组件设计模式（2）：高阶组件

在开发 React 组件过程中，很容易发现这样一种现象，某些功能是多个组件通用的，如果每个组件都重复实现这样的逻辑，肯定十分浪费，而且违反了“不要重复自己” (DRY，Don't Repeat Yourself)的编码原则，我们肯定想要把这部分共用逻辑提取出来重用。

我们说过，在 React 的世界里，组件是第一公民，首先想到的是当然是把共用逻辑提取为一个 React 组件。不过，有些情况下，这些共用逻辑还没法成为一个独立组件，换句话说，这些共用逻辑单独无法使用，它们只是对其他组件的功能加强。

举个例子，对于很多网站应用，有些模块都需要在用户已经登录的情况下才显示。比如，对于一个电商类网站，“退出登录”按钮、“购物车”这些模块，就只有用户登录之后才显示，对应这些模块的 React 组件如果连“只有在登录时才显示”的功能都重复实现，那就浪费了。

这时候，我们就可以利用“高阶组件（HoC）”这种模式来解决问题。

高阶组件的基本形式

“高阶组件”名为“组件”，其实并不是一个组件，而是一个函数，只不过这个函数比较特殊，它接受至少一个 React 组件为参数，并且能够返回一个全新的 React 组件作为结果，当然，这个新产生的 React 组件是对作为参数的组件的包装，所以，有机会赋予新组件一些增强的“神力”。

一个最简单的高阶组件是这样的形式：

```
const withDoNothing = (Component) => {
  const NewComponent = (props) => {
    return <Component {...props} />;
  };
  return NewComponent;
};
```

上面的函数 `withDoNothing` 就是一个高阶组件，作为一项业界通用的代码规范，高阶组件的命名一般都带 `with` 前缀，命名中后面的部分代表这个高阶组件的功能。

就如同 withDoNothing 这个名字所说的一样，这个高阶组件什么都没做，但是从中可以看出高阶组件的基本代码套路。

1. 高阶组件不能去修改作为参数的组件，高阶组件必须是一个纯函数，不应该有任何副作用。
2. 高阶组件返回的结果必须是一个新的 React 组件，这个新的组件的 JSX 部分肯定会包含作为参数的组件。
3. 高阶组件一般需要把传给自己的 props 转手传递给作为参数的组件。

用高阶组件抽取共同逻辑

接下来，我们对 withDoNothing 进行一些改进，让它实现“只有在登录时才显示”这个功能。

假设我们已经有一个函数 `getUserId` 能够从 cookies 中读取登录用户的 ID，如果用户未登录，这个 `getUserId` 就返回空，那么“退出登录按钮“就需要这么写：

```
const LogoutButton = () => {
  if (getUserId()) {
    return ...; // 显示”退出登录“的JSX
  } else {
    return null;
  }
};
```

同样，购物车的代码就是这样：

```
const ShoppintCart = () => {
  if (getUserId()) {
    return ...; // 显示”购物车“的JSX
  } else {
    return null;
  }
};
```

上面两个组件明显有重复的代码，我们可以把重复代码抽取出来，形成 `withLogin` 这个高阶组件，代码如下：

```
const withLogin = (Component) => {
  const NewComponent = (props) => {
    if (getUserId()) {
      return <Component {...props} />;
    } else {
      return null;
    }
  }

  return NewComponent;
};
```

如此一来，我们就只需要这样定义 `LogoutButton` 和 `ShoppintCart`：

```
const LogoutButton = withLogin((props) => {
  return ...; // 显示”退出登录“的JSX
});

const ShoppingCart = withLogin(() => {
  return ...; // 显示”购物车“的JSX
});
```

你看，我们避免了重复代码，以后如果要修改对用户是否登录的判断逻辑，也只需要修改 withLogin，而不用修改每个 React 组件。

高阶组件的高级用法

高阶组件只需要返回一个 React 组件即可，没人规定高阶组件只能接受一个 React 组件作为参数，完全可以传入多个 React 组件给高阶组件。

比如，我们可以改进上面的 withLogin，让它接受两个 React 组件，根据用户是否登录选择渲染合适的组件。

```
const withLoginAndLogout = (ComponentForLogin, ComponentForLogout) => {
  const NewComponent = (props) => {
    if (getUserId()) {
      return <ComponentForLogin {...props} />;
    } else {
      return <ComponentForLogout{...props} />;
    }
  }
  return NewComponent;
};
```

有了上面的 `withLoginAndLogout`，就可以产生根据用户登录状态显示不同的内容。

```
const TopButtons = withLoginAndLogout(
  LogoutButton,
  LoginButton
);
```

链式调用高阶组件

高阶组件最巧妙的一点，是可以链式调用。

假设，你有三个高阶组件分别是 `withOne`、`withTwo` 和 `withThree`，那么，如果要赋予一个组件 X 某个高阶组件的超能力，那么，你要做的就是挨个使用高阶组件包装，代码如下：

```
const X1 = withOne(X);
const X2 = withTwo(X1);
const X3 = withThree(X2);
const SuperX = X3; //最终的SuperX具备三个高阶组件的超能力
```

很自然，我们可以避免使用中间变量 `X1` 和 `X2`，直接连续调用高阶组件，如下：

```
const SuperX = withThree(withTwo(withOne(X)));
```

对于 `X` 而言，它被高阶组件包装了，至于被一个高阶组件包装，还是被 N 个高阶组件包装，没有什么差别。而高阶组件本身就是一个纯函数，纯函数是可以组合使用的，所以，我们其实可以把多个高阶组件组合为一个高阶组件，然后用这一个高阶组件去包装 `X`，代码如下：

```
const hoc = compose(withThree, withTwo, withOne);
const SuperX = hoc(X);
```

在上面代码中使用的 `compose`，是函数式编程中很基础的一种方法，作用就是把多个函数组合为一个函数，在很多开源的代码库中都可以看到，下面是一个参考实现：

```
export default function compose(...funcs) {
  if (funcs.length === 0) {
    return arg => arg
  }

  if (funcs.length === 1) {
    return funcs[0]
  }

  return funcs.reduce((a, b) => (...args) => a(b(...args)))
}
```

React 组件可以当做积木一样组合使用，现在有了 compose，我们就可以把高阶组件也当做积木一样组合，进一步重用代码。

假如一个应用中多个组件都需要同样的多个高阶组件包装，那就可以用 compose 组合这些高阶组件为一个高阶组件，这样在使用多个高阶组件的地方实际上就只需要使用一个高阶组件了。

不要滥用高阶组件

高阶组件虽然可以用一种可重用的方式扩充现有 React 组件的功能，但高阶组件并不是绝对完美的。

首先，高阶组件不得不处理 `displayName`，不然 debug 会很痛苦。当 React 渲染出错的时候，靠组件的 displayName 静态属性来判断出错的组件类，而高阶组件总是创造一个新的 React 组件类，所以，每个高阶组件都需要处理一下 displayName。

如果要做一个最简单的什么增强功能都没有的高阶组件，也必须要写下面这样的代码：

```
const withExample = (Component) => {
  const NewComponent = (props) => {
    return <Component {...props} />;
  }

  NewComponent.displayName = `withExample(${Component.displayName || Component.name || 'Component'})`

  return NewCompoennt;
};
```

每个高阶组件都这么写，就会非常的麻烦。

对于 React 生命周期函数，高阶组件不用怎么特殊处理，但是，如果内层组件包含定制的静态函数，这些静态函数的调用在 React 生命周期之外，那么高阶组件就必须要在新产生的组件中增加这些静态函数的支持，这更加麻烦。

其次，高阶组件支持嵌套调用，这是它的优势。但是如果真的一大长串高阶组件被应用的话，当组件出错，你看到的会是一个超深的 stack trace，十分痛苦。

最后，使用高阶组件，一定要非常小心，要避免重复产生 React 组件，比如，下面的代码是有问题的：

```
const Example = () => {
  const EnhancedFoo = withExample(Foo);
  return <EnhancedFoo />
}
```

像上面这样写，每一次渲染 Example，都会用高阶组件产生一个新的组件，虽然都叫做 `EnhancedFoo`，但是对 React 来说是一个全新的东西，在重新渲染的时候不会重用之前的虚拟 DOM，会造成极大的浪费。

正确的写法是下面这样，自始至终只有一个 EnhancedFoo 组件类被创建：

```
const EnhancedFoo = withExample(Foo);

const Example = () => {
  return <EnhancedFoo />
}
```

总之，高阶组件是重用代码的一种方式，但并不是唯一方式，在下一小节，我们会介绍一种更加精妙的方式。

小结

在这一小节中，我们介绍了高阶组件（HoC）这种重用逻辑的模式，读者应该能够学会：

1. 高阶组件的形式；
2. 高阶组件的链式调用方法；
3. 高阶组件的不足。

留言

评论将在后台进行审核，审核通过后对所有人可见

Robert 2017 前端工程师
这里怎么理解呢？能否具体些呢
对于 React 生命周期函数，高阶组件不用怎么特殊处理，但是，如果内层组件包含定制的静态函数，这些静态函数的调用在 React 生命周期之外，那么高阶组件就必须要在新产生的组件中增加这些静态函数的支持，这更加麻烦。
▲ 1 收起评论 1月前

xiari 高级前端工程师 @ 拼多多
被高阶组件A包裹的组件B， 如果B有某个 static 方法， 则A里必须显式调用下 B 的static 方法。比如在next.js 中就必须注意 getInitialProps 这个static 方法。
25天前

jeffacode同学 前端开发 @ 不好意思无业游民
我也想问这个，不过“这些静态函数的调用在React生命周期之外”这句还是好理解的，毕竟调用它们并不需要生成组件实例，但后一句是什么意思？前面一位老兄说的，为什么高阶组件A非得去显式调用B的静态方法？？
21天前

jeffacode同学 前端开发 @ 不好意思无业游民
哦哦，https://zhuanlan.zhihu.com/p/36178509
21天前

Robert 2017 前端工程师
回复 xiari：明白惹.hh 谢谢
20天前

评论审核通过后显示 评论

code11
大神，看这里， NewComponent.displayName = `withExample(${Component.displayName || Component.name || 'Component'})`; 这个是把高阶组件又调用了一次？ 还是?
▲ 0 收起评论 1月前

程墨 Hulu
这只是在拼字符串，没有调用。
1月前

卡西法 前端工程师
这个就是高阶组建本身，只是加了一个属性，不是调用。
1月前

评论审核通过后显示 评论

Ru_minnnn
高阶组件可以返回表现型组件和容器型组件，挺好
▲ 0 评论 1月前

Farris 前端工程师
沙发
▲ 0 评论 1月前