kit dnd - QuickStart

安装

1. 核心库：`npm install @dnd-kit/core`
   - 对等依赖（peer dependencies）：`react`、`react-dom`
   - 主要内容：
     - Context provider
     - Hooks for: Draggable、Droppable
     - Drag Overlay
     - Sensors for: Pointer、Mouse、Touch、Keyboard
     - Accessibility features
2. 子库
   a. Modifiers：`npm install @dnd-kit/modifiers`

   > 动态修改传感器（sensors）检测到的运动坐标（movement coordinates）

   b. Sortable：`npm install @dnd-kit/sortable`

   > 构建在 @dnd-kit/core 基础上的一个小层，专为构建一个可排序界面

   c. Utilities：`npm install @dnd-kit/utilities`

   > 提供了如 CSS 这样的工具，用于简化代码

核心

1. **Context Provider**（提供上下文）：为了使 useDraggable 和 useDroppable 钩子正常运行，您需要确保使用这两个钩子的组件被包裹在 <DndContext /> 组件中。

```
1 import React from 'react';
2 import {DndContext} from '@dnd-kit/core';
3
4 import {Draggable} from './Draggable';
5 import {Droppable} from './Droppable';
6
7 function App() {
8   return (
9     {/* DndContext 提供使 Hook 正常运行的上下文 */}
```

```
      <DndContext>
        {/* 使用了 useDraggable 的组件 */}
        <Draggable />
        {/* 使用了 useDroppable 的组件*/}
        <Droppable />
      </DndContext>
    )
}
```

2. **Droppable**（可放置组件）：使用 useDroppable 钩子使得组件为**可放置组件**（droppable）。
   a. useDroppable 钩子的参数：`{ id }`，表示为每一个可放置组件提供一个唯一的 id 属性。
   b. useDroppable 钩子的返回值：`{ isOver, setNodeRef }`
      i. isOver：当可拖拽组件移动到当前可放置组件上时，isOver 变为 true
      ii. setNodeRef：将该内容传递给 DOM 元素的 ref 属性，使其变为一个可放置组件
   c. 注意：Droppable 组件不会影响应用程序的结构。

```
import React from 'react';
import {useDroppable} from '@dnd-kit/core';

function Droppable(props) {
  const {isOver, setNodeRef} = useDroppable({
    id: 'droppable',
  });
  const style = {
    color: isOver ? 'green' : undefined,
  };

  return (
    <div ref={setNodeRef} style={style}>
      {props.children}
    </div>
  );
}
```

3. **Draggable**（可拖拽组件）：使用 useDraggable 钩子使得组件为**可拖拽组件**（draggable）。
   a. useDraggable 钩子的参数：`{ id }`，表示为每一个可拖拽组件提供一个唯一的 id 属性。
   b. useDraggable 钩子的返回值：`{ attributes, listeners, setNodeRef, transform }`

i. transform：当开始拖拽一个可拖拽组件时，transform 将会被填充为 `{ x: number, y: number, scaleX: number, scaleY: number }`。x，y 表示当前组件相对于其原始位置的水平和垂直偏移量；scaleX、scaleY 表示当前组件在水平方向和垂直方向上的缩放比例，大多数情况，这两个值取值为 1，表示组件大小不变

ii. setNodeRef：将该内容传递给 DOM 元素的 ref 属性，使其变为一个可拖拽组件

iii. listeners、attributes：要传递给可拖拽组件的监听器和属性。

c. 注意：Draggable 组件不会影响应用程序的结构。

```
import React from 'react';
import {useDraggable} from '@dnd-kit/core';

function Draggable(props) {
  const {attributes, listeners, setNodeRef, transform} = useDraggable({
    id: 'draggable',
  });
  const style = transform ? {
    transform: `translate3d(${transform.x}px, ${transform.y}px, 0)`,
  } : undefined;


  return (
    <button ref={setNodeRef} style={style} {...listeners} {...attributes}>
      {props.children}
    </button>
  );
}
```

4. **Droppable & Draggable**

- useDraggable 钩子用于定义一个**可拖拽（Draggable）组件**，其可以放置在使用 useDroppable 钩子定义的**可放置（Droppable）组件**（容器）中。
- 对于可拖拽组件
  - **建议使用 `transform` 这个 CSS 属性来移动组件**，这在性能上更佳。
  - 可以**设置 `z-index` 属性，以确保其出现在其他元素之上**。
  - 可以**使用 `<DragOverlay>` 组件，实现跨容器拖拽效果**。
  - 可以**使用 `@dnd-kit/utilities` 中的 `CSS` 将 transform 对象转换为字符串**，避免手工操作。

```
const style = {
```

```
  transform: `translate3d(${transform.x}px, ${transform.y}px, 0)`,
}

// 等价为

import {CSS} from '@dnd-kit/utilities';

// Within your component that receives `transform` from
`useDraggable`:
const style = {
  transform: CSS.Translate.toString(transform),
}
```

示例 - 单容器

将 <Draggable> 组件从外部移动到 <Droppable> 组件中，同时，监听 <DndContext> 组件的 onDragEnd 事件，以确认可拖拽组件是否被放到了可放置组件上。

```
// App.tsx
import React, {useState} from 'react';
import {DndContext} from '@dnd-kit/core';

import {Droppable} from './Droppable';
import {Draggable} from './Draggable';

function App() {
  const [isDropped, setIsDropped] = useState(false);
  const draggableMarkup = (
    <Draggable>Drag me</Draggable>
  );

  return (
    <DndContext onDragEnd={handleDragEnd}>
      {!isDropped ? draggableMarkup : null}
      <Droppable>
        {isDropped ? draggableMarkup : 'Drop here'}
      </Droppable>
    </DndContext>
  );

  function handleDragEnd(event) {
    if (event.over && event.over.id === 'droppable') {
      setIsDropped(true);
    }
```

```
  }
}
```

```tsx
// Droppable.tsx
import React from 'react';
import {useDroppable} from '@dnd-kit/core';

export function Droppable(props) {
  const {isOver, setNodeRef} = useDroppable({
    id: 'droppable',
  });
  const style = {
    color: isOver ? 'green' : undefined,
  };


  return (
    <div ref={setNodeRef} style={style}>
      {props.children}
    </div>
  );
}
```

```tsx
// Draggable.tsx
import React from 'react';
import {useDraggable} from '@dnd-kit/core';

export function Draggable(props) {
  const {attributes, listeners, setNodeRef, transform} = useDraggable({
    id: 'draggable',
  });
  const style = transform ? {
    transform: `translate3d(${transform.x}px, ${transform.y}px, 0)`,
  } : undefined;


  return (
    <button ref={setNodeRef} style={style} {...listeners} {...attributes}>
      {props.children}
    </button>
  );
}
```

示例 - 多容器

```tsx
// App.tsx
import React, {useState} from 'react';
import {DndContext} from '@dnd-kit/core';

import {Droppable} from './Droppable';
import {Draggable} from './Draggable';

function App() {
  const containers = ['A', 'B', 'C'];
  const [parent, setParent] = useState(null);
  const draggableMarkup = (
    <Draggable id="draggable">Drag me</Draggable>
  );

  return (
    <DndContext onDragEnd={handleDragEnd}>
      {parent === null ? draggableMarkup : null}

      {containers.map((id) => (
        // We updated the Droppable component so it would accept an `id`
        // prop and pass it to `useDroppable`
        <Droppable key={id} id={id}>
          {parent === id ? draggableMarkup : 'Drop here'}
        </Droppable>
      ))}
    </DndContext>
  );

  function handleDragEnd(event) {
    const {over} = event;

    // If the item is dropped over a container, set it as the parent
    // otherwise reset the parent to `null`
    setParent(over ? over.id : null);
  }
};
```

```tsx
// Droppable.tsx
import React from 'react';
import {useDroppable} from '@dnd-kit/core';

export function Droppable(props) {
```

```
  const {isOver, setNodeRef} = useDroppable({
    id: props.id,
  });
  const style = {
    color: isOver ? 'green' : undefined,
  };


  return (
    <div ref={setNodeRef} style={style}>
      {props.children}
    </div>
  );
}
```

```
// Draggable.tsx
import React from 'react';
import {useDraggable} from '@dnd-kit/core';

export function Draggable(props) {
  const {attributes, listeners, setNodeRef, transform} = useDraggable({
    id: props.id,
  });
  const style = transform ? {
    transform: `translate3d(${transform.x}px, ${transform.y}px, 0)`,
  } : undefined;


  return (
    <button ref={setNodeRef} style={style} {...listeners} {...attributes}>
      {props.children}
    </button>
  );
}
```