kanban-app

1. `src/components/KanbanBoard.js`

   > 新建

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


class KanbanBoard extends Component {
  // 参数验证
  static propTypes = {
    cards: PropTypes.arrayOf(PropTypes.object).isRequired
  }
  render () {
    console.log(this.props.cards)
    return (
      <div className="app">
        <List id="todo" title="To Do" cards={ this.props.cards.filter(
(card) => card.status === 'todo' ) } />
        <List id="in-progress" title="In Pro" cards={
this.props.cards.filter( (card) => card.status === 'in-progress' ) } />
        <List id="done" title="Done" cards={ this.props.cards.filter(
(card) => card.status === 'done' ) } />
      </div>
    )
  }
}

export default KanbanBoard
```

2. `src/components/List.js`

   > 列表渲染 使用key

```
import React, { Component } from 'react'
import Card from './Card.js'

class List extends Component {
  render () {
    // 列表渲染
    let cards = this.props.cards.map( (card) => {
      return <Card id={ card.id } title={ card.title } description={
card.description } color={ card.color } tasks={ card.tasks } key={
card.id } />
    } )
    return (
      <div className="list">
        <h1>
          { this.props.title }
        </h1>
        { cards }
```

```
      </div>
    )
  }
}


export default List
```

3. `src/components/Card.js`

```
import React, { Component } from 'react'
import PropTypes from 'prop-types'
// 使用Markdown
import marked from 'marked'

import CheckList from './CheckList'

// 自定义的校验器 自定义参数验证
let titlePropType = (props, propName, componentName) => {
  if (props[propName]) {
    let value = props[propName]
    if (typeof value !== 'string' || value.length > 80) {
      return new Error(
        `${propName} in ${componentName} is longer then 80 characters`
      )
    }
  }
}

// 使用自定义的参数验证
class Card extends Component {
  static propTypes = {
    id: PropTypes.number,
    // 自定义的校验
    title: titlePropType,
    description: PropTypes.string,
    color: PropTypes.string,
    tasks: PropTypes.arrayOf(PropTypes.object)
  }
  constructor () {
    super()
    this.state = {
      showDetails: false
    }
  }
  render () {
    let cardDetails
    if (this.state.showDetails) {
      cardDetails = (
        {/* 渲染HTML */}
        <div className="card__details">
          {/* html 渲染 xss */}
          <span dangerouslySetInnerHTML={ {
            __html: marked(this.props.description)
          } } />
          <CheckList cardId={ this.props.id } tasks={ this.props.tasks
} />
```

```
        </div>
      )
    }

    let sideColor = {
      position: 'absolute',
      zIndex: -1,
      top: 0,
      bottom: 0,
      left: 0,
      width: 7,
      backgroundColor: this.props.color
    }

    return (
      <div className="card">
        <div style={ sideColor } />
        <div className={ this.state.showDetails? "card__title
card__title--is-open" : "card__title" } onClick={ () => {
          // 快速的设置 setState
          this.setState({
            showDetails: !this.state.showDetails
          })
        } }>
          { this.props.title }
        </div>
        { cardDetails }
      </div>
    )
  }
}

export default Card
```

4. `src/components/CheckList.js`

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

class Card extends Component {

  render () {
    return (
      <div className="card">
        <div className="card__title">
          { this.props.title }
        </div>
        <div className="card__details">
          { this.props.description }
          <CheckList cardId={ this.props.id } tasks={ this.props.tasks
} />
        </div>
      </div>
    )
  }
}

```

```
21 export default Card
```

5. `src/components/CheckList.js`

> 列表渲染

```
1  import React, { Component } from 'react'
2
3  class CheckList extends Component {
4    render () {
5      let tasks = this.props.tasks.map((task, index) => {
6        return (
7          <li className="checklist__task" key={ index }>
8            <input type="checkbox" defaultChecked={ task.done } />
9            { task.name }
10           <i className="checklist__task--remove" />
11         </li>
12       )
13     })
14     return (
15       <div className="checklist">
16         <ul>
17           { tasks }
18         </ul>
19       </div>
20     )
21   }
22 }
23
24
25 export default CheckList
```

1 使用whatwg-fetch 获取服务器数据

`src/components/KanbanBoardContainer.js` 容器组件

```
1  import React, { Component } from 'react'
2  import 'whatwg-fetch'
3  import update from 'react-addons-update'
4
5  import KanbanBoard from './KanbanBoard'
6
7
8  const API_URL = 'http://kanbanapi.pro-react.com/'
9  const API_HEADERS = {
10   'Content-Type': 'application/json',
11   Authorization: 'magicwingzs@gmail'
12 }
13
14 class KanbanBoardContainer extends Component {
15   constructor () {
16     super()
17     this.state = {
18       cards: []
```

```
    }
  }

  componentWillMount () {
    fetch(`${API_URL}/cards`,{
      headers: API_HEADERS
    })
    .then((response) => response.json())
    .then((responseData) =>{
      this.setState({
        cards: responseData
      })
    })
    .catch(error => console.log('获取数据出错'))
  }

  // 添加任务
  addTask (cardId, taskName) {
    let cardIndex = this.state.cards.findIndex(card => card.id === cardId)

    let newTask = {
      id: Date.now(),
      name: taskName,
      done: false
    }

    let newCards = update(this.state.cards, {
      [cardIndex]: {
        tasks: {
          $push: [newTask]
        }
      }
    })

    this.setState({
      cards: newCards
    })

    fetch(`${API_URL}/cards/${cardId}/tasks`,{
      method: 'post',
      headers: API_HEADERS,
      body: JSON.stringify(newTask)
    })
    .then((response) => response.json())
    .then((responseData) =>{
      newTask.id = responseData.id
      this.setState({
        cards: newCards
      })
    })
    .catch(error => new Error('添加任务出错'))
  }

  // 删除任务
  deleteTask (cardId, taskId, taskIndex) {
    console.log(...arguments)
    let cardIndex = this.state.cards.findIndex(card => card.id === cardId)
    let newCards = update(this.state.cards, {
```

```
      [cardIndex]: {
        tasks: { $splice: [[taskIndex, 1]] }
      }
    })
    this.setState({
      cards: newCards
    })

    fetch(`${API_URL}/cards/${cardId}/tasks/${taskId}`, {
      method: 'delete',
      headers: API_HEADERS
    })
  }

  // 切换任务状态
  toggleTask (cardId, taskId, taskIndex) {
    let cardIndex = this.state.cards.findIndex(card => card.id === cardId)
    let newDonValue
    let newCards = update(this.state.cards, {
      [cardIndex]: {
        tasks: {
          [taskIndex]: {
            done: {
              $apply: (done) => {
                newDonValue = !done
                return newDonValue
              }
            }
          }
        }
      }
    })

    this.setState({
      cards: newCards
    })

    fetch(`${API_URL}/cards/${cardId}/tasks/${taskId}`, {
      method: 'put',
      headers: API_HEADERS,
      body: JSON.stringify({
        done: newDonValue
      })
    })
  }


  render () {
    return (
      <KanbanBoard cards={ this.state.cards } taskCallbacks={{
        add: this.addTask.bind(this),
        delete: this.deleteTask.bind(this),
        toggle: this.toggleTask.bind(this)
      }} />
    )
  }
}

```

```
135
136 export default KanbanBoardContainer
```

2 卡片的拖拽功能

> 实现卡片的拖放功能，需要把卡片做成可排序的，不仅可以在列表之间拖动卡片，还可以在同一个列表中交换它和其它卡片的顺序

2.1 安装 React DND2 和 HTML5后端

```
1 npm i react-dnd react-dnd-html5-backend -S
```

- 创建一个常量文件

  `utils/constants.js`

```
1 export default {
2   CARD: 'card'
3 }
```

2.1.1 跨列表拖拽

> 需要使用React Dnd 的高阶组件来设置

1. 拖拽源

   > DragSource 其实就是Card组件

2. 放置目标

   > DropTarget是List组件

3. 上下文

   > 是KanbanBoard组件

2.1.1.1 实现拖拽源 DragSource 的 Card 组件

`Card.js`

```
1  import React, { Component } from 'react'
2  import PropTypes from 'prop-types'
3  // ******* 【Drag_1】使用拖拽  *******
4  import { DragSource } from 'react-dnd'
5  import constants from '../utils/constants'
6
7
8  //  ******* 【Drag_2 书写Spec对象 】  *******
9  /*
10   描述了增强组件是如何响应拖拽和放置事件的，他是一个包含了若干函数的普通JavaScript对象，这些甘薯会在拖拽交互发生时被调用
11   对于DragSource 有
12   1. beginDrag
13   2. endDrag
14 */
15 const cardDragSpec = {
16   beginDrag (props) {
```

```
    return {
      id: props.id
    }
  }
}


// ******* 【Drag_3 书写collectDrag 函数对象 】 *******
/**
 * 通过collect 函数来控制哪些属性需要进行注入以及如何进行注入
 * 可以在注入前对属性进行预处理，改变其名称
 * @param {*} connect
 * @param {*} monitor
 */
let collectDrag = (connect, monitor) => {
  return {
    // 拖拽源头
    connectDragSource: connect.dragSource()
  }
}


class Card extends Component {
  static propTypes = {
    // ******* 【Drag_4 collectDrag中返回的属性会注入到props中 】 *******
    connectDragSource: PropTypes.func.isRequired
  }
  constructor () {
    super()
    this.state = {
      showDetails: false
    }
  }
  render () {
    //  ******* 【Drag_5 获取高阶组价connectDragSource 】  *******
    const { connectDragSource } = this.props
    ....
    return (
      // ******* 【Drag_6 使用高阶组件高阶组价connectDragSource 】 *******
      connectDragSource (
        <div className="card">
        ....
        </div>
      )
    )
  }
}

// ******* 【Drag_7 传入 type值(用来唯一标识), spec对象(响应拖拽事件), collect(函数 用来做属性的注入) 】 *******
export default DragSource(constants.CARD, cardDragSpec, collectDrag)(Card)
```

2.1.1.2 实现拖拽目标组件

·List.js

```
import React, { Component } from 'react'
```

```
import PropTypes from 'prop-types'
// ******* 【Drag_1】使用拖拽  作为拖拽目标  *******
import { DropTarget } from 'react-dnd'

// 引入常量文件
import constants from '../utils/constants'


// ******* 【Drag_2】 编写scpec对象  *******
const listTargetSpec = {
  hover(props, monitor) {
    // 获取拖拽源的id
    const draggedId = monitor.getItem().id
    // 执行更新卡片的状态的函数 从prop函数中取到父组件传递下来的更新函数
    props.cardCallbacks.updateStatus(draggedId, props.id)
  }
}

// ******* 【Drag_3】 编写collect函数  *******
let collect = (connect, monitor) => {
  return {
    connectDropTarget: connect.dropTarget()
  }

}

class List extends Component {
  static propTypes =  {
    // ******* 【Drag_4】 验证 connectDropTarget  *******
    connectDropTarget: PropTypes.func.isRequired
  }
  render () {
    // ******* 【Drag_5】 从props中获取connectDropTarget  *******
    const { connectDropTarget } = this.props

    return (
      // ******* 【Drag_6】 使用connectDropTarget高阶组件  *******
      connectDropTarget (
        <div className="list">
        ...
        </div>
      )
    )
  }
}

// ******* 【Drag_7】 使用使用高阶组件DropTarget  *******
export default DropTarget(constants.CARD, listTargetSpec, collect)(List)
```

2.1.1.3 实现拖拽上下文组件

> 其实就是Card 和 List 公共的父级组件，将其作为拖放上下文

```
import React, { Component } from 'react'
import PropTypes from 'prop-types'
// ******* 【Drag_1】使用拖拽  作为拖拽上下文对象  *******
import { DndProvider } from 'react-dnd'
```

```
// ******* 【Drag_2】使用HTML5后端  *******
import HTML5Backend from 'react-dnd-html5-backend'

import List from './List'

class KanbanBoard extends Component {
  static propTypes = {
    cards: PropTypes.arrayOf(PropTypes.object).isRequired,
    taskCallbacks: PropTypes.objectOf(PropTypes.func).isRequired,
    cardCallbacks: PropTypes.objectOf(PropTypes.func).isRequired,
  }
  render () {
    return (
    // ******* 【Drag_3】使用 DndProvider 对根元素进行包装 传入后端参数  ****
      <DndProvider backend={ HTML5Backend }>
        <div className="app">
        ....
        </div>
      </DndProvider>
    )
  }
}


export default KanbanBoard
```

2.1.2 同列表(卡片排序)

> 使用React Dnd 的关键在于同时把

3 快速调用父组件传递下来的函数

`KanbanBoardContainer.js`

```
// 有三个函数

addTask () {
....
}

deleteTask () {
...
}

toggleTask () {
  ...
}

  render () {
    return (
      // 向子组件传递
      <KanbanBoard cards={ this.state.cards } taskCallbacks={{
        add: this.addTask.bind(this),
        delete: this.deleteTask.bind(this),
        toggle: this.toggleTask.bind(this)
      }} />
```

```
    )
  }
```

`CheckList.js`

```
// 子组件接收
static propTypes = {
    taskCallbacks: PropTypes.objectOf(PropTypes.func).isRequired
  }

// 列表渲染 进行调用
let tasks = this.props.tasks.map((task, index) => {
      return (
        <li className="checklist__task" key={ task.id }>
          <input type="checkbox" checked={ task.done }
            // 调用传递下来的函数
            onChange={ this.props.taskCallbacks.toggle.bind(null,
this.props.cardId, task.id, index) }
          />
          { task.name }
          {/* 删除任务 */}
          <i className="checklist__task--remove"
                        // 快速调用 传递参数
            onClick={ this.props.taskCallbacks.delete.bind(null,
this.props.cardId, task.id, index) }
          />
        </li>
      )
    })
```