react 知识点三

react 知识点三

1. Refs 转发

Ref 转发是一项将 ref 自动地通过组件传递到其一子组件的技巧。对于大多数应用中的组件来说，这通常不是必需的。但其对某些组件，尤其是可重用的组件库是很有用的。

javascript 复制代码

```
const FancyButton = React.forwardRef((props, ref) => (
  <button ref={ref} className="FancyButton">
    {props.children}
  </button>
));

// 你可以直接获取 DOM button 的 ref：
const ref = React.createRef();
<FancyButton ref={ref}>Click me!</FancyButton>;
```

ref 不是 prop 属性。就像 key 一样，其被 React 进行了特殊处理。如果你对 HOC 添加 ref，该 ref 将引用最外层的容器组件，而不是被包裹的组件。

我们可以使用 React.forwardRef API 明确地将 refs 转发到内部的 FancyButton 组件。React.forwardRef 接受一个渲染函数，其接收 props 和 ref 参数并返回一个 React 节点。例如：

javascript 复制代码

```
function logProps(Component) {
  class LogProps extends React.Component {
    componentDidUpdate(prevProps) {
      console.log("old props:", prevProps);
      console.log("new props:", this.props);
    }

    render() {
      const { forwardedRef, ...rest } = this.props;

      // 将自定义的 prop 属性 “forwardedRef” 定义为 ref
```

```
      return <Component ref={forwardedRef} {...rest} />;
    }
  }

  // 注意 React.forwardRef 回调的第二个参数 "ref"。
  // 我们可以将其作为常规 prop 属性传递给 LogProps，例如 "forwardedRef"
  // 然后它就可以被挂载到被 LogProps 包裹的子组件上。
  return React.forwardRef((props, ref) => {
    return <LogProps {...props} forwardedRef={ref} />;
  });
}
```

2.在 DevTools 中显示自定义名称

React.forwardRef 接受一个渲染函数。React DevTools 使用该函数来决定为 ref 转发组件显示的内容。

javascript 复制代码

```
// 例如，以下组件将在 DevTools 中显示为 "ForwardRef"：

const WrappedComponent = React.forwardRef((props, ref) => {
  return <LogProps {...props} forwardedRef={ref} />;
});

// 如果你命名了渲染函数，DevTools 也将包含其名称（例如 "ForwardRef(myFunction)"）：

const WrappedComponent = React.forwardRef(function myFunction(props, ref) {
  return <LogProps {...props} forwardedRef={ref} />;
});

// 你甚至可以设置函数的 displayName 属性来包含被包裹组件的名称：

function logProps(Component) {
  class LogProps extends React.Component {
    // ...
  }

  function forwardRef(props, ref) {
    return <LogProps {...props} forwardedRef={ref} />;
  }

  // 在 DevTools 中为该组件提供一个更有用的显示名。
  // 例如 "ForwardRef(logProps(MyComponent))"
  const name = Component.displayName || Component.name;
  forwardRef.displayName = `logProps(${name})`;
```

```
  return React.forwardRef(forwardRef);
}
```

3. refs 用处

jsx 复制代码

```
##### 1. dom 节点上使用，通过 this.refs[refName]来引用真实的 dom 节点
<Text ref="inputRef" /> //this.refs['inputRef']来访问

##### 2. 回调函数
React 支持给任意组件添加特殊属性。ref 属性接受一个回调函数，它在组件被加载或卸载时会立即执行。
当给 HTML 元素添加 ref 属性时，ref 回调接收了底层的 DOM 元素作为参数。
当给组件添加 ref 属性时，ref 回调接收当前组件实例作为参数。
当组件卸载的时候，会传入 null
ref 回调会在 componentDidMount 或 componentDidUpdate 这些生命周期回调之前执行。
<input ref={(input) => {this.textInput = input;}} type="text" /> //HTML 元素添加 ref 属性时
<CustomInput ref={(input) => {this.textInput = input;}} /> //组件添加 ref 属性
```

3. Fragments

React 中的一个常见模式是一个组件返回多个元素。Fragments 允许你将子列表分组，而无需向 DOM 添加额外节点。

jsx 复制代码

```
// 例如
<table>
  <tr>
    <div>
      <td>Hello</td>
      <td>World</td>
    </div>
  </tr>
</table>
想要将多个td标签放到一起：

class Columns extends React.Component {
  render() {
    return (
      <div>
        <td>Hello</td>
        <td>World</td>
      </div>
    );
  }
```

```
}
```
将会多处div标签，可能会破坏table的表结构

react中使用fragment来代替div标签

```jsx
class Columns extends React.Component {
  render() {
    return (
      <React.Fragment>
        <td>Hello</td>
        <td>World</td>
      </React.Fragment>
    );
  }
}
```

也可以用来包裹列表中有其他的组件

```jsx
render() {
 return <React.Fragment>
        <A />
        <B />
        <C />
    </React.Fragment>
  )
}
```

4. 高阶组件

组件是将 props 转换为 UI，而高阶组件是将组件转换为另一个组件. react 之前使用 mixins 用于解决横切关注点相关的问题。但 mixins 会产生更多麻烦。

jsx 复制代码

```jsx
// 此函数接收一个组件...
function withSubscription(WrappedComponent, selectData) {
  // ...并返回另一个组件...
  return class extends React.Component {
    constructor(props) {
      super(props);
      this.handleChange = this.handleChange.bind(this);
      this.state = {
        data: selectData(DataSource, props),
      };
    }

    componentDidMount() {
      // ...负责订阅相关的操作...
      DataSource.addChangeListener(this.handleChange);
```

```
    }

    componentWillUnmount() {
      DataSource.removeChangeListener(this.handleChange);
    }

    handleChange() {
      this.setState({
        data: selectData(DataSource, this.props),
      });
    }

    render() {
      // ... 并使用新数据渲染被包装的组件!
      // 请注意，我们可能还会传递其他属性
      return <WrappedComponent data={this.state.data} {...this.props} />;
    }
  };
}
```

5. 深入 JSX

JSX 仅仅只是 React.createElement(component, props, ...children) 函数的语法糖

jsx 复制代码

```
<MyButton color="blue" shadowSize={2}>
  Click Me
</MyButton>;

会被编译为;

React.createElement(MyButton, { color: "blue", shadowSize: 2 }, "Click Me");

<div className="sidebar" />;
会编译为: React.createElement("div", { className: "sidebar" });
```

在 JSX 类型中使用点语法

在 JSX 中，你也可以使用点语法来引用一个 React 组件。当你在一个模块中导出许多 React 组件时，这会非常方便。例如，如果 MyComponents.DatePicker 是一个组件，你可以在 JSX 中直接使用：

jsx 复制代码

```jsx
import React from "react";

const MyComponents = {
  DatePicker: function DatePicker(props) {
    return <div>Imagine a {props.color} datepicker here.</div>;
  },
};

function BlueDatePicker() {
  return <MyComponents.DatePicker color="blue" />;
}
```

用户定义的组件必须以大写字母开头

jsx 复制代码

```jsx
错误
<hello toWhat="World" />;
正确
<Hello toWhat="World" />;
```

在运行时选择类型

可以动态的指定 jsx 类型

jsx 复制代码

```jsx
import React from "react";
import { PhotoStory, VideoStory } from "./stories";

const components = {
  photo: PhotoStory,
  video: VideoStory,
};

function Story(props) {
  // 正确！JSX 类型可以是大写字母开头的变量。
  const SpecificStory = components[props.storyType];
  return <SpecificStory story={props.story} />;
}
```

Props 默认值为 “True”

jsx 复制代码

```jsx
下面两个效果是一样的
<MyTextBox autocomplete />
```

```jsx
<MyTextBox autocomplete={true} />
```

属性展开

jsx 复制代码

```jsx
<Greeting {...props} />;

const { kind, ...other } = props;
const className = kind === "primary" ? "PrimaryButton" : "SecondaryButton";
return <button className={className} {...other} />;
```

jsx 子元素

可以用变量，函数等返回 jsx 子元素

jsx 复制代码

```jsx
布尔类型、Null 以及 Undefined 将会忽略
<div />

<div></div>

<div>{false}</div>

<div>{null}</div>

<div>{undefined}</div>

<div>{true}</div>
```

6.性能优化

文件压缩打包

1. 打包生产版本

2. 使用单文件 直接引入

```
<script src="https://unpkg.com/react@16/umd/react.production.min.js"></script>
<script src="https://unpkg.com/react-dom@16/umd/react-dom.production.min.js"></script>
```

3. terser-brunch 压缩 js 文件

```
# 如果你使用 npm
npm install --save-dev terser-brunch

# 如果你使用 Yarn
yarn add --dev terser-brunch
brunch build -p
```

4. Browserify 混淆，转译 js 文件

```
# 如果你使用 npm
npm install --save-dev envify terser uglifyify

# 如果你使用 Yarn
yarn add --dev envify terser uglifyify
```

5. Rollup 是一个 JavaScript 模块打包器，可以将小块代码编译成大块复杂的代码

```
# 如果你使用 npm
npm install --save-dev rollup-plugin-commonjs rollup-plugin-replace rollup-plugin-terser

# 如果你使用 Yarn
yarn add --dev rollup-plugin-commonjs rollup-plugin-replace rollup-plugin-terser
```

6. webpack

分析

1. 使用 Chrome Performance 标签分析组件

Chrome 开发者工具的 Performance 标签并按下 Record。

2. 使用开发者工具中的分析器对组件进行分析

3. 虚拟化长列表(大容量列表)

react-window 和 react-virtualized

7. Portals

用法

```jsx
render() {
  // React 并*没有*创建一个新的 div。它只是把子元素渲染到 `domNode` 中。
  // `domNode` 是一个可以在任何位置的有效 DOM 节点。
  return ReactDOM.createPortal(
    this.props.children,
    domNode
  );
}
```

冒泡事件

其行为和普通的 React 子节点行为一致