# 组件设计模式（1）：聪明组件和傻瓜组件

从这一节开始，我们来介绍 React 中的模式。

在 React 应用中，最简单也是最常用的一种组件模式，就是“聪明组件和傻瓜组件”。

其实，这个模式的名称很多，就我所知，除了“聪明组件和傻瓜组件”，还有这些称呼：

1. 容器组件和展示组件（Container and Presentational Components）；
2. 胖组件和瘦组件；
3. 有状态组件和无状态组件。

名字只是一个代号，关键还是要看本质，这种模式的本质，就是把一个功能分配到两个组件中，形成父子关系，外层的父组件负责管理数据状态，内层的子组件只负责展示。

在本小册中，都会以“聪明组件”和“傻瓜组件”称呼这种模式。

## 为什么要分割聪明组件和傻瓜组件

软件设计中有一个原则，叫做“责任分离”（Separation of Responsibility），简单说就是让一个模块的责任尽量少，如果发现一个模块功能过多，就应该拆分为多个模块，让一个模块都专注于一个功能，这样更利于代码的维护。

还记得我么说过 React 其实就是这样一个公式吗?

```
UI = f(data)
```

使用 React 来做界面，无外乎就是获得驱动界面的数据，然后利用这些数据来渲染界面。当然，你可以在一个组件中就搞定，但是，最好把获取和管理数据这件事和界面渲染这件事分开。做法就是，把获取和管理数据的逻辑放在父组件，也就是聪明组件；把渲染界面的逻辑放在子组件，也就是傻瓜组件。

这么做的好处，是可以灵活地修改数据状态管理方式，比如，最初你可能用 Redux 来管理数据，然后你想要修改为用 Mobx，如果按照这种模式分割组件，那么，你需要改的只有聪明组件，傻瓜组件可以保持原状。

## 随机笑话样例

我们利用一个显示“随机笑话”的功能来演示这种模式，所谓“随机笑话”，就是需要从服务器获取随机的一个笑话，展示在页面上。

功能可以分为两部分，第一部分是展示，也就是傻瓜组件，代码如下：

```
import SmileFace from './yaoming_simile.png';

const Joke = ({value}) => {
  return (
    <div>
      <img src={SmileFace} />
      {value || 'loading...' }
    </div>
  );
}
```

傻瓜组件 `Joke` 的功能很简单，显示一个笑脸，然后显示名为 value 的 props，也就是笑话的内容，如果没有 value 值，就显示一个“loading...”。

至于怎么获得笑话内容，不是 Joke 要操心的事，它只专注于显示笑话，所谓傻人有傻福，傻瓜组件虽然“傻”了一点，但是免去了数据管理的烦恼。

然后是聪明组件，这个组件不用管渲染的逻辑，只负责拿到数据，然后把数据传递给傻瓜组件，由傻瓜组件来完成渲染。

我们把聪明组件命名为 `RandomJoke`，代码如下：

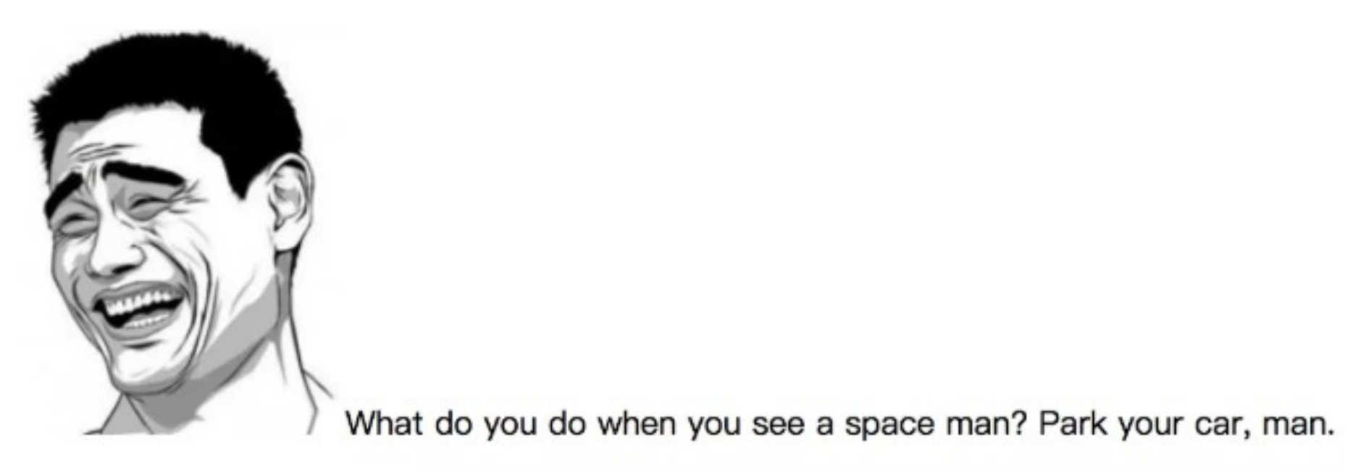
```
export default class RandomJoke extends React.Component {
  state = {
    joke: null
  }

  render() {
    return <Joke value={this.state.joke} />
  }

  componentDidMount() {
    fetch('https://icanhazdadjoke.com/',
      {headers: {'Accept': 'application/json'}}
    ).then(response => {
      return response.json();
    }).then(json => {
      this.setState({joke: json.joke});
    });
  }
}
```

可以看到，RandomJoke 的 `render` 函数只做一件事，就是渲染 Joke，并把 `this.state` 中的值作为 props 传进去。聪明组件的 `render` 函数一般都这样简单，因为渲染不是他们操心的业务，他们的主业是获取数据。

RandomJoke 获取数据的方法是在 `componentDidMount` 函数中调用一个 API (icanhazdadjoke.com/)，这个 API 随即返回一个英文笑话。实话说，这些英文笑话都很冷，也不大容易看懂，但是足够展示通过 API 获取数据的过程。

当 RandomJoke 被第一次渲染的时候，它的 state 中的 joke 值为 `null`，所以它传给 Joke 的 value 也是 `null`，这时候，Joke 会渲染一“loading...”。但是，在第一次渲染完毕的时候，`componentDidMount` 被调用，一个 API 请求发出去，拿到一个随机笑话，更新 state 中的 joke 值。因为对一个组件 state 的更新会引发一个新的渲染过程，所以 RandomJoke 的 `render` 再一次被调用，所以 Joke 也会再一次被渲染，这一次，传入的 value 值是一个真正的笑话，所以，笑话也就出现了。

最终界面类似这个样子：



当然，这个界面非常简陋，获取数据的方法也十分粗糙，但是，最妙的是，应用了这种方法之后，如果你要优化界面，只需要去修改傻瓜组件 Joke，如果你想改进数据管理和获取，只需要去修改聪明组件 RandomJoke。

如此一来，维护工作就简单多了，你甚至可以把两个组件分配各两个不同的开发者去维护开发。

如果应用 Redux 和 Mobx，也会应用到这种模式，我们在后面的小节中会详细介绍更多这种模式的应用。

## PureComponent

因为傻瓜组件一般没有自己的状态，所以，可以像上面的 Joke 一样实现为函数形式，其实，我们可以进一步改进，利用 `PureComponent` 来提高傻瓜组件的性能。

函数形式的 React 组件，好处是不需要管理 state，占用资源少，但是，函数形式的组件无法利用 `shouldComponentUpdate`。

看上面的例子，当 RandomJoke 要渲染 Joke 时，即使传入的 props 是一模一样的，Joke 也要走一遍完整的渲染过程，这就显得浪费了。

好一点的方法，是把 Joke 实现为一个类，而且定义 shouldComponentUpdate 函数，每次渲染过程中，在 render 函数执行之前 shouldComponentUpdate 会被调用，如果返回 `true`，那就继续，如果返回 `false`，那么渲染过程立刻停止，因为这代表不需要重画了。

对于傻瓜组件，因为逻辑很简单，界面完全由 props 决定，所以 shouldComponentUpdate 的实现方式就是比较这次渲染的 props 是否和上一次 props 相同。当然，让每一个组件都实现一遍这样简单的 shouldComponentUpdate 也很浪费，所以，React 提供了一个简单的实现工具 `PureComponent`，可以满足绝大部分需求。

改进后的 Joke 组件如下：

```
class Joke extends React.PureComponent {
  render() {
    return (
      <div>
        <img src={SmileFace} />
        {this.props.value || 'loading...' }
      </div>
    );
  }
}
```

值得一提的是，`PureComponent` 中 `shouldComponentUpdate` 对 props 做得只是浅层比较，不是深层比较，如果 props 是一个深层对象，就容易产生问题。

比如，两次渲染传入的某个 props 都是同一个对象，但是对象中某个属性的值不同，这在 PureComponent 眼里，props 没有变化，不会重新渲染，但是这明显不是我们想要的结果。

## React.memo

虽然 PureComponent 可以提高组件渲染性能，但是它也不是没有代价的，它逼迫我们必须把组件实现为 class，不能用纯函数来实现组件。

如果你使用 React v16.6.0 之后的版本，可以使用一个新功能 `React.memo` 来完美实现 React 组件，上面的 Joke 组件可以这么写：

```
const Joke = React.memo(() => (
    <div>
        <img src={SmileFace} />
        {this.props.value || 'loading...' }
    </div>
));
```

React.memo 既利用了 shouldComponentUpdate，又不要求我们写一个 class，这也体现出 React 逐步向完全函数式编程前进。

## 小结

在这一小节中，我们介绍了“聪明组件和傻瓜组件”这种做法简单的 React 设计模式，读者应该能够理解为什么有时候要把组件分为两个部分，在众多框架中，都可以应用“聪明组件和傻瓜组件”模式。

留言

评论将在后台进行审核，审核通过后对所有人可见

**fuxiang123**
请教一下：比如我一个页面上半部分是一个轮播图，下半部分是一个列表，两个部分的逻辑都不一样。我是在页面的最顶层写入所有逻辑呢，还是再用两个组件分别表示各自的逻辑比较好？
▲ 0 收起评论 4天前

程墨 Hulu
如果像你说的这样两个完全独立的模块，当然是各自实现逻辑更好。
3天前

评论审核通过后显示 评论

**刚好喜欢你丶**
我想问一个问题啊，例子里面只是简单的父子级关系，但是多级情况下，大佬能不能给个分析。谢谢。
▲ 0 收起评论 8天前

程墨 Hulu
接着看，关于Context、Redux和Mobx部分就是处理多级的情况。
3天前

评论审核通过后显示 评论

**17、a** 17
大佬貌似memo那个例子写错了吧？应该是React.memo((props) => {

})
▲ 1 收起评论 15天前

17、a 17
React.memo((props) => (
  <div>
    {props.value || 'loading...' }
  </div>
))
15天前

评论审核通过后显示 评论

**zweekend**
为什么不将shouldComponentUpdate放到容器组件中了？
▲ 0 收起评论 1月前

三郎本尊
一般容器组件都会配置react-redux的connect，connect内已经配置了shouldComponentUpdate，所以不用
20天前

评论审核通过后显示 评论

**GentleGuo**
如果给一个组件传递一个值，但组件内并没有使用这个值，那么当外部传递的这个值变化时，组件会重新渲染吗？
▲ 0 收起评论 1月前

程墨 Hulu
外部传递这个值变化的过程，其实必定会引起这个组件重新渲染。
1月前

评论审核通过后显示 评论

**Farris** 前端工程师
"傻瓜组件 Joke 的功能很简答，显示一个笑脸"

有错别字哈~
▲ 0 收起评论 1月前

Linmi 导游 @ 掘金
已经修改，感谢反馈！
1月前

评论审核通过后显示 评论

**blacker**
React.memo也是浅比较,不同的是，我们可以自定义比较规则
React.memo(youComponent,compareMethod)
▲ 3 收起评论 1月前

程墨 Hulu
React.memo的确支持第二个函数参数作比较，只是这个函数参数只能访问到props，没有state，这也是memo的一个局限。
1月前

晨风明悟 前端 @ Now.Now.Now
回复 程墨：程墨老师，我觉得这个应该也不算局限吧，因为如果使用了 state，那就表明这个组件自身是有状态的，那直接继承 PureComponent 我觉得才是最好的选择。
19天前

评论审核通过后显示 评论