React组件化2

课堂目标

1. 掌握第三方组件正确使用方式
2. 能设计并实现自己的组件
3. 了解常见组件优化技术

知识要点

1. 使用antd
2. 设计并实现表单控件
3. 实现弹窗类组件
4. 实现树组件
5. 使用PureComponent、memo

资源

umi

ant design

知识点

快速开始

(https://www.html.cn/create-react-app/docs/getting-started/)

```
npx create-react-app my-app

cd my-app

npm start
```

使用第三方组件

不必eject，直接安装：`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

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 customize-cra babel-plugin-import -D`

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

module.exports = override(
  fixBabelImports("import", {//antd按需加载
```

```
    libraryName: "antd",
    libraryDirectory: "es",
    style: "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

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

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

module.exports = override(
  ...,
  addDecoratorsLegacy()//配置装饰器
);
```

```
//按需加载和实现装饰器之后的页面如下：HocPage.js
import React, { Component } from "react";
import { Button } from "antd";

const foo = Cmp => props => {
  return (
    <div className="border">
      <Cmp {...props} />
    </div>
  );
};
const foo2 = Cmp => props => {
  return (
    <div className="border" style={{ border: "solid 1px red" }}>
      <Cmp {...props} />
    </div>
  );
};

@foo
@foo2
class Child extends Component {
  render() {
    return <div className="border">child</div>;
  }
}
/* function Child(props) {
  return <div className="border">child</div>;
} */
```

```
@foo2
class HocPage extends Component {
  render() {
    // const Foo = foo2(foo(Child));
    return (
      <div>
        <h1>HocPage</h1>
        <Child />
        <Button type="dashed">click</Button>
      </div>
    );
  }
}

export default HocPage;
```

表单组件设计与实现

antd表单试用

```
import React, { Component } from "react";
import { Form, Input, Icon, Button } from "antd";

const FormItem = Form.Item;

//校验规则
const nameRules = { required: true, message: "please input your name" };
const passwordRules = { required: true, message: "please input your password" };

@Form.create()
class FormPageDecorators extends Component {
  handleSubmit = () => {
    /*  const { getFieldsValue, getFieldValue } = this.props.form;
    console.log("submit", getFieldsValue()); */

    const { validateFields } = this.props.form;
    validateFields((err, values) => {
      if (err) {
        console.log("err", err);
      } else {
        console.log("submit", values);
      }
    });
  };
  render() {
    const { getFieldDecorator } = this.props.form;
    // console.log(this.props.form);
    return (
      <div>
        <h1>FormPageDecorators</h1>
        <Form>
```

```
          <FormItem label="姓名">
            {getFieldDecorator("name", { rules: [nameRules] })(
              <Input prefix={<Icon type="user" />} />,
            )}
          </FormItem>
          <FormItem label="密码">
            {getFieldDecorator("password", { rules: [passwordRules] })(
              <Input type="password" prefix={<Icon type="lock" />} />,
            )}
          </FormItem>
          <FormItem label="姓名">
            <Button type="primary" onClick={this.handleSubmit}>
              提交
            </Button>
          </FormItem>
        </Form>
      </div>
    );
  }
}
export default FormPageDecorators;
// export default Form.create()(FormPageDecorators);
```

表单组件设计思路

- 表单组件要求实现**数据收集**、**校验**、**提交**等特性，可通过高阶组件扩展
- 高阶组件给表单组件传递一个input组件**包装函数**接管其输入事件并统一管理表单数据
- 高阶组件给表单组件传递一个**校验函数**使其具备数据校验功能

表单组件实现

- 表单基本结构，创建MyFormPage.js

```
import React, { Component } from "react";
import kFormCreate from "../../components/kFormCreate";

const nameRules = { required: true, message: "please input your name!" };
const passwordRules = {
  required: true,
  message: "please input your password!",
};
class MyFormPage extends Component {
  handleSubmit = () => {
    const { getFieldValue } = this.props;
    const res = {
      name: getFieldValue("name"),
      password: getFieldValue("password"),
    };
    console.log("hah", res);
```

```
  };
  handleSubmit2 = () => {
    // 加入校验
    const { validateFields } = this.props;
    validateFields((err, values) => {
      if (err) {
        console.log("validateFields", err);
      } else {
        console.log("submit", values);
      }
    });
  };
  render() {
    const { getFieldDecorator } = this.props;
    return (
      <div>
        <h1>MyFormPage</h1>
        <div>
          {getFieldDecorator("name", { rules: [nameRules] })(
            <input type="text" />,
          )}
          {getFieldDecorator("password", [nameRules])(
            <input type="password" />,
          )}
        </div>
        <button onClick={this.handleSubmit2}>submit</button>
      </div>
    );
  }
}

export default kFormCreate(MyFormPage);
```

- 高阶组件kFormCreate：扩展现有表单，./components/KFormTest.js

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

export default function kFormCreate(Cmp) {
  return class extends 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 });
    };
    getFieldValue = field => {
      return this.state[field];
    };
    validateFields = callback => {
```

```
      const res = { ...this.state };
      const err = [];
      for (let i in this.options) {
        if (res[i] === undefined) {
          err.push({ [i]: "error" });
        }
      }
      if (err.length > 0) {
        callback(err, res);
      } else {
        callback(undefined, res);
      }
    };
    getFieldDecorator = (field, option) => {
      this.options[field] = option;
      return InputCmp => (
        <div>
          {// 由React.createElement生成的元素不能修改，需要克隆一份再扩展
          React.cloneElement(InputCmp, {
            name: field,
            value: this.state[field] || "", //控件值
            onChange: this.handleChange, //控件change事件处理
          })}
        </div>
      );
    };
    render() {
      return (
        <div className="border">
          <Cmp
            {...this.props}
            getFieldDecorator={this.getFieldDecorator}
            getFieldValue={this.getFieldValue}
            validateFields={this.validateFields}
          />
        </div>
      );
    }
  };
}
```

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。react中相当于弹窗内容看起来被render到一个组件里面去，实际改变的是网页上另一处的DOM结构，这个显然不符合正常逻辑。但是通过使用框架提供的特定API创建组件实例并指定挂载目标仍可完成任务。

```
// 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>
   }
</div>
```

具体实现

方案1：Portal

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

```
// Diallog.js
import React, { Component } from "react";
import { createPortal } from "react-dom";
import "./index.scss";

export default class Diallog extends Component {
  constructor(props) {
    super(props);
    const doc = window.document;
    this.node = doc.createElement("div");
    doc.body.appendChild(this.node);
  }
  componentWillUnmount() {
    window.document.body.removeChild(this.node);
  }
  render() {
    const { hideDialog } = this.props;
    return createPortal(
      <div className="dialog">
        {this.props.children}
        {typeof hideDialog === "function" && (
          <button onClick={hideDialog}>关掉弹窗</button>
        )}
      </div>,
      this.node,
    );
  }
}
```

```
// Diallog/index.scss
.dialog {
```

```
  position: absolute;
  top: 0;
  right: 0;
  bottom: 0;
  left: 0;
  line-height: 30px;
  width: 400px;
  height: 300px;
  transform: translate(50%, 50%);
  border: solid 1px gray;
  text-align: center;
}
```

方案2：unstable_renderSubtreeIntoContainer

在v16之前，实现“传送门”，要用到react中两个秘而不宣的React API

```
export class Dialog2 extends React.Component {
  render() {
    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() {
    unmountComponentAtNode(this.node);
    window.document.body.removeChild(this.node);
  }

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

> 总结一下：
>
> 1. Dialog什么都不给自己画，render返回一个null就够了；
> 2. 它做得事情是通过调用createPortal把要画的东西画在DOM树上另一个角落。

树形组件设计与实现

设计思路

递归：自己调用自己

如计算f(n)=f(n-1)*n; n>0, f(1)=1

```
function foo(n) {
  return n===1 ? 1 : n*foo(n-1)
}
```

react中实现递归组件更加纯粹，就是组件递归渲染即可。假设我们的节点组件是TreeNode，它的render中只要发现当前节点拥有子节点就要继续渲染自己。节点的打开状态可以通过给组件一个open状态来维护。

实现

//TreeNode.js

```
import React, { Component } from "react";
import TreeNode from "../../components/TreeNode";
//数据源
const treeData = {
  key: 0, //标识唯一性
  title: "全国", //节点名称显示
  children: [
    //子节点数组
    {
      key: 6,
      title: "北方区域",
      children: [
        {
          key: 1,
          title: "黑龙江省",
          children: [
            {
              key: 6,
              title: "哈尔滨",
            },
          ],
        },
        {
          key: 2,
          title: "北京",
        },
      ],
    },
    {
      key: 3,
      title: "南方区域",
      children: [
```

```
        {
          key: 4,
          title: "上海",
        },
        {
          key: 5,
          title: "深圳",
        },
      ],
    },
  ],
};
export default class TreePage extends Component {
  render() {
    return (
      <div>
        <h1>TreePage</h1>
        <TreeNode data={treeData} />
      </div>
    );
  }
}
```

TreeNode.js

```
import React, { Component } from "react";
import classnames from "classnames";//先安装下npm install classnames

export default class TreeNode extends Component {
  constructor(props) {
    super(props);
    this.state = {
      expanded: false,
    };
  }
  handleExpanded = () => {
    this.setState({
      expanded: !this.state.expanded,
    });
  };
  render() {
    const { title, children } = this.props.data;
    const { expanded } = this.state;
    const hasChildren = children && children.length > 0;
    return (
      <div>
        <div className="nodeInner" onClick={this.handleExpanded}>
          {hasChildren && (
            <i
              className={classnames("tri", expanded ? "tri-open" : "tri-close")}
            ></i>
          )}
          <span>{title}</span>
```

```
        </div>
        {expanded && hasChildren && (
          <div className="children">
            {children.map(item => {
              return <TreeNode key={item.key} data={item} />;
            })}
          </div>
        )}
      </div>
    );
  }
}
```

```
/* 树组件css */
.nodeInner {
  cursor: pointer;
}

.children {
  margin-left: 20px;
}

.tri {
  width: 20px;
  height: 20px;
  margin-right: 2px;
  padding-right: 4px;
}

.tri-close:after,
.tri-open:after {
  content: "";
  display: inline-block;
  width: 0;
  height: 0;
  border-top: 6px solid transparent;
  border-left: 8px solid black;
  border-bottom: 6px solid transparent;
}

.tri-open:after {
  transform: rotate(90deg);
}
```

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

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

export default class CommentList extends Component {
  constructor(props) {
    super(props);
    this.state = { comments: [] };
  }

  componentDidMount() {
    setInterval(() => {
      this.setState({
        comments: [
          {
            author: "小明",
            body: "这是小明写的文章",
          },
          {
            author: "小红",
            body: "这是小红写的文章",
          },
        ],
      });
    }, 1000);
  }
  render() {
    const { comments } = this.state;
    return (
      <div>
        <h1>CommentList</h1>
        {comments.map((c, i) => {
          return <Comment key={i} data={c} />;
        })}
      </div>
    );
  }
}

class Comment extends Component {
  shouldComponentUpdate(nextProps, nextState) {
    const { author, body } = nextProps.data;
    const { author: nowAuthor, body: nowBody } = this.props.data;
    if (body === nowBody && author === nowAuthor) {
      return false; //如果不执行这里，将会多次render
    }
    return true;
  }
  render() {
    console.log("hah");
    const { body, author } = this.props.data;
    return (
      <div>
        <p>作者：  {author}</p>
        <p>正文：{body}</p>
        <p>----------------------------------</p>
      </div>
```

```
    );
  }
}
```

PureComponent

定制了shouldComponentUpdate后的Component

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

export default class PuerComponentPage extends PureComponent {
  constructor(props) {
    super(props);
    this.state = {
      counter: 0,
      obj: {
        num: 100,
      },
    };
  }

  setCounter = () => {
    this.setState({
      counter: 1,
      obj: {
        num: 200,
      },
    });
    console.log("setCounter");
  };
  render() {
    console.log("render");
    const { counter, obj } = this.state;
    return (
      <div>
        <button onClick={this.setCounter}>setCounter</button>
        <div>counter: {counter}</div>
        <div>obj.num: {obj.num}</div>
      </div>
    );
  }
}
```

缺点是必须要用class形式，而且要注意是**浅比较**

```
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.memo(...)`是React v16.6引进来的新属性。它的作用和`React.PureComponent`类似，是用来控制函数组件的重新渲染的。`React.memo(...)`其实就是函数组件的`React.PureComponent`。

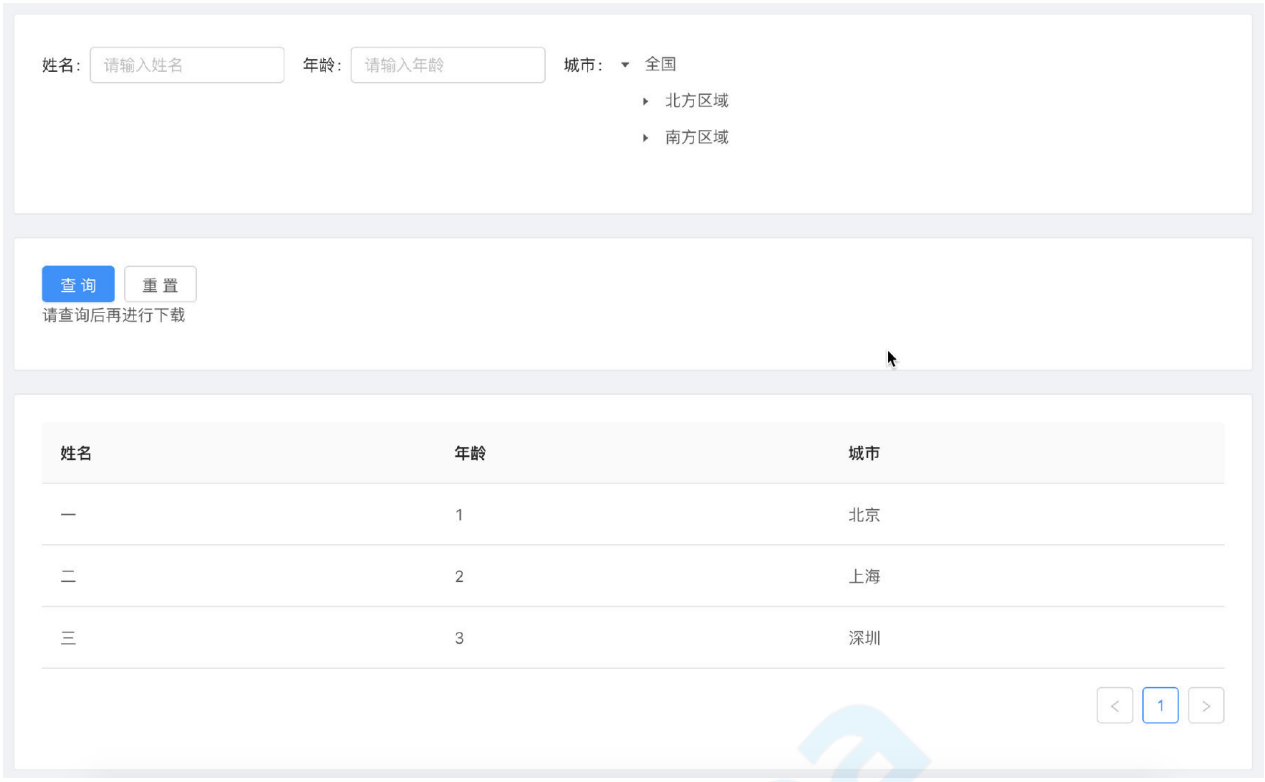
```
import React, { Component, PureComponent, memo } from "react";
```

```
export default class MemoPage extends Component {
  constructor(props) {
    super(props);
    this.state = {
      counter: 0,
      obj: { num: -1 },
    };
  }
  setCounter = () => {
    this.setState({
      counter: 1 /* ,
      obj: {
        num: 100,
      }, */,
    });
  };
  render() {
    const { counter } = this.state;
    return (
      <div>
        <h1>MemoPage</h1>
        <button onClick={this.setCounter}>按钮</button>
        {/* <PuerCounter counter={counter} obj={obj} /> */}
        <PuerCounter counter={counter} />
      </div>
    );
  }
}

const PuerCounter = memo(props => {
  console.log("render");
  return <div>{props.counter}</div>;
});
```

作业

实现下图：



提示：可以使用antd的 Card、Input、Tree、Button、Form、Table