知识补充

cra项目中的css模块化

1. 创建xx.module.css
2. 组件中写法

```
import style from './xx.module.css'

<div className={style.box}>
```

知识点

资源

ant design

使用第三方组件:

- 安装: `npm install antd --save`
- 配置按需加载

  安装react-app-rewired取代react-scripts，可扩展webpack的配置，类似vue.config.js

  `npm install react-app-rewired customize-cra babel-plugin-import -D`

```
//根目录创建config-overrides.js
const { override, fixBabelImports } = require("customize-cra");

module.exports = override(
  fixBabelImports("import", {
    libraryName: "antd",
    libraryDirectory: "es",
    style: "css"
  })
);

//修改package.json
  "scripts": {
    "start": "react-app-rewired start",
    "build": "react-app-rewired build",
    "test": "react-app-rewired test",
    "eject": "react-app-rewired eject"
  },
```

- 使用组件

```
import {Button} from 'antd'
```

容器组件 VS 展示组件

基本原则：容器组件负责数据获取，展示组件负责根据props显示信息

优势：更小、更专注、重用性高、高可用、易于测试、性能更好

```
import React, { Component } from "react";

// 容器组件
export default class CommentList extends Component {
  constructor(props) {
    super(props);
    this.state = {
      comments: []
    };
  }
  componentDidMount() {
    setTimeout(() => {
      this.setState({
        comments: [
          { body: "react is very good", author: "facebook" },
          { body: "vue is very good", author: "youyuxi" }
        ]
      });
    }, 1000);
  }
  render() {
    return (
      <div>
        {this.state.comments.map((c, i) => (
          <Comment key={i} data={c} />
        ))}
      </div>
    );
  }
}
// 展示组件
function Comment({ data }) {
  return (
    <div>
      <p>{data.body}</p>
      <p> --- {data.author}</p>
    </div>
  );
}
```

PureComponent

定制了shouldComponentUpdate后的Component

```
class Comp extends React.PureComponent {}
```

> 由于比较方式是浅比较，注意传值方式，值类型或者地址不变的且仅根属性变化的引用类型才能享受该特性

React.memo

React 16.6.0 使用 `React.memo` 让函数式的组件也有PureComponent的功能

```
const Joke = React.memo(() => (
    <div>
        {this.props.value || 'loading...' }
    </div>
));
```

高阶组件

高阶组件HOC（Higher-Order Components）是React中重用组件逻辑的高级技术，它不是react的api，而是一种组件增强模式。高阶组件是一个函数，它返回另外一个组件，产生新的组件可以对被包装组件属性进行包装，也可以重写部分生命周期

```
const withKaikeba = (Component) => {
  const NewComponent = (props) => {
    return <Component {...props} name="开课吧高阶组件" />;
  };
  return NewComponent;
};
```

上面withKaikeba组件，其实就是代理了Component，只是多传递了一个name参数

链式调用

```
import React, { Component } from 'react'
import {Button} from 'antd'

const withKaikeba = (Component) => {

  const NewComponent = (props) => {
    return <Component {...props} name="开课吧高阶组件" />;
  };
```

```
  return NewComponent;
};

const withLog = Component=>{
  class NewComponent extends React.Component{
    render(){
      return <Component {...this.props} />;
    }
    componentDidMount(){
      console.log('didMount',this.props)
    }
  }
  return NewComponent
}


class App extends Component {
  render() {
    return (
      <div className="App">
      <h2>hi,{this.props.name}</h2>
        <Button type="primary">Button</Button>
      </div>
    )
  }
}

export default withKaikeba(withLog(App))
```

装饰器写法

ES7中有一个优秀的语法—装饰器，可使代码更简洁。

安装:

```
npm install -D @babel/plugin-proposal-decorators
```

配置:

```
const { addDecoratorsLegacy } = require("customize-cra");

module.exports = override(
  ...,
  addDecoratorsLegacy()
);
```

应用:

```
import React, { Component } from 'react'
import {Button} from 'antd'
```

```
const withLog = Component=>{...}
const withKaikeba = Component=>{...}

@withKaikeba
@withLog
class App extends Component {
  render() {
    return (
      <div className="App">
      <h2>hi,{this.props.name}</h2>
        <Button type="primary">Button</Button>
      </div>
    )
  }
}

export default App
```

组件复合 - Composition

复合组件使我们以更敏捷的方式定义组件的外观和行为，比起继承的方式它更明确和安全。

```
// Dialog作为容器不关心内容和逻辑
function Dialog(props) {
  return <div style={{ border: "4px solid blue" }}>{props.children}</div>;
}
// WelcomeDialog通过复合提供内容
function WelcomeDialog() {
  return (
    <Dialog>
      <h1>欢迎光临</h1>
      <p>感谢使用react</p>
    </Dialog>
  );
}
```

通过属性通信

```
<Dialog color="red">

<div style={{ border: `4px solid ${props.color || 'blue'}` }}>
```

传递任意表达式均可

```
<Dialog footer={<button>确定</button>}>
```

children也是如此

```
<Dialog>
    {(value) => <p>{value}</p>}
</Dialog>
```

编辑children

```
React.Children.map(props.children, child=>child.type==='p')
```

> 类似的还有React.Children.forEach，React.Children.toArray等

传递进来的child如果是vdom，注意不能修改

```
React.Children.map(props.children, child=>child.props.name==='mvvm') // 错误
React.Children.map(props.children, child=>React.cloneElement(child, {name: 'mvvm'})) //
正确
```

组件跨层级通信 - Context

vuejs的privide&inject模式的来源---context

这种模式下有两个角色：

- Provider：外层提供数据的组件
- Consumer ： 内层获取数据的组件

使用：

```
const FormContext = React.createContext()
const FormProvider = FormContext.Provider
const FormConsumer = FormContext.Consumer

const store ={
  name:'开课吧',
}

class ContextTest extends Component {
  render() {
    return <FormProvider value={store}>
      <FormConsumer>
        {store=><p>{store.name}</p>}
      </FormConsumer>
    </FormProvider>
  }
}
```