# React组件化

## 资源

Hook

## 知识点

### Hook

#### 升级react、react-dom至16.8以上

```
npm i react react-dom -S
```

#### 状态钩子 State Hook

函数组件可以使用状态

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

export default function HooksTest() {
  const [fruit, setFruit] = useState("");
  return (
    <div>
      <p>{fruit === "" ? "请选择喜爱的水果: " : `您的选择是: ${fruit}`}</p>
    </div>
  );
}
```

### 副作用钩子 Effect Hook

函数组件执行副作用操作。

- 基本使用

```
import { useEffect } from "react";
useEffect(()=>{
    setTimeout(() => {
        setFruits(['香蕉','西瓜'])
    }, 1000);
})
```

- 设置依赖

```
useEffect(()=>{...}, [])
```

- 清除工作：有一些副作用是需要清除的，防止内存泄露

```
useEffect(() => {
    const timer = setInterval(() => {
        console.log('msg');
    }, 1000);

    return function(){
        clearInterval(timer);
    }
}, []);
```

## useReducer

useState的可选项，常用于组件有复杂状态逻辑时，类似于redux中reducer概念。

```
import { useReducer } from "react";

// 状态维护reducer
function fruitReducer(state, action) {
  switch (action.type) {
    case "init":
      return action.payload;
    case "add":
      return [...state, action.payload];
    default:
      return state;
  }
}

export default function HooksTest() {
  const [fruits, dispatch] = useReducer(fruitReducer, []);

  useEffect(() => {
    setTimeout(() => {
```

```
      // 变更状态
      dispatch({ type: "init", payload: ["香蕉", "西瓜"] });
    }, 1000);
  }, []);

  return (
    <div>
      {/*变更状态*/}
      <FruitAdd onAddFruit={pname => dispatch({type: 'add', payload: pname})} />
    </div>
  );
}
```

## useContext

useContext用于在快速在函数组件中导入上下文。

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

// 创建上下文
const Context = React.createContext();

export default function HooksTest() {
  // ...
  return (
    {/* 提供上下文的值 */}
    <Context.Provider value={{fruits,dispatch}}>
      <div>
        {/* 这里不再需要给FruitAdd传递变更函数，实现了解耦 */}
        <FruitAdd />
      </div>
    </Context.Provider>
  );
}

function FruitAdd(props) {
  // 获取上下文
  const {dispatch} = useContext(Context)
  const onAddFruit = e => {
    if (e.key === "Enter") {
      // 直接派发动作修改状态
      dispatch({ type: "add", payload: pname })
      setPname("");
    }
  };
  // ...
}
```

## Hook相关拓展

1. 基于useReducer的方式实现异步action
2. Hook规则、自定义Hook
3. 一堆第三方hook实现

# 组件设计与实现

## 表单组件实现

- KFormTest.js

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

export default class KFormTest extends Component {
  render() {
    return (
      <div>
        <Input type="text" />
        <Input type="password" />
        <Button>登录</Button>
      </div>
    );
  }
}
```

- 扩展现有表单KFormTest.js

```
function kFormCreate(Comp) {
  return class extends React.Component {
    constructor(props) {
      super(props);
      this.options = {}; // 选项
      this.state = {}; // 字段值
    }

    handleChange = e => {
      let { name, value } = e.target;
      this.setState({ [name]: value }, () => {
        // 校验
      });
    };

    // 返回包装输入控件的高阶组件
    getFieldDec = (field, option) => {
      this.options[field] = option;
      return InputComp => (
        <div>
          {/* vdom不能修改，克隆一份再扩展 */}
          {React.cloneElement(InputComp, {
            name: field, // 控件name
```

```
            value: this.state[field] || "",
            onChange: this.handleChange
          })}
        </div>
      );
    };
    render() {
      return (
        <div>
          <Comp
            {...this.props}
            getFieldDec={this.getFieldDec}
          />
        </div>
      );
    }
  };
}
```

- 应用高阶组件

```
@kFormCreate
class KFormTest extends Component {
  onSubmit = () => {
    console.log('submit')
  };
  render() {
    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;
```

- 添加校验

```
function kFormCreate(Comp) {
  return class extends React.Component {
```

```
    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);
    };

    render() {
      return (
        <div>
          <Comp validate={this.validate} />
        </div>
      );
    }
  };
}
```

- 调用校验函数

```
@kFormCreate
class KFormTest extends Component {
  onSubmit = () => {
    // 校验、提交
    this.props.validate((isValid, data) => {
      if (isValid) {
        console.log("提交登录", data);
      } else {
        alert("校验失败");
      }
    });
  };
  render() { ... }
}

export default KFormTest;
```