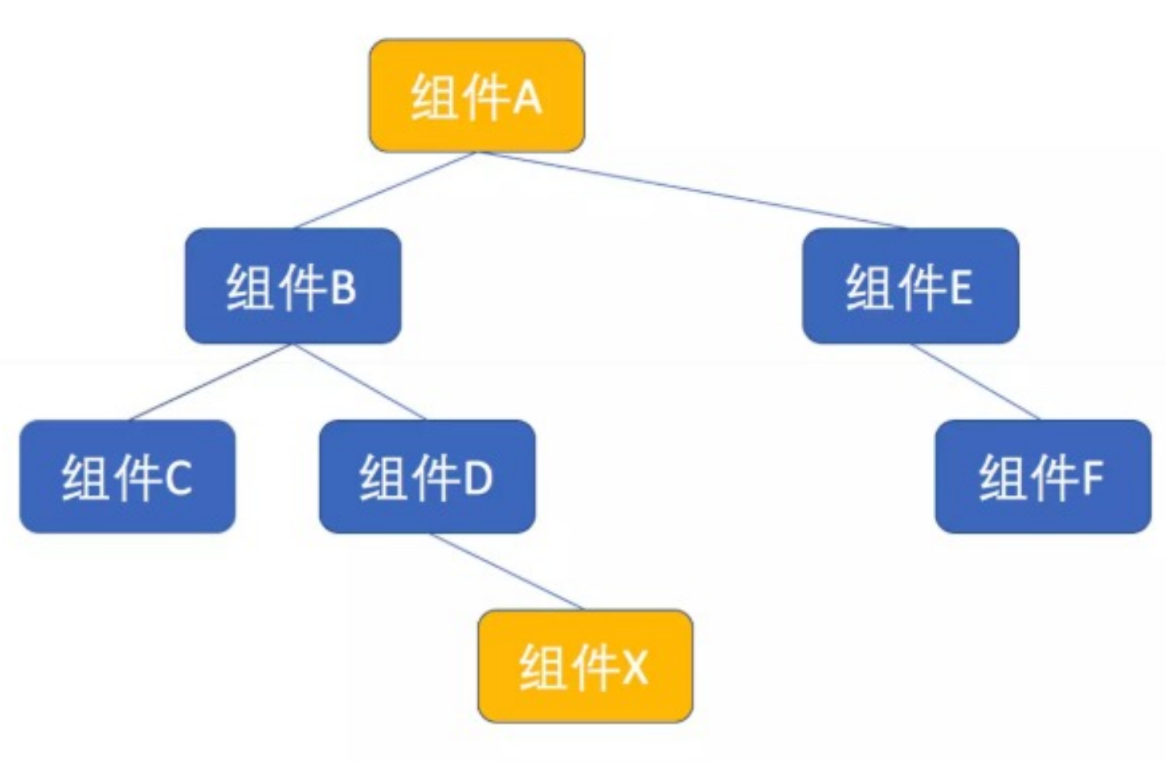
组件设计模式（4）：提供者模式

这一节我们来介绍 React 中的“提供者模式”（Provider Pattern）。

问题场景

在 React 中，props 是组件之间通讯的主要手段，但是，有一种场景单纯靠 props 来通讯是不恰当的，那就是两个组件之间间隔着多层其他组件，下面是一个简单的组件树示例图图：



在上图中，组件 A 需要传递信息给组件 X，如果通过 props 的话，那么从顶部的组件 A 开始，要把 props 传递给组件 B，然后组件 B 传递给组件 D，最后组件 D 再传递给组件 X。

其实组件 B 和组件 D 完全用不上这些 props，但是又被迫传递这些 props，这明显不合理，要知道组件树的结构会变化的，将来如果组件 B 和组件 D 之间再插入一层新的组件，这个组件也需要传递这个 props，这就麻烦无比。

可见，对于跨级的信息传递，我们需要一个更好的方法。

在 React 中，解决这个问题应用的就是“提供者模式”。

提供者模式

虽然这个模式叫做“提供者模式”，但是其实有两个角色，一个叫“提供者”（Provider），另一个叫“消费者”（Consumer），这两个角色都是 React 组件。其中“提供者”在组件树上居于比较靠上的位置，“消费者”处于靠下的位置。在上面的组件树中，组件 A 可以作为提供者，组件 X 就是消费者。

既然名为“提供者”，它可以提供一些信息，而且这些信息在它之下的所有组件，无论隔了多少层，都可以直接访问到，而不需要通过 props 层层传递。

避免 props 逐级传递，即是提供者的用途。

如何实现提供者模式

实现提供者模式，需要 React 的 Context 功能，可以说，提供者模式只不过是让 Context 功能更好用一些而已。

所谓 Context 功能，就是能够创造一个“上下文”，在这个上下文笼罩之下的所有组件都可以访问同样的数据。

在 React v16.3.0 之前，React 虽然提供了 Context 功能，但是官方文档上都建议尽量不要使用，因为对应的 API 他们并不满意，觉得迟早要废弃掉。即使如此，依然有很多库和应用使用 Context 功能，可见对这个需求的呼声有多大。

当 React 发布 v16.3.0 时，终于提供了“正式版本”的 Context 功能 API，和之前的有很大不同，当然，这也带来一些问题，我在后面会介绍。

提供者模式的一个典型用例就是实现“样式主题”（Theme），由顶层的提供者确定一个主题，下面的样式就可以直接使用对应主题里的样式。这样，当需要切换样式时，只需要修改提供者就行，其他组件不用修改。

为了方便比对，这里我会介绍提供者模式用不同 Context API 的实现方法。不过，你如果完全不在意老版本 React 如何实现的，可以略过下面一段。

React v16.3.0 之前的提供者模式

在 React v16.3.0 之前，要实现提供者，就要实现一个 React 组件，不过这个组件要做两个特殊处理。

1. 需要实现 `getChildContext` 方法，用于返回“上下文”的数据；
2. 需要定义 `childContextTypes` 属性，声明“上下文”的结构。

下面就是一个实现“提供者”的例子，组件名为 `ThemeProvider`：

```
class ThemeProvider extends React.Component {
  getChildContext() {
    return {
      theme: this.props.value
    };
  }

  render() {
    return (
      <React.Fragment>
        {this.props.children}
      </React.Fragment>
    );
  }
}

ThemeProvider.childContextTypes = {
  theme: PropTypes.object
};
```

在上面的例子中，getChildContext 只是简单返回名为 `value` 的 props 值，但是，因为 getChildContext 是一个函数，它可以有更加复杂的操作，比如可以从 state 或者其他数据源获得数据。

对于 ThemeProvider，我们创造了一个上下文，这个上下文就是一个对象，结构是这样：

```
{
  theme: {
    //一个对象
  }
}
```

接下来，我们来做两个消费（也就是使用）这个“上下文”的组件，第一个是 `Subject`，代表标题；第二个是 `Paragraph`，代表章节。

我们把 Subject 实现为一个类，代码如下：

```
class Subject extends React.Component {
  render() {
    const {mainColor} = this.context.theme;
    return (
      <h1 style={{color: mainColor}}>
        {this.props.children}
      </h1>
    );
  }
}

Subject.contextTypes = {
  theme: PropTypes.object
}
```

在 Subject 的 `render` 函数中，可以通过 `this.context` 访问到“上下文”数据，因为 ThemeProvider 提供的“上下文”包含 `theme` 字段，所以可以直接访问 `this.context.theme`。

千万不要忘了 Subject 必须增加 `contextTypes` 属性，必须和 ThemeProvider 的 `childContextTypes` 属性一致，不然，`this.context` 就不会得到任何值。

读者可能会问了，为什么这么麻烦呢？为什么要求“提供者”用 `childContextTypes` 定义一次上下文结构，又要求“消费者”再用 `contextTypes` 再重复定义一次呢？这不是很浪费吗？

React 这么要求，是考虑到“上下文”可能会嵌套，就是一个“提供者”套着另一个“提供者”，这时候，底层的消费者组件到底消费哪一个“提供者”呢？通过这种显示的方式指定。

不过，实话实说，这样的 API 设计的确麻烦了一点，难怪 React 官方在最初就不建议使用。

上面的 Subject 是一个类，其实也可以把消费者实现为一个纯函数组件，只不过访问“上下文”的方式有些不同，我们用纯函数的方式实现另一个消费者 `Paragraph`，代码如下：

```
const Paragraph = (props, context) => {
  const {textColor} = context.theme;
  return (
    <p style={{color: textColor}}>
      {props.children}
    </p>
  );
};

Paragraph.contextTypes = {
  theme: PropTypes.object
};
```

从上面的代码可以看到，因为 Paragraph 是一个函数形式，所以不可能访问 `this.context`，但是函数的第二个参数其实就是 `context`。

当然，也不要忘了设定 Paragraph 的 `contextTypes`，不然参数 `context` 也不会是上下文。

最后，我们看如何结合”提供者“和”消费者“。

我们做一个组件来使用 Subject 和 Paragraph，这个组件不需要帮助传递任何 props，代码如下：

```
const Page = () => (
  <div>
    <Subject>这是标题</Subject>
    <Paragraph>
      这是正文
    </Paragraph>
  </div>
);
```

上面的组件 `Page` 使用了 Subject 和 Paragraph，现在我们想要定制样式主题，只需要在 Page 或者任何需要应用这个主题的组件外面包上 ThemeProvider，对应的 JSX 代码如下：

```
  <ThemeProvider value={{mainColor: 'green', textColor: 'red'}} >
    <Page />
  </ThemeProvider>
```

最后，看到的效果如下：

当我们需要改变一个样式主题的时候，改变传给 ThemeProvider的 value 值就搞定了。

React v16.3.0 之后的提供者模式

到了 React v16.3.0 的时候，新的 Context API 出来了，这套 API 毫不掩饰自己就是“提供者模式”的实现，命名上就带 “Provider” 和 “Consumer”。

还是上面的样式主题的例子，首先，要用新提供的 `createContext` 函数创造一个“上下文”对象。

```
const ThemeContext = React.createContext();
```

这个“上下文”对象 `ThemeContext` 有两个属性，分别就是——对，你没猜错—— `Provider` 和 `Consumer`。

```
const ThemeProvider = ThemeContext.Provider;
const ThemeConsumer = ThemeContext.Consumer;
```

创造“提供者”极大简化了，都不需要我们创造一个 React 组件类。

使用“消费者”也同样简单，而且应用了上一节我们介绍的 render props 模式，比如，Subject 的代码如下：

```
class Subject extends React.Component {
  render() {
    return (
      <ThemeConsumer>
        {
          (theme) => (
            <h1 style={{color: theme.mainColor}}>
              {this.props.children}
            </h1>
          )
        }
      </ThemeConsumer>
    );
  }
}
```

上面的 `ThemeConsumer` 其实就是一个应用了 render props 模式的组件，它要求子组件是一个函数，会把“上下文”的数据作为参数传递给这个函数，而这个函数里就可以通过参数访问“上下文”对象。

在新的 API 里，不需要设定组件的 `childContextTypes` 或者 `contextTypes` 属性，这省了不少事。

可以注意到，Subject 没有自己的状态，没必要实现为类，我们用纯函数的形式实现 `Paragraph`，代码如下：

```
const Paragraph = (props, context) => {
  return (
    <ThemeConsumer>
      {
        (theme) => (
          <p style={{color: theme.textColor}}>
            {props.children}
          </p>
        )
      }
    </ThemeConsumer>
  );
};
```

实现 `Page` 的方式并没有变化，而应用 `ThemeProvider` 的代码和之前也完全一样：

```
  <ThemeProvider value={{mainColor: 'green', textColor: 'red'}} >
    <Page />
  </ThemeProvider>
```

两种提供者模式实现方式的比较

通过上面的代码，可以很清楚地看到，新的 Context API 更简洁，但是，也并不是十全十美。

在老版 Context API 中，“上下文”只是一个概念，并不对应一个代码，两个组件之间达成一个协议，就诞生了“上下文”。

在新版 Context API 中，需要一个“上下文”对象（上面的例子中就是 `ThemeContext`），使用“提供者”的代码和“消费者”的代码往往分布在不同的代码文件中，那么，这个 `ThemeContext` 对象放在哪个代码文件中呢？

最好是放在一个独立的文件中，这么一来，就多出一个代码文件，而且所有和这个“上下文”相关的代码，都要依赖于这个“上下文”代码文件，虽然这没什么大不了的，但是的确多了一层依赖关系。

为了避免依赖关系复杂，每个应用都不要滥用“上下文”，应该限制“上下文”的使用个数。

不管怎么说，新版本的 Context API 才是未来，在 React v17 中，可能就会删除对老版 Context API 的支持，所以，现在大家都应该使用第二种实现方式。

小结

这一小节我们介绍了“提供者模式”，读者应该能够理解：

1. 提供者模式解决的问题；
2. React 的 Context 功能对这种模式有很直接的支持；
3. 提供者模式中 render props 的应用。

在接下来关于 Redux 和 Mobx 的介绍中，可以看到“提供者模式”更广泛的应用。

留言

评论将在后台进行审核，审核通过后对所有人可见

AboyL 前端开发者 @ 保密

这样如果我希望在A组件里面去控制兄弟B组件里面的数据，则需要在A,B组件的父组件AB组件的中使用provider，然后给A组件传入一个函数用于改变context，感觉还是很麻烦啊。。。

▲ 0 评论 20天前

xiari 高级前端工程师 @ 拼多多

16.7 进一步简化了 context 的使用， 只需要添加静态属性 contextType, 就可以在组件里使用 this.context 访问到context了。详情地址 https://reactjs.org/docs/context.html#dynamic-context

▲ 1 收起评论 25天前

jeffacode同学 前端开发 @ 不好意思无业游民

这样就好多了！看这篇文章时候就觉得v16.3之后简化下Provider端就可以了，干嘛把Consumer端搞得那么复杂，以前好好的写个组件多好，非得搞成render props的形式。。现在v16.7终于改回来了😂

21天前

webLion200 前端开发 @ 等我成了大牛再告诉你

使用consumer就不需要contextType，岂不是更好

13天前

评论审核通过后显示 评论

Artyhacker 前端 @ 北京某小国企

在实际项目中，Context文件也只是创建Context、export Provider和Consumer，不需要也不会去做别的吗？

▲ 0 评论 1月前

Artyhacker 前端 @ 北京某小国企

在实际项目中使用时，Context文件内，只需要引入React、然后createContext、最后export一个Provider和一个Consumer就好了，不需要也不会做别的了吗？

▲ 0 收起评论 1月前

程墨 Hulu

没大明白你的问题，是问需要别的库吗？

1月前 回复

评论审核通过后显示 评论

blacker

官网的例子并没有说必须要使用render props模式吧 https://reactjs.org/docs/context.html

▲ 0 收起评论 1月前

程墨 Hulu

我不确定我理解你的问题，官网上给的例子就是render props模式啊，只不过官网上不说“render props”这个术语罢了。

1月前

评论审核通过后显示 评论

红谷滩陈冠希 前端 @ 无限996公司

这个“上下文”对象 ThemeContext 有两个属性，分别就是——对，你没猜错——Provider 和 Context。 这里的 Context 应该是 Consumer 吧, typo。

▲ 0 评论 1月前