React组件化

资源

ant design

customize-cra

知识点

使用第三方组件

安装：`npm install antd --save`

范例：试用 ant-design组件库

```
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

简化引入方式：安装react-app-rewired取代react-scripts，可以扩展webpack的配置

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

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

module.exports = override(
  fixBabelImports("import", {
    libraryName: "antd",libraryDirectory: "es",//意思是从antd里的
    es文件里找相关的ui组件
    style: "css"//样式从antd组件里的css文件里加载index.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"
  },
```

支持装饰器配置（加了另一个插件）：

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

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

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

表单组件设计与实现

antd表单试用

```
import React from 'react';
import { Form, Icon, Input, Button } from "antd";
class NormalLoginForm extends React.Component {
   handleSubmit = e => {
      e.preventDefault();//validateFields是全局校验，如何
      实现的？
      this.props.form.validateFields((err, values) =>
      { if (!err) {
          console.log("Received values of form: ", values);
        }
      });
   };

   render() {getFieldDecorator又是怎么来的？
     const { getFieldDecorator } = this.props.form;
     return (
       <Form onSubmit={this.handleSubmit} className="login-form">
         <Form.Item>
           {getFieldDecorator("userName", {
             rules: [{ required: true, message: "Please input your username!" }]
           })(
             <Input
               prefix={<Icon type="user" style={{ color: "rgba(0,0,0,.25)" }} />}
               placeholder="Username"
```

```
            />
          )}
        </Form.Item>
        <Form.Item>
          {getFieldDecorator("password", {
            rules: [{ required: true, message: "Please input your Password!" }]
          })(
            <Input
              prefix={<Icon type="lock" style={{ color: "rgba(0,0,0,.25)" }} />}
              type="password"
              placeholder="Password"
            />
          )}
        </Form.Item>
        <Form.Item>
          <Button
            type="primary"
            htmlType="submit"
            className="login-form-button"
          >
            Log in
          </Button>
        </Form.Item>
      </Form>
    );
  }
}

const WrappedNormalLoginForm = Form.create({ name: "normal_login" })(
  NormalLoginForm
);

export default WrappedNormalLoginForm;
```

表单组件实现思路

表单组件实现

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

// 2.扩展表单的高阶组件，提供输入控件包装、事件处理、表单校验等
function kFormCreate(Comp) {
  return class extends React.Component {
    constructor(props) {
      super(props);
      this.options = {};
      this.state = {};
    }
    handleChange = e => {
      let { name, value } = e.target;
      this.setState({ [name]: value }, () => {
```

```
      // 校验:注意回调中调用
      this.validateField(name);
    });
  };
  // 校验指定字段
  validateField = field => {
    const rules = this.options[field].rules; // 获取校验规则
    // 只要有任何一项校验失败就返回true跳出，对返回值取反表示校验失败
    const ret = !rules.some(rule => {
      if (rule.required) {
        // 仅验证必填项
        if (!this.state[field]) {
          // 校验失败
          this.setState({
            // 错误信息设置
            [field + "Message"]: rule.message
          });
          return true; // 若有校验失败，返回true
        }
      }
    });
    // 若校验成功，清除错误信息
    if (ret) this.setState({ [field + "Message"]: "" });
    return ret;
  };

  // 校验所有字段
  validateFields = cb => {
    // 将选项中所有field组成的数组转换为它们校验结果数组
    const ret = Object.keys(this.options).every(field =>
      this.validateField(field)
    );
    cb(ret, this.state);
  };

  getFieldDec = (field, option) => {
    this.options[field] = option;
    return InputComp => (
      <div>
        {React.cloneElement(InputComp, {
          name: field,
          value: this.state[field] || "",
          onChange: this.handleChange
        })}
        {/* 添加一个校验提示信息 */}
        {this.state[field + "Message"] && (
          <p style={{ color: "red" }}>{this.state[field + "Message"]}</p>
        )}
      </div>
    );
  };

  render() {
```

```
      return (
        <div>
          <Comp
            {...this.props}
            getFieldDec={this.getFieldDec}
            validate={this.validate} // 添加校验属性
          />
        </div>
      );
    }
  };
}

@kFormCreate
class KFormTest extends Component {
  onSubmit = () => {
    // 校验、提交
    this.props.validateFields((isValid, data) => {
      if (isValid) {
        console.log("提交登录", data);
      } else {
        alert("校验失败");
      }
    });
  };
  render() {
    // 结构出扩展的方法
    const { getFieldDec } = this.props;
    return (
      <div>
        {getFieldDec("uname", {
          rules: [{ required: true, message: "请输入用户名" }]
        })(<input type="text" />)}
        {getFieldDec("pwd", {
          rules: [{ required: true, message: "请输入密码" }]
        })(<input type="password" />)}
        <button onClick={this.onSubmit}>登录</button>
      </div>
    );
  }
}

export default KFormTest
```

弹窗类组件设计与实现

设计思路

弹窗类组件的要求弹窗内容在A处声明，却在B处展示。

```
// 常见用法如下：Dialog在当前组件声明，但是却在body中另一个div中显示
<div class="foo">
  <div> ... </div>
  { needDialog ?
    <Dialog>
       <header>Any Header</header>
       <section>Any content</section>
    </Dialog>
    : null }
</div>
```

具体实现

方案1：Portal
传送门
传送门，react v16之后出现的portal可以实现内容传送功能。

范例：Dialog组件

```
import React from 'react';
import {createPortal} from 'react-dom';

export default class Dialog extends React.Component {
  constructor(props) {
    super(props);

    const doc = window.document;
    this.node = doc.createElement('div');
    doc.body.appendChild(this.node);
  }

  render() {
    return createPortal(
      <div className="dialog">
        {this.props.children}
      </div>, //塞进传送门的JSX
      this.node //传送门的另一端DOM node
    );
  }

  componentWillUnmount() {
    window.document.body.removeChild(this.node);
  }
}
```

方案2：unstable_renderSubtreeIntoContainer

在v16之前，要用到react中两个秘而不宣的React API: unstable_renderSubtreeIntoContainer, unmountComponentAtNode

```
export class Dialog2 extends React.Component {
  render() {
  // render一个null，目的什么内容都不渲染
    return null;
  }

  componentDidMount() {
    const doc = window.document;
    this.node = doc.createElement("div");
    doc.body.appendChild(this.node);

    this.createPortal(this.props);
  }

  componentDidUpdate() {
  } this.createPortal(this.props);

  componentWillUnmount() {//清理节点
    unmountComponentAtNode(this.node);
  // 清理宿主window.document.body.removeChild
  }this.node);

  createPortal(props) {
    unstable_renderSubtreeIntoContainer(
      this, //当前组件
      <div className="dialog">{props.children}</div>, //要传送的内容 塞进传送门的JSX
      this.node // 传送门另一端的DOM node
    );
  }
}
```

树形组件设计与实现

设计思路

react中实现递归组件更加纯粹，就是组件递归渲染即可。

实现

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

class TreeNode extends Component {
  constructor(props) {
    super(props);

    this.state = {
      open: false
```

```
    };
  }

  toggle = () => {
    if (this.isFolder) {
      this.setState(nextState => ({ open: !nextState.open }));
    }
  };

  get isFolder() {
    return this.props.model.children && this.props.model.children.length;
  }

  render() {
    return (
      <ul>
        <li>
          <div onClick={this.toggle}>
            {this.props.model.title}
            {this.isFolder ? (
              <span>[{this.state.open ? "-" : "+"}]</span>
            ) : null}
          </div>
          {this.isFolder ? (
            <ul style={{ display: this.state.open ? "block" : "none" }}>
              {this.props.model.children.map(model => (
                <TreeNode model={model} key={model.title} />
              ))}
            </ul>
          ) : null}
        </li>
      </ul>
    );
  }
}

export default class Tree extends Component {
  treeData = {
    title: "Web全栈架构师",
    children: [
      {
        title: "Java架构师"
      },
      {
        title: "JS高级",
        children: [
          {
            title: "ES6"
          },
          {
            title: "动效"
          }
        ]
```

```
      },
      {
        title: "Web全栈",
        children: [
          {
            title: "Vue训练营",
            expand: true,
            children: [
              {
                title: "组件化"
              },
              {
                title: "源码"
              },
              {
                title: "docker部署"
              }
            ]
          },
          {
            title: "React",
            children: [
              {
                title: "JSX"
              },
              {
                title: "虚拟DOM"
              }
            ]
          },
          {
            title: "Node"
          }
        ]
      }
    ]
  };
  render() {
    return <TreeNode model={this.treeData} />;
  }
}
```

常见组件优化技术

定制组件的shouldComponentUpdate钩子

范例：通过shouldComponentUpdate优化组件

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

```
// 容器组件
export default class CommentList extends Component {
  constructor(props) {
    super(props);
    this.state = {
      comments: []
    };
  }
  componentDidMount() {
    setInterval(() => {
      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>
    );
  }
}
class Comment extends Component {
  render() {
    return (
      <div>
        <p>{this.props.data.body}</p>
        <p> --- {this.props.data.author}</p>
      </div>
    );
  }
}
```

添加shouldComponentUpdate判断

```
shouldComponentUpdate({ data: { body, author } (这里的参数是下一次要变更的属性)
}) {
    if (body === this.props.data.body && author === this.props.data.author) {
        return false;
    }

    return true;
}
```

PureComponent

定制了shouldComponentUpdate后的Component

```
class Comp extends React.PureComponent {
  render() {
    // 注意这里直接获取body和author原始值
    （字符串），不要用对象
    return (
      <div>
        <p>{this.props.body}</p>
        <p> --- {this.props.author}</p>
      </div>
    );
  }
}
```

> 缺点是必须要用class形式，而且要注意是**浅比较**

React.memo

React v16.6.0 之后的版本，可以使用一个新功能 `React.memo` 来完美实现让函数式的组件也有了PureComponent的功能

```
const Comment = React.memo(function({ body, author }) {
  console.log("render");

  return (
    <div>
      <p>{body}</p>
      <p> --- {author}</p>
    </div>
  );
});
```

作业：

尝试实现Form（布局、提交）、FormItem（错误信息）、Input（前缀图标）