组件设计模式（3）：render props 模式

在上一小节中，我们介绍了高阶组件，高阶组件并不是 React 中唯一的重用组件逻辑的方式，在这一小节中，我们会介绍另一种方式 render props。

render props

所谓 render props，指的是让 React 组件的 props 支持函数这种模式。因为作为 props 传入的函数往往被用来渲染一部分界面，所以这种模式被称为 render props。

一个最简单的 render props 组件 `RenderAll`，代码如下：

```
const RenderAll = (props) => {
  return(
     <React.Fragment>
        {props.children(props)}
     </React.Fragment>
  );
};
```

这个 `RenderAll` 预期子组件是一个函数，它所做的事情就是把子组件当做函数调用，调用参数就是传入的 props，然后把返回结果渲染出来，除此之外什么事情都没有做。

使用 RenderAll 的代码如下：

```
    <RenderAll>
      {() => <h1>hello world</h1>}
    </RenderAll>
```

可以看到，RenderAll 的子组件，也就是夹在 RenderAll 标签之间的部分，其实是一个函数。这个函数渲染出 `<h1>hello world</h1>`，这就是上面使用 RenderAll 渲染出来的结果。

当然，这个 RenderAll 没做任何实际工作，接下来我们看 render props 真正强悍的使用方法。

传递 props

和高阶组件一样，render props 可以做很多的定制功能，我们还是以根据是否登录状态来显示一些界面元素为例，来实现一个 render props。

下面是实现 render props 的 `Login` 组件，可以看到，render props 和高阶组件的第一个区别，就是 render props 是真正的 React 组件，而不是一个返回 React 组件的函数。

```
const Login = (props) => {
  const userName = getUserName();

  if (userName) {
    const allProps = {userName, ...props};
    return (
      <React.Fragment>
        {props.children(allProps)}
      </React.Fragment>
    );
  } else {
    return null;
  }
};
```

当用户处于登录状态，`getUserName` 返回当前用户名，否则返回空，然后我们根据这个结果决定是否渲染 `props.children` 返回的结果。

当然，render props 完全可以决定哪些 props 可以传递给 props.children，在 Login 中，我们把 `userName` 作为增加的 props 传递给下去，这样就是 Login 的增强功能。

一个使用上面 Login 的 JSX 代码示例如下：

```
  <Login>
    {({userName}) => <h1>Hello {userName}</h1>}
  </Login>
```

对于名为“程墨Morgan”的用户登录，上面的 JSX 会产生 `<h1>Hello 程墨Morgan</h1>`。

不局限于 children

在上面的例子中，作为 render 方法的 props 就是 `children`，在我写的《深入浅出React和Redux》中，将这种模式称为“以函数为子组件（function as child）”的模式，这可以算是 render props 的一种具体形式，也就利用 `children` 这个 props 来作为函数传递。

实际上，render props 这个模式不必局限于 children 这一个 props，任何一个 props 都可以作为函数，也可以利用多个 props 来作为函数。

我们来扩展 Login，不光在用户登录时显示一些东西，也可以定制用户没有登录时显示的东西，我们把这个组件叫做 `Auth`，对应代码如下：

```
const Auth= (props) => {
  const userName = getUserName();

  if (userName) {
    const allProps = {userName, ...props};
    return (
      <React.Fragment>
        {props.login(allProps)}
      </React.Fragment>
    );
  } else {
    <React.Fragment>
      {props.nologin(props)}
    </React.Fragment>
  }
};
```

使用 Auth 的话，可以分别通过 `login` 和 `nologin` 两个 props 来指定用户登录或者没登录时显示什么，用法如下：

```
  <Auth
    login={({userName}) => <h1>Hello {userName}</h1>}
    nologin={() => <h1>Please login</h1>}
  />
```

依赖注入

render props 其实就是 React 世界中的“依赖注入”（Dependency Injection)。

所谓依赖注入，指的是解决这样一个问题：逻辑 A 依赖于逻辑 B，如果让 A 直接依赖于 B，当然可行，但是 A 就没法做得通用了。依赖注入就是把 B 的逻辑以函数形式传递给 A，A 和 B 之间只需要对这个函数接口达成一致就行，如此一来，再来一个逻辑 C，也可以用一样的方法重用逻辑 A。

在上面的代码示例中，`Login` 和 `Auth` 组件就是上面所说的逻辑 A，而传递给组件的函数类型 props，就是逻辑 B 和 C。

render props 和高阶组件的比较

我们来比对一下这两种重用 React 组件逻辑的模式。

首先，render props 模式的应用，就是做一个 React 组件，而高阶组件，虽然名为“组件”，其实只是一个产生 React 组件的函数。

render props 不像上一小节中介绍的高阶组件有那么多毛病，如果说 render props 有什么缺点，那就是 render props 不能像高阶组件那样链式调用，当然，这并不是一个致命缺点。

render props 相对于高阶组件还有一个显著优势，就是对于新增的 props 更加灵活。还是以登录状态为例，假如我们扩展 withLogin 的功能，让它给被包裹的组件传递用户名这个 props，代码如下：

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

  return NewComponent;
};
```

这就要求被 withLogin 包住的组件要接受 `userName` 这个props。可是，假如有一个现成的 React 组件不接受 userName，却接受名为 `name` 的 props 作为用户名，这就麻烦了。我们就不能直接用 withLogin 包住这个 React 组件，还要再造一个组件来做 `userName` 到 `name` 的映射，十分费事。

对于应用 render props 的 Login，就不存在这个问题，接受 `name` 不接受 `userName` 是吗？这样写就好了：

```
<Login>
  {
    (props) => {
      const {userName} = props;
      return <TheComponent {...props} name={userName} />
    }
  }
</Login>
```

所以，当需要重用 React 组件的逻辑时，建议首先看这个功能是否可以抽象为一个简单的组件；如果行不通的话，考虑是否可以应用 render props 模式；再不行的话，才考虑应用高阶组件模式。

这并不表示高阶组件无用武之地，在后续章节，我们会对 render props 和高阶组件分别讲解具体的实例。

小结

在这一小节中，我们介绍了 render props 这种模式，也将 render props 和高阶组件两种模式进行了比较。

读者应该要明白：

1. render props 的形式；
2. render props 其实就是“依赖注入”；
3. 如何利用 render props 实现共享组件之间的逻辑。

留言

评论将在后台进行审核，审核通过后对所有人可见

娄童
希望能够给举例一些实际的应用场景，目前根据一个登录的例子，我还是不能灵活应用。
▲ 1 评论 10天前

xiari 高级前端工程师 @ 拼多多
renderProps 的应用可以参考 React 的 context 用法。 Consumer 这一块。
▲ 0 评论 25天前

阿五 web前端开发工程师
第四段代码，else没有return元素出来;
const Auth= (props) => {
 const userName = getUserName();

 if (userName) {
  const allProps = {userName, ...props};
  return (
   <React.Fragment>
    {props.login(allProps)}... 展开全部
▲ 0 评论 25天前

Robert 2017 前端工程师
// 这里既然拿到了username 直接 打印就好了吧，为什么还要在外部用 children 参数形式 传递呢？

const Login = (props) => {
 const userName = getUserName();

 if (userName) {
  const allProps = {userName, ...props};
  return (... 展开全部
▲ 1 收起评论 1月前

程墨 Hulu
因为使用userName的方式很多，可能是打印，可能是拼成其他东西，可能加一个写样式显示，这些决定权交给children。
1月前

阿五 web前端开发工程师
原来如此
25天前

Robert 2017 前端工程师
嗯，明白了。通用组件以提取不确定的放到调用层来带到自身的通用。
16天前

评论审核通过后显示 评论

之乎哲也.
请问`{props.nologin(props)}`这种类型的代码中，有必要将props再次传递给props函数（nologin）吗？直观上理解这种情况下直接call nologin()不需要传递props进去也是可以的，而且会更好理解一些？
▲ 0 收起评论 1月前

程墨 Hulu
作为一个通用的render props，并不知道子组件需要用到哪些props，所以应该都传递过去，用不用在他，但是传不传在你。
1月前

之乎哲也.
理解了。通用这个词用的好，这就意味着<Login />完全和传进来的children解耦了，并不干涉children到底用不用props/怎么用props，跟HOC一个道理。谢谢：）
1月前

评论审核通过后显示 评论

再来以后
第一段代码中RenderAll 的第二个React.Fragment 写成了 React.Fragement
▲ 0 收起评论 1月前

程墨 Hulu
改过来了，谢谢指正。
1月前

评论审核通过后显示 评论

Da'Mn° 前端开发工程师
<Login>
  {({userName}) => <h1>Hello {userName}</h1>}
 </Login>

const Login = (props) => {
 const userName = getUserName();

 if (userName) {
  const allProps = {userName, ...props}... 展开全部
▲ 0 收起评论 1月前

caiyangL
此处userName来自Login组件内部逻辑而不是Login的props
1月前

程墨 Hulu
并不是这样，你可以运行一下代码看看
1月前

肖炎 前端开发 @ 今日头条
render props 已经将props通过函数参数传下去了 尤其是userName 也是一个参数{userName, ...props};
1月前

Da'Mn° 前端开发工程师
回复 程墨：对，是这样的没错，是我自己写代码跟您这边有点不一样导致的，我内部没有写const userName = getUserName();类似这个方法去获取userName，不好意思，看岔了。
1月前

评论审核通过后显示 评论

void promise
const allProps = {userName, ...props};的目的是什么
▲ 0 收起评论 1月前

红谷滩陈冠希 前端 @ 无限996公司
如果登录了，那就有 userName 这个字段，添加到 allProps 中传递下去，如果没有userName, 那就直接传递 props 就行了。
1月前

Da'Mn° 前端开发工程师
其实我认为是不是写反了，因为后面的props扩展后如果有同名属性会覆盖前面的userName，是不是写成{...props, userName}比较恰当
1月前

一個抗爭的少年
回复 Da'Mn°：可以把props中的userName理解为调用Auth时指定的值，把通过getUserName()获取的值当作默认值。例如，用{ userName, ...props }这种写法，你可以这样写：
<Auth
  login={({userName}) => <h1>Hello {userName}</h1>}
  nologin={() => <h1>Please login</h1>}
  userName='Andrew'
 />
1月前

zhangyanling77 前端开发 @ 成都
回复 Da'Mn°：我觉得你说得有道理，应该是{...props, userName}比较好
1月前

之乎哲也.
请问`{props.nologin(props)}`这种类型的代码中，有必要将props再次传递给props函数（nologin）吗？直观上理解这种情况下直接call nologin()不需要传递props进去也是可以的，而且会更好理解一些？
1月前

1 2 下一页

评论审核通过后显示 评论