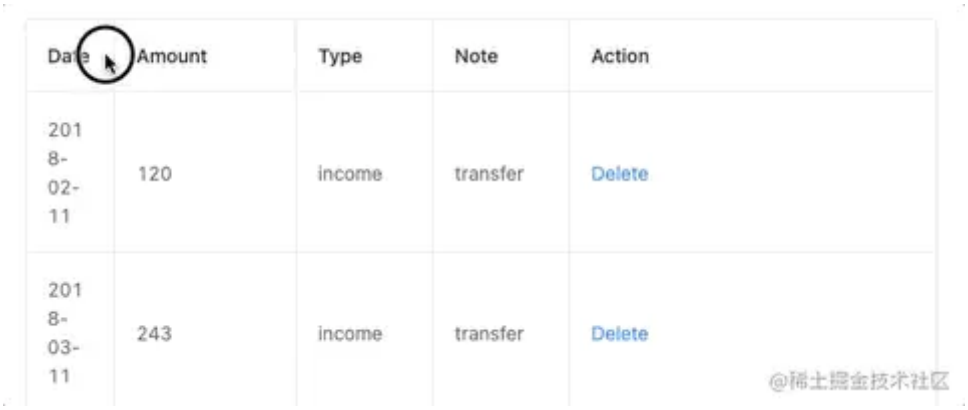
实现 Ant Table 可伸缩列（列宽度可拖拽）

概述

本文将介绍如何使用 react-resizable 实现 Ant Table 可伸缩列



在实际开发和调试遇到的问题概览：

1. 在 Ant Design 4.x 文档 上已经找不到 Ant Table 可伸缩列，本文可作为参考资料
2. 解决在 Windows 系统松开鼠标依然能拖动
3. 解决伸缩拖动与排序事件重叠
4. 实现持久化存储（localStorage 和 sessionStorage）
5. 支持最小、最大列宽度的配置

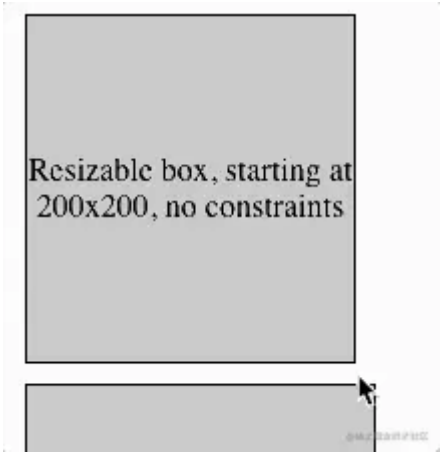
> `antd` 是一个基于 Ant Design 设计体系的 React UI 组件库，主要用于研发企业级中后台产品。地址：github.com/ant-design/...

接下来将从零开始讲述，您可以通过小标题跳到最感兴趣的位置

一、react-resizable实现可伸缩组件

在每次拖动宽高时，将触发 Resizable 组件 onResize 回调，修改 state 的 width 和 height，更新 container 组件的 width 和 height，达到可伸缩的效果。

效果：



演示代码如下：

jsx 复制代码

```jsx
import { Resizable, ResizableBox } from 'react-resizable';

export const Resizable = () => {
   const [width, setWidth] = useState(200);
   const [height, setHeight] = useState(200);
   onResize = (event, {element, size, handle}) => {
       setWidth(size.width);
       setHeight(size.height);
   };
   return (
       <Resizable
           width={width}
           height={height}
           onResize={onResize}
       >
           <div
               className="container"
               style={{
                   width,
                   height,
               }}>
               Resizable box, starting at 200x200, no constraints
```

```
            </div>
        </Resizable>
    )
}
```

更多示例和资料可点击 react-resizable

二、实现可伸缩表格列

接下来给 Ant Table 传入自定义的 header cell 组件，下方所示的 ResizableTitle，同时，在 Column 定义增加 `width: column.width` 和 `onResize: handleResize(index)` 参数，即可实现可伸缩 Ant Table 表头

jsx 复制代码

```jsx
import { Resizable } from 'react-resizable';

const ResizableTitle = props => {
    const { onResize, width, ...restProps } = props;

    // 没有原始宽度的列，不支持伸缩；会出现从自适应宽度一下子跳到拖动位置；也可以自行增加参数，如 disableRes
    if (!width) {
        return <th {...restProps} />;
    }

    return (
        <Resizable
            width={width}
            height={0}    // 不需要调整高度，设为 0
            onResize={onResize}
        >
            <th {...restProps} />
        </Resizable>
    );
};

// 原始的表格列 tableColumn
const OriginTableColumns = [
    {
        title: 'Date',
        dataIndex: 'date',
        width: 200,
    },
```

```
    {
        title: 'Amount',
        dataIndex: 'amount',
        width: 100,
    }
];

const ResizableTable = props => {
    const [columns, setColumns] = useState(
        // 每一列增加 width, onResize 作为 ResizableTitle 的 props
        OriginTableColumns.map((col, index) => ({
            ...col,
            onHeaderCell: (column) => ({
                width: column.width,
                onResize: handleResize(col.key),
            }),
        })
    );
    // 表格数据 data
    const data = [{
        key: 0,
        date: '2018-02-11',
        amount: 120,
    }];

    // 拖动时更新表格列
    handleResize = key => (e, { size }) => {
        setColumns((columns) => (
            columns.map((column) => {
                if (column.key === key) {
                    return {
                        ...column,
                        width: size.width,
                    };
                } else {
                    return column;
                }
            })
        ))
    };

    const components={{
        header: {
            cell: ResizableTitle,
        },
    }}
    return <Table bordered components={components} columns={columns} dataSource={data} />;
}
```

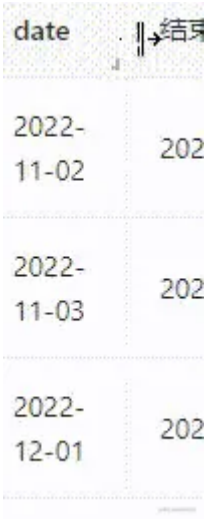
三、解决在 Windows 松开鼠标依然能拖动

本以为这样就完成了，后端小伙伴说在他 Windows 电脑上，时不时出现松开鼠标依然可以拖动，在我 Mac 电脑上却无法复现。

经过半天排查结果：

1. Windows 上 Chrome 和 Edge 浏览器会出现，Firefox 不会
2. Mac 上 Chrome、Firefox、Safari、Edge 浏览器均不会出现
3. 出现异常以前会误选中文本，再点击拖动按钮

现象如图：



解决思路：在点击拖动时，使用浏览器API Selection.removeAllRanges 清空原本误选的文本。

对于不同浏览器的兼容和触发时机代码如下：

```js
const clearSelection = () => {
        if (window.getSelection) {
                const selection = window.getSelection();
                if (selection) {
                        if (selection.empty) {  // Chrome
                                selection.empty();
                        } else if (selection.removeAllRanges) {  // Firefox
                                selection.removeAllRanges();
                        }
                }
        } else if (document.selection && document.selection.empty) {  // IE
```

```
                document.selection.empty();
        }
}
const ResizableTitle = props => {
        // ...
        return (
                <Resizable
                        // ...
                        draggableOpts={{
                                onMouseDown: (e: any) => {
                                        // fix: 修复在 Windows Chrome 和 Edge 松开鼠标依然能拖动
                                        clearSelection();
                                }
                        }}
                        >
                        <th {...restProps} />
                </Resizable>
        );
}
```

四、解决伸缩拖动与排序事件重叠

如下动图展示，在按住拖动到松开后，必然会触发排序事件，原因是排序按钮的点击区域，是覆盖整个表头的，点击拖动按钮时，事件也会冒泡到外层的排序按钮。

解决方案是阻止冒泡，具体实现如下：

```javascript
const ResizableTitle = props => {
    useEffect(() => {
        // 使用选择器，获取所有拖动按钮的 Dom 对象
        document.querySelectorAll('.react-resizable-handle').forEach((elem) => {
            if (elem.onclick) {
                return;
            }
            elem.onclick = (event) => {
                // 阻止冒泡
                event.stopPropagation();
                return false;
            };
        });
    }, []);
    // ...
    return (
        <Resizable
            // ...
        >
            <th {...restProps} />
        </Resizable>
    );
}
```

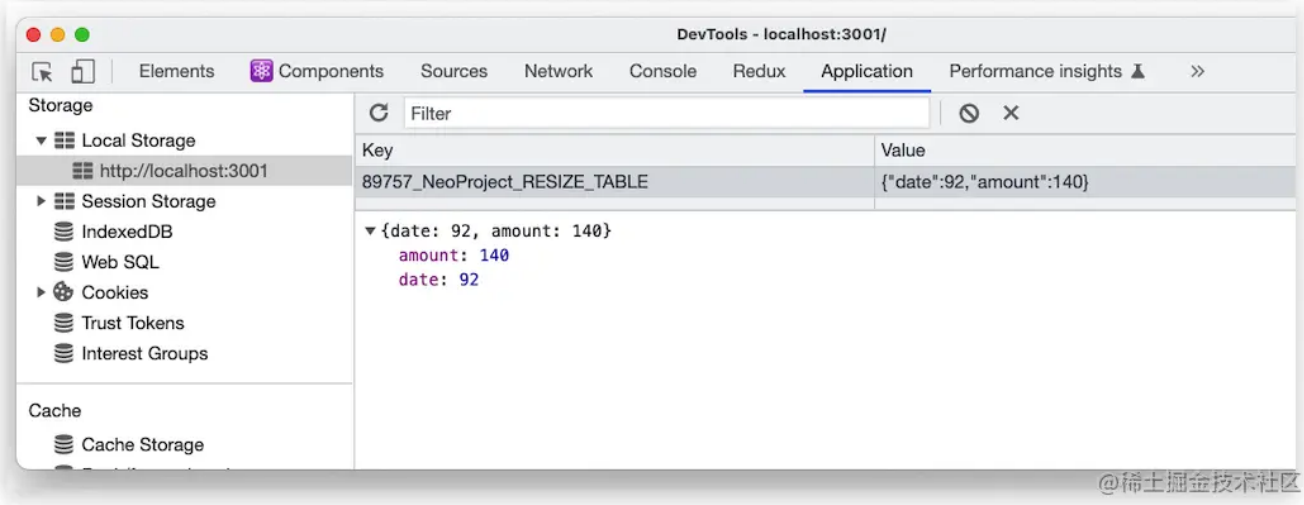

五、实现持久化存储

在将可伸缩列的数据做持久化存储之前，可以先思考🤔：

1. 存储的数据结构是怎样的?
2. 划分的维度是怎样的?
3. 何时存储和读取?

由于 indexedDB 兼容性不是很好，这里功能也简单，就使用 localStorage 和 sessionStorage 进行存储了。

存储效果展示：



1. 存储的数据结构，是一个对象 或 Map，对象的 key 是 Column 的 key，值是拖拽后得到的数值 width，这样就能最小化存储数据了
2. 划分的维度，是根据产品经理的要求去做划分，比如 `${userId}_{projectName}_RESIZE_TABLE`，把这个作为 Storage.setItem(key: string, value: string) 的 key, value 是上面提到的数据结构
3. 存储 Storage 时机，是在用户拖拽结束后，是用 Resizable 的参数 onResizeStop 进行监听;
4. 读取 Storage 时机，是在初始化 columns state 时读取的，useState 的 initialState 也可以传进一个函数

关键代码演示：

```js
// Storage setItem 和 getItem 使用的 key，根据业务具体设计，这里仅供参考
const STORAGE_KEY = `${userId}_{projectName}_RESIZE_TABLE`;
import { Resizable } from 'react-resizable';

const ResizableTitle = props => {
    const { onResize, onResizeStop, width, ...restProps } = props;
    if (!width) {
        return <th {...restProps} />;
    }
    return (
        <Resizable
            width={width}
            height={0}    // 不需要调整高度，设为 0
            onResize={onResize}
            // 监听 onResizeStop，用于在用户拖拽结束后，把宽度传给外层组件处理
            onResizeStop={onResizeStop}
        >
            <th {...restProps} />
        </Resizable>
    );
```

```
};

// ...
const ResizableTable = props => {
    const storage = window.localStorage; // 根据需要选择 window.localStorage 或 window.sessionStorage

    const [columns, setColumns] = useState(() => {
        // 读取 Storage 时机
        const columnWidthStr = storage.getItem(STORAGE_KEY);
        const columnWidthMap = JSON.parse(columnWidthStr);
        return OriginTableColumns
            .map((col, index) => ({
                ...col,
                onHeaderCell: (column) => ({
                    width: column.width,
                    onResize: handleResize(col.key),
                    onResizeStop: handleResizeStop(col.key),
                }),
            })
            .map((col) => {
                if (columnWidthMap[col.key]) {
                    // 如果之前调整过宽度，就对 column width 进行覆盖
                    return {
                        ...col,
                        width: columnWidthMap[col.key],
                    };
                } else {
                    return col;
                }
            })
    });

    // 存储 Storage 时机
    const handleResizeStop = key => (e, { size }) => {
        // 存储的数据结构: columnWidthMap key 是 column key, value 是 width
        const columnWidthMap = columns.reduce((acc, cur) => {
            // 如果 width 为空，说明没有初始宽度的列，直接返回
            if (cur.width === null || cur.width === undefined) return acc;
            return {
                ...acc,
                [cur.key]: cur.width,
            }
        }, {});
        storage.setItem(STORAGE_KEY, JSON.stringify(columnWidthMap));
    };
    //...
    return <Table bordered components={components} columns={columns} dataSource={data} />;
}
```

> 提示：JSON.parse 和 JSON.stringify 可能会抛出 Error，需要对他们进行 try...catch

六、支持最小、最大列宽度的配置

当列宽度太小会使得行特别长，影响展示效果；当列宽度太长，太多的空白是没有必要的，因而增加最小、最大列宽度的配置。

关键演示代码如下：

```js 复制代码
import { Resizable } from 'react-resizable';

const ResizableTitle = props => {
    const { onResize, width, minWidth, maxWidth, ...restProps } = props;
    // ...
    /**
        Resizable 暴露的参数有 minConstraints 和 maxConstraints，
        [number, number] 对应的是 [width, height]，这里我们只调整列宽度

        export type ResizableProps = {
            // ...
            minConstraints?: [number, number] | undefined;
            maxConstraints?: [number, number] | undefined;
        }
     */
    const minConstraints = minWidth ? [minWidth, -Infinity] : undefined;
    const maxConstraints = maxWidth ? [maxWidth, +Infinity] : undefined;
    return (
        <Resizable
            width={width}
            height={0}    // 不需要调整高度，设为 0
            onResize={onResize}
            minConstraints={minConstraints}
            maxConstraints={maxConstraints}
        >
            <th {...restProps} />
        </Resizable>
    );
};

const OriginTableColumns = [
```

```
    {
        title: 'Date',
        dataIndex: 'date',
        width: 200,
        minWidth: 50,
        maxWidth: 400,
    },
    {
        title: 'Amount',
        dataIndex: 'amount',
        width: 100,
        minWidth: 20,
        maxWidth: 200,
    }
];

const ResizableTable = props => {
    const [columns, setColumns] = useState(
        OriginTableColumns.map((col, index) => ({
            ...col,
            onHeaderCell: (column) => ({
                width: column.width,
                // 每一列增加 minWidth, maxWidth 作为 ResizableTitle 的 props
                minWidth: column.minWidth,
                maxWidth: column.maxWidth,
                onResize: handleResize(col.key),
            }),
        })
        //...
    );
    // ...
    return <Table bordered components={components} columns={columns} dataSource={data} />;
}
```