React.js 小书

<-- 返回首页

实战分析：评论功能（七）

- 作者：胡子大哈
- 原文链接： http://huziketang.com/books/react/lesson44
- 转载请注明出处，保留原文链接和作者信息。

（本文未审核）

从本节开始，我们开始用 Redux、React-redux 来重构第二阶段的评论功能。产品需求跟之前一样，但是会用 Redux、React-redux 来帮助管理应用状态，而不是“状态提升”。让整个应用更加接近真实的工程。

大家可以在第二阶段的代码上进行修改 comment-app2（非高阶组件版本）。如果已经忘了第二阶段评论功能的同学可以先简单回顾一下它的功能需求，实战分析：评论功能（四）。第一、二、三阶段的实战代码都可以在这里找到：react-naive-book-examples。

我们首先安装好依赖，现在 comment-app2 需要依赖 Redux、React-redux 了，进入工程目录执行命令安装依赖：

```
npm install redux react-redux --save
```

然后我们二话不说先在 `src` 下建立三个空目录：`components`、`containers`、`reducers`。

构建评论的 reducer

我们之前的 reducer 都是直接写在 `src/index.js` 文件里面，这是一个不好的做法。因为随着应用越来越复杂，可能需要更多的 reducer 来帮助我们管理应用（这里后面的章节会有所提及）。所以最好还是把所有 reducer 抽出来放在一个目录下 `src/reducers`。

对于评论功能其实还是比较简单的，回顾一下我们在状态提升章节里面不断提升的状态是什么？其实评论功能的组件之间共享的状态只有 `comments`。我们可以直接只在 `src/reducers` 新建一个 reducer `comments.js` 来对它进行管理。

思考一下评论功能对于评论有什么操作？想清楚我们才能写好 reducer，因为 reducer 就是用来描述数据的形态和相应的变更。**新增**和**删除**评论这两个操作是最明显的，大家应该都能够轻易想到。还有一个，我们的评论功能其实会从 LocalStorage

读取数据，读取数据以后其实需要保存到应用状态中。所以我们还有一个**初始化**评论的操作。所以目前能想到的就是三个操作：

```
// action types
const INIT_COMMENTS = 'INIT_COMMENTS'
const ADD_COMMENT = 'ADD_COMMENT'
const DELETE_COMMENT = 'DELETE_COMMENT'
```

我们用三个常量来存储 `action.type` 的类型，这样以后我们修改起来就会更方便一些。根据这三个操作编写 reducer：

```
// reducer
export default function (state, action) {
  if (!state) {
    state = { comments: [] }
  }
  switch (action.type) {
    case INIT_COMMENTS:
      // 初始化评论
      return { comments: action.comments }
    case ADD_COMMENT:
      // 新增评论
      return {
        comments: [...state.comments, action.comment]
      }
    case DELETE_COMMENT:
      // 删除评论
      return {
        comments: [
          ...state.comments.slice(0, action.commentIndex),
          ...state.comments.slice(action.commentIndex + 1)
        ]
      }
    default:
      return state
  }
}
```

我们只存储了一个 `comments` 的状态，初始化为空数组。当遇到 `INIT_COMMENTS` 的 action 的时候，会新建一个对象，然后用 `action.comments` 覆盖里面的 `comments` 属性。这就是初始化评论操作。

同样新建评论操作 `ADD_COMMENT` 也会新建一个对象，然后新建一个数组，接着把原来 `state.comments` 里面的内容全部拷贝到新的数组当中，最后在新的数组后面追加 `action.comment`。这样就相当新的数组会比原来的多一条评论。（这里不要担心数组拷贝的性能问题，`[...state.comments]` 是浅拷贝，它们拷贝的都是对象引用而已。）

对于删除评论，其实我们需要做的是**新建一个删除了特定下标的内容的数组**。我们知道数组 `slice(from, to)` 会根据你传进去的下标拷贝特定范围的内容放到新数组里面。所以我们可以利用 `slice` 把原来评论数组中 `action.commentIndex` 下标之前的内容拷贝到一个数组当中，把 `action.commentIndex` 坐标之后到内容拷贝到另外一个数组当中。然后把两个数组合并起，就相当于“删除”了 `action.commentIndex` 的评论了。

这样就写好了评论相关的 reducer。

action creators

之前我们使用 `dispatch` 的时候，都是直接手动构建对象：

```
dispatch({ type: 'INIT_COMMENTS', comments })
```

每次都要写 `type` 其实挺麻烦的，而且还要去记忆 action type 的名字也是一种负担。我们可以把 action 封装到一种函数里面，让它们去帮助我们去构建这种 action，我们把它叫做 action creators。

```
// action creators
export const initComments = (comments) => {
  return { type: INIT_COMMENTS, comments }
}

export const addComment = (comment) => {
  return { type: ADD_COMMENT, comment }
}

export const deleteComment = (commentIndex) => {
  return { type: DELETE_COMMENT, commentIndex }
}
```

所谓 action creators 其实就是返回 action 的函数，这样我们 `dispatch` 的时候只需要传入数据就可以了：

```
dispatch(initComments(comments))
```

action creators 还有额外好处就是可以帮助我们对传入的数据做统一的处理；而且有了 action creators，代码测试起来会更方便一些。这些内容大家可以后续在实际项目当中进行体会。

整个 `src/reducers/comments.js` 的代码就是：

```
// action types
const INIT_COMMENTS = 'INIT_COMMENTS'
const ADD_COMMENT = 'ADD_COMMENT'
```

```
const DELETE_COMMENT = 'DELETE_COMMENT'

// reducer
export default function (state, action) {
  if (!state) {
    state = { comments: [] }
  }
  switch (action.type) {
    case INIT_COMMENTS:
      // 初始化评论
      return { comments: action.comments }
    case ADD_COMMENT:
      // 新增评论
      return {
        comments: [...state.comments, action.comment]
      }
    case DELETE_COMMENT:
      // 删除评论
      return {
        comments: [
          ...state.comments.slice(0, action.commentIndex),
          ...state.comments.slice(action.commentIndex + 1)
        ]
      }
    default:
      return state
  }
}

// action creators
export const initComments = (comments) => {
  return { type: INIT_COMMENTS, comments }
}

export const addComment = (comment) => {
  return { type: ADD_COMMENT, comment }
}

export const deleteComment = (commentIndex) => {
  return { type: DELETE_COMMENT, commentIndex }
}
```

> 有些朋友可能会发现我们的 reducer 跟网上其他的 reducer 的例子不大一样。有些人喜欢把 action 单独切出去一个目录 `actions`，让 action 和 reducer 分开。个人观点觉得这种做法可能有点过度优化了，其实多数情况下特定的 action 只会影响特定的 reducer，直接放到一起可以更加清晰地知道这个 action 其实只是会影响到什么样的 reducer。而分开会给我们维护和理解代码带来额外不必要的负担，这有种矫枉过正的意味。但是这里没有放之四海皆准的规则，大家可以多参考、多尝试，找到适合项目需求的方案。

个人写 reducer 文件的习惯，仅供参考：

1. 定义 action types
2. 编写 reducer

3. 跟这个 reducer 相关的 action creators

> 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：实战分析：评论功能（八）

上一节：Smart 组件 vs Dumb 组件