React.js 小书

<-- 返回首页

动手实现 Redux（一）：优雅地修改共享状态

- 作者：胡子大哈
- 原文链接： http://huziketang.com/books/react/lesson30

（本文未审核）

从这节起我们开始学习 Redux，一种新型的前端“架构模式”。经常和 React.js 一并提出，你要用 React.js 基本都要伴随着 Redux 和 React.js 结合的库 React-redux。

要注意的是，Redux 和 React-redux 并不是同一个东西。Redux 是一种架构模式（Flux 架构的一种变种），它不关注你到底用什么库，你可以把它应用到 React 和 Vue，甚至跟 jQuery 结合都没有问题。而 React-redux 就是把 Redux 这种架构模式和 React.js 结合起来的一个库，就是 Redux 架构在 React.js 中的体现。

如果把 Redux 的用法重新介绍一遍那么这本书的价值就不大了，我大可把官网的 Reducers、Actions、Store 的用法、API、关系重复一遍，画几个图，说两句很玄乎的话。但是这样对大家理解和使用 Redux 都没什么好处，本书初衷还是跟开头所说的一样：希望大家对问题的根源有所了解，了解这些工具到底解决什么问题，怎么解决的。

现在让我们忘掉 React.js、Redux 这些词，从一个例子的代码 + 问题开始推演。

用 `create-react-app` 新建一个项目 `make-redux`，修改 `public/index.html` 里面的 `body` 结构为：

```
  <body>
    <div id='title'></div>
    <div id='content'></div>
  </body>
```

删除 `src/index.js` 里面所有的代码，添加下面代码，代表我们应用的状态：

```
const appState = {
  title: {
    text: 'React.js 小书',
    color: 'red',
  },
  content: {
    text: 'React.js 小书内容',
```

```
    color: 'blue'
  }
}
```

我们新增几个渲染函数，它会把上面状态的数据渲染到页面上：

```
function renderApp (appState) {
  renderTitle(appState.title)
  renderContent(appState.content)
}

function renderTitle (title) {
  const titleDOM = document.getElementById('title')
  titleDOM.innerHTML = title.text
  titleDOM.style.color = title.color
}

function renderContent (content) {
  const contentDOM = document.getElementById('content')
  contentDOM.innerHTML = content.text
  contentDOM.style.color = content.color
}
```

很简单，`renderApp` 会调用 `rendeTitle` 和 `renderContent`，而这两者会把 `appState` 里面的数据通过原始的 DOM 操作更新到页面上，调用：

```
renderApp(appState)
```

你会在页面上看到：

这是一个很简单的 App，但是它存在一个重大的隐患，我们渲染数据的时候，使用的是一个共享状态 `appState`，**每个人都可以修改它**。如果我在渲染之前做了一系列其他操作：

```
loadDataFromServer()
doSomethingUnexpected()
doSomthingMore()
// ...
renderApp(appState)
```

`renderApp(appState)` 之前执行了一大堆函数操作，你根本不知道它们会对 `appState` 做什么事情，`renderApp(appState)` 的结果根本没法得到保障。一个可以被不同模块任意修改共享的数据状态就是魔鬼，一旦数据可以任意修改，**所有对共享状态的操作都是不可预料的**（某个模块 `appState.title = null` 你一点意见都没有），出现问题的时候 debug 起来就非常困难，这就是老生常谈的尽量避免全局变量。

你可能会说我去看一下它们函数的实现就知道了它们修改了什么，在我们这个例子里面还算比较简单，但是真实项目当中的函数调用和数据初始化操作非常复杂，深层次的函数调用修改了状态是很难调试的。

但不同的模块（组件）之间确实需要共享数据，这些模块（组件）还可能需要修改这些共享数据，就像上一节的“主题色”状态（`themeColor`）。这里的矛盾就是：**“模块（组件）之间需要共享数据”，和“数据可能被任意修改导致不可预料的结果”之间的矛盾**。

让我们来想办法解决这个问题，我们可以学习 React.js 团队的做法，把事情搞复杂一些，提高数据修改的门槛：模块（组件）之间可以共享数据，也可以改数据。但是我们约定，这个数据并不能直接改，你只能执行某些我允许的某些修改，而且你修改的必须**大张旗鼓**地告诉我。

我们定义一个函数，叫 `dispatch`，它专门负责数据的修改：

```
function dispatch (action) {
  switch (action.type) {
    case 'UPDATE_TITLE_TEXT':
      appState.title.text = action.text
      break
    case 'UPDATE_TITLE_COLOR':
      appState.title.color = action.color
      break
    default:
      break
  }
}
```

所有对数据的操作必须通过 `dispatch` 函数。它接受一个参数 `action`，这个 `action` 是一个普通的 JavaScript 对象，里面必须包含一个 `type` 字段来声明你到底想干什么。`dispatch` 在 `swtich` 里面会识别这个 `type` 字段，能够识别出来的操作才会执行对 `appState` 的修改。

上面的 `dispatch` 它只能识别两种操作，一种是 `UPDATE_TITLE_TEXT` 它会用 `action` 的 `text` 字段去更新 `appState.title.text`；一种是 `UPDATE_TITLE_COLOR`，它会用 `action` 的 `color` 字段去更新 `appState.title.color`。可以看到，`action` 里面除了 `type` 字段是必须的以外，其他字段都是可以自定义的。

任何的模块如果想要修改 `appState.title.text`，必须大张旗鼓地调用 `dispatch`：

```
dispatch({ type: 'UPDATE_TITLE_TEXT', text: '《React.js 小书》' }) // 修改标题文
dispatch({ type: 'UPDATE_TITLE_COLOR', color: 'blue' }) // 修改标题颜色
```

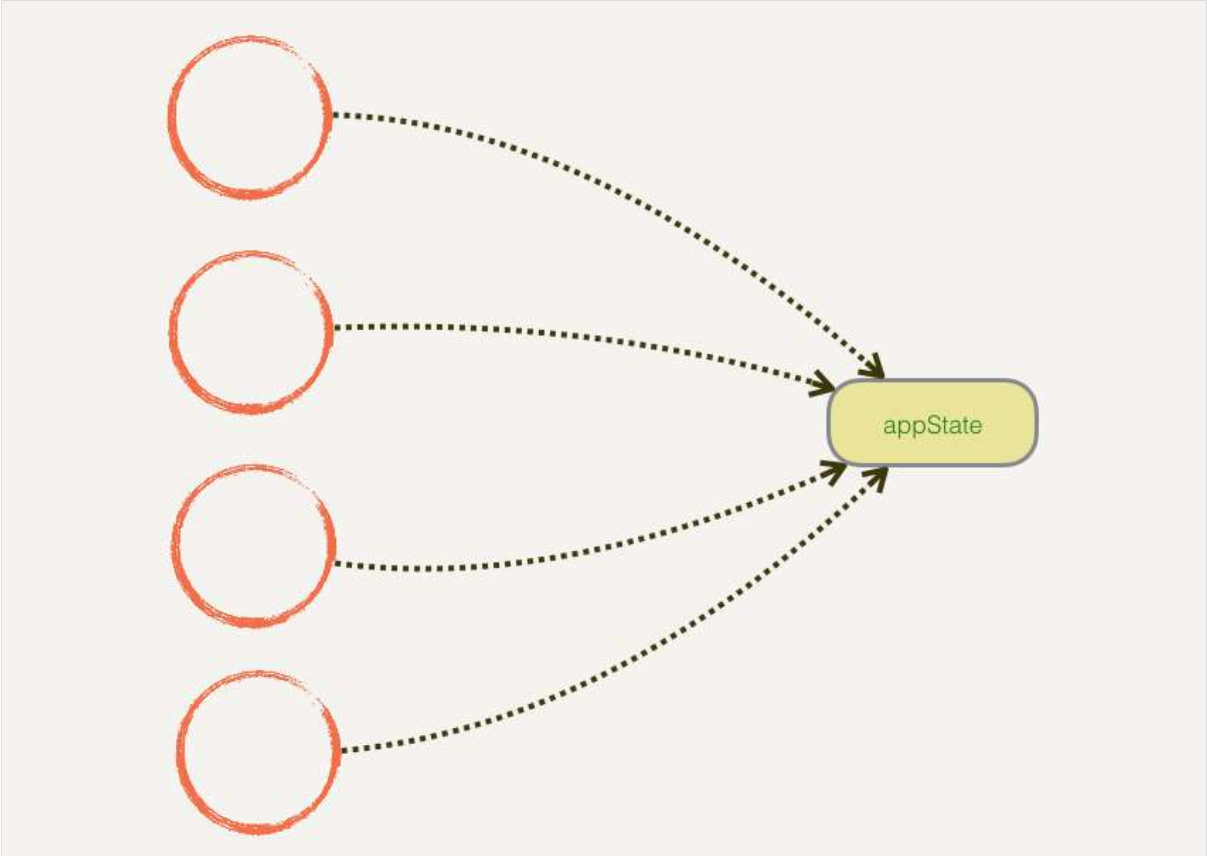
我们来看看有什么好处：

```
loadDataFromServer() // => 里面可能通过 dispatch 修改标题文本
doSomethingUnexpected()
doSomthingMore() // => 里面可能通过 dispatch 修改标题颜色
// ...
renderApp(appState)
```

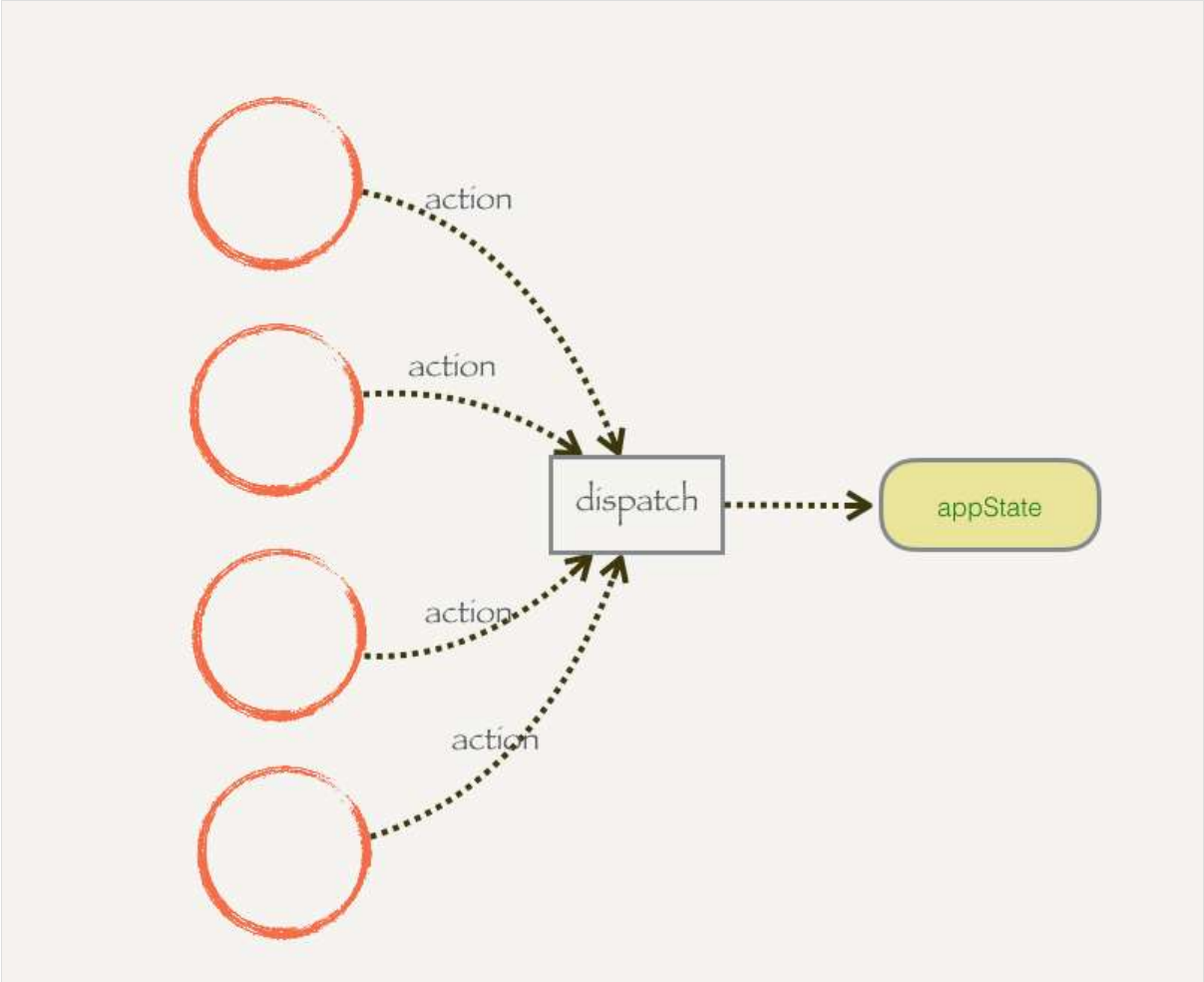
我们不需要担心 `renderApp(appState)` 之前的那堆函数操作会干什么奇奇怪怪得事情，因为我们规定不能直接修改 `appState`，它们对 `appState` 的修改必须只能通过 `dispatch`。而我们看看 `dispatch` 的实现可以知道，你只能修改 `title.text` 和 `title.color`。

如果某个函数修改了 `title.text` 但是我并不想要它这么干，我需要 debug 出来是哪个函数修改了，我只需要在 `dispatch` 的 `switch` 的第一个 `case` 内部打个断点就可以调试出来了。

原来模块（组件）修改共享数据是直接改的：



我们很难把控每一根指向 `appState` 的箭头，`appState` 里面的东西就无法把控。但现在我们必须通过一个“中间人” — `dispatch`，所有的数据修改必须通过它，并且你必须用 `action` 来大声告诉它要修改什么，只有它允许的才能修改：



我们再也不用担心共享数据状态的修改的问题，我们只要把控了 `dispatch`，所有的对 `appState` 的修改就无所遁形，毕竟只有一根箭头指向 `appState` 了。

本节完整的代码如下：

```
let appState = {
  title: {
    text: 'React.js 小书',
    color: 'red',
  },
  content: {
    text: 'React.js 小书内容',
    color: 'blue'
  }
}

function dispatch (action) {
  switch (action.type) {
    case 'UPDATE_TITLE_TEXT':
      appState.title.text = action.text
      break
    case 'UPDATE_TITLE_COLOR':
      appState.title.color = action.color
      break
    default:
      break
  }
```

```
}

function renderApp (appState) {
  renderTitle(appState.title)
  renderContent(appState.content)
}

function renderTitle (title) {
  const titleDOM = document.getElementById('title')
  titleDOM.innerHTML = title.text
  titleDOM.style.color = title.color
}

function renderContent (content) {
  const contentDOM = document.getElementById('content')
  contentDOM.innerHTML = content.text
  contentDOM.style.color = content.color
}

renderApp(appState) // 首次渲染页面
dispatch({ type: 'UPDATE_TITLE_TEXT', text: '《React.js 小书》' }) // 修改标题文
dispatch({ type: 'UPDATE_TITLE_COLOR', color: 'blue' }) // 修改标题颜色
renderApp(appState) // 把新的数据渲染到页面上
```

下一节我们会把这种 `dispatch` 的模式抽离出来，让它变得更加通用。

> 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 Redux（二）：抽离 store 和监控数据变化

上一节：React.js 的 context

3 条评论，4 人参与。

7

我有话说...

使用社交帐号登录

发布前先点击左边的按钮登录

最新评论

时光荏苒~ • 7月1日 10:54

厉害了

顶 • 回复 • 分享»

ぅ|く瓶ㄖㄅ • 4月26日 20:02

写的很棒

顶 • 回复 • 分享»

时间被海绵吃了 • 4月17日 23:15

赞赞赞

顶 • 回复 • 分享»

友言?