day05-高阶组件

1. HOC - 高阶组件

### 高阶函数

high-order-function(高阶函数)接受函数作为参数的函数,返回值是另一个函数.

高阶函数的应用有很多，函数防抖，函数节流，bind函数，函数柯里化，map，Promise的then函数等。

#### 防抖-应用:

短时间内多次触发同一事件，只执行最后一次，或者只执行最开始的一次，中间的不执行.

const debounce = (cb, delay = 1000) => {

let timerId;

return (...args) => {

clearTimeout(timerId);

timerId = setTimeout(() => {

cb(...args)

}, delay)

}

}

在组件中使用:

<input type="text" onChange={debounce((e) => {

console.log("搜索" + e.target.value)

})} />

#### 节流-应用:

连续触发事件但是在 n 秒中只执行一次函数。即 2n 秒内执行 2 次... 。节流如字面意思，会稀释函数的执行频率。

const throttle =(cb, delay = 1000)=>{

let timerId;

return (...args)=>{

if(!timerId){

timerId=setTimeout(()=>{

cb(...args);

timerId=undefined;

},delay)

}

}

}

在组件中使用:

<input type="text" onChange={throttle((e) => {

console.log("搜索" + e.target.value)

})} />

### 高阶组件

高阶组件就是一个函数，它接受一个组件作为输入，然后会返回一个新的组件作为结果，且所返回的新组件会进行相对增强.

高阶组件是react应用中很重要的一部分，最大的特点就是重用组件逻辑。它并不是由React API定义出来的功能，而是由React的组合特性衍生出来的一种设计模式.

import React,{Component} from 'react';

const MyContainer = (WraooedComponent) =>

return class extends Component {

render(){

return <WrappedComponent {...this.props} />

}

}

或者

function HOCFactoryFactory(...params){

return function HOCFactory(WrappedComponent){

return class HOC extends Component{

render(){

return <WrappedComponent {...this.props} />

}

}

}

}

使用高阶组件处理原始组件:

import Hoc from '../utils/Hoc'

class MyTitle extends Component {

render() {

return (

<div className="mytitle">

MyTitle组件

</div>

);

}

}

export default Hoc( MyTitle );

案例1: 封装一个高阶组件，让你的 UI 组件在鼠标滑入时显示为一只手

export default Component => class extends React.Component {

render() {

return <div style={{cursor: 'pointer', display: 'inline-block'}}>

<Component/>

</div>

}

}

案例2: 封装一个高阶组件，让你的 UI 组件具有某种特定功能(比如左滑删除或者拖拽)

**使用场景介绍**

代码重用

属性代理

1. 装饰器

使用装饰器可以让高阶组件代码更优雅, 同时也解决了多层嵌套的问题.

使用高阶组件处理原始组件( 使用装饰器之前 ):

import Hoc from '../utils/Hoc'

class MyTitle extends Component {

render() {

return (

<div className="mytitle">

MyTitle组件

</div>

);

}

}

export default Hoc( MyTitle );

使用高阶组件处理原始组件( 使用装饰器之后 ):

import Hoc from '../utils/Hoc'

@Hoc

class MyTitle extends Component {

render() {

return (

<div className="mytitle">

MyTitle组件

</div>

);

}

}

export default MyTitle;

使用了装饰器语法后, Hoc高阶组件在编译阶段就会执行, 并且可以自动将@下的组件作为参数传入高阶组件, 这使得代码书写更加优雅.

**注意事项**

1. 装饰器对类的行为的改变是在代码编译时发生的,而不是在运行时,这意味着,装饰器能在编译阶段运行代码,它本身就是编译时执行的函数

⒉ 装饰器只能用于类和类的方法,不能用于函数,因为它存在函数提升

### 介绍

使用装饰器可以大大简化代码, 往往配合高阶组件使用.

在react项目中引入装饰器语法需要添加一些webpack配置, 比较极端的方法是可以使用 npm run eject 命令将 webpack.config.js 暴露出来。然后在该配置文件中进行修改。但其一，该命令是不可逆的。也就是一旦执行了此命令。webpack.config.js 文件就永久的暴露出来。其二，如果只是修改一个很小的配置项。是否可以不执行 npm run eject 也能够配置 webpack 昵。这时我们可以用 customize-cra .

### 安装插件工具

npm install react-app-rewired customize-cra

customize-cra依赖于react-app-rewired. 两者需要一起安装.

### 配置装饰器

在当前项目根目录下面创建一个名称为`config-overrides.js`文件，对webpack进行配置

配置信息可以参考：https://www.npmjs.com/package/customize-cra

const path = require('path')

const { override, addDecoratorsLegacy } = require('customize-cra')

function resolve(dir) {

return path.join(\_\_dirname, dir)

}

const customize = () => (config, env) => {

config.resolve.alias['@'] = resolve('src')

if (env === 'production') {

config.externals = {

'react': 'React',

'react-dom': 'ReactDOM'

}

}

return config

};

module.exports = override(addDecoratorsLegacy(), customize())

### 修改`package.json`中的脚本命令为：

"scripts": {

"start": "react-app-rewired start",

"build": "react-app-rewired build",

"test": "react-app-rewired test",

"eject": "react-app-rewired eject"

}

### 在项目根目录下创建 .babelrc 并写入以下内容：

{

"presets": [

"@babel/preset-env"

],

"plugins": [

[

"@babel/plugin-proposal-decorators",

{

"legacy": true

}

]

]

}

参考来源：<https://www.npmjs.com/package/react-app-rewired>

1. 内置组件

### 1.Fragment

一种常见的需求是组件需要返回多个子元素，但是组件内只能有一个根元素，这时我们一般会在多个子元素外，额外包裹一个元素返回，这样平白无故多加了一个无用的元素，并不是最好的选择。

我们可以借助react的内置组件Fragment，用它包裹多个子元素并返回，同样可以实现相同的效果，而且<Fragment></Fragment>可以简写为<></>

### 2.PureComponent

React15.3中新加了一个PureComponent类，取代之前的PureRenderMixin，PureComponent可以进行react性能优化，减少不必要的render渲染次数，使用时只要把继承类从Component换成PureComponent。

PureComponent的原理是继承了Component类，自动加载shouldComponentUpdate函数，当组件更新时，shouldComponentUpdate对props和state进行了一层浅比较，如果组件的props和state都没有发生改变，render方法就不会触发，省去Virtual DOM的生成和对比过程，达到提升性能的目的.

综合案例：

用装饰器+高阶组件封装弹出层.