前端搭建解决的问题

低代码开发平台(LCDP)：开发者不需要传统的手写代码的方式进行编程，采用图形化拖拽的方式，配置参数完成开发工作

低代码核心：降低重复劳动 （营销活动、中后台系统）

组成：组件区、编辑区/预览区、属性区/事件区。通过拖拽组件生成JSON，通过JSON渲染出页面 （采用JsonSchema描述页面）

可视化拖拽编辑器(一)

一.项目初始化

> 通过*Vue/cli*创建项目

```
vue create vue-visual-editor
? Please pick a preset: Manually select features
? Check the features needed for your project:
 (*) Choose Vue version
 (*) Babel
 ( ) TypeScript
 ( ) Progressive Web App (PWA) Support
 ( ) Router
 ( ) Vuex
>(*) CSS Pre-processors
 ( ) Linter / Formatter
 ( ) Unit Testing
 ( ) E2E Testing
```

```
? Please pick a preset: Manually select features
? Check the features needed for your project: Choose Vue version, Babel,
CSS Pre-processors
? Choose a version of Vue.js that you want to start the project with 3.x
(Preview)
? Pick a CSS pre-processor (PostCSS, Autoprefixer and CSS Modules are
supported by default): Sass/SCSS (with da
rt-sass)
? Where do you prefer placing config for Babel, ESLint, etc.? In
dedicated config files
? Save this as a preset for future projects? No
```

1.实现基本编辑器布局

```
<template>
  <div class="app">
    <Editor></Editor>
  </div>
</template>
<script>
import Editor from './components/editor';
export default {
  name: 'App',
  components: { Editor }
}
</script>
<style lang="scss">
.app {
  position: fixed;
  top: 20px;
  left: 20px;
  right: 20px;
  bottom: 20px;
  background: rgba(0, 0, 0, 0.3);
}
</style>
```

> *editor.js*

```
import { defineComponent } from "vue";
import './editor.scss'
export default defineComponent({
    setup() {
        return () => (
            <div class="editor">
                <div class="editor-left">左侧菜单</div>
```

```
                <div class="editor-top">上面控制条</div>
                <div class="editor-right">右侧控制栏</div>
                <div class="editor-container">
                    <div class="editor-container-canvas">
                        <div class="editor-container-canvas__content">
                            这是画页面的位置
                        </div>
                    </div>
                </div>
            </div>
        )
    }
})
```

editor.scss

```
.editor {
    width: 100%;
    height: 100%;

    &-left,
    &-right {
        position: absolute;
        width: 275px;
        top: 0;
        bottom: 0;
        background: red;
    }

    &-left {
        left: 0
    }

    &-right {
        right: 0
    }

    &-top {
        right: 275px;
        left: 275px;
        height: 75px;
        position: absolute;
        background: blue;
    }

    &-container {
        padding-top: 75px;
        padding-left: 275px;
```

```
        padding-right: 275px;
        height: 100%;
        box-sizing: border-box;
        background: yellow;

        &-canvas {
            overflow: scroll;
            height: 100%;
            &__content{
                position: relative;
                margin: 20px auto;
                background: yellow;
            }
        }
    }
}
```

2.定义JSON Schema

> 外部可以修改容器的宽高，*blocks*则为容器中放置的组件

```
<template>
  <div class="app">
    <Editor v-model="state"></Editor>
  </div>
</template>
<script>
import Editor from './components/editor.jsx';
import { reactive } from 'vue'
export default {
  name: 'App',
  components: { Editor },
  setup() {
    const state = ref({
      container: {
        height: 550,
        width: 550
      },
      blocks: [
        { top: 100, left: 200, key: 'button',"zIndex":0 },
        { top: 200, left: 200, key: 'input',"zIndex":0 }
      ]
    });
    return {
      state
    }
  }
}
```

```
</script>
```

> 通过*jsonSchema*渲染对应的结果

```
export default defineComponent({
    props: {
        modelValue: { type: Object } // 接收数据
    },
    emits: ['update:modelValue'],
    setup(props, ctx) {
        const data = computed({
            get() {
                return props.modelValue
            },
            set(newValue) {
                ctx.emit('update:modelValue',deepcopy(newValue))
            }
        });
        const containerStyles = computed(()=>({
            width:data.value.container.width + 'px' ,
            height:data.value.container.height + 'px'
        }))
        return () => (
            <div class="editor">
                <div class="editor-left">左侧菜单</div>
                <div class="editor-top">上面控制条</div>
                <div class="editor-right">右侧控制栏</div>
                <div class="editor-container">
                    <div class="editor-container-canvas">
                        <div class="editor-container-canvas__content"
style={containerStyles.value}>
                            {(data.value.blocks || []).map(block => (
                                <EditorBlock block={block}>
</EditorBlock>
                            ))}
                        </div>
                    </div>
                </div>
            </div>
        )
    }
})
```

3.渲染EditorBlock组件

```
import { computed, defineComponent } from "vue";

export default defineComponent({
    props: {
        block: { type: Object }
    },
    setup(props) {
        const blockStyles = computed(()=>({
            top:`${props.block.top}px`,
            left:`${props.block.left}px`,
            zIndex:`${props.block.zIndex}`,
        }))
        return () => (<div style={blockStyles.value} class="editor-
block">
            这是一个代码块
        </div>)
    }
})
```

> 我们需要将代码块根据*key*的不同渲染成不同的组件

4.配置组件

utils/editor-config.jsx

```
import { ElButton, ElInput } from 'element-plus'
function createEditorConfig() {
    const componentList = []; // 稍后用于渲染菜单的列表
    const componentMap = {}; // 组件的映射关系
    return {
        componentList,
        componentMap,
        register: ( component) => {
            componentList.push(component);
            componentMap[component.key] = component;
        }
    }
}
export const registerConfig = createEditorConfig(); // 注册组件
registerConfig.register({
    label: '文本',
    preview: () => '预览文本',
    render: () => '渲染文本',
    key: 'text'
})
registerConfig.register({
```

```
    label: '按钮',
    preview: () => <ElButton>预览按钮</ElButton>,
    render: () => <ElButton>渲染按钮</ElButton>,
    key: 'button'
})

registerConfig.register({
    label: '输入框',
    preview: () => <ElInput placeholder="预览输入框"></ElInput>,
    render: () => <ElInput placeholder="渲染输入框"></ElInput>,
    key: 'input'
})
```

```
import { registerConfig as config } from './utils/editor-config'
provide('config', config); // 将组件信息提供出去
```

```
import { computed, defineComponent,inject } from "vue";
export default defineComponent({
    props: {
        block: { type: Object }
    },
    setup(props) {
        const styles = computed(() => ({
            top: `${props.block.top}px`,
            left: `${props.block.left}px`
        }));
        const config = inject('config'); // 注入组件信息
        return () => {
            const component = config.componentMap[props.block.key];
            const RenderComponent = component.render(); // 找到对应组件进行
渲染
            return (<div style={styles.value} class="editor-block">
                {RenderComponent}
            </div>)
        }
    }
})
```

二.组件拖拽实现

1.渲染菜单列表

```
<div class="editor-left">
    {config.componentList.map(component => (
        <div class="editor-left-item">
            <span class="editor-left-item__label">{component.label}
</span>
            <div class="editor-left-item__content">{component.preview()}
</div>
        </div>
    ))}
</div>
```

```
&-left {
        left:0;
        &-item {
            width: 250px;
            margin:20px auto;
            display: flex;
            background: #fff;;
            padding: 30px 20px;
            box-sizing: border-box;
            cursor: move;
            user-select: none;;
            min-height: 100px;;
            align-items: center;
            justify-content: center;
            position: relative;
            > span{
                position: absolute;
                top:0;
                left:0;
                background:rgb(114, 166, 214);
                color:#fff;
                padding:4px;
            }
            &::after{
                content:'';
                position: absolute;
                top:0;
                left:0;
                right:0;
                bottom:0;
                background: #ccc;
                opacity: 0.1;
            }
        }
    }
```

2.实现拖拽功能

```
{/* 菜单中的组件进行拖动 */}
<div class="editor-left">
    {config.componentList.map(component => (
        <div class="editor-left-item" draggable
            onDragstart={(e) => menuDragger.dragStart(e, component)}
            onDragend={menuDragger.dragend}>
            <span class="editor-left-item__label">{component.label}
</span>
            <div class="editor-left-item__content">{component.preview()}
</div>
        </div>
    ))}
</div>
<div class="editor-container-canvas__content" ref={containerRef}>
    {(data.value.blocks || []).map(block => (
        <EditorBlock block={block}></EditorBlock>
    ))}
</div>
```

useMenuDragger.js

```
function useMenuDragger(containerRef, data) {
    let currentComponent = null;
    const dragenter = (e) => { e.dataTransfer.dropEffect = 'move' }
    const dragover = (e) => { e.preventDefault() }
    const dragleave = (e) => { e.dataTransfer.dropEffect = 'none' }
    const drop = (e) => {
        let blocks = data.value.blocks || [];
        data.value = {
            ...data.value, blocks: [...blocks,
            {
                top: e.offsetY,
                left: e.offsetX,
                zIndex:1,
                key: currentComponent.key
            }]
        }
    }
    const dragStart = (e, component) => { // 绑定拖拽事件
        console.log('start')
        containerRef.value.addEventListener('dragenter', dragenter);
        containerRef.value.addEventListener('dragover', dragover)
        containerRef.value.addEventListener('dragleave', dragleave);
        containerRef.value.addEventListener('drop', drop);
        currentComponent = component
    }
```

```
    const dragend = (e) => { // 解绑事件
        containerRef.value.removeEventListener('dragenter', dragenter);
        containerRef.value.removeEventListener('dragover', dragover)
        containerRef.value.removeEventListener('dragleave', dragleave);
        containerRef.value.removeEventListener('drop', drop);
        currentComponent = null
    }
    return {
        dragStart,
        dragend
    }
}
```

> 让新生成的物料居中

```
const drop = (e) => {
     let blocks = data.value.blocks || [];
     data.value = {
         ...data.value, blocks: [...blocks,
         {
             top: e.offsetY,
             left: e.offsetX,
             key: currentComponent.key,
             alignCenter:true, // 设置居中
             zIndex:0
         }]
    }
}
```

```
const blockRef = ref(null)
onMounted(() => {
    const { offsetWidth, offsetHeight } = blockRef.value;
    if(props.block.alignCenter){ // 只有初始化的时候，设置居中
        props.block.left = props.block.left - offsetWidth / 2;
        props.block.top = props.block.top - offsetHeight / 2;
        props.block.alignCenter = false
    }
});
const config = inject('config'); // 注入组件信息
return () => {
    const component = config.componentMap[props.block.key];
    const RenderComponent = component.render(); // 找到对应组件进行渲染
    return (<div style={styles.value} class="editor-block" ref=
{blockRef}>
        {RenderComponent}
</div>)
}
```

3.实现物料的选中

```
.editor-block {
    position: absolute;
    &::after {
        content: '';
        position: absolute;
        top: 0;
        left: 0;
        right: 0;
        bottom: 0;
    }
}
.editor-block-focus {
    &::after {
        border:1px dashed red
    }
}
```

```
<div class="editor-container-canvas" onMousedown={containerMousedown}>
    {/* 拖拽到此容器中 */}
    <div class="editor-container-canvas__content" ref={containerRef}>
        {(data.value.blocks || []).map(block => (
            <EditorBlock
                class={block.focus ? 'editor-block-focus' : ''}
                block={block}
                onMousedown={(e) => blockMousedown(e, block)}
                ></EditorBlock>
        ))}
    </div>
</div>
```

```
// 2.选中组件
const { focusData, containerMousedown, blockMousedown } = useFocus(data,
(e)=>{
    // 拖拽逻辑
    console.log('选中逻辑')
});
```

```
import {computed} from 'vue'
export function useFocus(data,callback) {
    const focusData = computed(() => {
        let focus = [];    // 获取焦点的列表
        let unfocused = []; // 失去焦点的列表
        (data.value.blocks || []).forEach(block => (block.focus ? focus
: unfocused).push(block));
```

```
        return { focus, unfocused }
    })
    const clearBlocks = () => {
        data.value.blocks.forEach(block => block.focus = false);
    }
    const containerMousedown = (e) => {
        e.stopPropagation();
        clearBlocks();
    }
    const blockMousedown = (e, block) => {
        e.preventDefault();
        e.stopPropagation();
        if(e.shiftKey){
            block.focus = !block.focus;
        }else{
            if(!block.focus){
                clearBlocks();
                block.focus = true;
            }else{
                block.focus = false
            }
        }
        callback(e);
    }
    return {
        containerMousedown,
        blockMousedown,
        focusData
    }
}
```

4.实现多个物料拖拽逻辑

```
const blockDragger = useBlockDragger(focusData);
```

```
export function useBlockDragger(focusData) {
    let dragState = {
        startX: 0,
        startY: 0
    }
    const onMousedown = (e) => { // 当按下后记录当前位置
        dragState = {
            startX: e.clientX,
            startY: e.clientY,
            startPos: focusData.value.focus.map(({ top, left }) => ({
top, left }))
        }
        document.addEventListener('mousemove', onMousemove)
```

```
            document.addEventListener('mouseup', onMouseup)
        }
        const onmMousemove = (e) => {
            let {clientX:moveX,clientY:moveY} = e;
            let durX = moveX - dragState.startX;
            let durY = moveY - dragState.startY;
            focusData.value.focus.forEach((block,index)=>{
                block.top = dragState.startPos[index].top + durX
                block.bottom = dragState.startPos[index].bottom + durY
            })
        }
        const onMouseup = (e) => { // 鼠标抬起后，解绑事件
            document.removeEventListener('mousemove', onMousemove);
            document.removeEventListener('mouseup', onMouseup)
        }
        return {
            onMousedown
        }
    }
```