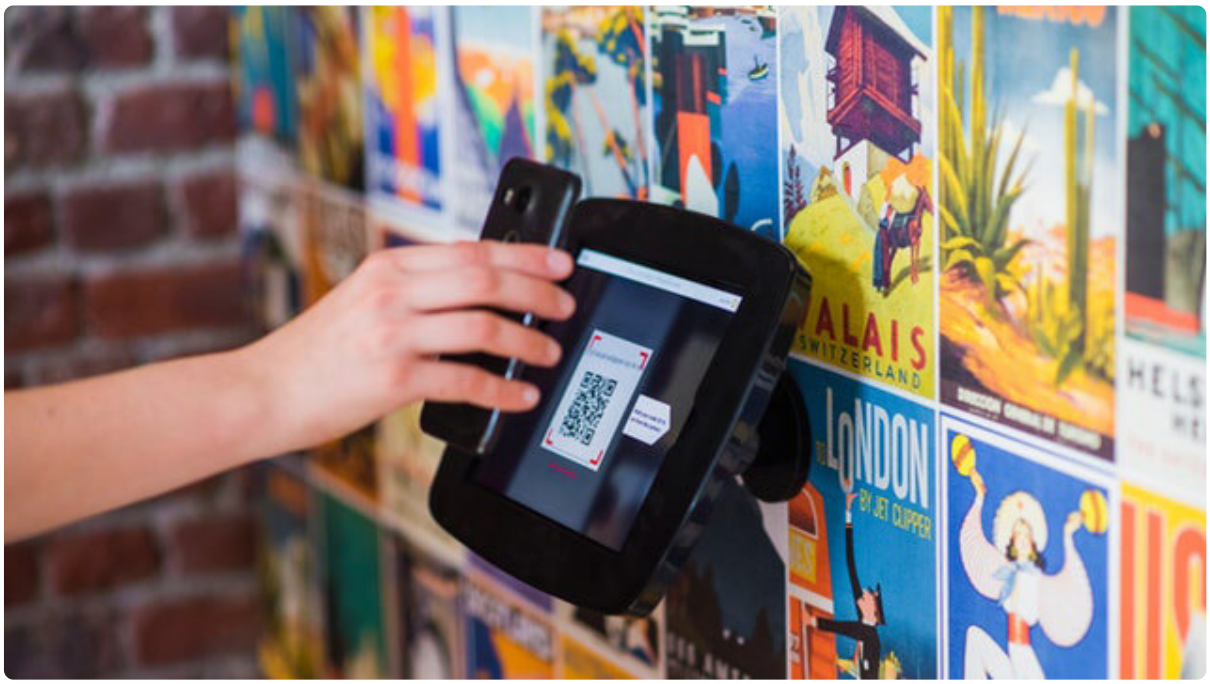
实现登录页并用Mock响应请求

更新时间：2019-07-26 09:34:06



> 勤学如春起之苗，不见其增，日有所长。
>
> ——陶潜

本小节我们来开发一个登录页，调用api请求，然后使用mockjs模拟请求响应，来返回登录成功与否的状态。登录成功之后，我们会跳到后台系统首页。如果没有登录，无论访问后台系统哪个页面的url，都会跳到登录页。接下来我们开始学习。

首先添加一个登录页，在*src/views*文件夹下创建一个*login*文件夹，所有和login页相关的文件都放到这个文件夹内，然后在这个文件夹创建一个*index.tsx*文件。因为项目默认添加的两个页面*About.vue*和*Home.vue*都是使用Vue单文件.vue后缀的形式，所以我们添加一个以 .tsx 为后缀的文件，来学习如何使用tsx写法写vue。

```
// src/views/login/index.tsx
import { Component, Vue } from 'vue-property-decorator'

@Component
export default class LoginPage extends Vue {
  protected render() {
    return (
      <div>login</div>
    )
  }
}
```

我们这里用到一个新的依赖"vue-property-decorator"，你需要先安装它，然后在这里引入，我们这里暂时只用到 Component 和 Vue，Component是一个装饰器工厂函数，可以用来修饰我们要作为组件实例的类的定义。它可以传入一个对象参数，用来配置组件的一些信息，也可以不传参数，我们待会儿会补充一些参数，现在我们先使用默认参数。

使用Component装饰器修饰之后，这个类LoginPage的定义再继承Vue，它的定义就包含了一些组件需要的属性等，然后我们这里就可以定义一个render方法，用来书写渲染的dom内容。这个方法名是固定的，Vue会拿到这个方法返回的内容去渲染实际的DOM。

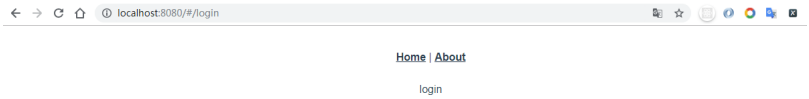
注意它的写法，`return`返回的内容用括号包住，然后里面就像写html一样写html标签，还可以写组件，当然JSX语法内容不是简答几句话可以概括完的，我们一点一点来看。这里我们就只渲染一个div标签，然后里面包含字符串"login"。

> 这里还有一点要特别注意，一个组件最外层只能由一个节点包裹，也就是这里你要将所有内容都包在这个`<div>`标签里，不能有和这个`<div>`标签同级的第二个标签了。

接下来这个组件就可以用了，我们定义一下路由，使得我们可以把这个组件作为一个页面访问。
在*src/router/routes.ts*文件的路由配置里，增加一个：

```
// ...
export default [
  // ...
  {
    path: '/login',
    name: 'login',
    component: () => import('@/views/login/index'),
  },
]
```

现在你可以在浏览器里访问login页面啦，默认情况下本地起的服务路径为`http://localhost:8080/#/login`，如果你起了别的服务，可能端口号8080会是别的，你可以直接在控制台查看服务的url。打开这个url之后，你会看到页面上有login这个词，说明你前面的这些步骤没有疏漏，如果出现问题，需要你仔细对照一下上面讲解的步骤啦。



接下来我们要添加两个输入框，一个用来输入用户名，一个用来输入密码。再添加一个按钮，用来请求登录api，我们对前面render方法做一些补充：

```
// src/views/login/index.tsx
// ...

@Component
export default class LoginPage extends Vue {
  public user_name: string = ''
  public password: string|number = ''

  protected render() {
    return (
      <div class='login-page'>
        <input v-model={ this.user_name }/>
        <p>输入的用户名是：{ this.user_name }</p>
      </div>
    )
  }
}
```

这里我们定义了两个实例属性user_name和password，用来指定它们的类型和初始值，并且保存输入的用户名和密码。这里我们先用只添加一个用户名输入框，好让大家先看下基础的JSX语法在Vue中的运用。

我们这里渲染了一个div标签，给它指定一个类名"login-page"。在它里面有两个标签，一个是input标签，我们使用Vue的双向绑定v-model指令绑定一个user_name变量，注意这里不像Vue模板语法那样写 v-model="user_name"，而是使用花括号 {} 代替引号，而且这里要访问的是实例上的属性，所以要写 this.user_name 。

> 在JSX中，需要使用实例变量或者要书写JS表达式的，都要用花括号 {} 括起来。

然后我们在p标签里输入一行文字："输入的用户名是："，后面来显示我们输入框中输入的内容，可以猜到这里要显示输入框输入的内容，也就是显示实例属性 user_name 的值，它是一个JS变量，所以我们需要使用花括号{}括起来。现在你再在浏览器中看下效果，你在input输入框内输入什么，下面就会对应显示什么。

接下来我们把密码输入框和登录按钮也补上：

```
// ...
export default class LoginPage extends Vue {
  public user_name: string = ''
  public password: string|number = ''

  public login() {
    //
  }

  protected render() {
    return (
      <div class='login-page'>
        <input v-model={ this.user_name }/>
        <input v-model={ this.password } type='password' style='margin-left: 10px;'/>
        <button style='margin-left: 10px;' on-click={ this.login }>登录</button>
      </div>
    )
  }
}
```

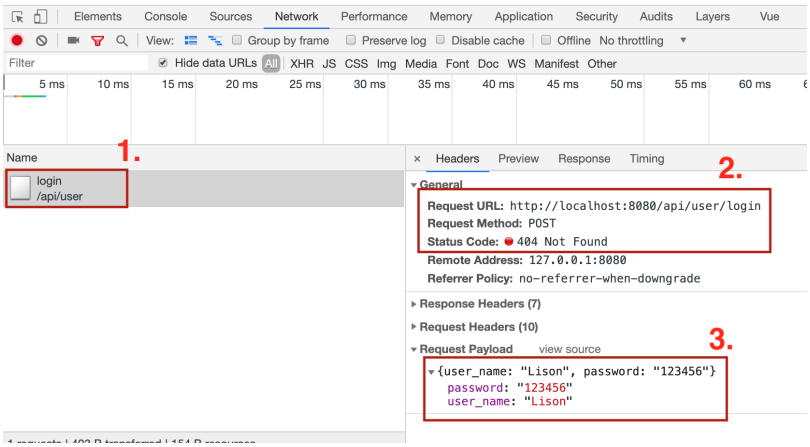
> @编注：可以在这里增加一张效果图，类似于访问login页面那样的。

可以看到，如果是指定字符串类型的html属性值，直接使用引号包裹即可，双引号单引号都可以，但是可能实际TSLint会要求使用统一引号。还有一个知识点就是事件的绑定，在JSX语法中，如果要绑定一个事件，使用 on- 作为前缀，连接事件名，作为要绑定事件的特定写法，需要给这个事件绑定的回调函数使用花括号括起来，然后在上面定义这个login方法。

下面我们就要调用登录接口了，怎么调用我们上个小节已经讲过了，通过 import { loginReq } from '@/api/user' 引入登录接口请求方法后，然后在login方法中调用：

```
public login() {
  loginReq({ user_name: this.user_name, password: this.password }).then((res) => {
    console.log(res.data)
  })
}
```

现在你可以输入用户名和密码，然后点击登录试一下，在浏览器的控制台你会发现有报错，打开Network栏，点击我们调用的login接口，你可以看到几个信息：



- (1) 我们调用接口的名称；
- (2) 我们调用接口的完整url，请求方法为post类型，接口响应状态码为404，也就是这个接口不存在；
- (3) 我们调用接口的时候传递的参数，这里user_name为"Lison"，password为"123456"。

因为我们没有起后端服务，所以没有这个接口。为了方便，我们在做前端开发的时候，可以使用mockjs(关于mock更多知识请参阅中文官方文档)来拦截请求，然后在完全不需要后端服务的情况下编写处理逻辑，并且模拟响应返回自定义数据。

首先我们要安装依赖：`npm install mockjs`，mockjs自身是没有提供声明文件的，所以我们需要另外安装：`npm install @types/mockjs -D`，安装完之后重新使用`npm run serve`启动本地开发服务即可。

接下来我们在项目根目录下创建一个*mock*文件夹，在这个文件夹创建一个*index.ts*文件，然后要让它生效，需要在*src/main.ts*文件里引入。而且我们往往只是在本地开发初期的时候使用mock来自己模拟响应，达到和后端同步开发的节奏，在部署到正式环境的时候并不需要mock，所以我们在引入的时候可以根据当前环境判断一下，然后按需引入：

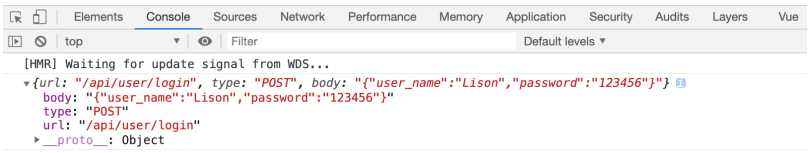
```
// src/main.ts
// ...
if (process.env.NODE_ENV === 'development') { require('../mock') }

Vue.config.productionTip = false

new Vue({
  // ...
})
```

紧接着我们定义*mock/index.ts*里面的内容：

```
import Mock from 'mockjs'

type MsgType = string | number

const success = (msg: MsgType = '', data?: any) => {
  // 这里定义一个成功返回的统一方法，返回我们在axios封装时指定的三个字段
  return {
    code: 0,
    msg,
    data
  }
}

const error = (code: number, msg: MsgType = '', data?: any) => {
  // 再定义一个返回错误状态的方法，一个必传参数是code，即错误码
  return {
    code,
    msg,
    data
  }
}

interface PostResInterface {
  body: string
  type: 'POST'
  url: string
}

Mock.mock(/\/api\/user\/login/, loginRes)

function loginRes(req: PostResInterface) {
  console.log(req)
  return success()
}
```

使用Mock.mock方法，可以指定要拦截的url的匹配规则，可以是字符串也可以是正则表达式；第二个参数是拦截到这个请求之后要做的处理和返回的数据，这里我们之所以使用function来定义这个回调函数，是因为function有变量提升的特点，可以先使用后定义。我们希望把拦截规则放到上面，具体的处理回调函数放到下面，这样看起来比较清晰。

这个回到函数有一个参数，是一个对象，我们定义一个接口 PostResInterface 来表明它的字段。如果是post请求就是这个结构；如果是get请求，是另外的样子。这里你可以打印出它，在控制台看一下它是什么样子的：



可以看到有三个参数：body是我们的参数对象的JSON字符串，type是接口请求类型，url是拦截到的url。我们要使用传来的参数，就要使用 JSON.parse 将参数JSON字符串转为对象，然后我们就能拿到传过来的参数user_name和password了：

```
function loginRes(req: PostResInterface) {
  const { user_name, password } = JSON.parse(req.body)
  return success()
}
```

> @编注：上面代码的原文是：
>
> ```
> const { user_name, password } = JSON.parse(req.body)
> ```

> 我改为了上面的样子。

在真实的后端登录逻辑中，会根据拿到的user_name和password去数据库比对，看用户名密码是否正确，我们这里是模拟，所以只是简单判断一下这个逻辑：

```
function loginRes(req: PostResInterface) {
  const { user_name, password } = JSON.parse(req.body)
  if (user_name === 'Lison' && String(password) === '123456') {
    return success()
  } else {
    return error(1001, '用户名或密码错误')
  }
}
```

这里模拟的逻辑是，用户名必须是"Lison"，密码必须是"123456"，才会返回成功状态，否则返回错误。现在再输入用户名和密码点击登录试一下，可以看到控制台打印出了接口请求返回的数据。

> @编注：在这里加一张返回成功状态的效果图。

在调用 loginReq 方法的地方，我们在then的回调函数里做判断，如果状态码是0，则说明登录成功，应该跳到后台首页，否则提示错误。我们还需要将登录状态持久化。这里我们使用一个插件**js-cookie**，先要安装它： npm install js-cookie 和它的声明文件： npm install @types/js-cookie -D 。然后我们在loginRes响应的回调函数里写下逻辑：

```
loginReq({ user_name: this.user_name, password: this.password }).then((res) => {
  const { data: { code, msg } } = res
  if (code === 0) {
    Cookies.set('token', 'value') // 这里实际开发中不会在这个地方写，而是抽离到路由守卫或store中
                                  // 而且一般这个值不会是写死的字符串，而是从服务端返回的随机且唯一的字符串
    this.$router.push('/home')
  } else {
    console.error(msg)
  }
})
```

> @编注：上面代码应该是在views/login/index.tsx文件中吧？
>
> 这个Cookies是不是应该在views/login/index.tsx文件中补充引入一下？
> import Cookies from 'js-cookie'

这样点击登录后，如果用户名密码正确，就会跳到home页。然后就剩下最后一个问题了，就是如果没有登录，在地址栏输入后台的任意url，都会跳到login登录页；如果已经登录了，访问login页，应该跳到首页或者不跳转。这个逻辑就要在路由的路由守卫里去处理了，我们将一个token字段存到了cookie里，这个是在有效期内持续存在的，可以用它来作为是否登录的判断，当然了这只是个简单的方法。

我们看下路由守卫的逻辑要怎么写：

```
// src/router/index.ts
import Vue from 'vue'
import Router from 'vue-router'
import routes from './routes'
import Cookies from 'js-cookie'

Vue.use(Router)

const router = new Router({
  routes,
})

router.beforeEach((to, from, next) => {
  const token = Cookies.get('token')
  console.log(token)
  if (token) { // 如果token不为空字符串或者undefined，说明登录了
    if (to.path === '/login') {
      // 如果登录了然后访问login页，不做跳转，从哪来回哪去
      next(from)
    } else {
      // 否则顺利跳转
      next()
    }
  } else { // 否则是没登录
    if (to.path === '/login') {
      // 如果没登录而且乖乖的到登录页去，轻松放行
      next()
    } else {
      // 如果没登录还想去登录后的页面，打回登录页
      next('/login')
    }
  }
})

export default router
```

加了这个逻辑之后，你可以试一下，登陆后，你在地址栏输入login页的路径，仍然停留在当前页比如home页；如果你清掉浏览器缓存把存在cookie里的token清掉，然后刷新页面，会跳大login页，就算在地址栏输入home页的地址，还是会跳到login页。

本小节的内容计算讲完了，我们事先简答的登录表单，用来输入用户名和密码，点击登录后调用接口，然后使用mock对请求进行模拟响应并返回状态。然后在路由守卫对登录状态做判断，从而跳转到合适的页面。

下个小节我们将使用第三方UI组件库，搭建一个简单的后台界面，并添加基本的部件。

封装接口请求

搭建后台界面布局和结合Vuex实现完整登录流程

精选留言 0

欢迎在这里发表留言，作者筛选后可公开显示

目前暂无任何讨论