

React.js 小书

[<-- 返回首页](#)

## 动手实现 React-redux（二）：结合 context 和 store

- 作者：[胡子大哈](#)
- 原文链接：<http://huziketang.com/books/react/lesson37>
- 转载请注明出处，保留原文链接和作者信息。

（本文未审核）

既然要把 store 和 context 结合起来，我们就先构建 store。在 `src/index.js` 加入之前创建的 `createStore` 函数，并且构建一个 `themeReducer` 来生成一个 store：

```
import React, { Component, PropTypes } from 'react'
import ReactDOM from 'react-dom'
import Header from './Header'
import Content from './Content'
import './index.css'

function createStore (reducer) {
  let state = null
  const listeners = []
  const subscribe = (listener) => listeners.push(listener)
  const getState = () => state
  const dispatch = (action) => {
    state = reducer(state, action)
    listeners.forEach((listener) => listener())
  }
  dispatch({}) // 初始化 state
  return { getState, dispatch, subscribe }
}

const themeReducer = (state, action) => {
  if (!state) return {
    themeColor: 'red'
  }
  switch (action.type) {
    case 'CHANGE_COLOR':
      return { ...state, themeColor: action.themeColor }
    default:
      return state
  }
}
```

```
const store = createStore(themeReducer)
...
```

`themeReducer` 定义了一个表示主题色的状态 `themeColor`，并且规定了一种操作 `CHNAGE_COLOR`，只能通过这种操作修改颜色。现在我们把 `store` 放到 `Index` 的 `context` 里面，这样每个子组件都可以获取到 `store` 了，修改 `src/index.js` 里面的 `Index`：

```
class Index extends Component {
  static childContextTypes = {
    store: PropTypes.object
  }

  getChildContext () {
    return { store }
  }

  render () {
    return (
      <div>
        <Header />
        <Content />
      </div>
    )
  }
}
```

如果有些同学已经忘记了 `context` 的用法，可以参考之前的章节：[React.js 的 context](#)。

然后修改 `src/Header.js`，让它从 `Index` 的 `context` 里面获取 `store`，并且获取里面的 `themeColor` 状态来设置自己的颜色：

```
class Header extends Component {
  static contextTypes = {
    store: PropTypes.object
  }

  constructor () {
    super()
    this.state = { themeColor: '' }
  }

  componentWillMount () {
    this._updateThemeColor()
  }

  _updateThemeColor () {
    const { store } = this.context
    const state = store.getState()
```

```
    this.setState({ themeColor: state.themeColor })
  }

  render () {
    return (
      <h1 style={{ color: this.state.themeColor }}>React.js 小书</h1>
    )
  }
}
```

其实也很简单，我们在 `constructor` 里面初始化了组件自己的 `themeColor` 状态。然后在生命周期中 `componentWillMount` 调用 `_updateThemeColor`，`_updateThemeColor` 会从 `context` 里面把 `store` 取出来，然后通过 `store.getState()` 获取状态对象，并且用里面的 `themeColor` 字段设置组件的 `state.themeColor`。

然后在 `render` 函数里面获取了 `state.themeColor` 来设置标题的样式，页面上就会显示：



如法炮制 `Content.js`：

```
class Content extends Component {
  static contextTypes = {
    store: PropTypes.object
  }

  constructor () {
    super()
    this.state = { themeColor: '' }
  }

  componentWillMount () {
    this._updateThemeColor()
  }

  _updateThemeColor () {
    const { store } = this.context
    const state = store.getState()
    this.setState({ themeColor: state.themeColor })
  }
}
```

```
render () {  
  return (  
    <div>  
      <p style={{ color: this.state.themeColor }}>React.js 小书内容</p>  
      <ThemeSwitch />  
    </div>  
  )  
}
```

还有 `src/ThemeSwitch.js`：

```
class ThemeSwitch extends Component {  
  static contextTypes = {  
    store: PropTypes.object  
  }  
  
  constructor () {  
    super()  
    this.state = { themeColor: '' }  
  }  
  
  componentWillMount () {  
    this._updateThemeColor()  
  }  
  
  _updateThemeColor () {  
    const { store } = this.context  
    const state = store.getState()  
    this.setState({ themeColor: state.themeColor })  
  }  
  
  render () {  
    return (  
      <div>  
        <button style={{ color: this.state.themeColor }}>Red</button>  
        <button style={{ color: this.state.themeColor }}>Blue</button>  
      </div>  
    )  
  }  
}
```

这时候，主题已经完全生效了，整个页面都是红色的：



当然现在点按钮还是没什么效果，我们接下来给按钮添加事件。其实也很简单，监听 `onClick` 事件然后 `store.dispatch` 一个 `action` 就好了，修改 `src/ThemeSwitch.js`：

```
class ThemeSwitch extends Component {
  static contextTypes = {
    store: PropTypes.object
  }

  constructor () {
    super()
    this.state = { themeColor: '' }
  }

  componentWillMount () {
    this._updateThemeColor()
  }

  _updateThemeColor () {
    const { store } = this.context
    const state = store.getState()
    this.setState({ themeColor: state.themeColor })
  }

  // dispatch action 去改变颜色
  handleSwitchColor (color) {
    const { store } = this.context
    store.dispatch({
      type: 'CHANGE_COLOR',
      themeColor: color
    })
  }

  render () {
    return (
      <div>
        <button
          style={{ color: this.state.themeColor }}
          onClick={this.handleSwitchColor.bind(this, 'red')}>Red</button>
        <button
          style={{ color: this.state.themeColor }}

```

```
      onClick={this.handleSwitchColor.bind(this, 'blue')}>Blue</button>
    </div>
  )
}
}
```

我们给两个按钮都加上了 `onClick` 事件监听，并绑定到了 `handleSwitchColor` 方法上，两个按钮分别给这个方法传入不同的颜色 `red` 和 `blue`，`handleSwitchColor` 会根据传入的颜色 `store.dispatch` 一个 `action` 去修改颜色。

当然你现在点击按钮还是没有反应的。因为点击按钮的时候，只是更新 `store` 里面的 `state`，而并没有在 `store.state` 更新以后去重新渲染数据，我们其实就是忘了 `store.subscribe` 了。

给 `Header.js`、`Content.js`、`ThemeSwitch.js` 的 `componentWillMount` 生命周期都加上监听数据变化重新渲染的代码：

```
...
componentWillMount () {
  const { store } = this.context
  this._updateThemeColor()
  store.subscribe(() => this._updateThemeColor())
}
...
```

通过 `store.subscribe`，在数据变化的时候重新调用 `_updateThemeColor`，而 `_updateThemeColor` 会去 `store` 里面取最新的 `themeColor` 然后通过 `setState` 重新渲染组件，这时候组件就更新了。现在可以自由切换主题色了：



我们顺利地把 `store` 和 `context` 结合起来，这是 `Redux` 和 `React.js` 的第一次胜利会师，当然还有很多需要优化的地方。

因为第三方评论工具有问题，对本章节有任何疑问的朋友可以移步到 [React.js 小书的论坛](#) 发帖，我会回答大家的疑问。

[下一节：动手实现 React-redux（三）：connect 和 mapStateToProps](#)

[上一节：动手实现 React-redux（一）：初始化工程](#)

如果你觉得小书写得还不错，可以请胡子大哈喝杯茶 :)

赞赏

或者传播一下知识也是一个很好的选择

4 条评论，3 人参与。

★ 0



我有话说...

使用社交帐号登录

发布前请点击左边的按钮登录

最新评论



无名 • 3月31日 19:52

按照步骤来，报ThemeSwitch.js:30Uncaught TypeError: Cannot read property 'getState' of undefined错误怎么解决？

顶 • 回复 • 分享»



无名 无名 • 3月31日 19:58

已经解决了，谢谢，

顶 • 回复 • 分享»



Faire • 3月30日 09:49

请问这个react.js还在更新么

顶 • 回复 • 分享»



Mook-Livoras Faire • 3月30日 14:37

一直在更新

顶 • 回复 • 分享»

友言？