

形状, 圆圈

描述已自动生成

**JISHOU　UNIVERSITY**

**专业综合实训报告**

|  |  |
| --- | --- |
| **题 目：** | 前端渲染框架 |
| **作 者：** | 杨武 |
| **学 号：** | 2020404112 |
| **所属学院：** | 计算机科学与工程学院 |
| **专业年级：** | 计算机科学与技术，2020计科1 |
| **完成时间：** | 2023年 11月23日 |

目录

[**JISHOU　UNIVERSITY** 1](#_Toc152087973)

[问题描述 4](#_Toc152087974)

[1.组件渲染问题 4](#_Toc152087975)

[1.1.组件是什么 4](#_Toc152087976)

[1.2.组件是怎么渲染的 4](#_Toc152087977)

[1.3.组件如何传值的 4](#_Toc152087978)

[1.4.组件是如何修改的 4](#_Toc152087979)

[1.5.组件是如何卸载的 4](#_Toc152087980)

[框架接口 4](#_Toc152087981)

[2.1.开发者如何在不同时期操控组件 4](#_Toc152087982)

[2,2,开发者如何确认组件渲染情况 5](#_Toc152087983)

[3.音乐平台 5](#_Toc152087984)

[3.1.何处理音频传输时延过大问题 5](#_Toc152087985)

[需求文档 5](#_Toc152087986)

[1.前端渲染框架 5](#_Toc152087987)

[1.1.页面渲染 5](#_Toc152087988)

[1.2.页面自动更新 5](#_Toc152087989)

[1.3.开发者插口 5](#_Toc152087990)

[1.4.反馈 5](#_Toc152087991)

[2.音乐平台 6](#_Toc152087992)

[2.1用户登录 6](#_Toc152087993)

[2.2用户注册 6](#_Toc152087994)

[2.3 查看歌单 6](#_Toc152087995)

[2.4 添加歌单 6](#_Toc152087996)

[2.5 删除歌单 6](#_Toc152087997)

[2.6 切换歌单 6](#_Toc152087998)

[3.歌曲管理 7](#_Toc152087999)

[3.1 添加歌曲 7](#_Toc152088000)

[3.2 下载歌曲 7](#_Toc152088001)

[3.3 播放歌曲 7](#_Toc152088002)

[3.4 添加歌曲至其他歌曲 7](#_Toc152088003)

[3.5 切换歌曲 7](#_Toc152088004)

[3.6 切换播放模式 7](#_Toc152088005)

[3.7 暂停歌曲 7](#_Toc152088006)

[3.8 播放上一曲歌曲 8](#_Toc152088007)

[3.9 播放下一首歌曲 8](#_Toc152088008)

[3.10 更改音乐播放进度 8](#_Toc152088009)

[概要设计文档 8](#_Toc152088010)

[1. 引言 8](#_Toc152088011)

[1.1 目的 8](#_Toc152088012)

[1.2 技术 8](#_Toc152088013)

[1.3 参考文献 8](#_Toc152088014)

[2. 系统架构 9](#_Toc152088015)

[2.1 总体架构 9](#_Toc152088016)

[2.2 模块划分 9](#_Toc152088017)

[3 数据流程 10](#_Toc152088018)

[3.1.渲染流程 10](#_Toc152088019)

[3.2.更新流程 10](#_Toc152088020)

[4 接口设计 10](#_Toc152088021)

[详细设计文档 10](#_Toc152088022)

[1.功能拆分 10](#_Toc152088023)

[2.流程图 12](#_Toc152088024)

[2.1组件渲染流程 12](#_Toc152088025)

[2.2组件更新逻辑 13](#_Toc152088026)

[2.3. 登录流程图 14](#_Toc152088027)

[组件解析 15](#_Toc152088028)

[3.1.词法解析 15](#_Toc152088029)

[3.2.语法解析 15](#_Toc152088030)

[3.3.自定义指令解析 15](#_Toc152088031)

[4.组件渲染 15](#_Toc152088032)

[4.1.组件渲染 15](#_Toc152088033)

[4.2.组件更新 16](#_Toc152088034)

[4.3.组件卸载 16](#_Toc152088035)

[5.数值代理 16](#_Toc152088036)

[1.proxy代理 16](#_Toc152088037)

[2.Watcher监听 16](#_Toc152088038)

[6.路由管理 17](#_Toc152088039)

[6.1.路由表 17](#_Toc152088040)

[6.2..路由 17](#_Toc152088041)

[7. 页面设计 17](#_Toc152088042)

[7.1.登录页面 17](#_Toc152088043)

[7.2.注册平台 18](#_Toc152088044)

[7.3.音乐平台 18](#_Toc152088045)

[8. 数据库设计 19](#_Toc152088046)

[8.1.用户表 19](#_Toc152088047)

[8.2.歌单表 19](#_Toc152088048)

[8.3.歌曲信息表 19](#_Toc152088049)

[8.3.歌曲表 19](#_Toc152088050)

[9. 安全设计 19](#_Toc152088051)

[测试文档 20](#_Toc152088052)

[1. 渲染器测试 20](#_Toc152088053)

[a) 渲染器渲染结点 20](#_Toc152088054)

[1.2. 渲染器更新结点 22](#_Toc152088055)

## 问题描述

## 1.组件渲染问题

### 1.1.组件是什么

组件是一个类，采用可面向对象的思想。我们在使用组件时，根据需要，生成不同的对象。每个对象拥有独立的数据，共用组件的函数。

### 1.2.组件是怎么渲染的

编辑器先读取组件，将其编译为js数据树，然后js数据树进行数据绑定生成proxy同时生生成节点，将节点挂载在父节点上。

### 1.3.组件如何传值的

组件传值的方式包括了父子传值，不同的组件传值。

1. 父子组件各有各的数据体。父组件的参数是写在子组件调用标签上的。在子组件生成时，数据会补上子组件中数据中props空位。同时生成proxy绑定关系，修改子组件属性是，会通过proxy修改父组件对应属性。

2. 不同组件传值，不同组件传值主要是用到了第三方proxy。

### 1.4.组件是如何修改的

Proxy中保留这组件渲染函数，绑定数据修改时会自动出发渲染函数。因为此时父节点有节点，判定为修改节点。此时对比组件数据和标签数据，有差异则修改。

### 1.5.组件是如何卸载的

组件的卸载组要是自定义指令v-if 和 v-for，v-if 和v-for 节点对比修改数据时发现节点消失，则会出发卸载函数unmount函数。

## 框架接口

### 2.1.开发者如何在不同时期操控组件

框架为开发者提供了一系列生命函数，在路由跳转前，路由跳转后，组件绑定数值前，组件绑定数值后，组件渲染后，组件更新后，组件卸载后等一系列生命周期函数。

### 2,2,开发者如何确认组件渲染情况

框架提供了报错、警告等一系列信息。在控制台，开发者可以得知组件渲染中信息。

## 3.音乐平台

### 3.1.何处理音频传输时延过大问题

采用音频切片，上传时音频切片。用户收听时一个个发过来。

# 需求文档

## 1.前端渲染框架

## 1.1.页面渲染

页面按照次序以此渲染

## 1.2.页面自动更新

用户修改绑定函数时，页面会自动修改

## 1.3.开发者插口

在页面渲染，更新，卸载的不同时段提供不同的入口，方便用户操作

## 1.4.反馈

组件语法错误，产生逻辑错误时，能够给予开发者合理的反馈

## 2.音乐平台

### 2.1用户登录

用户输入账号和密码并提交，账号和密码均正确之后跳入音乐平台主页面。若账号或密码不符合则停留在用户页面，应提示登陆失败。

### 2.2用户注册

用户输入账号和密码并提交，注册成功之后跳转到登录页面并提示注册成功。

（2）歌单管理

### 2.3 查看歌单

用户打开歌单列表并查询用户的所有歌单。

### 2.4 添加歌单

用户点击添加添加歌单按钮，输入歌单名称并提交。

### 2.5 删除歌单

用户点击对应歌单删除按钮，删除此歌单。

### 2.6 切换歌单

用户点击对应歌单，切换到对应歌单，歌曲列表切换。

## 3.歌曲管理

### 3.1 添加歌曲

用户点击添加歌曲，填入相关信息和音乐文件后提交。

### 3.2 下载歌曲

用户点击对应歌曲下载按钮并下载对应音乐的源文件。

### 3.3 播放歌曲

用户点击对应歌曲播放按钮并播放对应对应歌曲。

### 3.4 添加歌曲至其他歌曲

用户点击歌曲和待加入的歌单将该歌曲加入目标歌单。

### 3.5 切换歌曲

用户点击对应歌曲播放按钮并播放对应对应歌曲。

### 3.6 切换播放模式

用户点击播放模式并切换到下一个播放模式。

### 3.7 暂停歌曲

用户点击暂停播放按钮，歌曲停止播放，暂停图标变成待播放图标。

### 3.8 播放上一曲歌曲

用户点击上一曲按钮，开始播放上一首歌曲。

### 3.9 播放下一首歌曲

用户点击下一曲按钮，开始播放下一首歌曲。

### 3.10 更改音乐播放进度

用户点击进度条，音乐切换到对应进度并刷新进度条。

概要设计文档

## 1. 引言

### 1.1 目的

本项目旨在搭建一个前段渲染框架并在此基础上搭建一个在线音乐平台测试前段框架的性能。采用node.js模仿vue3的渲染逻辑，并对其部分逻辑进行一定的优化。在此基础上，搭建一个在线音乐平台，测试框架的性能。

### 1.2 技术

1.node.js

2.Spingboot

### 1.3 参考文献

1.《Vue.js设计与实现》

2.《基于Vue+SpringCloud博客的设计与实现》

## 2. 系统架构

### 2.1 总体架构

渲染框架采用MVVM的渲染架构，将视图（view）和模型（model）完全分离，通过一个视图模型（viewModel）来实现两者之间的交互。

在MVVM架构中，视图只负责显示用户界面，模型只负责存储数据，视图模型则负责处理业务逻辑和数据操作，并通过数据绑定和命令模式来与视图进行通信。

后端采用MVC的架构, [MVC模式中，Model（模型）表示应用程序核心（比如数据库记录列表）；View（视图）显示数据（数据库记录）；Controller（控制器）处理输入（写入数据库记录）。MVC模式定义Web应用程序带有三个逻辑层：业务层（模型逻辑）、显示层（视图逻辑）和输入控制（控制器逻辑）1。Model（模型）是应用程序中用于处理应用程序数据逻辑的部分。通常模型对象负责在数据库中存取数据1。View（视图）是应用程序中处理数据显示的部分。通常视图是依据模型数据创建的1。Controller（控制器）是应用程序中处理用户交互的部分。通常控制器负责从视图读取数据，控制用户输入，并向模型发送数据](https://www.runoob.com/aspnet/mvc-intro.html)。

### 2.2 模块划分

**1.解析器**

1解析器负责读取组件，并将组件编译成相应的虚拟DOM

**2,渲染器**

1. 渲染器负责将虚拟DOM和数据绑定
2. 渲染器负责生成对应的DOM，并绑定到父节点上
3. 渲染器负责更新组件
4. 渲染器负责卸载组件

**3,数据代理**

1. 数据代理负责在绑定数据触发时出发渲染器
2. 数据代理负责监听数据，监听数据变化时触发绑定函数

**4.路由**

1. 路由负责提供当前页面组件

2. 路由负责传值

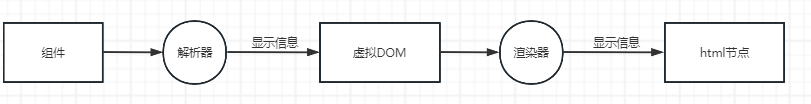
3. 路由负责控制是否有进入的权限

**5.音乐平台组件**

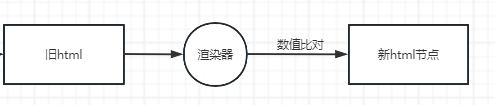
1.音乐平台细分为登录，注册，歌曲平台

## 3 数据流程

### 3.1.渲染流程



### 3.2.更新流程



## 4 接口设计

|  |  |  |  |
| --- | --- | --- | --- |
| 接口名称 | 接口参数 | 执行时间 | 备注 |
| BeforeEnterRouter | 无 | 进入路由之前执行 |  |
| AfterEnterRouter | 无 | 进入路由之后执行 |  |
| BeforeCreate | 组件虚拟DOM | 组件数据绑定之前执行 |  |
| Mouted | 组件挂载父节点 | 组件挂载之后执行 |  |
| UnMouted | 组件挂载父节点 | 组件卸载之后执行 |  |

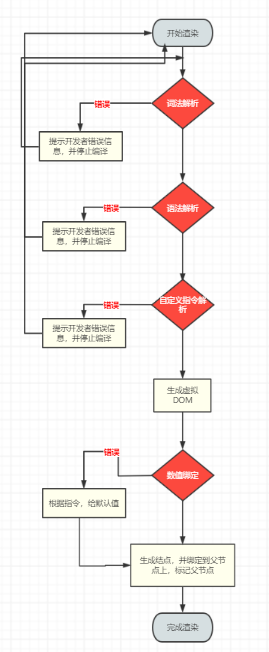
详细设计文档

## 1.功能拆分

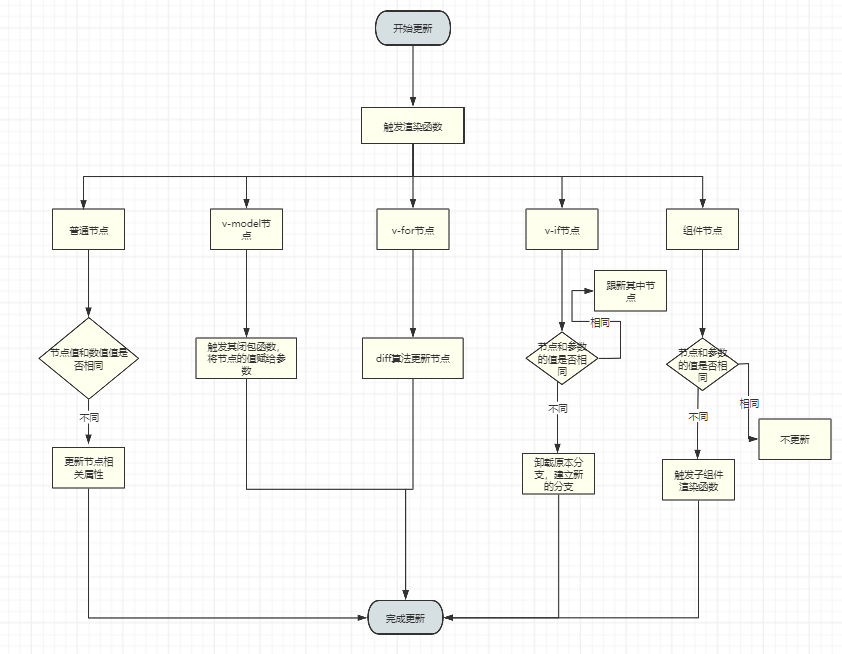
1. 前端动态编译框架功能要求：路由控制，缓存控制，动态编译，响应式更新，组件化开发
   1. 路由控制：开发者通过使用框架所提供函数修改路由，从而跳转页面
   2. 缓存控制：开发者通过使用框架所提供函数即可操作缓存数据
   3. 动态编译：开发者设定命令，编译时根据数据动态的生成节点
   4. 响应式更新：开发者设定响应式数据和组件内页面元素的绑定关系，修改响应式数据会触发组件更新，从而修改页面
   5. 组件化开发：开发者设定组件、组件和其他组件数据和逻辑关系、组件内的数据，编译时组件会读取组件，导入外部数据和逻辑关系，从而生成一个具有单独逻辑单元的一段html代码
2. 应用要求功能：用户和管理员注册登录、创建歌单，修改歌单名称，删除歌单。上传歌曲，下载歌曲，播放歌曲，修改歌曲所属的歌单
   1. 管理员可以查看和删除所有歌单和歌曲，而用户只有使用的功能。
   2. 上传歌曲：会采取转码，切片上传方式，来保证用户收听时的稳定性和快速性。

## 2.流程图

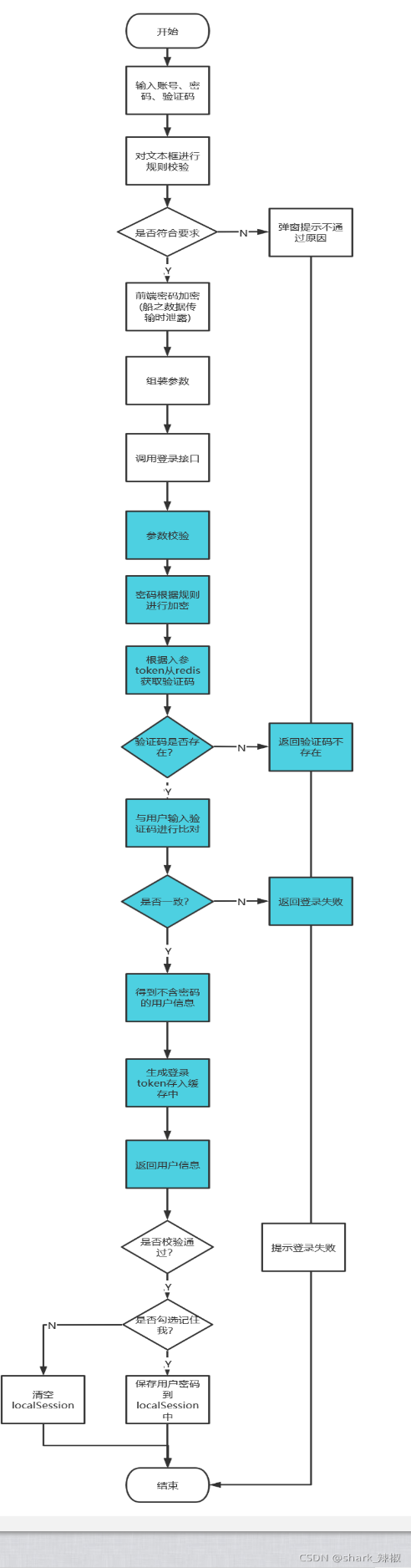
### 2.1组件渲染流程



### 2.2组件更新逻辑



### 2.3. 登录流程图



## 组件解析

### 3.1.词法解析

词法解析中状态机一共有四种状态，根据读取组件模板的字符而改变状态机状态。

状态1：标签开始

状态2：标签简单属性

状态3：标签复杂属性

状态4：标签闭合

词法解析读取组件模板并生成属性数组

### 3.2.语法解析

语法解析负责解析属性数组，并生成语法树。

语法解析便利属性数组，遇到状态1则递归一层，状态四则返回上一层。

状态二和状态三则添加属性至attr属性

### 3.3.自定义指令解析

自定义指令解析器负责解析v-if、v-for、v-model、{{}}四种自定义指令，并生成模板树

自定义指令解析器先序遍历语法树，对于自定义指令有不同的处理逻辑

1. v-for指令

为其附加动态节点标记和v-for标记，新建一个区域结点将其包裹，将其区域结点添加到动态结点库中

1. v-if指令

为其附加动态节点标记和v-if标记，新建一个区域结点将其包裹，将其区域结点添加到动态结点库中

1. v-model指令

为其附加动态节点标记和v-model标记，并将该节点添加到动态节点库中

1. {{}}指令

为其附加动态节点标记，将其区域结点添加到动态结点库中

## 4.组件渲染

### 4.1.组件渲染

组件先序遍历模板树，在其中进行数值绑定，生成虚拟DOM和结点，结点绑定到父节点上，虚拟DOM保存父节点的属性data\_visual\_dom中

组件渲染中面对不同的标签有不同的处理逻辑

**1.普通结点:** 普通结点指的除下列特殊结点外的结点，直接生成结点并返回

**2.v-if结点**：v-if结点根据逻辑判断渲染的分支，生成对应分支结点并返回

3.**v-for结点**：v-for结点遍历数组，并产生分支，生成对应的结点数组并返回

4.**v-mode结点**：v-model使用闭包包裹监听函数，生成结点并返回

5.**{{}}结点**：数值代替占位符，生成结点后返回

6.组件结点：数值代替占位符最为标签属性，再将其保存再props，触发子组件渲染函数

### 4.2.组件更新

绑定数据变化是，数据代理自动触发渲染器，渲染器触发组件更新函数，更新函数从父节点属性data\_visual\_dom获取虚拟dom。将虚拟dom的动态结点库和组件数据对比更新。

组件更新中面对不同的标签有不同的处理逻辑

**1.v-if结点**：v-if结点根据逻辑判断渲染的分支，若分支不变则更新分支。分支改变则卸载原分支，再挂载新分支。分支消失则卸载原分支。

2.**v-for结点**：v-for结点采用快速diff算法对比前后数组变化

3.**v-mode结点**：判断绑定函数时候变化，变化给闭包新监听函数，再赋值。

4.**{{}}结点**：数值代替占位符。

5.组件结点：一次遍历数组属性，判断数值是否变化。变化则修改子组件props数值，再触发子组件更新函数。

### 4.3.组件卸载

组件卸载从底至上卸载子组件，并以此删除父节点data\_visual\_dom属性

## 5.数值代理

### 1.proxy代理

Proxy代理数据，采用set和get函数，可以在绑定元素赋值和被访问时触发。绑定数据被访问时，将对应组件渲染函数保存到WeakMap-Map-Set的三层结构中，写作WeakMap(data, Map(属性名 Set(渲染函数)))，WeakMap保存组件的数值data，Map的键值对key(data属性名)—value(Set(渲染函数))，Set中保留渲染函数。

绑定数据被赋值时，获取该Set，并遍历Set依次触发渲染函数。

### 2.Watcher监听

Watcher参数为数据体（object）, 数据体属性名，绑定函数。会将该绑定函数以闭包的方式保存。Watcher触发时，会先触发绑定函数，再赋值。

## 6.路由管理

系统进入时先生成路由，路由采用单例模式。

### 6.1.路由表

路由表的结构为

{

path: '/',

component: LoginView,

meta: {

title: '欢迎登陆',

},

children: []

}

Path为路径，component为页面组件，meta为附加信息，childrem为子路由。

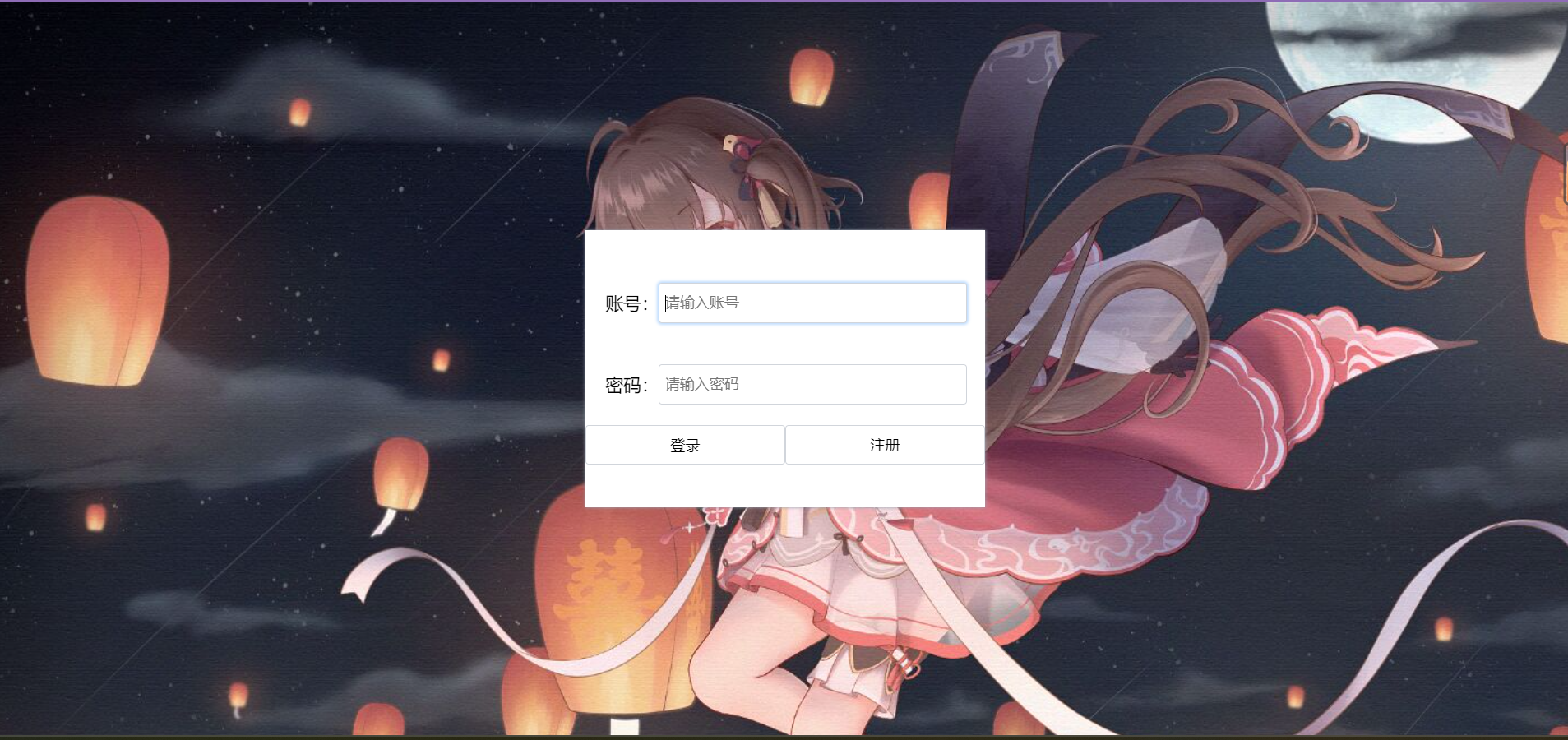
路由表为层级结构

### 6.2..路由

路由读取路由表并生成对象，该对象保留当前路由链接，并更具链接从左到右返回路由表中对应的路由组件。

## 7. 页面设计

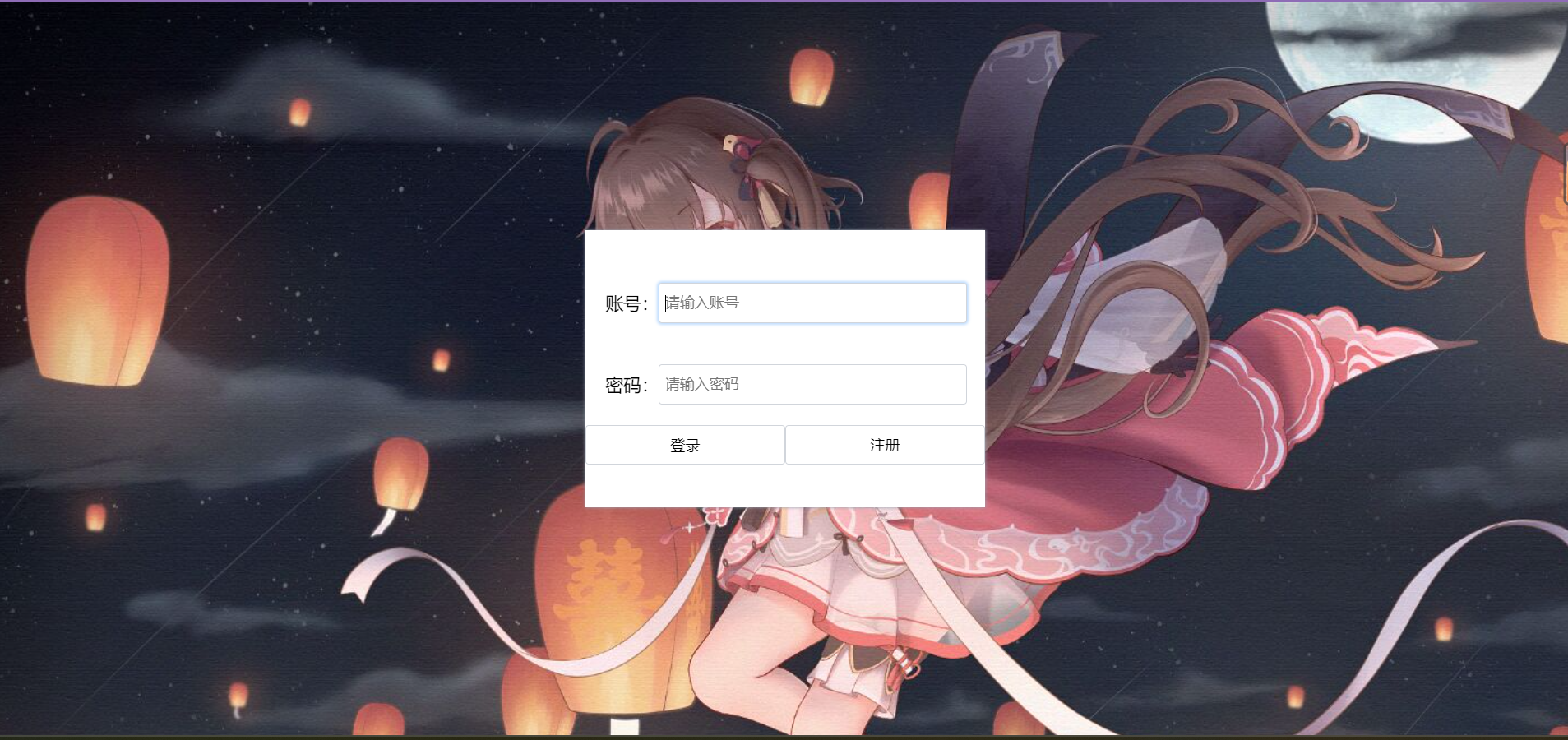
### 7.1.登录页面



整体采用了表单白色居中的设计

登陆时验证进行表单验证，用户的账号和密码需在6位以上

### 7.2.注册平台



整体采用了表单白色居中的设计

注册时验证进行表单验证，用户的账号和密码需在6位以上，并且两次密码需要相同

### 7.3.音乐平台



整体采用半透明居中设计

整体为黑色，字体为白色，按钮为红色和半白色。通过白色透明度创造不同的区域，整体给人一种磨砂的质感

## 8. 数据库设计

### 8.1.用户表

|  |  |  |
| --- | --- | --- |
| 列名 | 类型 | 备注 |
| Id | int | 主键 |
| username | Varchar(255) |  |
| password | Varchar(255) |  |
| salt | Varchar(255) |  |

### 8.2.歌单表

|  |  |  |
| --- | --- | --- |
| 列名 | 类型 | 备注 |
| Id | int | 主键 |
| name | Varchar(255) |  |
| uid | int | 外键，绑定用户表id |

### 8.3.歌曲信息表

|  |  |  |
| --- | --- | --- |
| 列名 | 类型 | 备注 |
| Id | int | 主键 |
| SongName | Varchar(255) |  |
| author | Varchar(255) |  |
| time | int |  |

### 8.3.歌曲表

|  |  |  |
| --- | --- | --- |
| 列名 | 类型 | 备注 |
| sid | int | 外键，绑定歌曲信息id |
| song | longtext |  |

## 9. 安全设计

1.前段路由做了路由守卫，根据用户是否登录和用户类型来判断用户是否能进入

2.后端采用了shiro，用户注册时shiro加密用户

7. 性能设计

后端采用redis缓存，保存用户信息

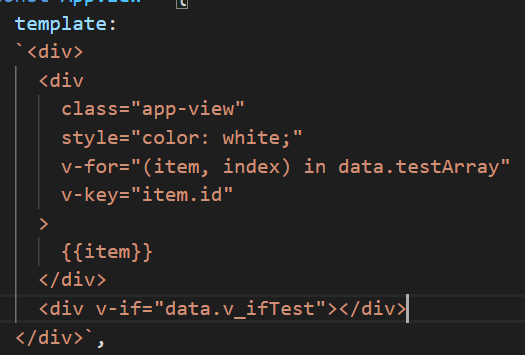
前端采用浏览器缓存，用户最近的音频文件保留在浏览器中，减少对服务器的访问

测试文档

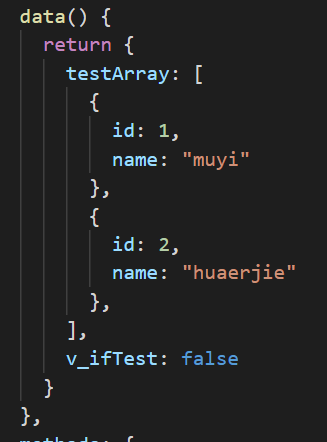
## 渲染器测试

### 渲染器渲染结点

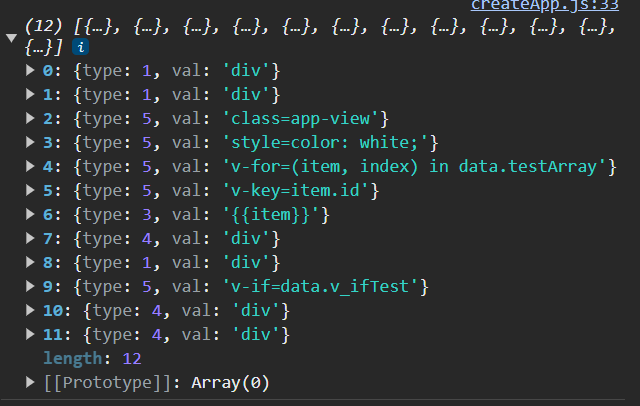
组件模板：



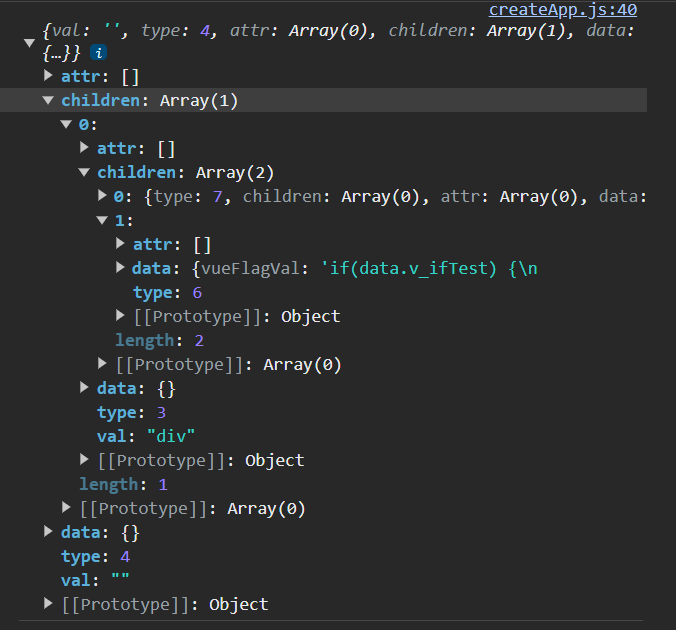
组件数值：



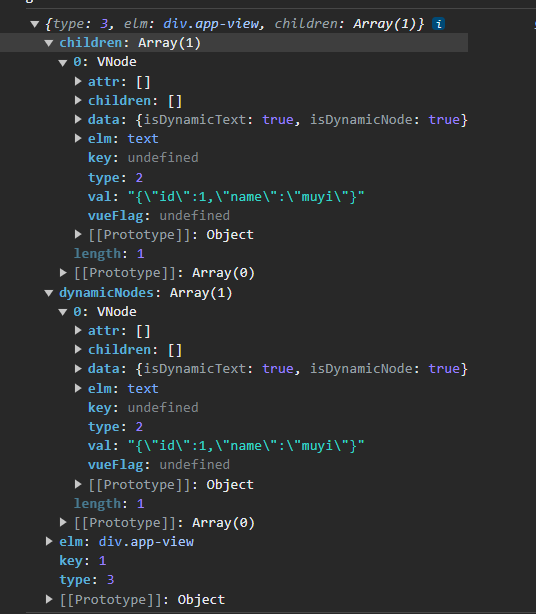
词法分析结构：



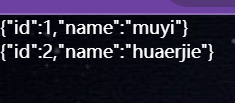
语法分析结构：



自定义指令结构：



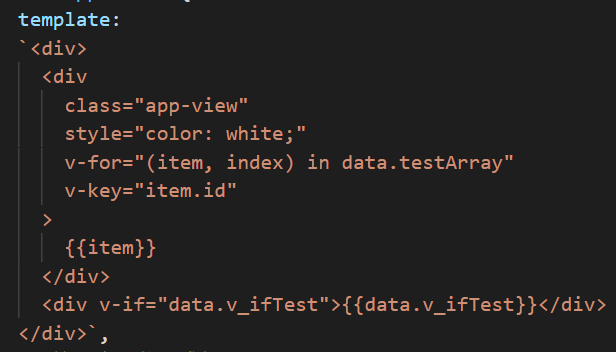
效果展现：



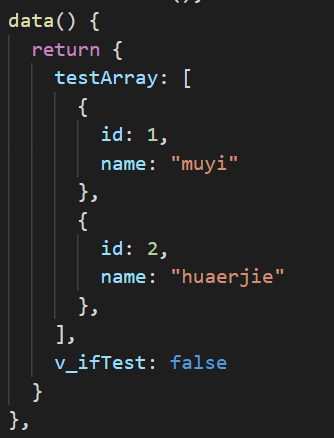
### 渲染器更新结点

1. **测试v-if**

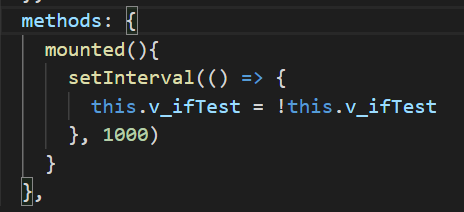
模板：



数据：

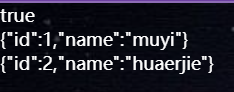


函数：

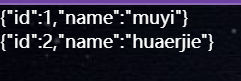


渲染效果：

开始：



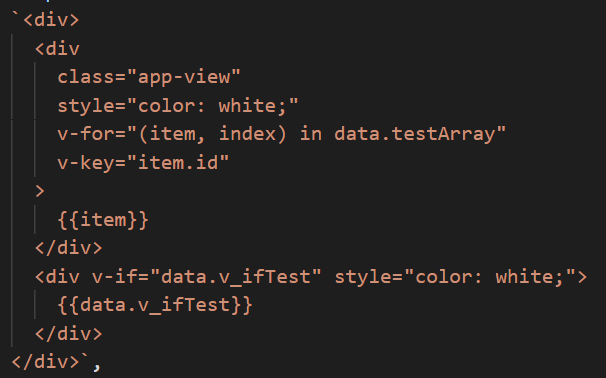
1s后：



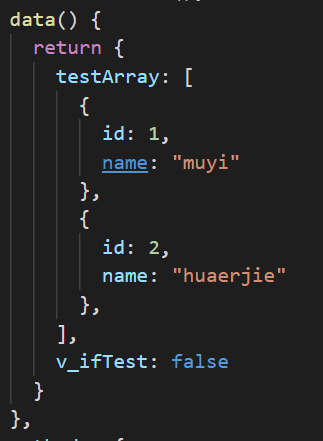
结论：v-if分支随绑定值改变渲染分支改变。

1. **测试v-for:**

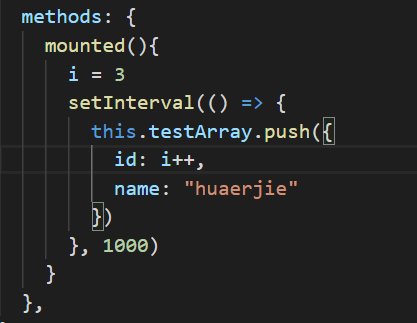
组件模板：



组件数据：



渲染函数:

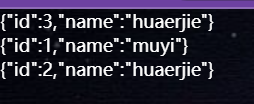


渲染效果：

刚开始



1s后：



2s后：

