可视化拖拽编辑器(三)

一.实现导入导出功能

```
{label: '导出', icon: 'icon-export', handler: () => {console.log('导出')}},
{label: '导入',icon: 'icon-import', handler: () => {console.log('导入')}}
```

> 增加两个按钮用于实现导入导出的功能

1.实现Dialog组件

```
let vm;
export const $dialog = (option) => {
    if (!vm) {
        const el = document.createElement('div');
        vm = createVNode(DialogComponent, { option }); // 渲染组件
        document.body.appendChild((render(vm, el),el));
    }
    let { showDialog } = vm.component.exposed;
    showDialog(option);
}
```

```
import { ElDialog ,ElButton,ElInput} from "element-plus";
import { createVNode, defineComponent, render,reactive } from "vue";

const DialogComponent = defineComponent({
    props: {
        option: { type: Object },
    },
    setup(props, ctx) {
        const state = reactive({
            option: props.option,
            isShow: false
        });
        const onCancel = () => { // 取消时关闭窗口
            state.isShow = false;
        }
        const onConfirm = () => { // 确认时调用用户回调
            state.option.onConfirm &&
state.option.onConfirm(state.option.content);
            state.isShow = false;
```

```
        }

        ctx.expose({
            showDialog: (option) => {
                state.option = option;
                state.isShow = true;
            }
        })
        return () => {
            return <ElDialog v-model={state.isShow} title=
{state.option.title}>
                {{
                    default: () => <div>
                        <ElInput type="textarea" v-model=
{state.option.content} rows={10}></ElInput>
                    </div>,
                    footer: () => state.option.footer && <div>
                        <ElButton onClick={onCancel}>取消</ElButton>
                        <ElButton onClick={onConfirm}>确定</ElButton>
                    </div>
                }}
            </ElDialog>
        }
    }
});
```

```
{
    label: '导出', icon: 'icon-export', handler: () => {
        $dialog({
            title:'导出JSON数据',
            content:JSON.stringify(data.value)
        })
    },
},
{
    label: '导入',icon: 'icon-import', handler: () => {
        $dialog({
            title:'导入JSON数据',
            content:'',
            footer:true,
            onConfirm(content) {
                data.value = JSON.parse(content);
            }
        })
    }
}
```

2.导入后支持撤回功能

> 注册能放到对应中的更新数据方法

```
 registry({ // 更新容器
     name: 'updateContainer',
     pushQueue: true,
     execute(val) {
         let state = {
             before: data.value,
             after: val
         }
         return {
             redo: () => {
                 data.value = state.after;
             },
             undo: () => {
                 data.value = state.before;
             }
         }
     }
 })
$dialog({
    title:'导入JSON数据',
    content:'',
    footer:true,
    onConfirm(content) {
        commands.updateContainer(JSON.parse(content)); // 调指令的时候需要传递参数
    }
})
```

二.实现置顶、置底功能

1.置顶的实现

> 置顶计算*zIndex*最大的在基础上*+1*

```
registry({
    name: 'placeTop',
    pushQueue: true,
    execute() {
        let before = deepcopy(data.value.blocks);
        let after = (() => {
            const { focus, unfocused } = focusData.value;
            let maxZIndex = unfocused.reduce((prev, block) => {
```

```
                return Math.max(prev, block.zIndex)
            }, -Infinity) + 1;
            focus.forEach(block => block.zIndex = maxZIndex);
            return data.value.blocks
        })();
        return {
            undo: () => {
                data.value = { ...data.value, blocks: before }
            },
            redo: () => {
                data.value = { ...data.value, blocks: after }
            }
        }
    }
})
```

2.置底的实现

> 置底则是在*zIndex*中最小的值*-1*，如果已经小于*0*则让其他*+1*

```
registry({ // 置底
    name: 'placeBottom',
    pushQueue: true,
    execute() {
        let before = deepcopy(data.value.blocks);
        let after = (() => {
            const { focus, unfocused } = focusData.value;
            let minZIndex = unfocused.reduce((prev, block) => {
                return Math.min(prev, block.zIndex)
            }, Infinity) -1
            if(minZIndex < 0){
                const dur = Math.abs(minZIndex);
                unfocused.forEach(block=> block.zIndex += dur);
                minZIndex = 0
            }
            focus.forEach(block => block.zIndex = minZIndex);
            return data.value.blocks
        })()
        return {
            undo: () => {
                data.value = { ...data.value, blocks: before }
            },
            redo: () => {
                data.value = { ...data.value, blocks: after }
            }
        }
    }
})
```

```
[
    { label: '置顶', icon: 'icon-place-top', tip: 'ctrl+up', handler: ()
=> commands.placeTop() },
    { label: '置底', icon: 'icon-place-bottom', tip: 'ctrl+down',
handler: () => commands.placeBottom() }
]
```

三.实现删除功能

```
{ label: '删除', icon: 'icon-reset', tip: '无', handler: () =>
commands.delete() }
```

```
registry({ // 删除
    name: 'delete',
    keyboard: 'delete',
    pushQueue: true,
    execute: () => {
        let state = {
            before: data.value.blocks || [], // 删除前
            after: focusData.value.unfocused // 删除的
        }
        return {
            redo: () => {
                data.value = { ...data.value, blocks: state.after }
            },
            undo: () => {
                data.value = { ...data.value, blocks: state.before }
            }
        }
    }
})
```

四.实现预览功能

```
const previewRef = ref(false);
let { blockMousedown, focusData, containerMousedown, lastSelectBlock,
clearBlockFocus } = useFocus(data, previewRef, (e) => {
    // 获取焦点后进行拖拽
    mousedown(e)
});
```

> 预览时阻止拖拽事件

```
const containerMousedown = () => {
    if(previewRef.value) return
}
const blockMousedown = (e, block,index) => {
    if(previewRef.value) return
}
```

```
const buttons = [
    {
        label: () => previewRef.value ? '编辑' : '预览',
        icon: () => previewRef.value ? 'icon-edit' : 'icon-browse',
        handler: () => {
            previewRef.value = !previewRef.value;
            // 如果是预览则清空获取焦点
            if (previewRef.value) clearBlockFocus(); // 需要导入方法

        }
    }
];
```

> 渲染按钮

```
{buttons.map((btn, index) => {
    const label = typeof btn.label == 'function' ? btn.label() :
btn.label;
    const icon = typeof btn.icon == 'function' ? btn.icon() : btn.icon
    return <div class="editor-top-button" onClick={btn.handler}>
        <i class={icon}></i>
        <span>{label}</span>
    </div>
})}
```

```
(data.value.blocks.map((block, index) => (
    <EditorBlock
        class={block.focus ? 'editor-block-focus' : ''}
        class={previewRef.value ? 'editor-block-editing' :''}
        block={block}
        onMousedown={(e) => blockMousedown(e, block, index)}
        ></EditorBlock>
)))
```

> 添加预览状态类名

```
.editor-block-editing {
    &:after {
        display: none;
    }
}
```

五.实现取消编辑功能

```
const editorRef = ref(true);
{
    label: '关闭', icon: 'icon-close', handler: () => {
        editorRef.value = false;
        clearBlockFocus()
    }
}
```

```
!editorRef.value ? <>
    <div
        class="editor-container-canvas__content"
        style={containerStyles.value}
        style="margin:0"
    >
        {
            (data.value.blocks.map((block, index) => (
                <EditorBlock
                    class='editor-block-editing'
                    block={block}
                    formData={props.formData}
                    ></EditorBlock>
            )))
        }
    </div>
    <div>
        <ElButton onClick={()=>editorRef.value = true}>编辑</ElButton>
    </div>
 </>
```

六.菜单展现

1.展现下拉菜单

```
(data.value.blocks.map((block, index) => (
    <EditorBlock
        class={block.focus ? 'editor-block-focus' : ''}
        block={block}
        onMousedown={(e) => blockMousedown(e, block, index)}
+       onContextmenu={(e) => onContextmenuBlock(e, block)}
    ></EditorBlock>
)))
```

```
const onContextmenuBlock = (e, block) => {
    e.preventDefault();
    $dropdown({
        el: e.target // 以哪个block显示下拉菜单
    });
}
```

```
import { computed, createVNode, defineComponent, render, reactive,
onMounted, onBeforeUnmount,ref } from "vue";
const DropdownComponent = defineComponent({
    props: {
        option: { type: Object },
    },
    setup(props, ctx) {
        const state = reactive({
            option: props.option,
            isShow: false,
            top: 0, // 显示的位置
            left: 0
        });
        ctx.expose({
            showDropdown: (option) => {
                state.option = option;
                state.isShow = true;
                // 计算显示的位置
                const { top, left, height } =
option.el.getBoundingClientRect();
                state.top = top + height;
                state.left = left;
            }
        })
        const classes = computed(() => [ // 下拉菜单的样式
            'dropdown',
            {
                'dropdown-show': state.isShow
            }
        ]);
```

```
        const el = ref(null);
        const onMousedownDocument = (e) => {
            if (!el.value.contains(e.target)) {
                state.isShow = false;
            }
        }
        // 处理点击其他元素收起下拉菜单
        onMounted(() => document.body.addEventListener('mousedown',
onMousedownDocument,true));
        onBeforeUnmount(() =>
document.body.removeEventListener('mousedown', onMousedownDocument))
        const styles = computed(() => ({
            top: `${state.top}px`,
            left: `${state.left}px`,
        }))
        return () => <div class={classes.value} style={styles.value}
ref={el}>
            内容区域
        </div>
    }
})
let vm;
export const $dropdown = (option) => {
    if (!vm) {
        const el = document.createElement('div');
        vm = createVNode(DropdownComponent, { option }); // 渲染组件
        document.body.appendChild((render(vm, el), el));
    }
    let { showDropdown } = vm.component.exposed;
    showDropdown(option);
}
```

```
.dropdown {
    display: none;
    position: fixed;
    background: #fff;
    box-shadow: 2px 2px #ccc;
}
.dropdown-show {
    display: block;
}
```

> 配置渲染到下拉菜单中的内容

```
const onContextmenuBlock = (e, block) => {
    e.preventDefault();
    $dropdown({
        el: e.target,
        content: () => <>
        <DropdownItem label="置顶" icon="icon-place-top" onClick={()=>
{}}></DropdownItem>
        <DropdownItem label="置底" icon="icon-place-bottom" onClick={()=>
{}}></DropdownItem>
        <DropdownItem label="删除" icon="icon-delete" onClick={()=>{}}>
</DropdownItem>
        <DropdownItem label="查看" icon="icon-browse" onClick={()=>{}}>
</DropdownItem>
        <DropdownItem label="导入" icon="icon-import" onClick={()=>{}}>
</DropdownItem>
        </>
    });
}
```

> 渲染*content*内容

```
<div class={classes.value} style={styles.value} ref={el}>
    {state.option.content()}
</div>
```

> 渲染下拉单中的内容

```
export const DropdownItem = defineComponent({
    props:{
        label:{type:String},
        icon:{type:String}
    },
    setup(props){
        return () => <div class="dropdown-item">
            <i class={props.icon}></i>
            <span>{props.label}</span>
        </div>
    }
})
```

```
.dropdown{
    display: none;
    position: fixed;
    background: #fff;
    box-shadow: 2px 2px #ccc;
}
```

```
.dropdown-show{
    display: block;
}
.dropdown-item {
    line-height: 30px;
    width: 100px;
    border-bottom: 1px solid #ccc;
    text-align: center;
    cursor: pointer;
    user-select: none;
}
.dropdown-item span{
    margin-left: 5px;;
}
```

2.实现点击菜单隐藏功能

```
const dropdown = {
    showDropdown: (option) => {
        state.option = option;
        state.isShow = true;
        // 计算显示的位置
        const { top, left, height } = option.el.getBoundingClientRect();
        state.top = top + height;
        state.left = left;
    },
    hideDropdown:() =>{
        state.isShow = false
    }
}
provide('dropdown',dropdown); // 将dropdown中的方法暴露出去
```

```
const {hideDropdown} = inject('dropdown')
return () => <div class="dropdown-item" onClick={hideDropdown}>
    <i class={props.icon}></i>
    <span>{props.label}</span>
</div>
```

3.实现菜单功能

```
const onContextmenuBlock = (e, block) => {
    e.preventDefault();
    $dropdown({
        el: e.target,
        content: () => <>
        <DropdownItem label="置顶" icon="icon-place-top" onClick={()=>{
                commands.placeTop()
            }}></DropdownItem>
```

```
        <DropdownItem label="置底" icon="icon-place-bottom" onClick={()=>
{
                commands.placeBottom()
            }}></DropdownItem>
        <DropdownItem label="删除" icon="icon-delete" onClick={()=>{
                commands.delete()
            }}></DropdownItem>
        <DropdownItem label="查看" icon="icon-browse" onClick={()=>{
                $dialog({
                    title:'节点数据',
                    content:JSON.stringify(block)
                })
            }}></DropdownItem>
        <DropdownItem label="导入" icon="icon-import" onClick={()=>{
                $dialog({
                    title:'请输入节点数据',
                    content:'',
                    footer:true,
                    onConfirm(text){
                        text = JSON.parse(text || '');
                        commands.updateBlock(text,block)
                    }
                })
            }}></DropdownItem>
        </>
    });
 }
```

4.实现更新某个节点

```
registry({
    name: 'updateBlock',
    pushQueue: true,
    execute(newBlock, oldBlock) {
        let state = {
            before: data.value.blocks,
            after: (() => {
                let blocks = [...data.value.blocks];
                const index = data.value.blocks.indexOf(oldBlock);
                if (index > -1) {
                    blocks.splice(index, 1, newBlock);
                }
                return blocks
            })()
        }
        return {
            redo: () => {
                data.value = { ...data.value, blocks: state.after };
            },
```

```
            undo: () => {
                data.value = { ...data.value, blocks: state.before };
            }
        }
    }
})
```

七.属性操作

> 注册组件对应所需要的属性

```
const createInputProp = (label) => ({ type: 'input', label });
const createColorProp = (label) => ({ type: 'color', label });
const createSelectProp = (label, options) => ({ type: 'select', label,
options });
```

1.注册组件属性操作栏

```
registerConfig.register({
    label: '文本',
    preview: () => '预览文本',
    render: () => '渲染文本',
    key: 'text',
    props: {
        text: createInputProp('文本内容'),
        color: createColorProp('字体颜色'),
        size: createSelectProp('字体大小', [
            { label: '14px', value: '14px' },
            { label: '20px', value: '20px' },
            { label: '24px', value: '24px' }
        ])
    }
})
registerConfig.register({
    label: '按钮',
    preview: () => <ElButton>预览按钮</ElButton>,
    render: () => <ElButton>渲染按钮</ElButton>,
    key: 'button',
    props: {
        text: createInputProp('按钮内容'),
        type: createSelectProp('按钮类型', [
            { label: '基础', value: 'primary' },
            { label: '成功', value: 'success' },
            { label: '警告', value: 'warning' },
            { label: '危险', value: 'danger' },
            { label: '提示', value: 'info' },
            { label: '文本', value: 'text' }
```

```
        ]),
        size: createSelectProp('按钮大小', [
            { label: '默认', value: '' },
            { label: '中等', value: 'medium' },
            { label: '小', value: 'small' },
            { label: '极小', value: 'mini' },
        ])
    }
})
```

```
<div class="editor-right">
    <EditorOperator block={lastSelectBlock.value}></EditorOperator>
</div>
```

2.根据属性渲染页面

```
import deepcopy from "deepcopy";
import { ElColorPicker, ElForm, ElFormItem, ElInput, ElInputNumber,
ElOption, ElSelect ,ElButton} from "element-plus";
import { defineComponent, inject, reactive } from "vue";

export default defineComponent({
    props: {
        block: { type: Object },
    },
    setup(props) {
        const config = inject('config');
        const apply = () => { // 实现应用重置功能
            console.log('应用')
        }
        const reset = () => {
            console.log('重置')
        }
        return () => {
            let content = []; // 如果没有选中元素默认修改的是容器
            if (!props.block) {
                content.push(<>
                    <ElFormItem label="容器宽度">
                        <ElInputNumber></ElInputNumber>
                    </ElFormItem>
                    <ElFormItem label="容器高度">
                        <ElInputNumber></ElInputNumber>
                    </ElFormItem>
                </>);
            } else { // 如果选中则找到组件定义的属性进行渲染
                const { key } = props.block;
                const component = config.componentMap[key];
                if (component && component.props) {
```

```
                    content.push(
                        Object.entries(component.props).map(([propName,
propConfig]) => {
                            return <ElFormItem label={propConfig.label}>
                                {{
                                    input: () => (<ElInput/>),
                                    color: () => (<ElColorPicker >
</ElColorPicker>),
                                    select: () => <ElSelect>
                                        {propConfig.options.map(opt => (
                                            <ElOption label={opt.label}
value={opt.value}>{opt.value}</ElOption>
                                        ))}
                                    </ElSelect>
                                }[propConfig.type]()}
                            </ElFormItem>
                        })
                    )
                }
            }
            return <ElForm labelPosition="top" style="padding:30px;">
                {content}
                <ElFormItem>
                    <ElButton type="primary" onClick={apply}>应用
</ElButton>
                    <ElButton onClick={reset}>重置</ElButton>
                </ElFormItem>
            </ElForm>
        }
    }
})
```

3.实现数据绑定

```
<EditorOperator block={lastSelectBlock.value} data={data.value}>
</EditorOperator>
```

```
const state = reactive({
    editData:{}
});
const reset = () => {
    if (!props.block) { // 重置容器的值
        state.editData = deepcopy(props.data.container);
    } else {             // 根据组件属性重置值
        state.editData = deepcopy(props.block || {});
    }
}
watch(() => props.block, () => { // 切换选中时更新
    reset();
}, { immediate: true });
```

> 绑定容器的值

```
content.push(<>
    <ElFormItem label="容器宽度">
        <ElInputNumber v-model={state.editData.width}></ElInputNumber>
    </ElFormItem>
    <ElFormItem label="容器高度">
        <ElInputNumber v-model={state.editData.height}></ElInputNumber>
    </ElFormItem>
</>);
```

> 更新*data.json*文件, 添加组件的*props*

```
{
    "container":{
        "width":350,
        "height":350
    },
    "blocks":[
        {"top":100,"left":100,"zIndex":1,"key":"text","props":{
            "text":"珠峰架构",
            "color":"#ff0000",
            "size":"14px"
        }},
        {"top":200,"left":100,"zIndex":1,"key":"button","props":{}},
        {"top":300,"left":100,"zIndex":1,"key":"input","props":{}}
    ]
}
```

```
const RenderComponent = component.render({
    props:props.block.props // 渲染时传入属性
});
```

```
registerConfig.register({
    label: '文本',
    preview: () => '预览文本',
    render: ({props}) =><span style=
{{color:props.color,fontSize:props.size}}>{props.text || '默认文本'}
</span>,
    key: 'text',
    // ...
})
registerConfig.register({
    label: '按钮',
    preview: () => <ElButton>预览按钮</ElButton>,
    render: ({props}) => <ElButton type={props.type}>{props.text || '默认
按钮'}</ElButton>,
    key: 'button',
    // ...
})
```

```
<ElFormItem label={propConfig.label}>
    {{
        input: () => (<ElInput v-model={state.editData.props[propName]}
/>),
        color: () => (<ElColorPicker v-model=
{state.editData.props[propName]}></ElColorPicker>),
        select: () => <ElSelect v-model=
{state.editData.props[propName]}>
            {propConfig.options.map(opt => (
                <ElOption label={opt.label} value={opt.value}>
{opt.label}</ElOption>
            ))}
        </ElSelect>
    }[propConfig.type]()}
</ElFormItem>
```

> 在菜单拖拽完毕后也添加*props*属性

4.实现修改功能

```
<EditorOperator
    block={lastSelectBlock.value}
    data={data.value}
    updateBlock={commands.updateBlock}
    updateContainer={commands.updateContainer}
 ></EditorOperator>
const apply = () => { // 实现应用重置功能
    if (!props.block) {
        // 修改的容器属性
```

```
            props.updateContainer({ ...props.data, container: state.editData
})
        } else {
            // 修改组件属性
            props.updateBlock(state.editData, props.block);
        }
}
```

> 每次数据变化后重新渲染组件

八.实现数据双向绑定

1.输入框元素的数据绑定

> 将需要绑定的数据传入

```
<Editor v-model="state" :formData="formData"></Editor>
const formData = ref({
    username: 'zfjg',
    password: '123',
})
```

> 增添*model*属性，实现双向数据绑定

```
registerConfig.register({
    label: '输入框',
    preview: () => <ElInput placeholder="预览输入框"></ElInput>,
    render: ({ model }) => <ElInput placeholder="渲染输入框"></ElInput>,
    key: 'input',
    model: {
        default: '绑定字段'
        // default 等会绑定的model = {modelValue,onUpdate:modelValue}
    }
});
```

> 将用户填写的数据绑定到对应*block*的*model*属性上

```
if (component && component.model) {
    content.push(<>
        {Object.entries(component.model).map(([modelName, label]) => {
        return <ElFormItem label={label}>
            <ElInput v-model={state.editData.model[modelName]}>
</ElInput>
        </ElFormItem>
    })}
</>);
}
```

```
const RenderComponent = component.render({
    props: props.block.props,
    model: Object.keys(component.model || {}).reduce((prev, propName) =>
{
        const modelName = props.block.model[propName]; // 获取对应需要绑定
的字段
        prev[propName] = {
            modelValue: props.formData[modelName], // 进行数据绑定
            'onUpdate:modelValue': (val) => {
                props.formData[modelName] = val;
            }
        }; // 将model数据传递给渲染组件
        return prev; // 绑定的数据
    }, {})
});
```

> 绑定数据

```
render: ({ model }) => <ElInput placeholder="渲染输入框"
{...model.default}></ElInput>
```

2.范围选择数据绑定

```
registerConfig.register({
    label: '数字范围输入框',
    preview: () => <NumberRange style={{ width: '100%' }}>
</NumberRange>,
    render: ({ model, size }) => {
        return <NumberRange {...{
            start: model.start.modelValue,
            end: model.end.modelValue,
            'onUpdate:start': model.start['onUpdate:modelValue'],
            'onUpdate:end': model.end['onUpdate:modelValue'],
        }
        }></NumberRange>
    },
```

```
    model: {
        start: '开始绑定字段',
        end: '结束绑定字段'
    },
    key: 'range',
});
```

```
import deepcopy from "deepcopy";
import { defineComponent, computed } from "vue";
export const NumberRange = defineComponent({
    props: {
        start: { type: Number },
        end: { type: Number },
    },
    emits: ['update:start', 'update:end'],
    setup(props, ctx) {
        const start = computed({
            get() {
                return props.start;
            },
            set(newValue) {
                ctx.emit('update:start', deepcopy(newValue))
            }
        });
        const end = computed({
            get() {
                return props.end;
            },
            set(newValue) {
                ctx.emit('update:end', deepcopy(newValue))
            }
        });
        return () => {
            return <div class="number-range">
                <input type="text" v-model={start.value} />
                <span>~</span>
                <input type="text" v-model={end.value} />
            </div>
        }
    }
})
```

九.实现下拉框选项添加组件

1.注册下拉菜单组件

```
registerConfig.register({
    label: '下拉框',
    preview: () => <ElSelect modelValue=""></ElSelect>,
    render: ({ props }) => <ElSelect>
        {(props.options || []).map((opt, index) => {
            return <ElOption label={opt.label} value={opt.value} key=
{index}></ElOption>
        })}
    </ElSelect>,
    props: {
        options: createTableProp('下拉选项', {
            options: [
                { label: '显示值', field: 'label' },
                { label: '绑定值', field: 'value' },
            ],
            key: 'label' // 显示在页面上的用label属性
        })
        /**
         * [{label:'吃饭',value:'eat'},{label:'喝水',value:'drink'}]
         */
    }
})
```

```
<ElFormItem label={propConfig.label}>
    {{
        input: () => (<ElInput v-model={state.editData.props[propName]}
/>),
        color: () => (<ElColorPicker v-model=
{state.editData.props[propName]}></ElColorPicker>),
        select: () => <ElSelect v-model=
{state.editData.props[propName]}>
            {propConfig.options.map(opt => (
                <ElOption label={opt.label} value={opt.value}>
{opt.label}</ElOption>
            ))}
        </ElSelect>,
        // 渲染表格
        table: () => <TableEditor
            propConfig={propConfig}
            v-model={state.editData.props[propName]}>
        </TableEditor>
    }[propConfig.type]()}
</ElFormItem>
```

2.属性编辑菜单

```
export default defineComponent({
    props: {
        modelValue: { type: Array }, // 选项列表
        propConfig: { type: Object } // {options,key}
    },
    emits: ['update:modelValue'],
    setup(props, ctx) {
        // 获取用户选项
        const data = computed({
            get() {
                return props.modelValue || [];
            },
            set(newValue) {
                ctx.emit('update:modelValue', deepcopy(newValue))
            }
        });
        const add = () => {
            $TableEditor({
                config: props.propConfig,
                data: data.value,
                onConfirm(value) { // 确定的时候更新值
                    data.value = value;
                }
            })
        }
        return () => {
            return <div>
                {/* 如果用户选项没有则显示添加按钮 */}
                {(!data.value || data.value.length == 0) && <ElButton
onClick={add}>添加</ElButton>}
                {/* 如果有用户选项 */}
                {(data.value || []).map(item => <ElTag>
                    {item[props.propConfig.table.key]}
                </ElTag>)}
            </div>
        }
    }
})
```

3.注册表格服务

```
const TableComponent = defineComponent({
    props: {
        option: { type: Object },
    },
    setup(props, ctx) {
        const state = reactive({
```

```
            option: props.option,
            isShow: false,
            editData: []
        });
        const methods = {
            show(option) {
                state.option = option;
                state.editData = deepcopy(option.data)
                state.isShow = true;
            },
            hide() {
                state.isShow = false
            },
            add() {
                state.editData.push({})
            },
            reset: () => {
                state.editData = deepcopy(state.option.data)
            }
        }
        ctx.expose(methods);
        const onConfirm = () => {
            state.option.onConfirm(state.editData);
            methods.hide()
        }
        const onCancel = () => {
            methods.hide();
        }
        return () => {
            return state.isShow && <ElDialog v-model={state.isShow}>
                {{
                    default: () => (
                        <div>
                            <div>
                                <ElButton onClick={methods.add}>添加
</ElButton>
                                <ElButton onClick={methods.reset}>重置
</ElButton>
                            </div>
                            <ElTable data={state.editData}>
                                <ElTableColumn type="index">
</ElTableColumn>

{state.option.config.table.options.map((item, index) => {
                                    return <ElTableColumn label=
{item.label}>
                                        {{
                                            default: ({ row }) =>
<ElInput v-model={row[item.field]}></ElInput>
```

```
                                        }}
                                    </ElTableColumn>
                                })}
                                <ElTableColumn label={'操作'}>
                                    <ElButton type="danger">删除
</ElButton>
                                </ElTableColumn>
                            </ElTable>
                        </div>
                    ),
                    footer: () => <>
                        <ElButton onClick={onCancel}>取消</ElButton>
                        <ElButton onClick={onConfirm}>确定</ElButton>
                    </>
                }}
            </ElDialog>
        }
    }
})
let vm;
export const $TableEditor = (option) => {
    if (!vm) {
        const el = document.createElement('div')
        vm = createVNode(TableComponent, { option })
        let r = render(vm, el);
        document.body.appendChild(el);
    }
    let { show } = vm.component.exposed;
    show(option); // 重新再次渲染
}
```