---

link: null
title: 珠峰架构师成长计划
description: null
keywords: null
author: null
date: null
publisher: 珠峰架构师成长计划
stats: paragraph=385 sentences=802, words=5662

1.拖拽应用场景

- 拖拽的业务很常见，比如拖拽菜单 、拖拽卡片排序和拖拽评分等

2.React DnD

- react-dnd (https://www.npmjs.com/package/react-dnd)是React和Redux核心作者 Dan创造的一组React实用程序，可帮助您构建复杂的拖放界面
- react-dnd官网 (http://react-dnd.github.io/react-dnd/)

3.核心知识

3.1 HTML拖放API

- HTML 拖放（Drag and Drop） (https://developer.mozilla.org/zh-CN/docs/Web/API/HTML_Drag_and_Drop_API)接口使应用程序能够在浏览器中使用拖放功能。例如，用户可使用鼠标选择可拖拽（draggable）元素，将元素拖拽到可放置（droppable）元素，并释放鼠标按钮以放置这些元素。拖拽操作期间，会有一个可拖拽元素的半透明快照跟随着鼠标指针

事件 触发时刻 dragstart 当用户开始拖拽一个元素时触发 dragover 当元素被拖到一个可释放目标上时触发(每100毫秒触发一次) dragend 当拖拽操作结束时触发

3.2 触摸事件

- HTML拖放API不支持触摸事件，所以它不适用于平板电脑和移动设备
- 触摸事件(Touch_events) (https://developer.mozilla.org/zh-CN/docs/Web/API/Touch_events)可为特定程序提供多点触控交互的支持

事件 触发时刻 touchstart 触摸开始 touchmove 接触点移动 touchend 触摸结束

3.3 getBoundingClientRect

- getBoundingClientRect (https://developer.mozilla.org/zh-CN/docs/Web/API/Element/getBoundingClientRect)方法返回元素的大小及其相对于视口的位置

3.4 clientX/Y

- event.clientX 鼠标相对于浏览器左上角x轴的坐标； 不随滚动条滚动而改变
- event.clientY 鼠标相对于浏览器左上角y轴的坐标； 不随滚动条滚动而改变

3.5 React Hooks

- `useState` 通过在函数组件里调用它来给组件添加一些内部 state,React 会在重复渲染时保留这个 state
- `useCallback` 把内联回调函数及依赖项数组作为参数传入 useCallback，它将返回该回调函数的 memoized 版本，该回调函数仅在某个依赖项改变时才会更新
- `useMemo` 把创建函数和依赖项数组作为参数传入 useMemo，它仅会在某个依赖项改变时才重新计算 memoized 值
- `useLayoutEffect` 给函数组件增加了操作副作用的能力

3.6 高阶组件

- 高阶组件就是一个函数，传给它一个老组件，它返回一个新的组件
- 高阶组件的作用其实就是为了组件之间的代码复用

```
const NewComponent = higherOrderComponent(OldComponent)
```

4. 核心概念

4.1 DndProvider

- DndProvider 组件为您的应用程序提供 React-DnD 功能
- 使用时必须通过 `backend`属性注入一个后端
    - backend: 必需属性 React DnD 后端

4.2 backend(后端)

- React DnD 使用 HTML5 拖放 API。它会截取拖动的 DOM 节点并将其用作开箱即用的 `&#x62D6;&#x52A8;&#x9884;&#x89C8;`。 当光标移动时，您不必进行任何绘图，这很方便
- 后端抽象出浏览器差异并处理原生 DOM 事件,并将 DOM 事件转换为 React DnD 可以处理的内部 Redux 操作
- react-dnd-html5-backend (https://react-dnd.github.io/react-dnd/docs/backends/html5)是 React-DnD 支持的主要后端,它本质上是使用 HTML5 拖放 API
- HTML5 后端不支持触摸事件。 所以它不适用于平板电脑和移动设备。 您可以使用react-dnd-touch-backend (https://react-dnd.github.io/react-dnd/docs/backends/touch)

4.3 DragDropManager(管理器)和Registry(注册器)

- 管理整个拖拽应用
- 包含仓库和全局monitor
- 全局monitor里包含registry(注册器)
- 注册器里包含了handlerId和对应的DragSource或DropTarget的对应关系
- 每一个项目都会有一个唯一的handlerId

4.4 Items and Types(项目和类型)

- 与Redux一样，React DnD 使用数据而不是视图作为来源
- 当您在屏幕上拖动某物时，我们并不是说正在拖动组件或 DOM 节点。 相反，我们说某种类型的项目正在被拖动
- 什么是项目？ 一个项目是一个简单的 JavaScript 对象，描述被拖动的内容。 例如，在看板应用程序中，当您拖动卡片时，项目可能看起来像是 `{type:'card', id: 'card1' }`,将拖动的数据描述为普通对象可以帮助您保持组件解耦
- 那什么是type(类型)呢？ 类型是一个字符串(或Symbol)，它唯一地标识了应用程序中的项目。 在看板应用程序中，您可能有一个代表可拖动卡片的 `&#x5361;&#x7247;`类型
- 类型很有用，因为随着您的应用程序的增长，您可能希望使更多内容可拖动，但您不一定希望所有现有的放置目标突然开始对新项目做出反应。 这些类型允许您指定兼容的拖放源和放置目标

4.5 Monitors(监听器)

- 拖放本质上是有状态的。 要么正在进行拖动操作，要么不在。 要么有当前类型和当前项目，要么没有。 这种状态必须存在于某个地方
- React DnD 通过监视器封装内部存储状态,并将此状态公开给您的组件
- 监视器允许您更新组件的属性以响应拖放状态的变化
- 对于需要跟踪拖放状态的每个组件，您可以定义一个收集函数，从监视器获取它的相关属性
- React DnD 然后负责及时调用您的收集函数并将其返回值合并到组件的属性对象中

4.5.1 DragSourceMonitor(拖动源监听器)

- `DragSourceMonitor` 是传递给基于钩子或基于装饰器的拖动源的收集函数的对象
- 它的方法可让您获取有关特定拖动源的拖动状态的信息
- 绑定到该监视器的特定拖动源在下面称为监视器的所有者
- 方法
    - isDragging(): 如果拖动操作正在进行中，并且所有者发起拖动，则返回 true
    - getItemType(): 返回标识当前拖动项目类型的字符串或Symbol
    - getItem(): 返回一个表示当前拖动项目的普通对象。 每个拖动源都必须通过从其 `beginDrag()` 方法返回一个对象来指定它
    - getClientOffset(): 在拖动操作正在进行时，返回最后记录的 { x, y } 鼠标的客户端偏移量

4.5.2 DropTargetMonitor(放置目标监听器)

- DropTargetMonitor 是传递给基于钩子或基于装饰器的收集函数的对象
- 它的方法可让您获取有关特定放置目标的拖动状态的信息
- 绑定到该监视器的特定放置目标在下面称为监视器的所有者
- 方法
    - getItemType(): 返回标识当前拖动项目类型的字符串或Symbol
    - getItem(): 返回一个表示当前拖动项目的普通对象。 每个拖动源都必须通过从其 beginDrag() 方法返回一个对象来指定它
    - getClientOffset(): 在拖动操作正在进行时，返回最后记录的 { x, y } 鼠标的客户端偏移量

4.6 Connectors(连接器)

- 如果后端处理 DOM 事件，但是组件使用 React 来描述 DOM，后端如何知道要监听哪些 DOM 节点？
- 连接器允许您将预定义角色之一（拖动源或放置目标）分配给渲染函数中的 DOM 节点
- 事实上，连接器作为第一个参数传递给我们上面描述的收集函数
- 在组件的渲染方法中，我们可以访问从监视器获得的数据和从连接器获得的函数

4.6.1 DragSourceConnector(拖动源连接器)

- DragSourceConnector 是传递给 DragSource 的收集函数的对象
- 它提供了将 React 组件绑定到 Drag Source 角色的能力
- 属性
    - dragSource 返回一个必须被 prop-injected 到你的组件中并在该组件的 render() 方法中使用的函数。 您可以将此函数传递给此方法一个 react 组件、一个 DOM 元素或一个 ref 对象

4.6.2 DragTargetConnector(放置目标连接器)

- DropTargetConnector 是传递给 DropTarget 的收集函数的对象。 它提供了将 React 组件绑定到 Drop Target 角色的能力
- 属性
    - dropTarget() => (Element | Node | Ref) 返回一个必须被 prop-injected 到你的组件中并在该组件的 render() 方法中使用的函数。 您可以将此函数传递给此方法一个 react 组件、一个 DOM 元素或一个 ref 对象

4.7 DragSources(拖动源)和DropTargets(放置目标)

- 上面我们已经介绍了与 DOM 一起工作的后端、由项目和类型表示的数据以及收集功能，依靠监视器和连接器，这些功能让您可以描述 React DnD 应该注入到您的组件中的属性对象中
- 但是我们如何配置我们的组件来实际注入这些属性对象中呢？ 我们如何执行响应拖放事件的副作用？ React DnD 的主要抽象单元拖动源和放置目标可以将类型、项目、副作用和收集功能与您的组件联系在一起
- 每当您想让一个组件或其某些部分可拖动时，您需要将该组件包装到拖动源声明中。 每个拖动源都针对特定类型注册，并且必须实现一个方法，从组件的 props 生成一个项目。 拖动源声明还允许您为给定组件指定收集功能
- 放置目标与拖动源非常相似。 唯一的区别是单个放置目标可以同时注册多个项目类型

4.7.1 DragSource

- 参数
    - type: 必需的。 一个字符串、一个符号或一个函数，返回给定组件的属性。 只有为相同类型注册的放置目标才会对此拖动源生成的项目做出反应
    - spec: 必需的。 一个简单的 JavaScript 对象，上面有一些允许的方法。 它描述了拖动源如何对拖放事件做出反应
        - beginDrag(props, monitor, component): 必需的。 当拖动开始时，beginDrag被调用。 您必须返回一个描述被拖动数据的普通 JavaScript 对象。 您返回的是有关拖放源的放置目标可用的唯一信息，因此选择他们需要知道的最少数据很重要
    - collect: 必需的。 收集功能。 它应该返回一个普通的属性对象以注入到您的组件中。 它接收两个参数：connect 和 monitor

4.7.2 DropTarget

- 参数
    - types: 必需的。 给定组件的 props，一个字符串、一个符号、一个数组或一个返回其中任何一个的函数。 这个放置目标只会对指定类型的拖拽源产生的项目做出反应
    - spec: 必需的。 一个简单的 JavaScript 对象，上面有一些允许的方法。 它描述了放置目标如何对拖放事件做出反应
        - hover(props, monitor, component): 可选的。 当项目悬停在组件上时调用
    - collect: 必需的。 收集功能。 它应该返回一个普通的属性对象以注入到您的组件中。 它接收两个参数：connect 和 monitor

4.8 useDrag

- useDrag hook 提供了一种将组件作为拖动源连接到 DnD 系统的方法
- 通过将规范传入 useDrag，您可以声明性地描述正在生成的可拖动的类型、表示拖动源的项目对象、要收集的属性等
- useDrag hooks 返回几个关键项：收集的属性，以及可能附加到拖动源的 `refs`
- 参数
    - spec 规范对象或创建规范对象的函数
        - type: 必需的。 这必须是字符串或Symbol。 只有为相同类型注册的放置目标才会对此项目做出反应
        - item: 必需的 (对象或者函数) 当这是一个对象时，它是一个描述被拖动数据的普通 JavaScript 对象。 这是拖放目标唯一可用的有关拖动源的信息。当这是一个函数时，它在拖动操作开始时被触发，并返回一个表示拖动操作的对象
        - collect: 可选的。 收集功能。 它应该返回一个普通的属性对象，以返回以注入到您的组件属性中。 它接收两个参数，monitor 和 props
- 返回值数组
    - [0] - Collected Props: 包含从 collect 函数收集的属性的对象。 如果没有定义 collect 函数，则返回一个空对象
    - [1] - DragSource Ref: 拖动源的连接器功能。 这必须附加到 DOM 的可拖动部分

4.9 useDrop

- useDrop hook (https://react-dnd.github.io/react-dnd/docs/api/use-drop) 为您提供了一种将组件连接到 DnD 系统作为放置目标的方法
- 通过将规范传入 useDrop hook，您可以指定放置目标将接受哪些类型的数据项，要收集哪些属性，等等
- 此函数返回一个要附加到 Drop Target 节点的ref和收集到的属性
- 参数
    - specA 规范对象或创建规范对象的函数
        - accept 必填项 一个字符串或一个Symbol,这个放置目标只会对指定类型的拖拽源产生的项目做出反应
        - collect: 可选的。收集功能。 它应该返回一个普通的属性对象，以返回以注入到您的组件属性中。 它接收两个参数，monitor 和 props
        - hover(item, monitor): 可选的。 当在组件发生hover事件时调用
- 返回值

- Collected Props: 包含从 collect 函数收集的属性的对象。 如果没有定义collect function，则返回一个空对象
- DropTarget Ref: 放置目标的连接器函数。 这必须附加到 DOM 的放置目标部分

5. 拖拽排序实战

5.1 安装

```
npm install react-dnd react-dnd-html5-backend --save
```

5.2 绘制容器

5.2.1 src\index.js

src\index.js

```
import React from 'react';
import ReactDOM from 'react-dom';
import Container from './Container';
import { DndProvider } from 'react-dnd';
import { HTML5Backend } from 'react-dnd-html5-backend';

ReactDOM.render(
  <DndProvider backend={HTML5Backend}>
    <Container />
  DndProvider>,
  document.getElementById('root')
);
```

5.2.2 Container.js

src\Container.js

```
import React from 'react'
const style = {
    width: '300px'
}
function Container() {
    return (
        <div style={style}>
            Container
        div>
    )
}

export default Container;
```

5.3 绘制卡片

5.3.1 src\Card.js

src\Card.js

```
import React from 'react'
const style = {
    backgroundColor: 'red',
    padding: '5px',
    margin: '5px',
    border: '1px dashed gray',
    cursor: 'move'
}
export default function Card({ text }) {
    return (
        <div style={style}>
            {text}
        div>
    )
}
```

5.3.2 src\Container.js

```
+import React, { useState } from 'react'
+import Card from './Card';
const style = {
    width: '300px'
}
function Container() {
+   const [cards, setCards] = useState([
+       { id: 'card1', text: '卡片1' },
+       { id: 'card2', text: '卡片2' },
+       { id: 'card3', text: '卡片3' }
+   ]);
    return (

+           {
+               cards.map((card, index) => (
+
+               ))
+           }

    )
}
export default Container;
```

5.4 拖动卡片

5.4.1 ItemTypes.js

src\ItemTypes.js

```
export const CARD = 'card';
```

5.4.2 Container.js

src\Container.js

```
import React, { useState } from 'react'
import Card from './Card';
const style = {
    width: '200px'
}
function Container() {
    const [cards, setCards] = useState([
        { id: 'card1', text: '卡片1' },
        { id: 'card2', text: '卡片2' },
        { id: 'card3', text: '卡片3' }
    ]);
    return (

            {
                cards.map((card, index) => (
+
                ))
            }

    )
}

export default Container;
```

5.4.3 Card.js

src\Card.js

```
import React, { useRef } from 'react'
+import { useDrag } from 'react-dnd';
+import { CARD } from './ItemTypes';
const style = {
    backgroundColor: 'red',
    padding: '5px',
    margin: '5px',
    border: '1px dashed gray',
    cursor: 'move'
}
export default function Card({ text, id, index }) {
+   const ref = useRef(null);
+   //useDrag hook 提供了一种将组件作为拖动源连接到 DnD 系统的方法
+   //Collected Props: 包含从 collect 函数收集的属性的对象。 如果没有定义 collect 函数，则返回一个空对象
+   //DragSource Ref: 拖动源的连接器功能。 这必须附加到 DOM 的可拖动部分
+   const [{ isDragging }, drag] = useDrag({//spec
+       //必需的。 这必须是字符串或Symbol。 只有为相同类型注册的放置目标才会对此项目做出反应
+       type: CARD,
+       //item: 必需的 (对象或者函数) 当这是一个对象时，它是一个描述被拖动数据的普通 JavaScript 对象
+       item: () => ({ id, index }),
+       //collect:  收集功能。 它应该返回一个普通的属性对象，以返回以注入到您的组件属性中
+       //它接收两个参数，monitor 和 props
+       collect: (monitor) => ({//要收集的属性
+           isDragging: monitor.isDragging()
+       })
+   });
+   const opacity = isDragging ? 0.1 : 1;
+   drag(ref)
    return (
+
            {text}

    )
}
```

5.5 放置卡片

5.5.1 Container.js

src\Container.js

```
import React, { useState } from 'react'
import Card from './Card';
const style = {
    width: '200px'
}
function Container() {
    const [cards, setCards] = useState([
        { id: 'card1', text: '卡片1' },
        { id: 'card2', text: '卡片2' },
        { id: 'card3', text: '卡片3' }
    ]);
+   const moveCard = (dragIndex, hoverIndex) => {
+       const dragCard = cards[dragIndex];
+       const cloneCards = [...cards];
+       cloneCards.splice(dragIndex, 1);
+       cloneCards.splice(hoverIndex, 0, dragCard);
+       setCards(cloneCards);
+   }
    return (

            {
                cards.map((card, index) => (
+
                ))
            }

    )
}
export default Container;
```

5.5.2 src\Card.js

src\Card.js

```
import React, { useRef } from 'react'
+import { useDrag, useDrop } from 'react-dnd';
import { CARD } from './ItemTypes';
const style = {
    backgroundColor: 'red',
    padding: '5px',
    margin: '5px',
    border: '1px dashed gray',
    cursor: 'move'
}
export default function Card({ text, id, index, moveCard }) {
    const ref = useRef(null);
+   //useDrop hook 为您提供了一种将组件连接到 DnD 系统作为放置目标的方法
+   // Collected Props: 包含从 collect 函数收集的属性的对象
+   //DropTarget Ref: 放置目标的连接器函数。 这必须附加到 DOM 的放置目标部分
+   const [, drop] = useDrop({
+       //一个字符串或一个Symbol,这个放置目标只会对指定类型的拖拽源产生的项目做出反应
+       accept: CARD,
+       //收集功能。 它应该返回一个普通的属性对象，以返回以注入到您的组件属性中
+       collect: (monitor) => ({}),
+       //当在组件发生hover事件时调用
+       hover(item, monitor) {
+           //被拖动卡片的索引
+           const dragIndex = item.index;
+           //hover卡片的索引
+           const hoverIndex = index;
+           //如果一样什么都不做
+           if (dragIndex === hoverIndex) {
+               return;
+           }
+           //获取hover卡片的位置信息
+           const { top, bottom } = ref.current.getBoundingClientRect();
+           //获取hover卡片高度的一半
+           const halfOfHoverHeight = (bottom - top) / 2;
+           //获取鼠标最新的x和y坐标
+           const { y } = monitor.getClientOffset();
+           const hoverClientY = y - top;
+           if ((dragIndex < hoverIndex && hoverClientY > halfOfHoverHeight)
+               || (dragIndex > hoverIndex && hoverClientY < halfOfHoverHeight)) {
+               moveCard(dragIndex, hoverIndex);
+               item.index = hoverIndex;
+           }
+       }
+   });
    //useDrag hook 提供了一种将组件作为拖动源连接到 DnD 系统的方法
    //Collected Props: 包含从 collect 函数收集的属性的对象。 如果没有定义 collect 函数，则返回一个空对象
    //DragSource Ref: 拖动源的连接器功能。 这必须附加到 DOM 的可拖动部分
    const [{ isDragging }, drag] = useDrag({//spec
        //必需的。 这必须是字符串或Symbol。 只有为相同类型注册的放置目标才会对此项目做出反应
        type: CARD,
        //item: 必需的 (对象或者函数) 当这是一个对象时，它是一个描述被拖动数据的普通 JavaScript 对象
        item: () => ({ id, index }),
        //collect:  收集功能。 它应该返回一个普通的属性对象，以返回以注入到您的组件属性中
        //它接收两个参数，monitor 和 props
        collect: (monitor) => ({//要收集的属性
            isDragging: monitor.isDragging()
        })
    });
    const opacity = isDragging ? 0.1 : 1;
    drag(ref)
+   drop(ref)
    return (

            {text}

    )
}
```

5.实现Provider

5.1 react-dnd\index.js

src\react-dnd\index.js

```
export * from './core'
```

5.2 core\index.js

src\react-dnd\core\index.js

```
export { default as DndContext } from './DndContext';
export { default as DndProvider } from './DndProvider';
```

5.3 DndContext.js

src\react-dnd\core\DndContext.js

```
import React from 'react';
const DndContext = React.createContext({});
export default DndContext;
```

5.4 DndProvider.js

src\react-dnd\core\DndProvider.js

```
import DndContext from './DndContext';
import { createDragDropManager } from '../../dnd-core';
function DndProvider({ backend, children }) {
    let value = { dragDropManager: createDragDropManager(backend) };
    return (
        <DndContext.Provider value={value}>
            {children}
        DndContext.Provider>
    )
}
export default DndProvider;
```

5.5 dnd-core\index.js

src\dnd-core\index.js

```
export { default as createDragDropManager } from './createDragDropManager';
```

5.6 createDragDropManager.js

src\dnd-core\createDragDropManager.js

```
import { createStore } from 'redux';
import reducer from './reducers';
import DragDropManagerImpl from './classes/DragDropManagerImpl';
function createDragDropManager(backendFactory) {

    const store = createStore(reducer);
    const manager = new DragDropManagerImpl(store);
    const backend = backendFactory(manager);
    manager.receiveBackend(backend);
    return manager;
}
export default createDragDropManager;
```

5.7 DragDropManagerImpl.js

src\dnd-core\classes\DragDropManagerImpl.js

```
class DragDropManagerImpl {
    store;
    backend;
    constructor(store) {
        this.store = store;
    }
    receiveBackend(backend) {
        this.backend = backend;
    }
    getBackend() {
        return this.backend;
    }
}
export default DragDropManagerImpl;
```

5.8 reducers\index.js

src\dnd-core\reducers\index.js

```
function reducer(state = {}, action) {
    return {};
}
export default reducer;
```

5.9 react-dnd-html5-backend\index.js

src\react-dnd-html5-backend\index.js

```
import HTML5BackendImpl from './HTML5BackendImpl';
export function HTML5Backend(manager) {
    return new HTML5BackendImpl(manager);
}
```

5.10 HTML5BackendImpl.js

src\react-dnd-html5-backend\HTML5BackendImpl.js

```
class HTML5BackendImpl {}
export default HTML5BackendImpl;
```

6.实现Monitor

- 拖放本质上是有状态的。 要么正在进行拖动操作，要么不在。 要么有当前类型和当前项目，要么没有。 这种状态必须存在于某个地方
- React DnD 通过监视器封装内部存储状态,并将此状态公开给您的组件
- 监视器允许您更新组件的属性以响应拖放状态的变化
- 对于需要跟踪拖放状态的每个组件，您可以定义一个收集函数，从监视器获取它的相关属性
- React DnD 然后负责及时调用您的收集函数并将其返回值合并到组件的属性对象中

6.1 react-dnd\index.js

src\react-dnd\index.js

```
export * from './core'
+export * from './hooks';
```

6.2 hooks\index.js

src\react-dnd\hooks\index.js

```
export { default as useDrag } from './useDrag';
```

6.3 useDrag\index.js

src\react-dnd\hooks\useDrag\index.js

```
import useDragSourceMonitor from './useDragSourceMonitor';

function useDrag(spec) {
    const monitor = useDragSourceMonitor();
    console.log(monitor);
    return [{}, () => { }];
}
export default useDrag;
```

6.4 useDragSourceMonitor.js

src\react-dnd\hooks\useDrag\useDragSourceMonitor.js

```
import { useMemo } from 'react';
import useDragDropManager from '../useDragDropManager';
import { DragSourceMonitorImpl } from '../../internals'
function useDragSourceMonitor() {
    const manager = useDragDropManager();
    return useMemo(() => new DragSourceMonitorImpl(manager), [manager]);
}
export default useDragSourceMonitor;
```

6.5 useDragDropManager.js

src\react-dnd\hooks\useDragDropManager.js

```
import { useContext } from 'react';
import { DndContext } from '../core';
function useDragDropManager() {
    const { dragDropManager } = useContext(DndContext);
    return dragDropManager;
}
export default useDragDropManager;
```

6.6 internals\index.js

src\react-dnd\internals\index.js

```
export { default as DragSourceMonitorImpl } from './DragSourceMonitorImpl';
```

6.7 DragSourceMonitorImpl.js

src\react-dnd\internals\DragSourceMonitorImpl.js

```
class DragSourceMonitorImpl {
    internalMonitor
    constructor(manager) {
        this.internalMonitor = manager.getGlobalMonitor();
    }
}
export default DragSourceMonitorImpl;
```

6.8 dnd-core\index.js

src\dnd-core\index.js

```
export { default as createDragDropManager } from './createDragDropManager';
export { default as DragDropManagerImpl } from './classes/DragDropManagerImpl';
+export { default as DragDropMonitorImpl } from './classes/DragDropMonitorImpl';
```

6.9 createDragDropManager.js

src\dnd-core\createDragDropManager.js

```
import { createStore } from 'redux';
import reducer from './reducers';
import DragDropManagerImpl from './classes/DragDropManagerImpl';
+import DragDropMonitorImpl from './classes/DragDropMonitorImpl';
function createDragDropManager(backendFactory) {
    //创建redux仓库
    const store = createStore(reducer);
+   const globalMonitor = new DragDropMonitorImpl(store);
+   const manager = new DragDropManagerImpl(store, globalMonitor);
    const backend = backendFactory(manager);
    manager.receiveBackend(backend);
    return manager;
}
export default createDragDropManager;
```

6.10 DragDropMonitorImpl.js

src\dnd-core\classes\DragDropMonitorImpl.js

```
class DragDropMonitorImpl {
    store
    constructor(store) {
        this.store = store
    }
}
export default DragDropMonitorImpl;
```

6.11 DragDropManagerImpl.js

src\dnd-core\classes\DragDropManagerImpl.js

```
class DragDropManagerImpl {
    store;
    backend;
+   globalMonitor;
+   constructor(store, globalMonitor) {
        this.store = store;
+       this.globalMonitor = globalMonitor;
    }
    receiveBackend(backend) {
        this.backend = backend;
    }
    getBackend() {
        return this.backend;
    }
+   getGlobalMonitor() {
+       return this.globalMonitor;
+   }
}
export default DragDropManagerImpl;
```

7.实现Registry

- 每一个拖动源和放置目标都会有一个唯一标识handlerId,生成规则为S或T加一个递增数字
- Registry注册表里会存放handlerId和拖动源和放置目标的对应关系

7.1 HandlerRegistryImpl.js

src\dnd-core\classes\HandlerRegistryImpl.js

```
class HandlerRegistryImpl {
    store;
    constructor(store) {
        this.store = store;
    }
}
export default HandlerRegistryImpl;
```

7.2 createDragDropManager.js

src\dnd-core\createDragDropManager.js

```
import { createStore } from 'redux';
import reducer from './reducers';
+import HandlerRegistryImpl from './classes/HandlerRegistryImpl'
import DragDropManagerImpl from './classes/DragDropManagerImpl';
import DragDropMonitorImpl from './classes/DragDropMonitorImpl';
function createDragDropManager(backendFactory) {
    //创建redux仓库
    const store = createStore(reducer);
+   const registry = new HandlerRegistryImpl(store);
+   const globalMonitor = new DragDropMonitorImpl(store, registry);
    const manager = new DragDropManagerImpl(store, globalMonitor);
    const backend = backendFactory(manager);
    manager.receiveBackend(backend);
    return manager;
}
export default createDragDropManager;
```

7.3 DragDropMonitorImpl.js

src\dnd-core\classes\DragDropMonitorImpl.js

```
class DragDropMonitorImpl {
    store
+   registry
    constructor(store, registry) {
        this.store = store
+       this.registry = registry;
    }
}
export default DragDropMonitorImpl;
```

8.useDragSourceConnector

- 如果后端处理 DOM 事件，但是组件使用 React 来描述 DOM，后端如何知道要监听哪些 DOM 节点？
- 连接器允许您将预定义角色之一（拖动源或放置目标）分配给渲染函数中的 DOM 节点

8.1 useDrag\index.js

src\react-dnd\hooks\useDrag\index.js

```
import useDragSourceMonitor from './useDragSourceMonitor';
+import useDragSourceConnector from './useDragSourceConnector';
function useDrag(spec) {
    const monitor = useDragSourceMonitor();
+   const connector = useDragSourceConnector();
+   console.log(connector);
    return [{}, () => { }];
}
export default useDrag;
```

8.2 useDragSourceConnector.js

src\react-dnd\hooks\useDrag\useDragSourceConnector.js

```
import { useMemo } from 'react';
import useDragDropManager from '../useDragDropManager';
import { SourceConnector } from '../../internals';
function useDragSourceConnector() {
    const manager = useDragDropManager();
    const connector = useMemo(() => new SourceConnector(manager.getBackend()), [manager]);
    return connector;
}
export default useDragSourceConnector;
```

8.3 internals\index.js

src\react-dnd\internals\index.js

```
export { default as DragSourceMonitorImpl } from './DragSourceMonitorImpl';
+export { default as SourceConnector } from './SourceConnector';
```

8.4 SourceConnector.js

src\react-dnd\internals\SourceConnector.js

```
class SourceConnector {
    backend
    constructor(backend) {
        this.backend = backend;
    }
    connect() {
        console.log('连接React和DOM');
    }
}

export default SourceConnector;
```

9.创建DragSource和DragType

- 拖动源和放置目标可以将类型、项目、副作用和收集功能与您的组件联系在一起

9.1 src\react-dnd\hooks\useDrag\index.js

src\react-dnd\hooks\useDrag\index.js

```
import useDragSourceMonitor from './useDragSourceMonitor';
import useDragSourceConnector from './useDragSourceConnector';
+import useRegisteredDragSource from './useRegisteredDragSource';
function useDrag(spec) {
//  const monitor = useDragSourceMonitor();
    const connector = useDragSourceConnector();
+   useRegisteredDragSource(spec, monitor, connector);
    return [{}, () => { }];
}
export default useDrag;
```

9.2 useRegisteredDragSource.js

src\react-dnd\hooks\useDrag\useRegisteredDragSource.js

```
import useDragDropManager from '../useDragDropManager';
import useDragSource from './useDragSource';
import { useDragType } from './useDragType';
function useRegisteredDragSource(spec, monitor, connector) {
    const manager = useDragDropManager();
    const handler = useDragSource(spec, monitor, connector);
    console.log('handler', handler);
    const itemType = useDragType(spec);
    console.log(itemType);
}
export default useRegisteredDragSource;
```

9.3 useDragSource.js

src\react-dnd\hooks\useDrag\useDragSource.js

```

import { useMemo, useEffect } from 'react';
import { DragSourceImpl } from './DragSourceImpl';
function useDragSource(spec, monitor, connector) {
    const handler = useMemo(() => {
        return new DragSourceImpl(spec, monitor, connector);
    }, [monitor, connector]);
    useEffect(() => {
        handler.spec = spec;
    }, [handler, spec]);
    return handler;
}
export default useDragSource;
```

9.4 DragSourceImpl.js

src\react-dnd\hooks\useDrag\DragSourceImpl.js

```
export class DragSourceImpl {
    spec;
    monitor;
    connector;
    constructor(spec, monitor, connector) {
        this.spec = spec;
        this.monitor = monitor;
        this.connector = connector;
    }
}
```

9.5 useDragType.js

src\react-dnd\hooks\useDrag\useDragType.js

```
import { useMemo } from 'react';
export function useDragType(spec) {
    return useMemo(() => spec.type, [spec]);
}
```

10.registerSource

- 注册拖动源

10.1 useRegisteredDragSource.js

src\react-dnd\hooks\useDrag\useRegisteredDragSource.js

```
+import { useLayoutEffect } from 'react';
import useDragDropManager from '../useDragDropManager';
import useDragSource from './useDragSource';
import { useDragType } from './useDragType';
+import { registerSource } from '../../internals';
function useRegisteredDragSource(spec, monitor, connector) {
    const manager = useDragDropManager();
    const handler = useDragSource(spec, monitor, connector);
    const itemType = useDragType(spec);
+   useLayoutEffect(function () {
+       const handlerId = registerSource(itemType, handler, manager);
+       monitor.receiveHandlerId(handlerId);
+       connector.receiveHandlerId(handlerId);
+       console.log('dragSources', manager.globalMonitor.registry.dragSources);
+       console.log('types', manager.globalMonitor.registry.types);
+   }, [connector, handler, itemType, manager, monitor]);
}
export default useRegisteredDragSource;
```

10.2 DragSourceMonitorImpl.js

src\react-dnd\internals\DragSourceMonitorImpl.js

```
class DragSourceMonitorImpl {
    internalMonitor
+   handlerId
    constructor(manager) {
        this.internalMonitor = manager.getGlobalMonitor();
    }
+   receiveHandlerId(handlerId) {
+       this.handlerId = handlerId;
+   }
}
export default DragSourceMonitorImpl;
```

10.3 SourceConnector.js

src\react-dnd\internals\SourceConnector.js

```
class SourceConnector {
    backend
+   handlerId
    constructor(backend) {
        this.backend = backend;
    }
    connect() {
        console.log('连接React和DOM');
    }
+   receiveHandlerId(handlerId) {
+       this.handlerId = handlerId;
+   }
}

export default SourceConnector;
```

10.4 registerSource.js

src\react-dnd\internals\registerSource.js

```
function registerSource(type, handler, manager) {
    const registry = manager.getRegistry();
    const handlerId = registry.addSource(type, handler);
    return handlerId;
}
export default registerSource;
```

10.5 internals\index.js

src\react-dnd\internals\index.js

```
export { default as DragSourceMonitorImpl } from './DragSourceMonitorImpl';
export { default as SourceConnector } from './SourceConnector';
+export { default as registerSource } from './registerSource';
```

10.6 DragDropManagerImpl.js

src\dnd-core\classes\DragDropManagerImpl.js

```
class DragDropManagerImpl {
    store;
    backend;
    globalMonitor;
    constructor(store, globalMonitor) {
        this.store = store;
        this.globalMonitor = globalMonitor;
    }
    receiveBackend(backend) {
        this.backend = backend;
    }
    getBackend() {
        return this.backend;
    }
    getGlobalMonitor() {
        return this.globalMonitor;
    }
+   getRegistry() {
+       return this.globalMonitor.registry;
+   }
}
export default DragDropManagerImpl;
```

10.7 HandlerRegistryImpl.js

src\dnd-core\classes\HandlerRegistryImpl.js

```
+import { HandlerRole } from '../interfaces';
+import { getNextUniqueId } from '../utils';
+import { addSource } from '../actions/registry';
class HandlerRegistryImpl {
    store;//redux仓库
+   types = new Map(); //不同handlerId的类型
+   dragSources = new Map();//拖放来源
    constructor(store) {
        this.store = store;
    }
+   addSource(type, source) {
+       const handlerId = this.addHandler(HandlerRole.SOURCE, type, source);
+       this.store.dispatch(addSource(handlerId));
+       return handlerId;
+   }
+   addHandler(role, type, handler) {
+       const handlerId = getNextHandlerId(role);//获取下一个handleId
+       this.types.set(handlerId, type);
+       if (role === HandlerRole.SOURCE) {
+           this.dragSources.set(handlerId, handler);
+       }
+       return handlerId;
+   }
+   getSource(handlerId) {
+       return this.dragSources.get(handlerId);
+   }
+   getSourceType(handlerId) {
+       return this.types.get(handlerId);
+   }
}
+function getNextHandlerId(role) {
+    const id = getNextUniqueId().toString();
+    switch (role) {
+        case HandlerRole.SOURCE:
+            return `S${id}`;
+        case HandlerRole.TARGET:
+            return `T${id}`;
+        default:
+            throw new Error(`Unknown Handler Role: ${role}`);
+    }
+}
export default HandlerRegistryImpl;
```

10.8 interfaces.js

src\dnd-core\interfaces.js

```
export var HandlerRole = {
    "SOURCE": "SOURCE",
    "TARGET": "TARGET"
}
```

10.9 getNextUniqueId.js

src\dnd-core\utils\getNextUniqueId.js

```
let nextUniqueId = 0;
function getNextUniqueId() {
    return nextUniqueId++;
}
export default getNextUniqueId;
```

10.10 utils\index.js

src\dnd-core\utils\index.js

```
export { default as getNextUniqueId } from './getNextUniqueId';
```

10.11 registry.js

src\dnd-core\actions\registry.js

```
export const ADD_SOURCE = 'dnd-core/ADD_SOURCE';
export function addSource(handlerId) {
    return {
        type: ADD_SOURCE,
        payload: { handlerId }
    };
}
```

11.connectDragSource

- 绑定拖动事件

11.1 useDrag\index.js

src\react-dnd\hooks\useDrag\index.js

```
import useDragSourceMonitor from './useDragSourceMonitor';
import useDragSourceConnector from './useDragSourceConnector';
import useRegisteredDragSource from './useRegisteredDragSource';
+import useConnectDragSource from './useConnectDragSource';
function useDrag(spec) {
    const monitor = useDragSourceMonitor();
    const connector = useDragSourceConnector();
    useRegisteredDragSource(spec, monitor, connector);
    return [
        {},
+       useConnectDragSource(connector)
    ];
}
export default useDrag;
```

11.2 useConnectDragSource.js

src\react-dnd\hooks\useDrag\useConnectDragSource.js

```
import { useMemo } from 'react';
function useConnectDragSource(connector) {
    return useMemo(() => connector.receiveDragSource, [connector]);
}
export default useConnectDragSource
```

11.3 SourceConnector.js

src\react-dnd\internals\SourceConnector.js

```
class SourceConnector {
    backend
    handlerId  //拖动源唯一标识
+   dragSourceRef //DOM节点
    constructor(backend) {
        this.backend = backend;
    }
+   get dragSource() {
+       return this.dragSourceRef && this.dragSourceRef.current;
+   }
+   connect() {
+       if (!this.handlerId || !this.dragSource) {
+           return;
+       }
+       this.backend.connectDragSource(this.handlerId, this.dragSource);
+   }

    //useRegisteredDragSource中useLayoutEffect赋值
    receiveHandlerId(handlerId) {
        this.handlerId = handlerId;
+       this.connect();
    }
+   receiveDragSource = (dragSourceRef) => {
```