附件1

****

**《Java设计模式》期终考试**

**软件设计文档**

**题 目 可视化搭建移动端H5页面编辑器**

**教 师** 吴育锋

**院 系** 工学院

**专 业** 计算机科学与技术

**班 级** 计算机181

**学 号** 18103330135

**姓 名** 张超哲

|  |  |  |
| --- | --- | --- |
| PPT讲解50% | 设计文档50% | 得分 |
|  |  |  |

二〇 　年　　月　　日

## 第一章 概述

1.1 课题简介

这是一个基于typescript + vue开发的支持可视化编辑、自由拖拽排版、实时预览的移动h5（画布可调）建站系统，类似于易企秀、兔展的h5在线制作编辑系统。

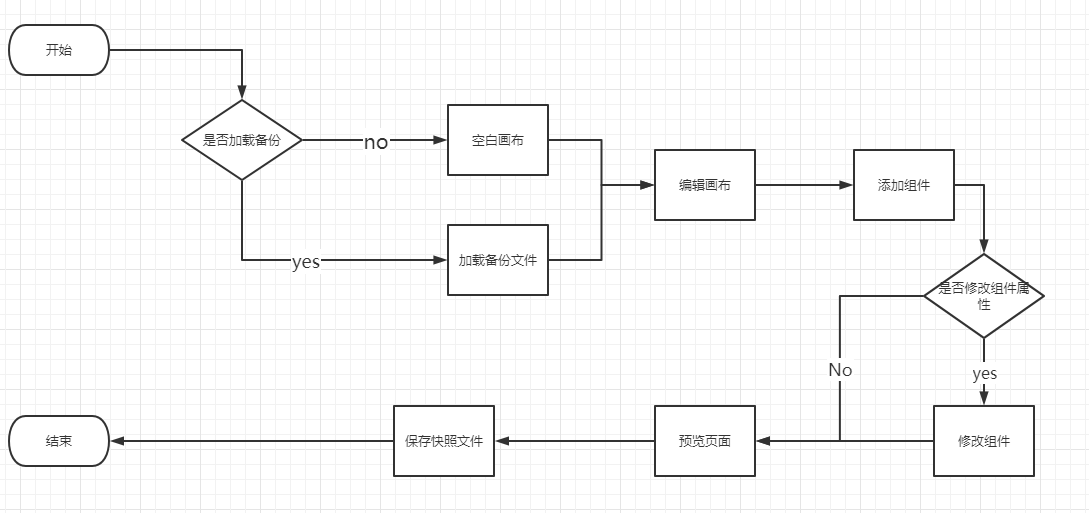
1.2 模式应用说明（说明设计模式会用在设计内容的哪些方面？）

作为一个编辑器，显然应该具有撤销、重做的功能，故**命令模式**必不可少，将命令封装成对象，每次访问逻辑层而不是直接在视图层去操作。

除去修改页面内容，用户不可能愿意每次都从零开始设计页面，那么就需要有存档读档的功能，结合**备忘录模式**，备忘录模式将创建状态快照（Snapshot）的工作委派给实际状态的拥有者原发器（Originator）对象。 这样其他对象就不再需要从“外部”复制编辑器状态了，编辑器类拥有其状态的完全访问权，因此可以自行生成快照。

项目提供了一系列的组件库，从侧边列表拖拽到画布上生成组件，在这过程中运用了**工厂模式**，不同的组件会调用不同的生成方法，然后再画布上将其实例化。

## 第二章 需求分析

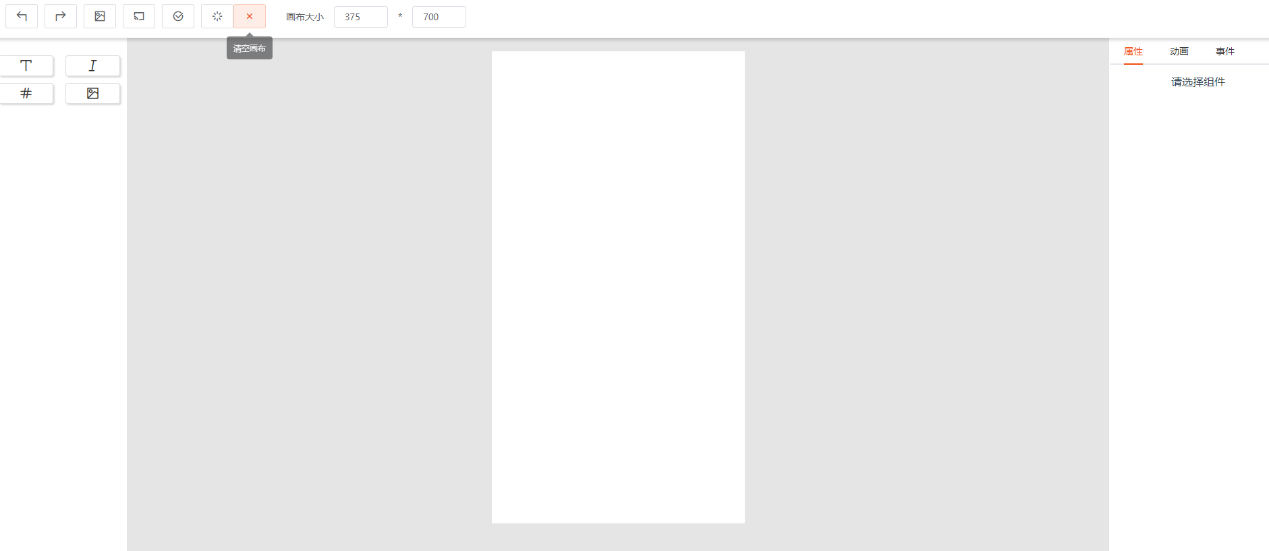
2.1 业务流程

2.2 功能模块

2.2.1 编辑器布局

顶部为编辑器的工具栏，包括撤销、重做、插入图片、预览、保存、读取清空等功能

左侧是已提供的组件，通过拖拽到中间画布，可以实例化组件

右侧是属性栏，能够操作所选择的组件的基础属性

2.2.2 拖拽

在画布中移动：

在画布中移动，需要监听鼠标事件，同时设置每个组件是absolute的position

1. mousedown ：鼠标（持续）按下，选中当前组件，记录当前位置状态
2. mousemove： 在鼠标按下状态移动鼠标，监听鼠标位置反馈给组件，从而动态更新组件位置
3. mouseup： 松开鼠标，结束移动，记录组件位置

从组件列表拖拽到画布上：

在页面中必须给元素添加一个 draggable 属性，才能实现拖拽效果。另外，在将组件列表中的组件拖拽到画布中，还有两个事件（具体可以查阅MDN有关Drag的文档）是起到关键作用的：

1. dragstart 事件，在拖拽刚开始时触发。它主要用于将拖拽的组件信息传递给画布。
2. drop 事件，在拖拽结束时触发。主要用于接收拖拽的组件信息。

此外还要设置鼠标事件的dataTransfer.dropEffect为copy，确保松开鼠标时，元素能在画布上

2.2.3 组件吸附+标线

所谓的组件吸附，就是在拖拽组件时，计算是否和另一个组件距离比较近的时候吸附在一起。



吸附的实现原理即确定一个diff值（例如4），如上图所示当移动按钮组件的时候，如果按钮的x坐标和下方组件的x坐标差小于diff值，则判定按钮坐标和下方相等，吸附功能对布局比较友好

标线用于对齐，在页面上创建 6 条线，分别是三横三竖

1. 上下方向的两个组件左边、中间、右边对齐时会出现竖线
2. 左右方向的两个组件上边、中间、下边对齐时会出现横线

2.2.4 放大缩小组件

在组件Component类外包装一层，也就是Shape类，不仅提供了移动组件的能力，也提供了8个坐标点（仅在选择当前组件时才显示），通过鼠标拖动能够改变组件大小



## 第三章 软件设计

3.1 设计类图

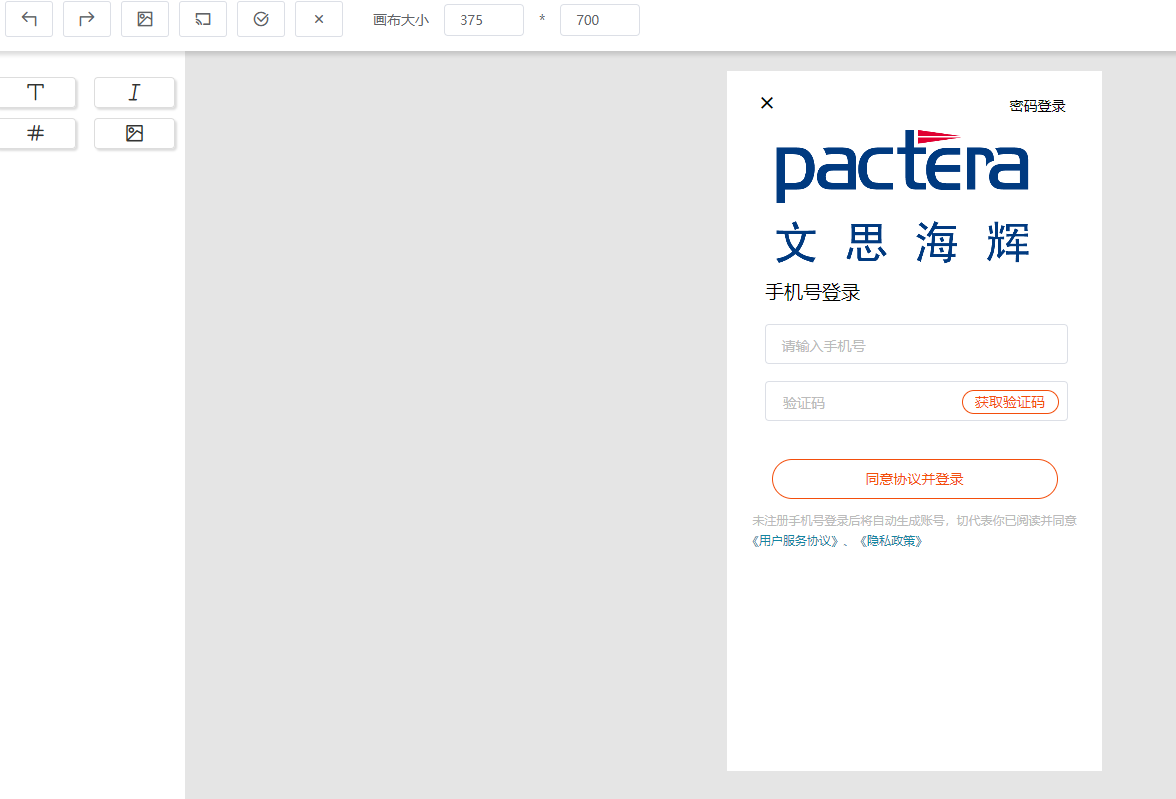
工厂模式（组件的生成）

命令模式（撤销、重做）

备忘录模式

3.2 设计模式的应用

…





## 第四章 源代码

4.1 关键类清单

4.1.1.Command类

CommandManger类

import { Editor } from '../Editor/Editor'

import { Command } from './Command'

import { AddCommand } from './CommonCommands'

class CommandManger {

redoStack: Array<any> = []

undoStack: Array<any> = []

editor: Editor

commandList: Array<any> = []

constructor(editor?: Editor) {

if (!editor) {

this.editor = new Editor()

} else {

this.editor = editor

}

this.resigterCommandClass(AddCommand)

}

execute(name: any, ...args: any[]) {

const CommandClass = this.commandList[name]

if (!CommandClass) throw new Error(`editor has not the ${name} command`)

const command = new CommandClass(this.editor, ...args) // 创建 command 实例

this.undoStack.push(command)

this.redoStack = []

}

undo() {

if (this.undoStack.length === 0) {

console.log('undo stack is empty, can not undo')

return

}

const command = this.undoStack.pop()

this.redoStack.push(command)

command.undo()

command.afterUndo()

}

redo() {

if (this.redoStack.length === 0) {

console.log('redo stack is empty, can not redo')

return

}

const command = this.redoStack.pop()

this.undoStack.push(command)

command.redo()

command.afterRedo()

}

resigterCommandClass(commandClass: any) {

const name = commandClass.name

this.commandList[name] = commandClass

}

}

export default CommandManger

4.1.2

4.2 代码清单

4.2.1. 深拷贝代码:

画布中任意组件的通用属性大概如下

{

component: 'umr-button',

label: '按钮',

propValue: '按钮',

icon: 'icon-hashtag',

animations: [],

events: {},

style: {

},

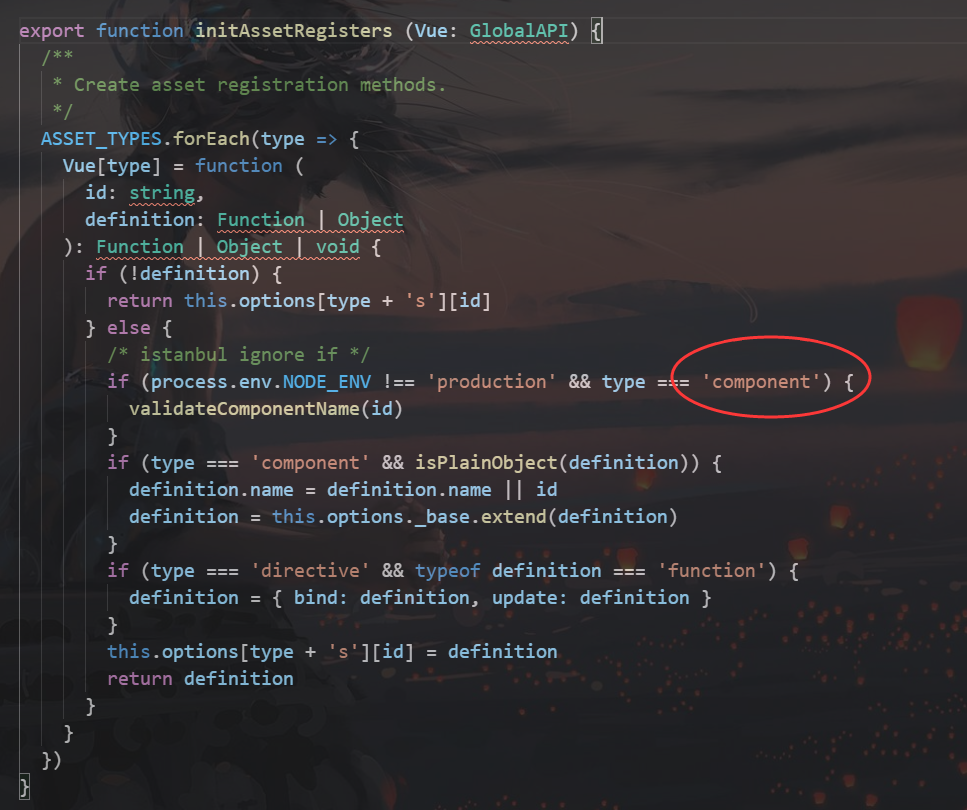
}

在画布中的 移动组件、修改组件属性、撤销、重做等操作就需要频繁的更新 componentData ，如果使用Object.assign()的浅拷贝只能拷贝一层的属性，难以对组件下的style拷贝



因此写了deepCopy函数递归的把target的拷贝对象返回

4.2.2. 解读Vue.component这个api的源码

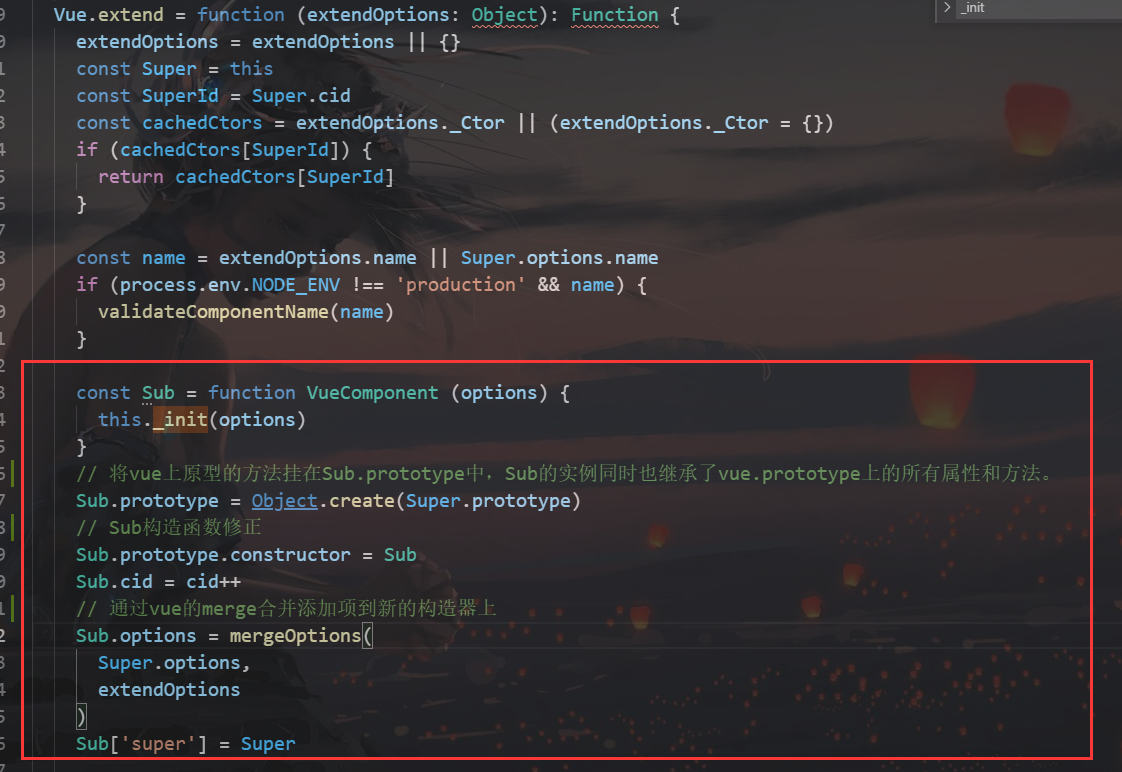


InitAssertRegisters这里包含了Vue.component和Vue.directive两个全局Api

众所周知的是，在vue中使用Vue.component('UmrText',UmrText)可以将UmrText注册为一个组件，可以在.vue文件中用<umr-text></umr-text> 来渲染

Vue.component(id, definition)中的definition是一个Object，也就是 .vue中default export 出来的一个对象

根据上图，使用vue.component会调用vue.extend(definition)，此时返回的definition已经不是object 而是一个 构造函数（一个组件实例的生成器）

extend的核心代码

<component

        class="component"

        :is="item.component"

        :style="getComponentStyle(item.style)"

        :propValue="item.propValue"

/>

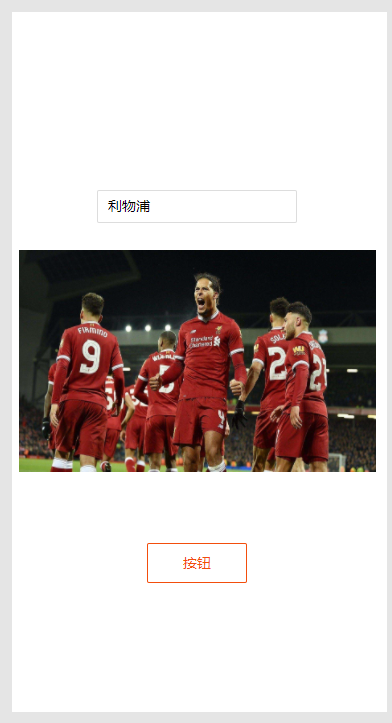
然后利用内置组件<component>,:is对应已注册组件的id，实现动态渲染组件（画布上的按钮、文字等组件）的效果

4.2.3. 工厂模式

**工厂方法模式**是一种创建型设计模式， 其在父类中提供一个创建对象的方法， 允许子类决定实例化对象的类型。

3.1.1 给定一个场景，即画布当前状态如下左图，

此时的画布中的内容与右图componentData对应





3.1.2 实现

需求即，将抽象的数据生成为具体的组件

遍历componentData,将传递 defaultStyle、style、active等给内部

通过item.component决定要生成的组件

<Shape

v-for="(item, index) in componentData"

:defaultStyle="item.style"

:style="getShapeStyle(item.style, index)"

:key="item.id"

:active="item === curComponent"

:element="item"

:zIndex="index"

>

<component

v-if="item.component != 'umr-text'"

class="component"

:is="item.component"

:style="getComponentStyle(item.style)"

:propValue="item.propValue"

/>

<component

v-else

class="component"

:is="item.component"

:style="getComponentStyle(item.style)"

:propValue="item.propValue"

@input="handleInput"

:element="item"

/>

</Shape>

4.2.4. 外观模式

**外观模式**是一种结构型设计模式， 能为程序库、 框架或其他复杂类提供一个简单的接口

前面说到工厂模式中，把<component>这个动态组件包裹在<Shape></Shape>中，通过遍历componentData渲染。如果纯粹是渲染组件，展示出来的就是静态页面了，而本项目是一个编辑器，给画布中的组件增加一系列可视化编辑的效果恰恰是**外观模式**的体现

Shape.vue 的html部分：

<template>

  <div

    class="baseEl"

    @click="selectCurComponent"

    @mousedown="handleMouseDown"

    @contextmenu="handleContextMenu"

>

    <div

      class="position-point"

      v-for="(item, index) in active ? pointList : []"

      :key="index"

      @mousedown="handleMouseDownPoint(item)"

      :style="getPointStyle(item)"

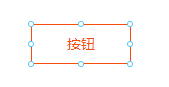
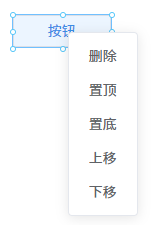
    ></div>

    <slot ></slot>

  </div>

</template>

<slot>插槽对应的是内部包裹的组件，外层的position-point给组件提供了用于可视化拉伸的接口

 最外层的baseEl提供了对应鼠标单击，鼠标按下，鼠标右键的接口，单击显示8个坐标点，鼠标按下（持续）用于移动组件，鼠标右键打开菜单

## 第五章 总结

注：可根据情况，调整目录结构。