drag 事件

```
addEventListener('drag', (event) => {});
ondrag = (event) => { };
```

- 设置标签属性: `draggable = “true”`
- `dragstart`：当用户开始拖动一个元素或选中的文本时触发。
- `drag`：当元素或选中的文本被拖动时持续触发。
- `dragend`：当拖放操作结束时触发（例如，当用户放开鼠标按钮或按下 Esc 键）。
- `dragenter`：当被拖动的元素或选中的文本进入有效的放置目标时触发。
- `dragover`：当被拖动的元素或选中的文本在有效的放置目标上移动时持续触发。
- `dragleave`：当被拖动的元素或选中的文本离开有效的放置目标时触发。
- `drop`：当被拖动的元素或选中的文本在有效的放置目标上放开时触发。

> 注意dragenter,dragover,dragleave,drop 都是在被进入的对象而不是被拖动的对象上绑定的事件

实例

```
<div id="drag-source">拖动我</div>
<div id="drop-target">将我拖到这里</div>
```

```
let dropTarget = document.querySelector('#drop-target');
dropTarget.addEventListener('dragenter', function(event) {
    dropTarget.style.backgroundColor = 'lightblue';
});
```

实例：拖拽排序

伪代码：

获取拖动的元素和被进入的对象，然后如果拖动的元素进入被拖动的对象，两个就**交换位置，**交换位置用到insertBefore,**只有Before,没有After**所以要判断两个的位置那个在前那个在后

判断：通过获取此元素在ul中的下标来判断，这个下标应该是现场取现场用的，拖动元素的下标和被进入元素的下标在每一次调换之后都会发生变化

insertBefore：

> `Node.insertBefore()` 在参考结点前插入一个有指定父元素的子节点，如果给定的子节点是对当前文档的引用，那么会把他移动到这个位置，不需要再去删除，（一个结点不会同时存在两个地方，除非用cloneNode()，如果引用结点为null，那么将指定节点添加到子节点列表的末端

```
var insertedNode = parentNode.insertBefore(newNode, referenceNode);
```

Copy to Clipboard

- `insertedNode` 被插入节点 (newNode)
- `parentNode` 新插入节点的父节点
- `newNode` 用于插入的节点
- `referenceNode` `newNode` 将要插在这个节点之前

⚠️ 注意，如果要实现动画效果，在结点移动的时候会额外触发两次enter事件，要注意标识动画是否在运行中，若在，则不执行代码 `animation = false`

```
let dragList = document.querySelector('.content .edit ul')//还有一个所以要再来一次
let draggedElement;
let draggedOrder;
let animation = false;//标记做动画的过程，在这个过程中不能再次触发动画了
dragList.addEventListener('dragstart', (e) => {
    draggedElement = e.target
    draggedOrder = Array.from(draggedElement.parentNode.children).indexOf(draggedElement)
})

dragList.addEventListener('dragenter', (e) => {//只有进入的时候会执行
    e.preventDefault()
    let order = Array.from(e.target.parentNode.children).indexOf(e.target)
    //判断先后，执行动画，调换位置：
    if (e.target !== draggedElement && !animation) {
        if ((order > draggedOrder)) {
            animation = true
            draggedElement.classList.add("dragSortingDown")
            e.target.classList.add("dragSortingUp")
            e.target.addEventListener("animationend", () => {
                dragList.insertBefore(e.target, draggedElement)
                e.target.classList.remove("dragSortingUp")
                draggedElement.classList.remove("dragSortingDown")
                animation = false
                //更新拖动元素的下标
                draggedOrder = Array.from(e.target.parentNode.children).indexOf(draggedElement)

                return;
            })
        }
        else if ((order < draggedOrder)) {
            animation = true
            e.target.classList.add("dragSortingDown")
            draggedElement.classList.add("dragSortingUp")
            e.target.addEventListener("animationend", () => {
                dragList.insertBefore(draggedElement, e.target)
                e.target.classList.remove("dragSortingDown")
                draggedElement.classList.remove("dragSortingUp")
                animation = false
                //更新拖动元素的下标
                draggedOrder = Array.from(e.target.parentNode.children).indexOf(draggedElement)

                return;
            })
        }
    }
})

dragList.addEventListener('dragover', (e) => {
    e.preventDefault()
})
```