React.js 小书

<-- 返回首页

动手实现 Redux（二）：抽离 store 和监控数据变化

- 作者：胡子大哈
- 原文链接： http://huziketang.com/books/react/lesson31

（本文未审核）

抽离出 store

上一节 的我们有了 `appState` 和 `dispatch`：

```
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
  }
}
```

现在我们把它们集中到一个地方，给这个地方起个名字叫做 `store`，然后构建一个函数 `createStore`，用来专门生产这种 `state` 和 `dispatch` 的集合，这样别的 App 也可以用这种模式了：

```
function createStore (state, stateChanger) {
  const getState = () => state
```

```
  const dispatch = (action) => stateChanger(state, action)
  return { getState, dispatch }
}
```

`createStore` 接受两个参数，一个是表示应用程序状态的 `state`；另外一个是 `stateChanger`，它来描述应用程序状态会根据 action 发生什么变化，其实就是相当于本节开头的 `dispatch` 代码里面的内容。

`createStore` 会返回一个对象，这个对象包含两个方法 `getState` 和 `dispatch`。`getState` 用于获取 `state` 数据，其实就是简单地把 `state` 参数返回。

`dispatch` 用于修改数据，和以前一样会接受 `action`，然后它会把 `state` 和 `action` 一并传给 `stateChanger`，那么 `stateChanger` 就可以根据 `action` 来修改 `state` 了。

现在有了 `createStore`，我们可以这么修改原来的代码，保留原来所有的渲染函数不变，修改数据生成的方式：

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

function stateChanger (state, action) {
  switch (action.type) {
    case 'UPDATE_TITLE_TEXT':
      state.title.text = action.text
      break
    case 'UPDATE_TITLE_COLOR':
      state.title.color = action.color
      break
    default:
      break
  }
}

const store = createStore(appState, stateChanger)

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

针对每个不同的 App，我们可以给 `createStore` 传入初始的数据 `appState`，和一个描述数据变化的函数 `stateChanger`，然后生成一个 `store`。需要修改数据的时候通过 `store.dispatch`，需要获取数据的时候通过 `store.getState`。

监控数据变化

上面的代码有一个问题，我们每次通过 `dispatch` 修改数据的时候，其实只是数据发生了变化，如果我们不手动调用 `renderApp`，页面上的内容是不会发生变化的。但是我们总不能每次 `dispatch` 的时候都手动调用一下 `renderApp`，我们肯定希望数据变化的时候程序能够智能一点地自动重新渲染数据，而不是手动调用。

你说这好办，往 `dispatch` 里面加 `renderApp` 就好了，但是这样 `createStore` 就不够通用了。我们希望用一种通用的方式“监听”数据变化，然后重新渲染页面，这里要用到观察者模式。修改 `createStore`：

```
function createStore (state, stateChanger) {
  const listeners = []
  const subscribe = (listener) => listeners.push(listener)
  const getState = () => state
  const dispatch = (action) => {
    stateChanger(state, action)
    listeners.forEach((listener) => listener())
  }
  return { getState, dispatch, subscribe }
}
```

我们在 `createStore` 里面定义了一个数组 `listeners`，还有一个新的方法 `subscribe`，可以通过 `store.subscribe(listener)` 的方式给 `subscribe` 传入一个监听函数，这个函数会被 `push` 到数组当中。

我们修改了 `dispatch`，每次当它被调用的时候，除了会调用 `stateChanger` 进行数据的修改，还会遍历 `listeners` 数组里面的函数，然后一个个地去调用。相当于我们可以通过 `subscribe` 传入数据变化的监听函数，每当 `dispatch` 的时候，监听函数就会被调用，这样我们就可以在每当数据变化时候进行重新渲染：

```
const store = createStore(appState, stateChanger)
store.subscribe(() => renderApp(store.getState()))

renderApp(store.getState()) // 首次渲染页面
store.dispatch({ type: 'UPDATE_TITLE_TEXT', text: '《React.js 小书》' }) // 修改
store.dispatch({ type: 'UPDATE_TITLE_COLOR', color: 'blue' }) // 修改标题颜色
// ...后面不管如何 store.dispatch，都不需要重新调用 renderApp
```

> 对观察者模式不熟悉的朋友可能会在这里晕头转向，建议了解一下这个设计模式的相关资料，然后进行练习： 实现一个 EventEmitter 再进行阅读。

我们只需要 `subscribe` 一次，后面不管如何 `dispatch` 进行修改数据，`renderApp` 函数都会被重新调用，页面就会被重新渲染。这样的订阅模式还有好处就是，以后我们还可以拿同一块数据来渲染别的页面，这时 `dispatch` 导致的变化也会让每个页面都重新渲染：

```
const store = createStore(appState, stateChanger)
store.subscribe(() => renderApp(store.getState()))
store.subscribe(() => renderApp2(store.getState()))
store.subscribe(() => renderApp3(store.getState()))
...
```

本节的完整代码：

```
function createStore (state, stateChanger) {
  const listeners = []
  const subscribe = (listener) => listeners.push(listener)
  const getState = () => state
  const dispatch = (action) => {
    stateChanger(state, action)
    listeners.forEach((listener) => listener())
  }
  return { getState, dispatch, subscribe }
}

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
}

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

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

const store = createStore(appState, stateChanger)
store.subscribe(() => renderApp(store.getState())) // 监听数据变化

renderApp(store.getState()) // 首次渲染页面
store.dispatch({ type: 'UPDATE_TITLE_TEXT', text: '《React.js 小书》' }) // 修改
store.dispatch({ type: 'UPDATE_TITLE_COLOR', color: 'blue' }) // 修改标题颜色
```

总结

现在我们有了一个比较通用的 `createStore`，它可以产生一种我们新定义的数据类型 `store`，通过 `store.getState` 我们获取共享状态，而且我们约定只能通过 `store.dispatch` 修改共享状态。`store` 也允许我们通过 `store.subscribe` 监听数据数据状态被修改了，并且进行后续的例如重新渲染页面的操作。

> 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（三）：纯函数（Pure Function）简介

上一节：动手实现 Redux（一）：优雅地修改共享状态

2 条评论，1 人参与。

0

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

Mockingjay • 4月25日 16:11

listeners.forEach((listener) => listener()) 这个地方为什么是listener(),es6函数写法不是很了解，还有第二次dispatch的时候，listeners数组里有几个元素

顶 • 回复 • 分享»

Mockingjay Mockingjay • 4月25日 16:40

已经想明白了

顶 • 回复 • 分享»

友言?