React高阶组件

Why

What

How

Why

在使用React构建项目的过程中，经常会碰到在不同的组件中需要用到相同功能的情况。会导致重复工作。

What

在使用React构建项目的过程中，经常会碰到在不同的组件中需要用到相同功能的情况。不过我们知道，去这些组件中到处编写同样的代码并不优雅。

How

1. 属性代理
2. 反向继承

属性代理

属性代理是最常见的实现方式，将被处理组件的props和新的props一起传递给新组件。

|  |
| --- |
| import React from 'react';  export default function HOC(WrappedComponent){  return class HOC extends React.Component {  render(){  const newProps = {type:'HOC'};  return <div>  <WrappedComponent {...this.props} {...newProps}/>  </div>  }  }  }  import React from 'react';  import HOC from './BaseHOCComponent'  // @HOC  class OriginComponent extends React.Component {  render(){  return <div>  <h2>这是原始组件</h2>  <p>{this.props.type}</p>  <p>{this.props.say}</p>  </div>  }  }  const newComponent = HOC(OriginComponent)  export default newComponent; |

使用属性代理的好处就是，可以把常用的方法独立出来并多次复用。比如我们实现了一个加法函数，那么我们把加法函数改造成形如上述HOC函数的形式，之后对其他组件进行包裹，就可以在组件里使用这个方法了。

反向继承

|  |
| --- |
| export default function HOC(WrapperComponent){  return class HOC extends WrapperComponent{  test1(){  return this.test2()+5;  }  componentDidMount(){  console.log('1');  this.setState({number:2});  }  render(){  //使用super调用传入组件的render方法  return super.render();  }  }  } |

|  |
| --- |
| import React from 'react';  import HOC from './ReverseInheritanceComponent'  class OriginComponent extends React.Component {  constructor(props){  super(props);  this.state = {number:1}  }    test2(){  return 4;  }  componentDidMount(){  console.log('2');  }    render(){  return (  <div>  {this.state.number}{'and'}  {this.test1()}  这是原始组件  </div>  )  }  }    const newComponent = HOC(OriginComponent);  export default newComponent; |

<https://segmentfault.com/a/1190000010307650>

<https://segmentfault.com/a/1190000010307650#articleHeader2>

高阶组件最常见的函数签名形式是这样的：

HOC([param])([WrappedComponent])

function withPersistentData = (key) => (**WrappedComponent**) => {

**return** **class** **extends** **Component** {

传值的时候

const **MyComponent2WithPersistentData** = withPersistentData('data')(**MyComponent2**);

connect定义

connect([mapStateToProps], [mapDispatchToProps], [mergeProps], [options])

这个函数会将一个React组件连接到Redux 的 store。在连接的过程中，connect通过函数参数mapStateToProps，从全局store中取出当前组件需要的state，并把state转化成当前组件的props；同时通过函数参数mapDispatchToProps，把当前组件用到的Redux的action creator，以props的方式传递给当前组件。connect并不会修改传递进去的组件的定义，而是它会返回一个新的组件。

const ConnectedComponentA = connect(componentASelector, componentAActions)(ComponentA);

// connect 是一个函数，返回值enhance也是一个函数

const enhance = connect(componentASelector, componentAActions);

// enhance是一个高阶组件

const ConnectedComponentA = enhance(ComponentA);

与父组件区别

有些同学可能会觉得高阶组件有些类似父组件的使用。例如，我们完全可以把高阶组件中的逻辑放到一个父组件中去执行，执行完成的结果再传递给子组件。从逻辑的执行流程上来看，高阶组件确实和父组件比较相像，但是高阶组件强调的是逻辑的抽象。高阶组件是一个函数，函数关注的是逻辑；父组件是一个组件，组件主要关注的是UI/DOM。如果逻辑是与DOM直接相关的，那么这部分逻辑适合放到父组件中实现；如果逻辑是与DOM不直接相关的，那么这部分逻辑适合使用高阶组件抽象，如数据校验、请求发送等。