React组件化

课堂目标

1. 掌握函数化组件Hooks
2. 掌握上下文的使用
3. 设计并实现一个表单组件

知识要点

1. Hook API
2. Context API
3. 组件设计思路
4. 组件实现实践

资源

1. Context
2. Hook

知识点

Hook

Hook是React16.8一个新增项，它可以让你在不编写 class 的情况下使用 state 以及其他的 React 特性。

状态钩子 - State Hook

- 创建HooksTest.js

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

export default function HooksTest() {
  // useState(initialState)，接收初始状态，返回一个状态变量和其更新函数
  const [count, setCount] = useState(0);
  return (
    <div>
      <p>You clicked {count} times</p>
      <button onClick={() => setCount(count + 1)}>Click me</button>
    </div>
  );
}
```

副作用钩子 - Effect Hook

- 更新HooksTest.js

```
import React, { useState, useEffect } from "react";
useEffect(() => {
    // Update the document title using the browser API
    document.title = `您点击了 ${count} 次`;
});
```

自定义钩子 - Custom Hook

其他Hook

组件跨层级通信 - Context

上下文提供一种不需要每层设置props就能跨多级组件传递数据的方式

Context相关API

- React.createContext
- Context.Provider
- Class.contextType
- Context.Consumer

基本用法

- 创建上下文

```
const MyContext = React.createContext();
```

- 提供上下文

```
const { Provider } = MyContext;
export default function App() {
  return (
    <div>
      <Provider value={{foo: "bar"}}>
        <Child />
      </Provider>
    </div>
  );
}
```

- 消费上下文

```
function Child3(props){
    return <div>{props.foo}</div>
}

export default function App() {
  return (
    <div>
      <Provider value={{ foo: "bar" }}>
        <Consumer>
            {value => <Child2 {...value}/>}
        </Consumer>
      </Provider>
    </div>
  );
}
```

组件设计与实现

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 = {}; // 各字段选项
      this.state = {}; // 各字段值
    }

    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
  validate = 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 rets = Object.keys(this.options).map(field => {
      return this.validateField(field);
    });
    // 校验结果中每一项都要求true
    const ret = rets.every(v => v == true);
    cb(ret, this.state);
  };

  // 返回包装输入控件的高阶组件，代理其事件处理、赋值等操作
  // field字段名，options选项，InputComp输入控件
  getFieldDec = (field, option) => {
    this.options[field] = option;
    return InputComp => (
      <div>
        {/* 由React.createElement生成的元素不能修改，需要克隆一份再扩展 */}
        {React.cloneElement(InputComp, {
          name: field, // 控件name
          value: this.state[field] || "", // 控件值
          onChange: this.handleChange // 控件change事件处理
        })}
        {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}
          />
        </div>
      );
    }
  };
}

@kFormCreate
class KFormTest extends Component {
  onSubmit = () => {
    this.props.validate((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;
```

作业：

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