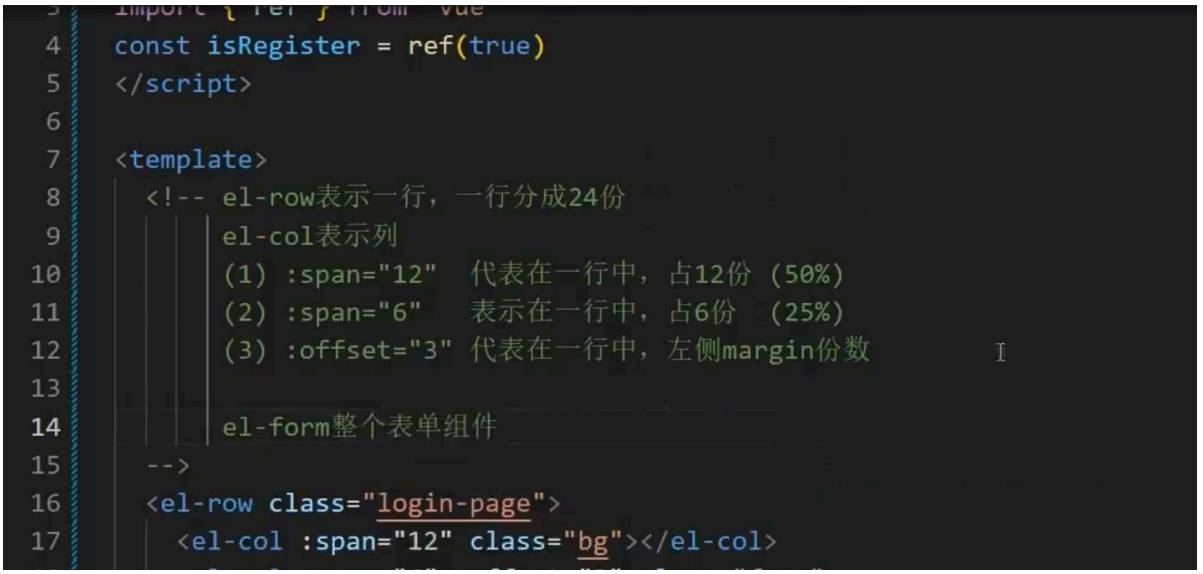
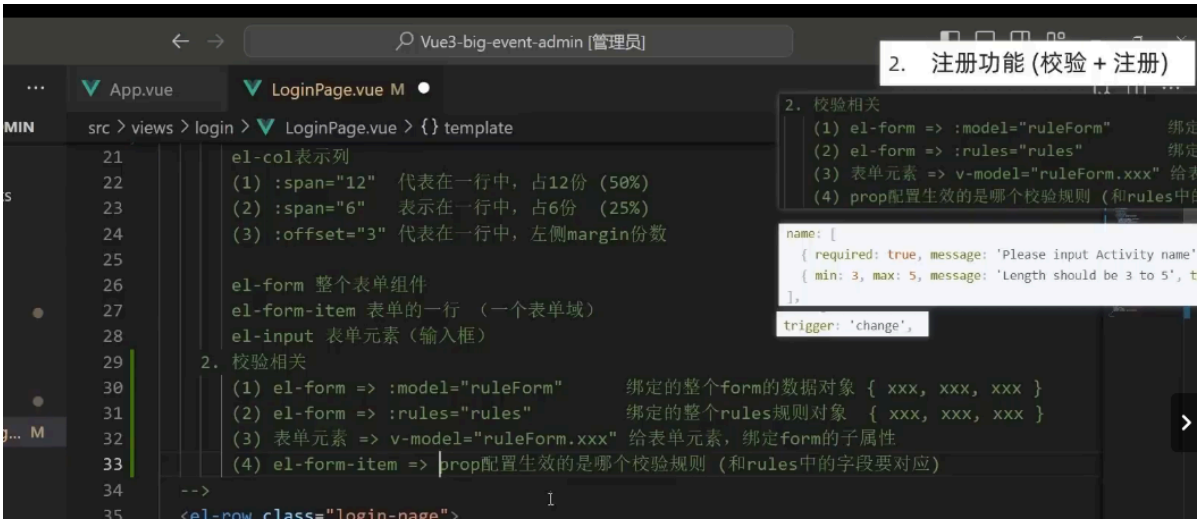
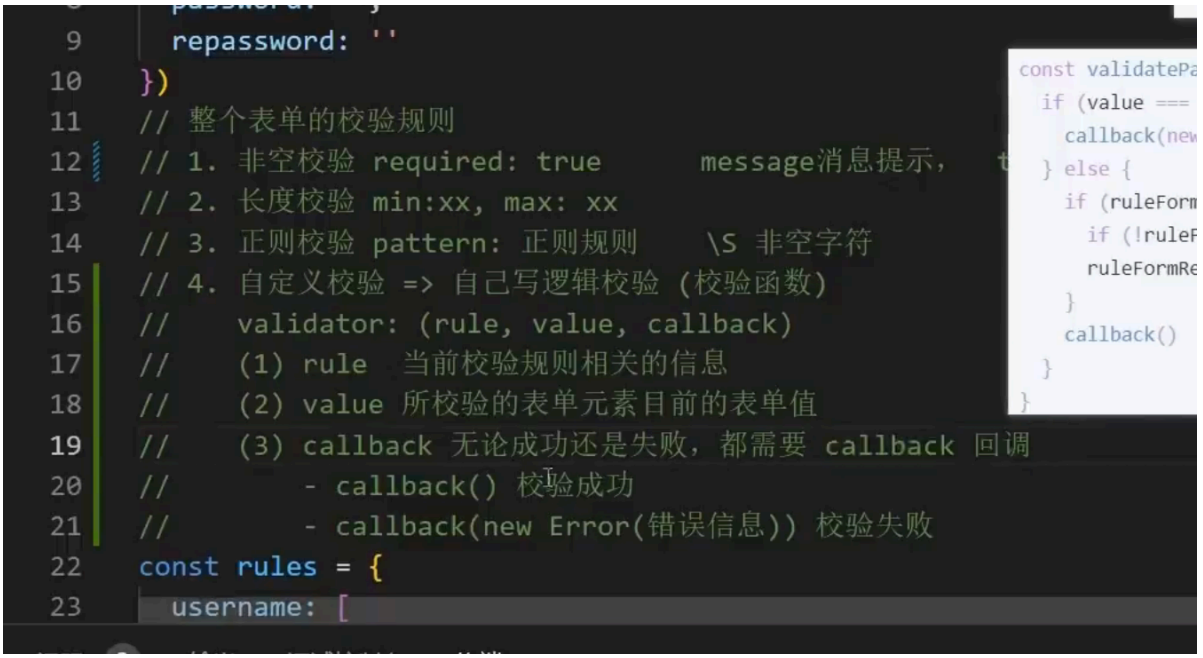
# el-row e-col


const isRegister = ref(true)
</script>
<template>
<!-- el-row表示一行，一行分成24份
el-col表示列
(1) :span="12" 代表在一行中，占12份 (50%)
(2) :span="6" 表示在一行中，占6份 (25%)
(3) :offset="3" 代表在一行中，左侧margin份数
el-form整个表单组件
-->
<el-row class="login-page">
<el-col :span="12" class="bg"></el-col>


Vue3-big-event-admin [管理员]
App.vue
LoginPage.vue M
src > views > login > LoginPage.vue > {} template
el-col表示列
(1) :span="12" 代表在一行中，占12份 (50%)
(2) :span="6" 表示在一行中，占6份 (25%)
(3) :offset="3" 代表在一行中，左侧margin份数
el-form 整个表单组件
el-form-item 表单的一行 （一个表单域）
el-input 表单元素（输入框）
2. 校验相关
(1) el-form => :model="ruleForm" 绑定的整个form的数据对象 { xxx, xxx, xxx }
(2) el-form => :rules="rules" 绑定的整个rules规则对象 { xxx, xxx, xxx }
(3) 表单元素 => v-model="ruleForm.xxx" 给表单元素，绑定form的子属性
(4) el-form-item => prop配置生效的是哪个校验规则 (和rules中的字段要对应)
2. 注册功能 (校验 + 注册)
trigger: 'change',


repassword: ''
})
// 整个表单的校验规则
// 1. 非空校验 required: true message消息提示,
// 2. 长度校验 min:xx, max: xx
// 3. 正则校验 pattern: 正则规则 \S 非空字符
// 4. 自定义校验 => 自己写逻辑校验 (校验函数)
// validator: (rule, value, callback)
// (1) rule 当前校验规则相关的信息
// (2) value 所校验的表单元素目前的表单值
// (3) callback 无论成功还是失败，都需要 callback 回调
// - callback() 校验成功
// - callback(new Error(错误信息)) 校验失败
const rules = {
username: [

# Router 拦截

```
10  const router = createRouter({ ···
42  })
43
44  // 登录访问拦截 => 默认是直接放行的
45  // 根据返回值决定，是放行还是拦截
46  // 返回值:
47  // 1. undefined / true  直接放行
48  // 2. false 拦回from的地址页面
49  // 3. 具体路径 或 路径对象  拦截到对应的地址
50  //     '/login'    { name: 'login' }
51  router.beforeEach(() => {
52    if
53  })
54
```

# await 作用

```
21  })
22  const handleCommand = async (key) => {
23    if (key === 'logout') {
24      // 退出  await: 只有成功了才会执行下面内容
25      await ElMessageBox.confirm('确定退出吗? ', '提示', {
26        type: 'warning',
27        confirmButtonText: '确认',
28        concelButtonText: '取消'
29      })
30      userStore.removeToken()
31      router.push('/login')
32    } else {
33      router.push(`/user/${key}`)
34    }
35  }
```

# async和await的作用

## async

- `async`是一个用于函数前的关键字，用于声明一个异步函数。
- 异步函数意味着函数的执行不会阻塞其他代码的执行，它会立即返回一个Promise对象。
- 如果异步函数返回一个值，那么Promise会立即以该值resolve。
- 如果异步函数抛出异常，Promise会以该异常reject。

例子:

```
async function fetchData() {
  // ...异步操作
  return data;
}
```

## await

- `await`关键字用于等待一个Promise完成，它只能在`async`函数内部使用。
- 使用`await`可以暂停当前`async`函数的执行，等待右侧的表达式（通常是一个返回Promise的函数调用）返回一个结果或者抛出异常。
- 如果等待的Promise被resolve，`await`表达式返回的是resolve的值。
- 如果Promise被reject，`await`会抛出一个错误，通常需要用try...catch语句来捕获处理。

例子：

```
async function getData() {
  try {
    const response = await fetch('url'); // 等待fetch请求完成
    const data = await response.json();  // 等待将响应转换为JSON
    return data;
  } catch (error) {
    console.error('Error fetching data: ', error);
  }
}
```