React.js 小书

<-- 返回首页

实战分析：评论功能（九）

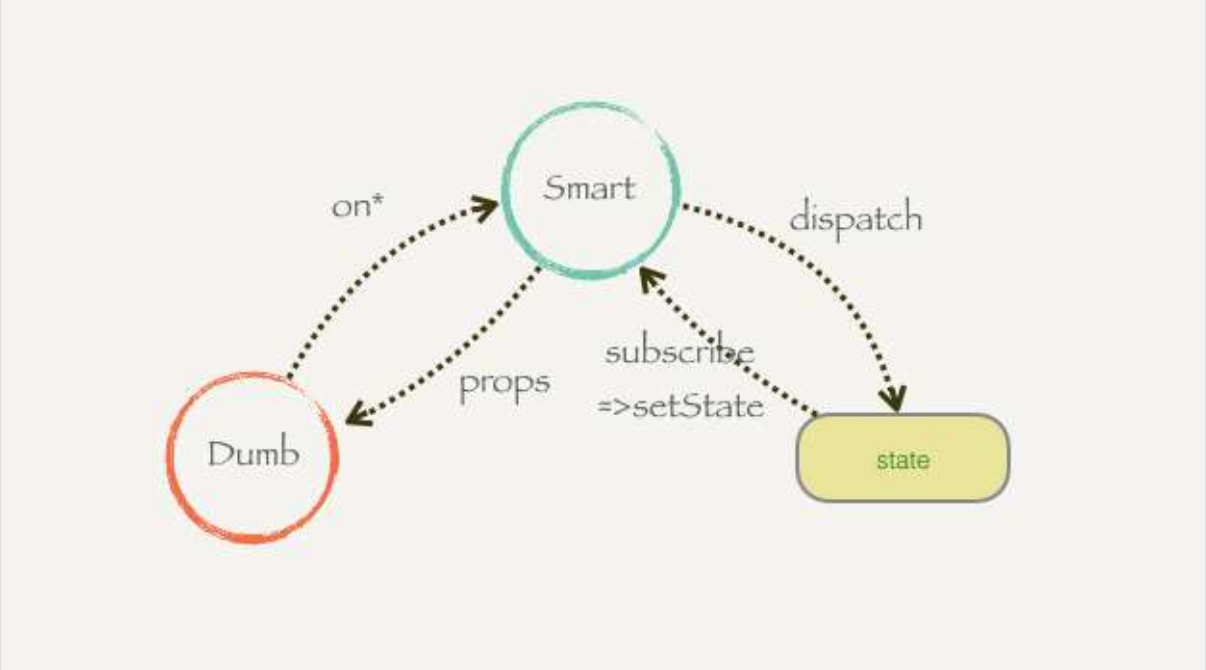
- 作者：胡子大哈
- 原文链接： http://huziketang.com/books/react/lesson46

- 转载请注明出处，保留原文链接和作者信息。

（本文未审核）

现在我们有三个 Dumb 组件，一个控制评论的 reducer。我们还缺什么？需要有人去 LocalStorage 加载数据，去控制新增、删除评论，去把数据保存到 LocalStorage 里面。之前这些逻辑我们都是零散地放在各个组件里面的（主要是 `CommentApp` 组件），那是因为当时我们还没对 Dumb 和 Smart 组件类型划分的认知，状态和视图之间也没有这么泾渭分明。

而现在我们知道，这些逻辑是应该放在 Smart 组件里面的：



了解 MVC、MVP 架构模式的同学应该可以类比过去，Dumb 组件就是 View（负责渲染），Smart 组件就是 Controller（Presenter），State 其实就有点类似 Model。其实不能完全类比过去，它们还是有不少差别的。但是本质上兜兜转转还是把东西分成了三层，所以说前端很喜欢炒别人早就玩烂的概念，这话果然不假。废话不多说，我们现在就把这些应用逻辑抽离到 Smart 组件里面。

Smart CommentList

对于 `CommentList` 组件，可以看到它接受两个参数：`comments` 和 `onDeleteComment`。说明需要一个 Smart 组件来负责把 `comments` 数据传给它，并且还得响应它删除评论的请求。我们新建一个 Smart 组件 `src/containers/CommentList.js` 来干这些事情：

```
import React, { Component, PropTypes } from 'react'
import { connect } from 'react-redux'
import CommentList from '../components/CommentList'
import { initComments, deleteComment } from '../reducers/comments'

// CommentListContainer
// 一个 Smart 组件，负责评论列表数据的加载、初始化、删除评论
// 沟通 CommentList 和 state
class CommentListContainer extends Component {
  static propTypes = {
    comments: PropTypes.array,
    initComments: PropTypes.func,
    onDeleteComment: PropTypes.func
  }

  componentWillMount () {
    // componentWillMount 生命周期中初始化评论
    this._loadComments()
  }

  _loadComments () {
    // 从 LocalStorage 中加载评论
    let comments = localStorage.getItem('comments')
    comments = comments ? JSON.parse(comments) : []
    // this.props.initComments 是 connect 传进来的
    // 可以帮我们把数据初始化到 state 里面去
    this.props.initComments(comments)
  }

  handleDeleteComment (index) {
    const { comments } = this.props
    // props 是不能变的，所以这里新建一个删除了特定下标的评论列表
    const newComments = [
      ...comments.slice(0, index),
      ...comments.slice(index + 1)
    ]
    // 保存最新的评论列表到 LocalStorage
    localStorage.setItem('comments', JSON.stringify(newComments))
    if (this.props.onDeleteComment) {
      // this.props.onDeleteComment 是 connect 传进来的
      // 会 dispatch 一个 action 去删除评论
      this.props.onDeleteComment(index)
    }
  }

  render () {
    return (
      <CommentList
```

```
        comments={this.props.comments}
        onDeleteComment={this.handleDeleteComment.bind(this)} />
    )
  }
}

// 评论列表从 state.comments 中获取
const mapStateToProps = (state) => {
  return {
    comments: state.comments
  }
}

const mapDispatchToProps = (dispatch) => {
  return {
    // 提供给 CommentListContainer
    // 当从 LocalStorage 加载评论列表以后就会通过这个方法
    // 把评论列表初始化到 state 当中
    initComments: (comments) => {
      dispatch(initComments(comments))
    },
    // 删除评论
    onDeleteComment: (commentIndex) => {
      dispatch(deleteComment(commentIndex))
    }
  }
}

// 将 CommentListContainer connect 到 store
// 会把 comments、initComments、onDeleteComment 传给 CommentListContainer
export default connect(
  mapStateToProps,
  mapDispatchToProps
)(CommentListContainer)
```

代码有点长，大家通过注释应该了解这个组件的基本逻辑。有一点要额外说明的是，我们一开始传给 `CommentListContainer` 的 `props.comments` 其实是 reducer 里面初始化的空的 `comments` 数组，因为还没有从 LocalStorage 里面取数据。

而 `CommentListContainer` 内部从 LocalStorage 加载 `comments` 数据，然后调用 `this.props.initComments(comments)` 会导致 `dispatch`，从而使得真正从 LocalStorage 加载的 `comments` 初始化到 state 里面去。

因为 `dispatch` 了导致 `connect` 里面的 `Connect` 包装组件去 state 里面取最新的 `comments` 然后重新渲染，这时候 `CommentListContainer` 才获得了有数据的 `props.comments`。

这里的逻辑有点绕，大家可以回顾一下我们之前实现的 `react-redux.js` 来体会一下。

Smart CommentInput

对于 `CommentInput` 组件，我们可以看到它有三个参数：`username`、`onSubmit`、`onUserNameInputBlur`。我们需要一个 Smart 的组件来管理用户名在 LocalStorage 的加载、保存；用户还可能点击“发布”按钮，所以还需要处理评论发布的逻辑。我们新建一个 Smart 组件 `src/containers/CommentInput.js` 来干这些事情：

```
import React, { Component, PropTypes } from 'react'
import { connect } from 'react-redux'
import CommentInput from '../components/CommentInput'
import { addComment } from '../reducers/comments'

// CommentInputContainer
// 负责用户名的加载、保存，评论的发布
class CommentInputContainer extends Component {
  static propTypes = {
    comments: PropTypes.array,
    onSubmit: PropTypes.func
  }

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

  componentWillMount () {
    // componentWillMount 生命周期中初始化用户名
    this._loadUsername()
  }

  _loadUsername () {
    // 从 LocalStorage 加载 username
    // 然后可以在 render 方法中传给 CommentInput
    const username = localStorage.getItem('username')
    if (username) {
      this.setState({ username })
    }
  }

  _saveUsername (username) {
    // 看看 render 方法的 onUserNameInputBlur
    // 这个方法会在用户名输入框 blur 的时候的被调用，保存用户名
    localStorage.setItem('username', username)
  }

  handleSubmitComment (comment) {
    // 评论数据的验证
    if (!comment) return
    if (!comment.username) return alert('请输入用户名')
    if (!comment.content) return alert('请输入评论内容')
    // 新增评论保存到 LocalStorage 中
    const { comments } = this.props
    const newComments = [...comments, comment]
```

```
    localStorage.setItem('comments', JSON.stringify(newComments))
    // this.props.onSubmit 是 connect 传进来的
    // 会 dispatch 一个 action 去新增评论
    if (this.props.onSubmit) {
      this.props.onSubmit(comment)
    }
  }

  render () {
    return (
      <CommentInput
        username={this.state.username}
        onUserNameInputBlur={this._saveUsername.bind(this)}
        onSubmit={this.handleSubmitComment.bind(this)} />
    )
  }
}

const mapStateToProps = (state) => {
  return {
    comments: state.comments
  }
}

const mapDispatchToProps = (dispatch) => {
  return {
    onSubmit: (comment) => {
      dispatch(addComment(comment))
    }
  }
}

export default connect(
  mapStateToProps,
  mapDispatchToProps
)(CommentInputContainer)
```

同样地，对代码的解释都放在了注释当中。这样就构建了一个 Smart 的 `CommentInput`。

Smart CommentApp

接下来的事情都是很简单，我们用 `CommentApp` 把这两个 Smart 的组件组合起来，把 `src/CommentApp.js` 移动到 `src/containers/CommentApp.js`，把里面的内容替换为：

```
import React, { Component } from 'react'
import CommentInput from './CommentInput'
import CommentList from './CommentList'

export default class CommentApp extends Component {
  render() {
    return (
```

```
      <div className='wrapper'>
        <CommentInput />
        <CommentList />
      </div>
    )
  }
}
```

原本很复杂的 `CommentApp` 现在变得异常简单，因为它的逻辑都分离到了两个 Smart 组件里面去了。原来的 `CommentApp` 确实承载了太多它不应该承担的责任。分离这些逻辑对我们代码的维护和管理也会带来好处。

最后一步，修改 `src/index.js`：

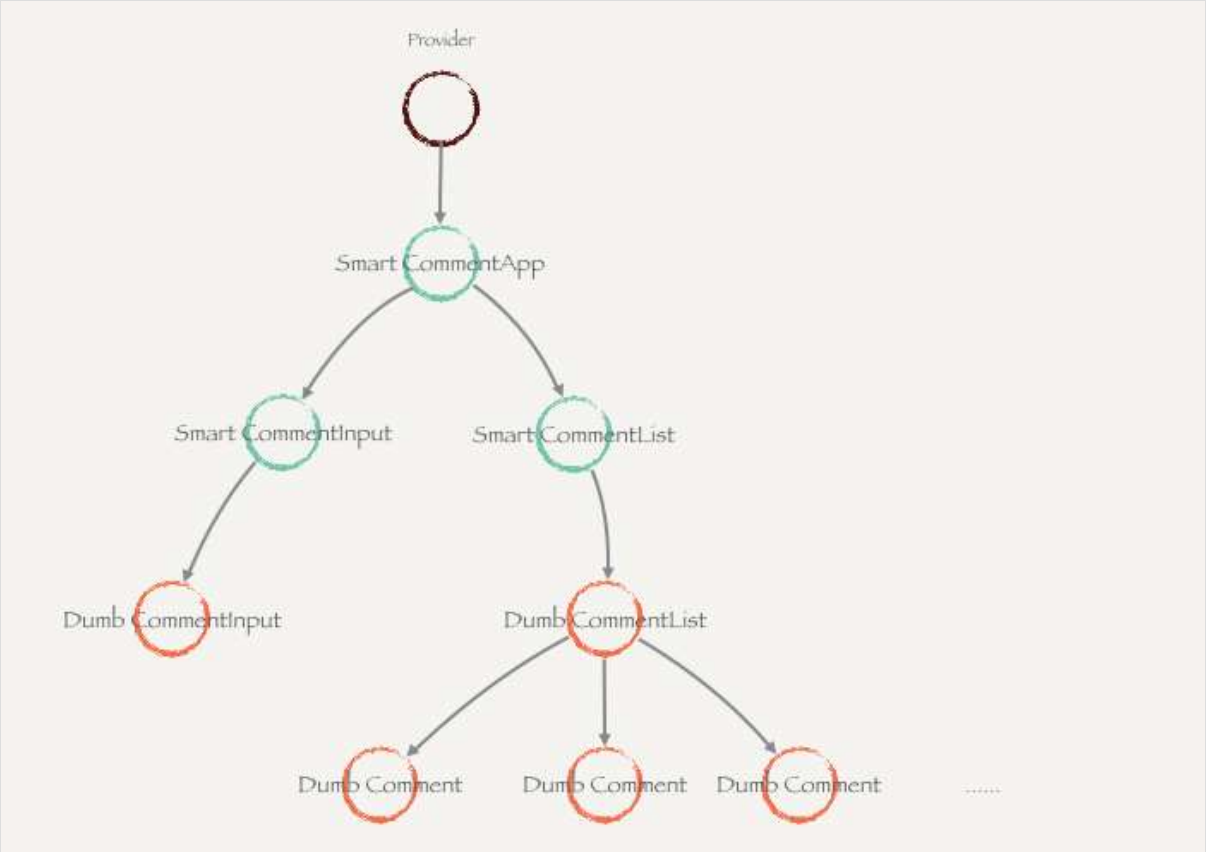
```
import React from 'react'
import ReactDOM from 'react-dom'
import { createStore } from 'redux'
import { Provider } from 'react-redux'
import CommentApp from './containers/CommentApp'
import commentsReducer from './reducers/comments'
import './index.css'

const store = createStore(commentsReducer)

ReactDOM.render(
  <Provider store={store}>
    <CommentApp />
  </Provider>,
  document.getElementById('root')
);
```

通过 `commentsReducer` 构建一个 `store`，然后让 `Provider` 把它传递下去，这样我们就完成了最后的重构。

我们最后的组件树是这样的：



文件目录：

```
src
├── components
│   ├── Comment.js
│   ├── CommentInput.js
│   └── CommentList.js
├── containers
│   ├── CommentApp.js
│   ├── CommentInput.js
│   └── CommentList.js
│  reducers
│     └── comments.js
├── index.css
└── index.js
```

所有代码可以在这里找到： comment-app3。

> 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。

上一节：实战分析：评论功能（八）

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

6 条评论，6 人参与。

5

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

晨思的海 • 7月6日 11:47

棒棒的，非常希望作者继续努力出其他小书～～

顶 • 回复 • 分享»

undef_0_0 • 6月22日 16:26

完结撒花，写的太好啦~觉得可以再出一个教程，侧重于实战，写个稍微大点的应用~

顶 • 回复 • 分享»

天堂鸟 • 4月24日 18:55

我要爆粗口了，我太TM喜欢这个教程了，真心希望出个结合react-router(v4)，react-redux的最佳实践

顶 • 回复 • 分享»

这个世界需要更多的Superman • 4月18日 10:30

希望把这个评论加上ajax从服务端取数据也能读localstroge里面的数据做个例子

顶 • 回复 • 分享»

安字符 • 4月17日 20:24

希望能加入react-router的动手实现

顶 • 回复 • 分享»

似梦飞花 • 4月11日 16:07

坐等更新 写的非常好

顶 • 回复 • 分享»

友言?