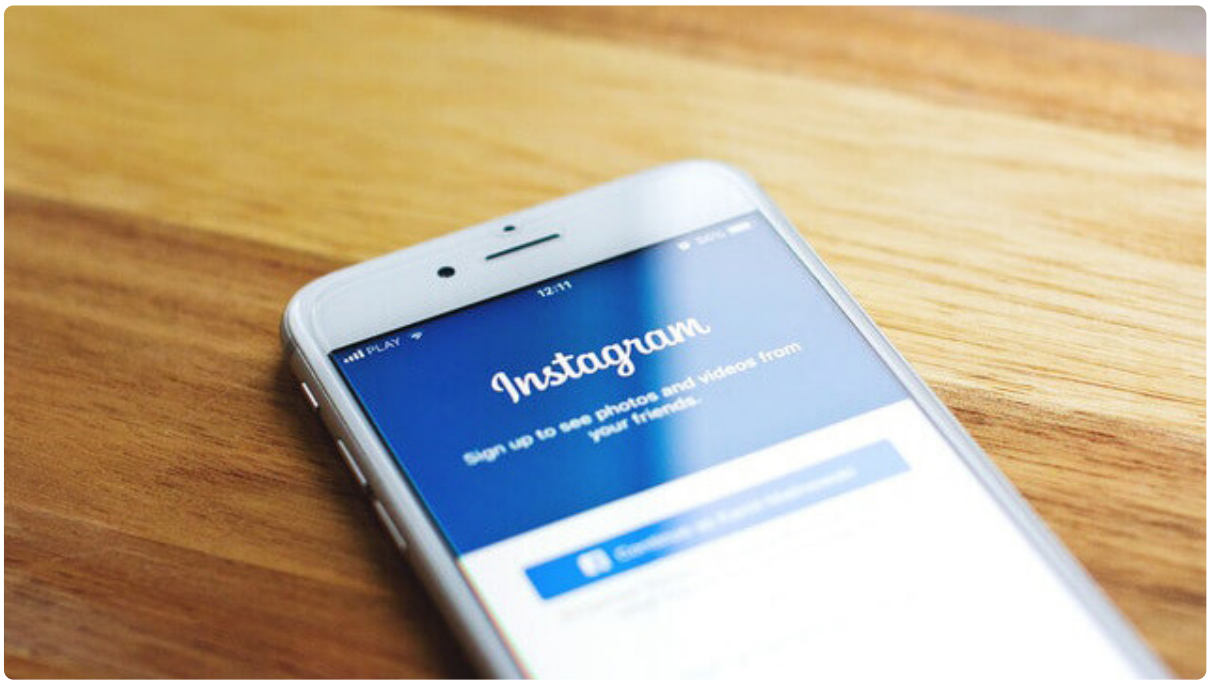
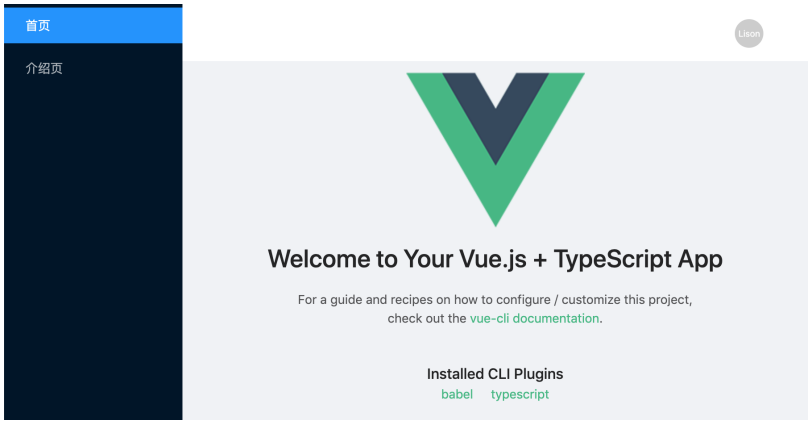
搭建后台界面布局和结合Vuex实现完整登录流程

更新时间：2019-07-29 09:26:01



> 理想必须要人们去实现它，它不但需要决心和勇敢而且需要知识。
>
> ——吴玉章

本小节我们来搭建一个后台基本框架，所有的页面除了登录页，都在这个框架里展示，左侧显示菜单，右侧显示页面，我们先看下要实现的效果：



接下来我们继续上个小节的节奏进行。在开始之前我们需要安装一个UI组件库，Ant Design Vue，它是React版本的Ant Design组件库的Vue版本，因为Ant Design是使用TypeScript编写的，类型声明文件较为完善，而Ant Design Vue延用这些声明文件，所以可以更好地在TypeScript环境下使用。首先来安装它：`npm install ant-design-vue`。

接下来我们在入口文件引入组件库及其样式，并且全局注册它：

```
// src/main.ts
// ...
import AntVue from 'ant-design-vue'
import 'ant-design-vue/dist/antd.css';
Vue.use(AntVue)
// ...
```

顺便修改一下*src/App.vue*的样式，我们在App.vue文件的样式里，增加如下样式，使页面具有高度：

```
html,body{
  height: 100%;
  margin: 0;
  padding: 0;
}
#app{
  height: 100%;
  .home{
    text-align: center;
  }
}
```

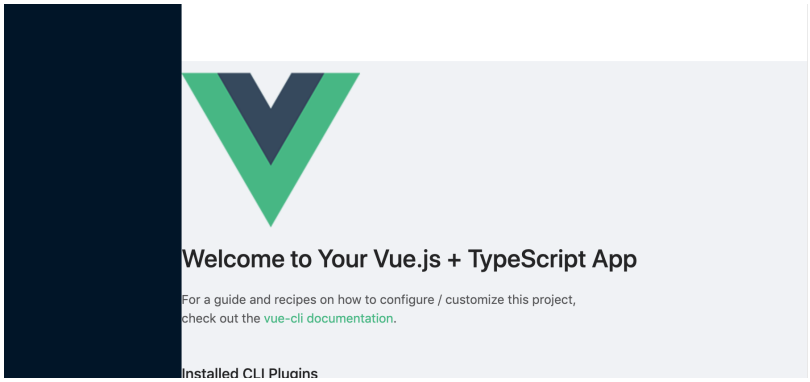
接下来我们在*src/components*文件夹下创建一个*TMain*文件夹，之所以叫这个，是因为根据Vue开发规范强烈建议组件名使用大写字母开头，然后使用驼峰形式命名，而且不要使用单个单词，应该统一加个前缀，所以我们这个框架组件叫做TMain，然后在*TMain*文件夹下创建一个*index.vue*文件。我们来使用单文件的形式编写组件。我们先只添加基本的代码：

```
<template>
  <div class="t-main-wrap">
    <!-- 布局容器组件 -->
    <a-layout style="height: 100%">
      <!-- 左侧容器组件 -->
      <a-layout-sider></a-layout-sider>
      <a-layout>
        <!-- 顶部容器组件 -->
        <a-layout-header style="background: #fff"></a-layout-header>
        <!-- 内容容器组件 -->
        <a-layout-content>
          <!-- 路由视图渲染组件 -->
          <router-view></router-view>
        </a-layout-content>
      </a-layout>
    </a-layout>
  </div>
</template>

<script lang="ts">
import { Component, Vue } from 'vue-property-decorator'

@Component({
  name: 'TMain'
})
export default class TMain extends Vue {}
</script>

<style lang="less">
.t-main-wrap{
  height: 100%;
}
</style>
```

然后修改下路由，这里需要用到嵌套路由，所有要在框架里展示的页面，都要作为子路由去配置，父路由的组件就是这里的TMain，我们来看下之前的路由配置如何修改：

```ts
// src/router/routes.ts
import Home from '../views/Home.vue'

export default [
  {
    path: '/',
    component: () => import('@/components/TMain/index.vue'),
    children: [
    {
      path: '/home',
      name: 'home',
      component: Home,
    },
    {
      path: '/about',
      name: 'about',
      component: () => import('../views/About.vue'),
    }
    ]
  },
  {
    path: '/login',
    name: 'login',
    component: () => import('../views/login/index'),
  },
]
```

可以看到，我们要让home页和about页在框架内显示，所以作为TMain组件的子路由，登录页是不希望在框架内显示的，所以和TMain组件所在路由同级。修改路由保存成功后，此时如果你访问**localhost:8080/#/home**(端口需要看你实际端口为多少)，就会发现原来的home页在右侧区域内，但因为我们删掉了样式，所有样式有变化，效果如图：



修改浏览器地址栏的url，改为**localhost:8080/#/about**，就会在右边区域展示about页的内容。

接下来我们要添加左侧菜单，在左侧容器组件`<a-layout-sider></a-layout-sider>`中添加菜单：

```
<!-- 省略部分代码 -->
<!--src\components\TMain\index.vue-->
      <a-layout-sider>
        <a-menu
          @click="handleClickMenu"
          theme="dark"
          style="width: 100%"
          mode="inline">
          <a-menu-item key="/home">首页</a-menu-item>
          <a-menu-item key="/about">介绍页</a-menu-item>
        </a-menu>
      </a-layout-sider>
<!-- 省略部分代码 -->
<script lang="ts">
import { Component, Vue } from 'vue-property-decorator'

@Component({
  name: 'TMain'
})
export default class TMain extends Vue {
  public handleClickMenu({ item, key, keyPath }) {
    this.$router.push(key)
  }
}
</script>
```

现在再在浏览器看下，左侧有了两个菜单项，点击菜单项右边的页面会相应变化。但当我们刷新页面后，左侧菜单并不会高亮相应的菜单项，即使右侧显示的页面是相应的页面。所以这里我们就可以使用一下路由的一些数据，搭配AMenu组件的api。我们知道vue-router可以在组件中通过`&lt;$route.path`即当前页面的路径，和我们在写菜单项的时候，给`<a-menu-item></a-menu-item>`设置的key属性是对应的，所以我们可以给`<a-menu></a-menu>`添加一个`selectedKeys`属性，它是一个数组，我们这里只需要如下设置即可：

```
<!-- 省略部分代码 -->
      <a-layout-sider>
        <a-menu
          :selected-keys="[this.$route.path]"
          @click="handleClickMenu"
          theme="dark"
          style="width: 100%"
          mode="inline">
<!-- 省略部分代码 -->
```

现在你选中任意一下进行页面跳转，然后刷新浏览器，你会发现菜单会高亮当前打开的页面对应的菜单项。利用路由对象还可以做很多事，比如可以根据路由对象的`matched`来实现面包屑，显示当前打开页面的层级关系等。

下面我们增加一个用户头像，用户头像我们这里不用图片了，而是用用户名，这样我们可以演示如何结合TypeScript使用Vuex。点击用户头像显示下拉菜单，显示退出登录。点击退出登录后清掉保存的登录cookie信息。在`<a-layout-header></a-layout-header>`标签中添加一个带下拉菜单的用户头像：

```
<!-- 省略部分代码 -->
      <a-layout-header style="background: #fff;text-align: right;">
        <a-dropdown>
          <a-avatar>USER</a-avatar>
          <a-menu slot="overlay">
            <a-menu-item key="logout">退出登录</a-menu-item>
          </a-menu>
        </a-dropdown>
      </a-layout-header>
<!-- 省略部分代码 -->
```

现在你会发现顶部右侧有一个原型头像框，中间是我们这里写死的"USER"字符，鼠标放上去会出现下拉菜单。接下来我们补充一下前面的登录逻辑，登录成功之后返回用户ID，然后再通过用户ID去拿用户信息，将用户信息保存在store中，然后从用户信息中将用户名在这里展示。其实在实际开发中，登录接口调用成功登录后，会返回token，然后携带token去请求获取用户信息，我们这里简化逻辑了。

首先修改下mock中登录接口拦截的逻辑，登录成功后返回用户ID：

```
// src/mock/index.ts
Mock.mock(/\/api\/user\/login/, loginRes)

function loginRes(req: PostResInterface) {
  const { user_name, password } = JSON.parse(req.body)
  if (user_name === 'Lison' && String(password) === '123456') {
    return success('登录成功', { user_id: 101 })
  } else {
    return error(1001, '用户名或密码错误')
  }
}
```

然后我们再添加一个返回用户信息接口的mock拦截：

```
// src/mock/index.ts
Mock.mock(/\/api\/user\/get_info/, getInfoRes)

function getInfoRes(req: PostResInterface) {
  return success('', {
    user_name: 'Lison',
    avatar: '',
    email: 'xxx@xx.com'
  })
}
```

接下来在*src/api/user.ts*里封装获取信息接口调用方法：

```
// src/api/user.ts
interface GetInfoReqArguInterface {
  user_id: string
}
export const getInfoReq = (data: GetInfoReqArguInterface): AxiosPromise<ResponseData> => {
  return axios.request({
    url: '/api/user/get_info',
    data,
    method: 'POST'
  })
}
```

然后我们要把原来写在*src/views/login/index.tsx*的login方法的逻辑写在store的actions里，然后这里通过dispatch调用。首先在*src/store/index.ts*文件中增加一个登录的actions：

```
// src/store/index.ts
import Vue from 'vue'
import Vuex from 'vuex'
import { loginReq,getInfoReq } from '@/api/user'
import Cookies from 'js-cookie'

Vue.use(Vuex)

export default new Vuex.Store({
  state: {
    user_name: '',
    email: ''
  },
  mutations: {
    // 设置存储在store里的用户信息
    setUserInfoMutations(state, info) {
      const { user_name, email } = info
      state.user_name = user_name
      state.email = email
    }
  },
  actions: {
    loginActions({ commit, dispatch }, { user_name, password }) {
      return new Promise((resolve, reject) => {
        loginReq({ user_name, password }).then((response) => { // 首先调登录接口
          const { data: { code, msg, data } } = response
          if (code === 0) {// 成功返回后，将token存到cookie中，然后携带token去请求获取信息接口
            Cookies.set('token', 'value')
            dispatch('getInfoActions').then(() => {
              resolve()
            })
          } else {
            console.error(msg)
          }
        })
      })
    },
    getInfoActions({ commit }) {
      return new Promise((resolve, reject) => {
        getInfoReq().then((res) => {
          const { data: { code, data } } = res
          if (code === 0) {
            commit('setUserInfoMutations', data)
            resolve() // 全部操作做完后，调用resolve方法
          }
        })
      })
    }
  },
})
```

修改成功之后，我们需要在登录成功之后跳到home页，这个逻辑就放在login页，也就是*src/views/login/index.tsx*文件里去写。首先要在点击登录后，调用store里的这个actions：loginActions方法。这里需要安装一个插件：`npm install vuex-class`，我们需要用它来方便进行开发。安装好后，启动服务，我们修改*src/views/login/index.tsx*：

```
import { Component, Emit, Prop, Vue, Watch } from 'vue-property-decorator'
import { State, Action } from 'vuex-class'

@Component
export default class LoginPage extends Vue {
  // ... 省略之前讲的部分代码

  @Action('loginActions') public loginAction // 这里通过@Action('loginActions')装饰器指定loginAction是store里的loginActions方法

  public login() {
    // 然后这里就可以直接调用loginAction方法
    // 效果和this.$store.dispatch('loginActions', { 参数 })是一样的
    this.loginAction({
      user_name: this.password,
      password: this.password
    }).then(() => {
      // 在store中的loginActions定义中，执行resolve方法的时机就是这里then中传入的这个函数执行的时机
      this.$router.push('/home') // 在这跳转到home页
    })
  }

  protected render() {
    // ...
  }
}
```

接下来我们登录成功之后，就会将用户信息保存在store中，所以我们可以在用户头像的地方用真实的用户名了。首先在TMain组件中引入这个state：

```
<script lang="ts">
import { Component, Vue } from 'vue-property-decorator'
import { State } from 'vuex-class'

@Component({
  name: 'TMain'
})
export default class TMain extends Vue {
  @State('user_name') public userName // 将store.state.user_name赋给userName
}
  // ...
</script>
```

然后你就可以在模板语法中使用了，我们使用userName替换上面写死的字符串"USER"：

```
<!-- 省略部分代码 -->
      <a-dropdown>
        <a-avatar>{{ userName }}</a-avatar>
        <a-menu slot="overlay">
<!-- 省略部分代码 -->
```

这样，你就可以先清掉cookie重新登录，登陆后会进入home页，你会发现头像处已经显示的是我们获取到的用户名"Lison"了。但还有一个问题，存储在store中的数据是存在内存中的，当页面刷新后，内存中的数据就被清掉了，所以我们刷新后，需要重新调用获取用户信息的api请求，因为存在cookie中的token是不会随刷新被清掉的，所以我们实际中获取用户信息等都是携带token，然后后端根据token的信息，返回响应信息。这里我就不写这么复杂的逻辑了，我们在路由守卫里来处理这个逻辑：

在每次路由跳转的时候，判断store中的user_name是否为空字符串。如果为空字符串，可能是因为刷新页面导致的store中的user_name丢失，所以要重新请求获取用户信息：

```
import Vue from 'vue'
import Router from 'vue-router'
import routes from './routes'
import store from '@/store' // 引入store实例
import Cookies from 'js-cookie'

Vue.use(Router)

const router = new Router({
  routes,
})

const turn = (to, from, next) => {
  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()
  }
}

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 (!store.state.user_name) { // 判断store.state.user_name是否为空，为空则需要获取
      store.dispatch('getInfoActions').then(() => {
        turn(to, from, next) // 获取之后再跳转页面
      })
    } else {
      turn(to, from, next) // 如果store.state.user_name不为空，直接跳转
    }
  } 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
```

现在无论你怎么刷新页面，你都可以看到头像显示的是正式的用户名。

最后我们还有一个功能，就是退出登录。点击“退出登录”后，需要清楚cookie中存的token，然后跳到登录页，并且清空store中存的用户信息。首先我们在*src/components/TMain/index.vue*组件里，从*vuex-class*中引入Mutation装饰器：

```
import { State, Mutation } from 'vuex-class'
import Cookie from 'js-cookie' // 注意，这个清楚cookie中token的逻辑，最好放到store中用专门的mutation去清除store中用户信息的同时清楚token
```

然后在组件类TMain中引入store中`setUserInfoMutations`方法，并且再增加两个方法：

handleClickAvatarMenu用于给头像下拉菜单绑定点击事件，logout用于做登出操作：

```
export default class TMain extends Vue {
  // ... 省略部分代码
  @Mutation('setUserInfoMutations') public setUserInfo
  public handleClickAvatarMenu({ item, key, keyPath }) {
    if (key === 'logout') {
      this.logout()
    }
  }
  public logout() {
    this.setUserInfo({ user_name: '', email: '' })
    Cookie.set('token', '')
    this.$router.push('/login')
  }
}
```

最后记得要在头像下拉菜单绑定这个handleClickAvatarMenu方法：

```
<!-- 省略部分代码 -->
          <a-dropdown>
            <a-avatar>{{ userName }}</a-avatar>
            <a-menu @click="handleClickAvatarMenu" slot="overlay">
              <a-menu-item key="logout">退出登录</a-menu-item>
            </a-menu>
          </a-dropdown>
<!-- 省略部分代码 -->
```

现在，点击头像下拉菜单的退出登录后，就会跳到登录页，而且不登录无法跳到Home等页。

到这里，我们本小节的内容就讲完了。下个小节，我们将结合echarts和其他插件，封装几个组件，来搭建一个好看而且实用的仪表盘首页。

实现登录页并用Mock响应请求

使用TypeScript开发Vue组件和使用Vue组件

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