Higher-order Components 高阶组件

作者 jacobbubu (/u/8ca9803d6c8a) 关注
2015.08.03 12:09* 字数 866 阅读 3826 评论 2 喜欢 9
(/u/8ca9803d6c8a)

原文来自 https://gist.github.com/sebmarkbage/ef0bf1f338a7182b6775 (https://gist.github.com/sebmarkbage/ef0bf1f338a7182b6775)

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

export const Enhance = (ComposedComponent) => class extends Component {
    constructor() {
        this.state = { data: null };
    }
    componentDidMount() {
        this.setState({ data: 'Hello' });
    }
    render() {
        return <ComposedComponent {...this.props} data={this.state.data} />;
    }
};
```

`Enhance` 是一个方法，当传入一个 Component（ `ComposedComponent` ）的时候，它将自动为该 Component 进行扩展并返回新的类定义。上例中，就返回了一个扩展的 `Component` 类，为构造函数中添加了 `state` ，也在 React 生命周期函数 `componentDidMount` 中添加了处理逻辑，而 `render` 方法则使用了传入的参数，完成了渲染。

新的业务逻辑由 `Enhance` 提供（通过返回新的 Component），传入的 `ComposedComponent` 就像一个回调函数。看看怎么使用：

```
import  { Component }  from "React";
import { Enhance } from "./Enhance";

class MyComponent = class extends Component {
    render() {
        if (!this.props.data) return <div>Waiting...</div>;
        return <div>{this.data}</div>;
    }
}

export default Enhance(MyComponent); // Enhanced component
```

`MyComponent` 就是一个高阶组件（类似于高阶函数-回调函数），负责对特定的数据进行渲染。`MyComponent` 仅仅知道别人会把数据通过 `this.prop.data` 传进来，其他就都不关心了。可以看到，和 Mixins 的扩展方式相比，`MyComponent` 的工作要轻松很多。

Mixins

在 React 中，Mixins 是传统的为 Component 进行扩展的做法。Mixins 的做法很像传统的命令式编程，即要扩展的组件决定需要哪些扩展(Mixins)，以及了解所有扩展(Mixins)的细节，从而避免状态污染。当 Mixins 多了之后，被扩展组件需要维护的状态和掌握的"知识"越来越多，因此也就越来越难维护，因为责任都被交给了"最后一棒"(Last Responsible Moment)。

而高阶组件的思路则是函数式的，每一个扩展（ `Enhance` ）就是一个函数，接受要扩展的组件作为参数。如果要进行 3 个扩展，那么则可以级联，看起来就是：

```
const newComponent = Enhance3(Enhance2(Enhance1(MyComponent)));
```

高阶组件的方式则使得每一个 `Enhance` 以及被扩展的组件都只关心自己手里那点事。`Enhance` 不知道别人会怎么用它，被扩展的组件也不关心别人会怎么扩展它。负责人是那个将它们连在一起的"水管工"，即最后写串联代码的人。

高阶组件的用法虽然用到了 ES6 的类继承，但是实际上却只是把它当个工具使用，而不是真的借助于 OO 的继承。在 React 中使用高阶组件部分替代 Mixins，仍然是非常函数化的思维方式，即针对 ***转换*** 编程。只不过是组件定义替代了函数而已。

Decorators

除了函数方式扩展，通过 ES7 草案中的 Decorator(https://medium.com/google-developers/exploring-es7-decorators-76ecb65fb841 (https://medium.com/google-developers/exploring-es7-decorators-76ecb65fb841)) 也是可以的。Decorator 可以通过返回特定的 descriptor 来"修饰" 类属性，也可以直接"修饰"一个类。即传入一个已有的类，通过 Decorator 函数"修饰"成了一个新的类。那么我们之前的 `Enhance` 方法就可以直接被当成 Decorator 来用了。

```
import { Component } from "react";
import { Enhance } from "./Enhance";

@Enhance
class MyComponent extends Component {
  render() {
    if (!this.data) return <div>Waiting...</div>;
    return <div>{this.data}</div>;
  }
}

export default MyComponent; // Enhanced component
```

用或不用 Decorator，只是习惯问题。

一个真实世界的 Enhance 的例子(https://github.com/kriasoft/react-decorators (https://github.com/kriasoft/react-decorators))

用法代码如下：

```
import React from 'react';
import withViewport from 'react-decorators/withViewport';

@withViewport
class MyComponent {
  render() {
    let { width, height } = this.props.viewport;
    return <div>Viewport: {width + 'x' + height}</div>;
  }
}

React.render(<MyComponent />, document.body);
```

`withView` 是一个 Enhance 函数，或者 Class Decorator，传入一个被扩展组件（`MyComponent`），返回一个新的组件。新组建具有获取窗口尺寸变化的能力（ viewport ），并且将viewport 通过 `this.props.viewport` 传递给被扩展组件(MyComponent)。