

## 课堂目标

1. 学习react组件化
2. 掌握容器组件 VS 展示组件
3. 掌握高阶组件
4. PureComponent
5. 掌握render props
6. 了解异步渲染组件
7. 了解函数化组件Hooks

## 知识要点

1. 组件化
2. antd组件库使用
3. 组件通信
4. 组件的多个模式

## 资源

1. [antd](#)

## 起步

### 试用 ant-design组件库

安装: `npm install antd --save`

试用button

```
import React, { Component } from 'react'
import Button from 'antd/lib/button'
import "antd/dist/antd.css"
class App extends Component {
  render() {
    return (
```

```

    <div className="App">
      <Button type="primary">Button</Button>
    </div>
  )
}
}

export default App

```

## 配置按需加载

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

```
npm install react-app-rewired@2.0.2-next.0 babel-plugin-import --save
```

## 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

```

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 (浅比较)

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

实现原理:

```
import shallowEqual from './shallowEqual'
import Component from './Component'

export default function PureComponent(props, context) {
  Component.call(this, props, context)
}

PureComponent.prototype = Object.create(Component.prototype)
PureComponent.prototype.constructor = PureComponent
PureComponent.prototype.isPureReactComponent = true
PureComponent.prototype.shouldComponentUpdate = shallowCompare

function shallowCompare(nextProps, nextState) {
  return !shallowEqual(this.props, nextProps) ||
    !shallowEqual(this.state, nextState)
}
```

```

export default function shallowEqual(objA, objB) {
  if (objA === objB) {
    return true
  }

  if (typeof objA !== 'object' || objA === null || typeof objB !== 'object' || objB === null) {
    return false
  }

  var keysA = Object.keys(objA)
  var keysB = Object.keys(objB)

  if (keysA.length !== keysB.length) {
    return false
  }

  // Test for A's keys different from B.
  for (var i = 0; i < keysA.length; i++) {
    if (!objB.hasOwnProperty(keysA[i]) || objA[keysA[i]] !== objB[keysA[i]]) {
      return false
    }
  }

  return true
}

```

## React.memo

React v16.6.0 之后的版本，可以使用 `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

在React里就有了HOC (Higher-Order Components) 的概念

高阶组件也是一个组件，但是他返回另外一个组件，产生新的组件可以对属性进行包装，甚至重写部分生命周期

```

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

## 高阶链式调用

高阶组件最巧妙的一点，是可以链式调用。

```
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))
```

## 高阶组件装饰器写法

ES7装饰器可用于简化高阶组件写法

```
npm install --save-dev babel-plugin-transform-decorators-legacy
```

```
const { injectBabelPlugin } = require('react-app-rewired')
```

```

module.exports = function override(config) {
  config = injectBabelPlugin(
    ['import', { libraryName: 'antd', libraryDirectory: 'es', style: 'css' }],
    config,
  )

  config = injectBabelPlugin(
    ['@babel/plugin-proposal-decorators', { "legacy": true }],
    config,
  )

  return config
}

```

## 使用装饰器

```

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(Component.name , 'didMount', this.props)
    }
  }
  return NewComponent
}

@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
```

## 组件跨层级通信 - 上下文

组件跨层级通信可使用Context

这种模式下有两个角色，Provider和Consumer

Provider为外层组件，用来提供 数据；内部需要数据时用Consumer来读取

使用上下文

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

let store = {
  name: '开课吧',
  sayHi() {
    console.log(this.name)
  }
}

let withForm = Component=>{
  const NewComponent = (props) => {
    return <FormProvider value={store}>
      <Component {...props} />
    </FormProvider>
  };
  return NewComponent;
}

@withForm
class App extends Component {
  render() {
    return <FormConsumer>
      {
        store=>{
          return <Button onClick={()=>store.sayHi()}>
            {store.name}
          </Button>
        }
      }
    </FormConsumer>
  }
}
```