时候实现所有上级组件的更新（即重新生成内容相同的新对象），保证即使是浅对比也能发现更新**

**2.路由**

通过一个路由的配置文件设定好路由结构，在该路由文件中引用所有节点所需要用到的组件，最后在根节点通过声明

{ renderRoutes(routes) }

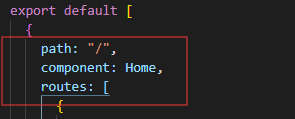
使用路由。

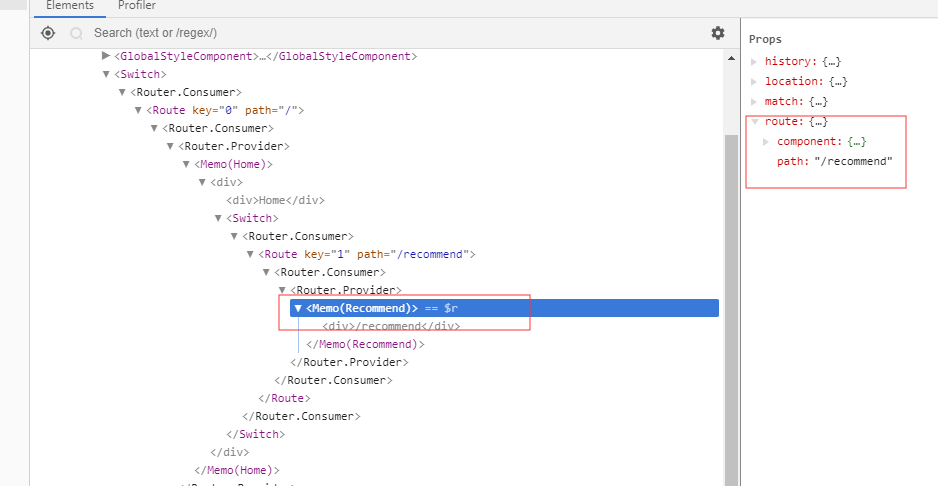
重点：嵌套路由。需要在产生嵌套的父组件内再次声明

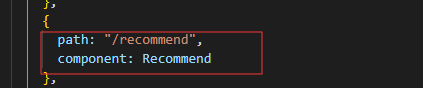
{ renderRoutes(routes) }

这里有个重点，每个组件的props默认都携带了route对象，且route对象的内容与react-route-config文件中的内容保持一致。









1. **图标的使用**

<https://segmentfault.com/a/1190000016235900>

1. **Redux**

使用immutable的目的在于性能优化，只需要momo进行浅层比较即可完成diff过程。

1. **关于NavLink**

每一个一级路由代表的都是一整各页面，一个整的页面中可能会包含许多小页面，从而形成了二级路由三级路由...但是如果NavLink作为二级路由的导航栏，则NavLink应该被放置在一级路由层次中，这样在点击导航跳转时只有对应的二级路由部分会重新渲染，NavLink所在的一级路由部分不会被渲染，从而也就能保证NavLink不会被渲染冲刷掉。

1. **通过设置margin\padding属性实现空div的占位**

容器的margin\padding使用百分比时是以父元素的width为基准的，所以容器的'padding-top:100%'就是父元素的width，同时不能使用margin的理由是如果用了margin容器就没有高度了，但是用padding有，而架设一个<img />元素是容器的子元素，设置他的height：100%，这样高度酒取决于容器的高度，如果用了margin，img高度就为0，反之padding则不会。

1. **react中ref的使用**

**问题1：ref是用来干嘛的？**

**答：**可以像document.getElementByXXX一样获取dom节点或者直接获取创建的class组件，他用dom或者组件的相关属性和方法。

**问题2：既然有了dom为什么还要使用ref？**

**答：**据网上资料说，react的思想是尽量减少开发者直接对dom的操作，如果真的要操作dom可以通过react提供的ref方法来进行操作。

**使用教程：**

**方式1：**通过React.createRef()直接将创建好的ref绑定到DOM节点或者Class组件。

**DOM节点：**

class MyComponent extends React.Component {

  constructor(props) {

    super(props);

    this.myRef = React.createRef();

  }

  render() {

    return <div ref={this.myRef} />;

  }

}

**Class组件：**

class AutoFocusTextInput extends React.Component {

  constructor(props) {

    super(props);

    this.textInput = React.createRef();

  }

  componentDidMount() {

    this.textInput.current.focusTextInput();

  }

  render() {

    return (

      <CustomTextInput ref={this.textInput} />

    );

  }

}

**方式2：**通过设置DOM节点或者class组件的ref属性绑定回调函数，从而通过ref将该节点或class组件传递给该回调函数，从而实现在其他地方调用该组件或者函数的相关属性或者参数。

class CustomTextInput extends React.Component {

  constructor(props) {

    super(props);

    this.textInput = null;

    this.setTextInputRef = element => {

      this.textInput = element;

    };

    this.focusTextInput = () => {

      // 直接使用原生 API 使 text 输入框获得焦点

      if (this.textInput) this.textInput.focus();

    };

  }

  componentDidMount() {

    // 渲染后文本框自动获得焦点

    this.focusTextInput();

  }

  render() {

    // 使用 `ref` 的回调将 text 输入框的 DOM 节点存储到 React

    // 实例上（比如 this.textInput）

    return (

      <div>

        <input

          type="text"

          ref={this.setTextInputRef}

        />

        <input

          type="button"

          value="Focus the text input"

          onClick={this.focusTextInput}

        />

      </div>

    );

  }

}

**方式1和方式2的比较：**

**方式1：**先创建ref，再将ref绑定到DOM节点

**方式2：**先指定DOM节点，绑定ref回调函数，再在回调函数中使用获取到的DOM节点

**注意点：函数式组件**不能使用ref，但是**函数式组件内部**可以使用ref，如下面这个是可以的。

function CustomTextInput(props) {

  return (

    <div>

      <input ref={props.inputRef} />

    </div>

  );

}

1. **更好的ref使用方式：ref转发**

通俗的解释：将ref作为参数传递给组件的子节点。例：

const FancyButton = React.forwardRef((props, ref) => (

  <button ref={ref} className="FancyButton">

    {props.children}

  </button>

));

// 你可以直接获取 DOM button 的 ref：

const ref = React.createRef();

<FancyButton ref={ref}>Click me!</FancyButton>;

这样上例中的ref就可以直接调用button的相关方法属性了。

1. **useRef和useCallback**

UseRef相当于7中的方式1，useCallback相当于7中的方式2。

**useRef:**

function TextInputWithFocusButton() {

  const inputEl = useRef(null);

  const onButtonClick = () => {

    // `current` 指向已挂载到 DOM 上的文本输入元素

    inputEl.current.focus();

  };

  return (

    <>

      <input ref={inputEl} type="text" />

      <button onClick={onButtonClick}>Focus the input</button>

    </>

  );

}

**useCallback:**

function MeasureExample() {

  const [height, setHeight] = useState(0);

  const measuredRef = useCallback(node => {

    if (node !== null) {

      setHeight(node.getBoundingClientRect().height);

    }

  }, []);

  return (

    <>

      <h1 ref={measuredRef}>Hello, world</h1>

      <h2>The above header is {Math.round(height)}px tall</h2>

    </>

  );

}

**10.propTypes**

import PropTypes from 'prop-types';

class Greeting extends React.Component {

  render() {

    return (

      <h1>Hello, {this.props.name}</h1>

    );

  }

}

Greeting.propTypes = {

  name: PropTypes.string

};

// 指定 props 的默认值：

Greeting.defaultProps = {

  name: 'Stranger'

};

**两个关键点：**

1. 通过‘组件名.propTypes’设置需要传递的props类型
2. 通过‘组件名.defaultProps’设置props的默认值

### 11.useImperativeHandle

用途：自定义ref转发时传递给父组件的实例值

1. **Axios**

axios使用步骤：

   1.创建一个axios实例并配置axios相关属性，export出去以供调用

   2.调用上一步创建好的axios实例，使用它的get，post...方法

   3.可以设置拦截器在axios请求发出前和接收后对数据进行处理

1. **immutable.js**

在immutable.js中，object经过fromjs函数以后默认转成map，array经过fromjs以后默认转成list

1. 柯里化
2. redux-thunk
3. Suspense和lazy