全屏插件的引用

目标：实现页面的全屏功能

> 全屏功能可以借助一个插件来实现

第一步，安装全局插件**screenfull**

```
$ npm i screenfull
```

第二步，封装全屏显示的插件·· `src/components/ScreenFull/index.vue`

```
<template>
  <!-- 放置一个图标 -->
  <div>
    <!-- 放置一个svg的图标 -->
    <svg-icon icon-class="fullscreen" style="color:#fff; width: 20px; height:
20px" @click="changeScreen" />
  </div>
</template>

<script>
import ScreenFull from 'screenfull'
export default {
  methods: {
    //   改变全屏
    changeScreen() {
      if (!ScreenFull.isEnabled) {
        // 此时全屏不可用
        this.$message.warning('此时全屏组件不可用')
        return
      }
      // document.documentElement.requestFullscreen()  原生js调用
      //   如果可用 就可以全屏
      ScreenFull.toggle()
    }
  }
}
</script>

<style>

</style>
```

第三步，全局注册该组件 `src/components/index.js`

```
import ScreenFull from './ScreenFull'
Vue.component('ScreenFull', ScreenFull) // 注册全屏组件
```

第四步，放置于`layout/navbar.vue`中

```
<screen-full class="right-menu-item" />
.right-menu-item {
   vertical-align: middle;
}
```

提交代码

`本节任务`： 实现页面的全屏功能

动态主题的设置

`目标`： 实现动态主题的设置

> 我们想要实现在页面中实时的切换颜色，此时页面的主题可以跟着设置的颜色进行变化

简单说明一下它的原理： element-ui 2.0 版本之后所有的样式都是基于 SCSS 编写的，所有的颜色都是基于几个基础颜色变量来设置的，所以就不难实现动态换肤了，只要找到那几个颜色变量修改它就可以了。 首先我们需要拿到通过 `package.json` 拿到 element-ui 的版本号，根据该版本号去请求相应的样式。拿到样式之后将样色，通过正则匹配和替换，将颜色变量替换成你需要的，之后动态添加 `style` 标签来覆盖原有的 css 样式。

第一步， 封装颜色选择组件 `ThemePicker` 代码地址：@/components/ThemePicker。

> 注意：本章节重点在于集成，内部的更换主题可以先不用关心。

实现代码

```
<template>
  <el-color-picker
    v-model="theme"
    :predefine="['#409EFF', '#1890ff', '#304156','#212121','#11a983', '#13c2c2',
'#6959CD', '#f5222d', ]"
    class="theme-picker"
    popper-class="theme-picker-dropdown"
  />

</template>

<script>
const version = require('element-ui/package.json').version // element-ui version
from node_modules
const ORIGINAL_THEME = '#409EFF' // default color
export default {
  data() {
    return {
      chalk: '', // content of theme-chalk css
      theme: ''
    }
  },
  computed: {
    defaultTheme() {
      return this.$store.state.settings.theme
    }
  },
  watch: {
    defaultTheme: {
      handler: function(val, oldVal) {
```

```
        this.theme = val
      },
      immediate: true
    },
    async theme(val) {
      const oldVal = this.chalk ? this.theme : ORIGINAL_THEME
      if (typeof val !== 'string') return
      const themeCluster = this.getThemeCluster(val.replace('#', ''))
      const originalCluster = this.getThemeCluster(oldVal.replace('#', ''))
      console.log(themeCluster, originalCluster)
      const $message = this.$message({
        message: '  Compiling the theme',
        customClass: 'theme-message',
        type: 'success',
        duration: 0,
        iconClass: 'el-icon-loading'
      })
      const getHandler = (variable, id) => {
        return () => {
          const originalCluster =
this.getThemeCluster(ORIGINAL_THEME.replace('#', ''))
          const newStyle = this.updateStyle(this[variable], originalCluster,
themeCluster)
          let styleTag = document.getElementById(id)
          if (!styleTag) {
            styleTag = document.createElement('style')
            styleTag.setAttribute('id', id)
            document.head.appendChild(styleTag)
          }
          styleTag.innerText = newStyle
        }
      }
      if (!this.chalk) {
        const url = `https://unpkg.com/element-ui@${version}/lib/theme-
chalk/index.css`
        await this.getCSSString(url, 'chalk')
      }
      const chalkHandler = getHandler('chalk', 'chalk-style')
      chalkHandler()
      const styles = [].slice.call(document.querySelectorAll('style'))
        .filter(style => {
          const text = style.innerText
          return new RegExp(oldVal, 'i').test(text) && !/Chalk
Variables/.test(text)
        })
      styles.forEach(style => {
        const { innerText } = style
        if (typeof innerText !== 'string') return
        style.innerText = this.updateStyle(innerText, originalCluster,
themeCluster)
      })
      this.$emit('change', val)
      $message.close()
    }
  },
  methods: {
    updateStyle(style, oldCluster, newCluster) {
      let newStyle = style
```

```
      oldCluster.forEach((color, index) => {
        newStyle = newStyle.replace(new RegExp(color, 'ig'), newCluster[index])
      })
      return newStyle
    },
    getCSSString(url, variable) {
      return new Promise(resolve => {
        const xhr = new XMLHttpRequest()
        xhr.onreadystatechange = () => {
          if (xhr.readyState === 4 && xhr.status === 200) {
            this[variable] = xhr.responseText.replace(/@font-face{[^}]+}/, '')
            resolve()
          }
        }
        xhr.open('GET', url)
        xhr.send()
      })
    },
    getThemeCluster(theme) {
      const tintColor = (color, tint) => {
        let red = parseInt(color.slice(0, 2), 16)
        let green = parseInt(color.slice(2, 4), 16)
        let blue = parseInt(color.slice(4, 6), 16)
        if (tint === 0) { // when primary color is in its rgb space
          return [red, green, blue].join(',')
        } else {
          red += Math.round(tint * (255 - red))
          green += Math.round(tint * (255 - green))
          blue += Math.round(tint * (255 - blue))
          red = red.toString(16)
          green = green.toString(16)
          blue = blue.toString(16)
          return `#${red}${green}${blue}`
        }
      }
      const shadeColor = (color, shade) => {
        let red = parseInt(color.slice(0, 2), 16)
        let green = parseInt(color.slice(2, 4), 16)
        let blue = parseInt(color.slice(4, 6), 16)
        red = Math.round((1 - shade) * red)
        green = Math.round((1 - shade) * green)
        blue = Math.round((1 - shade) * blue)
        red = red.toString(16)
        green = green.toString(16)
        blue = blue.toString(16)
        return `#${red}${green}${blue}`
      }
      const clusters = [theme]
      for (let i = 0; i <= 9; i++) {
        clusters.push(tintColor(theme, Number((i / 10).toFixed(2))))
      }
      clusters.push(shadeColor(theme, 0.1))
      return clusters
    }
  }
}
</script>
```

```
<style>
.theme-message,
.theme-picker-dropdown {
  z-index: 99999 !important;
}
.theme-picker .el-color-picker__trigger {
  height: 26px !important;
  width: 26px !important;
  padding: 2px;
}
.theme-picker-dropdown .el-color-dropdown__link-btn {
  display: none;
}
.el-color-picker {
  height: auto !important;
}
</style>
```

注册代码

```
import ThemePicker from './ThemePicker'
Vue.component('ThemePicker', ThemePicker)
```

第二步， 放置于`layout/navbar.vue`中

```
    <!-- 放置全屏插件 -->
     <theme-picker class="right-menu-item" />
```

提交代码

多语言实现

`目标`实现国际化语言切换

初始化多语言包

本项目使用国际化 i18n 方案。通过 vue-i18n而实现。

第一步，我们需要首先国际化的包

```
 $ npm i vue-i18n
```

第二步，需要单独一个多语言的实例化文件`src/lang/index.js`

```
import Vue from 'vue' // 引入Vue
import VueI18n from 'vue-i18n' // 引入国际化的包
import Cookie from 'js-cookie' // 引入cookie包
import elementEN from 'element-ui/lib/locale/lang/en' // 引入饿了么的英文包
import elementZH from 'element-ui/lib/locale/lang/zh-CN' // 引入饿了么的中文包
Vue.use(VueI18n) // 全局注册国际化包
export default new VueI18n({
  locale: Cookie.get('language') || 'zh', // 从cookie中获取语言类型 获取不到就是中文
  messages: {
    en: {
```

```
      ...elementEN // 将饿了么的英文语言包引入
    },
    zh: {
      ...elementZH // 将饿了么的中文语言包引入
    }
  }
})
```

> 上面的代码的作用是将Element的两种语言导入了

第三步，在main.js中对挂载 i18n的插件，并设置element为当前的语言

```
// 设置element为当前的语言
Vue.use(ElementUI, {
  i18n: (key, value) => i18n.t(key, value)
})

new Vue({
  el: '#app',
  router,
  store,
  i18n,
  render: h => h(App)
})
```

引入自定义语言包

> 此时，element已经变成了zh，也就是中文，但是我们常规的内容怎么根据当前语言类型显示?

这里，针对英文和中文，我们可以提供两个不同的语言包 `src/lang/zh.js , src/lang/en.js`

> 该语言包，我们已经在资源中提供

第四步，在index.js中同样引入该语言包

```
import customZH from './zh' // 引入自定义中文包
import customEN from './en' // 引入自定义英文包
Vue.use(VueI18n) // 全局注册国际化包
export default new VueI18n({
  locale: Cookie.get('language') || 'zh', // 从cookie中获取语言类型 获取不到就是中文
  messages: {
    en: {
      ...elementEN, // 将饿了么的英文语言包引入
      ...customEN
    },
    zh: {
      ...elementZH, // 将饿了么的中文语言包引入
      ...customZH
    }
  }
})
```

在左侧菜单中应用多语言包

> 自定义语言包的内容怎么使用?

第五步，在左侧菜单应用

当我们全局注册i18n的时候，每个组件都会拥有一个`$t`的方法，它会根据传入的key，自动的去寻找当前语言的文本，我们可以将左侧菜单变成多语言展示文本

`layout/components/SidebarItem.vue`

```
<item :icon="onlyOneChild.meta.icon||(item.meta&&item.meta.icon)"
:title="$t('route.'+onlyOneChild.name)" />
```

注意：当文本的值为嵌套时，可以通过`$t(key1.key2.key3...)`的方式获取

> 现在，我们已经完成了多语言的接入，现在封装切换多语言的组件

封装多语言插件

第六步，封装多语言组件 `src/components/lang/index.vue`

```
<template>
  <el-dropdown trigger="click" @command="changeLanguage">
    <!-- 这里必须加一个div -->
    <div>
      <svg-icon style="color:#fff;font-size:20px" icon-class="language" />
    </div>
    <el-dropdown-menu slot="dropdown">
      <el-dropdown-item command="zh" :disabled="'zh'=== $i18n.locale ">中文</el-
dropdown-item>
      <el-dropdown-item command="en" :disabled="'en'=== $i18n.locale ">en</el-
dropdown-item>
    </el-dropdown-menu>
  </el-dropdown>
</template>

<script>
import Cookie from 'js-cookie'
export default {
  methods: {
    changeLanguage(lang) {
      Cookie.set('language', lang) // 切换多语言
      this.$i18n.locale = lang // 设置给本地的i18n插件
      this.$message.success('切换多语言成功')
    }
  }
}
</script>
```

第七步，在Navbar组件中引入

```
<!-- 放置切换多语言 -->
      <lang class="right-menu-item" />
      <!-- 放置主题 -->
      <theme-picker class="right-menu-item" />
      <!-- 放置全屏插件 -->
      <screen-full class="right-menu-item" />
```

最终效果

提交代码

tab页的视图引入

目标： 实现tab页打开路由的功能

> 当前我们实现的打开页面，看到一个页面之后，另一个页面就会关闭，为了显示更加有效率，我们可以引入多页签组件

多页签的组件的代码过于繁杂，开发实际需要的是集成和调用能力，所以我们只是将开发好的组件集成到当前的功能项中即可。

> 在资源目录中，`多页签`目录下放置的是 组件和vuex模块

第一步，将组件TagsView目录放置到`src/components`，并全局注册

```
import TagsView from './TagsView'
Vue.component('TagsView', TagsView)
```

第二步，将Vuex模块`tagsView.js`放置到`src/store/modules`

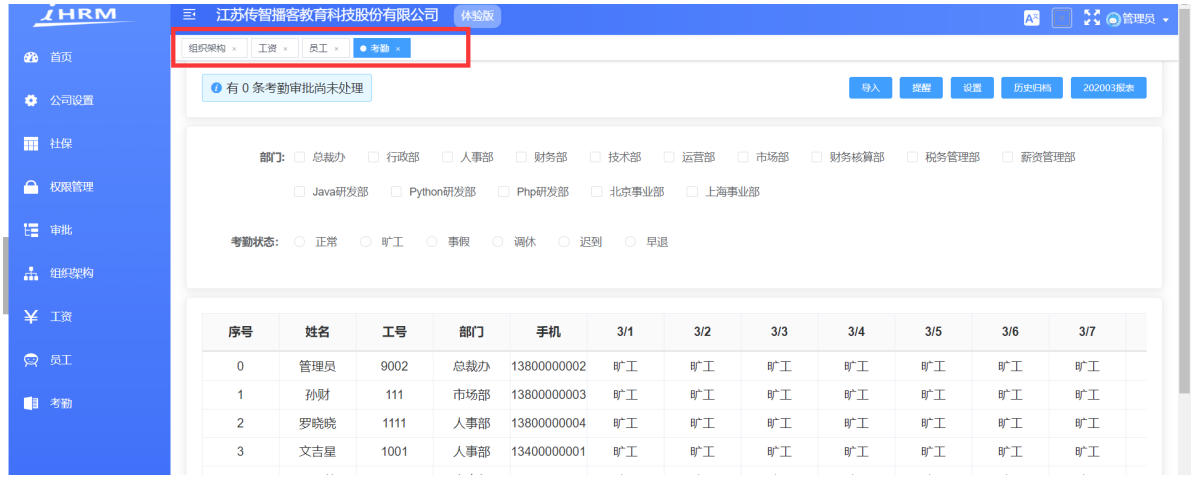
并在store中引入该模块

```
import tagsView from './modules/tagsView'
const store = new Vuex.Store({
  modules: {
    app,
    settings,
    user,
    permission,
    tagsView
  },
  getters
})
```

第三步，在`src/layout/Index.vue`中引入该组件

```
<template>
  <div :class="classObj" class="app-wrapper">
    <div v-if="device==='mobile'&&sidebar.opened" class="drawer-bg"
@click="handleClickOutside" />
    <sidebar class="sidebar-container" />
    <div class="main-container">
      <div :class="{'fixed-header':fixedHeader}">
        <navbar />
        <!-- 放置tabsview -->
        <tags-view />
      </div>
      <app-main />
    </div>
  </div>
</template>
```

效果如下



提交代码