Vue + Less + Css变量实现动态换肤功能

前言

每个网站都会有自己的一个主题色，但是随着行业内卷，越来越多的网站为了凸显特点，也为了更加迎合用户的需求，推出网站换肤功能。用户可以自己选择网站的主题：比如黑夜主题等等，从而实现了个性化定制。

市面上常见的换肤功能主要有以下2种：

- 网站自带几套固定主题，用户只能选择有限的几个主题。
- 主题色由用户随意更改，真正做到用户的个性化定制。

1 常见前端换肤方案

1.1 利用class命名空间

优点：简单、容易理解，非常容易实现。

缺点：需要定义class、手动维护、容易混乱

1.2 预备多套CSS主题（推荐）

优点：非常好理解，一套主题定义一套css

缺点：需要手写多套css样式代码，且更换主题需要下载新的css样式代码。

1.3 动态换肤

原理：主要是基于element-ui换肤方案的实现，生成一套主题， 将主题配色配置写在js中，在浏览器中用js动态修改style标签覆盖原有的CSS。

优点：通过定义函数的形式自动替换、操作性较强

缺点：需要有统一打包出来的index.css，实现难度较高。

1.4 less在线编译实现

原理：使用 `modifyVars()` 方法, 基于 less 在浏览器中的编译来实现。在引入less文件的时候需要通过link方式引入，然后基于less.js中的方法来进行修改less变量。

特点：编译速度依赖客户端性能、切换不及时，运行时编译、需要额外引入less.main.js、样式文件通过link方式引入。

这种方式一般不推荐。

1.5 CSS变量换肤（推荐）

优点：只需一套CSS文件；换肤不需要延迟等候；对浏览器性能要求低；可自动适配多种主题色

缺点：不兼容IE

2 要实现的需求

针对市场上常见的主题换肤方案，最符合用户个性化定制的方案无疑是让用户自定义主题色，实现热换肤。

最终需求：

1. 默认有几套主题色供用户选择
2. 用户也可以自定义主题色
3. 用户选定主题后需立即生效，无需重启项目或重新打包项目

3 采用的方案

为了满足上面的几种需求，且实现起来简单 一点，我们最终采用了less+css变量结合的方式来实行热换肤。因为我们需要热换肤，所以像iview这类的UI组件库提供的主题则不考虑，因为它们每次更换主题后需要重启项目才行。

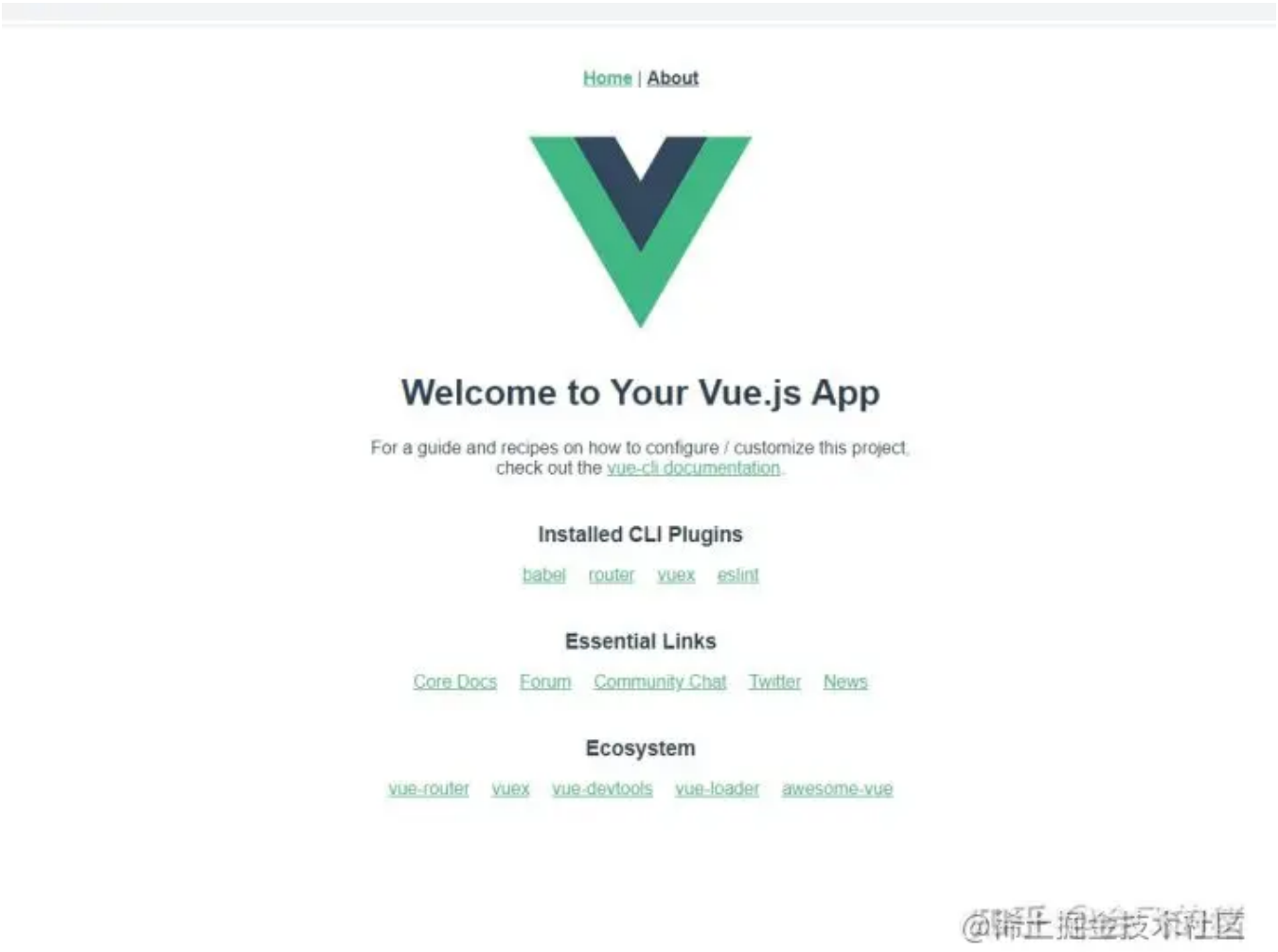
准备工作：

- Vue项目
- 安装less

4 具体实现

4.1 初始化vue项目

任意初始化一个Vue项目，当然你也可以在已有的项目里面更改：



4.2 安装必要插件

在这里我们会用到两个样式处理插件，项目执行以下命令：

复制代码

```
npm install style-resources-loader -D
npm install vue-cli-plugin-style-resources-loader -D
```

为了让我们的Vue项目里面能够使用less，还需要安装less相关插件：

复制代码

```
npm install less-loader@5.0.0 --save
npm install less --save
```

Vue项目如何配置less这里不做过多的介绍，因为我们的重点不在这里，我们的最终目的就是在项目里面能够使用less。

4.3 新建style.less

style.less用于配置全局的默认样式，也可以是默认主题或字体颜色。在项目src目录下新建theme文件夹，然后新建style.less，代码如下：

复制代码

```
/src/theme/style.less

// 默认的主题颜色
@primaryColor: var(--primaryColor, #000);
@primaryTextColor: var(--primaryTextColor, green);
// 导出变量
:export {
  name: "less";
  primaryColor: @primaryColor;
  primaryTextColor: @primaryTextColor;
}
```

4.4 配置vue.config.js

在项目根目录新建vue.config.js文件，编写配置，代码如下：

复制代码

```
const path = require("path");
module.exports = {
  pluginOptions: {
    "style-resources-loader": {
      preProcessor: "less",
      patterns: [
        // 这个是加上自己的路径,不能使用(如下:alias)中配置的别名路径
        path.resolve(__dirname, "./src/theme/style.less"),
      ],
    },
  },
};
```

当我们配置好vue.config.js文件后，就可以在项目的任何地方使用我们预先定义的less变量了，示例代码如下：

复制代码

```
<style lang="less" scoped>
p {
    color: @primaryTextColor;
}
</style>
```

我们随意更改一下我们的Vue项目：

修改HelloWorld组件，在组件中使用less语法以及刚刚我们定义的全局变量。

组件代码如下：

复制代码

```
<template>
  <div class="hello">
    <p>我是测试文字</p>
  </div>
</template>

<script>
export default {
  name: "HelloWorld",
};
</script>
<style scoped lang="less">
.hello {
  p {
    color: @primaryTextColor;
  }
}
</style>
```

此时的文字颜色便是我们刚刚设置的绿色了。

4.5 配置几套可选主题

在/src/theme目录下新建model.js，编写自定义主题代码，代码如下：

复制代码

```
// 一套默认主题以及一套暗黑主题
// 一套默认主题以及一套暗黑主题
export const themes = {
  default: {
    primaryColor: `${74}, ${144},${226}`,
    primaryTextColor: `${74}, ${144},${226}`,
  },
  dark: {
    primaryColor: `${0},${0},${0}`,
    primaryTextColor: `${0},${0},${0}`,
  },
};
```

这里我们定义了两套默认主题，目的就是为了让用户能够在这两套主题中切换，其中主题颜色我们采用了rgba的写法，因为有可能我们需要在不用的地方使用同一主题色，但是透明度不一样。

4.6 编写修改主题的方法

全局的颜色变量以及两套默认主题我们都编写好了，在这里我们已经实现了静态更改主题，即可以更改颜色，项目重新运行后便可生效。但是这还达不到我们的目的，我们需要动态更改主题，所以我们还需要编写一个能够动态更改主题的方法。

在/src/theme文件夹下新建theme.js文件，代码如下：

复制代码

```
import { themes } from "./model";
// 修改页面中的样式变量值
const changeStyle = (obj) => {
  for (let key in obj) {
    document
      .getElementsByTagName("body")[0]
      .style.setProperty(`--${key}`, obj[key]);
  }
};
// 改变主题的方法
export const setTheme = (themeName) => {
  localStorage.setItem("theme", themeName); // 保存主题到本地，下次进入使用该主题
  const themeConfig = themes[themeName];
  // 如果有主题名称，那么则采用我们定义的主题
  if (themeConfig) {
    localStorage.setItem("primaryColor", themeConfig.primaryColor); // 保存主题色到本地
    localStorage.setItem("primaryTextColor", themeConfig.primaryTextColor); // 保存文字颜色到本
    changeStyle(themeConfig); // 改变样式
  } else {
    let themeConfig = {
      primaryColor: localStorage.getItem("primaryColor"),
      primaryTextColor: localStorage.getItem("primaryTextColor"),
    };
    changeStyle(themeConfig);
  }
};
```

这里我们编写了两个方法，一个是更改全局css变量值的方法，达到更改样式的作用，另一个是更改主题的方法，可以让用户选择我们准备的几套主题或者自定义颜色。

4.7 动态变换主题

修改我们的HelloWorld组件，演示如何动态修改主题。

测试代码如下：

复制代码

```
<template>
  <div class="hello">
    <div class="box-1"></div>
    <div class="box-2"></div>
    <p>我是测试文字</p>
    <button @click="defaultTheme">默认主题</button>
    <button @click="dark">暗黑主题</button>
    <button @click="custom">自定义主题</button>
  </div>
</template>


<script>
import { setTheme } from "../theme/theme";
export default {
  name: "HelloWorld",
  mounted() {
    this.init(); // 初始化主题
  },
  methods: {
    init() {
      setTheme("default"); // 初始化未默认主题
    },
    // 更改为默认主题
    defaultTheme() {
      setTheme("default");
    },
    // 更改为暗黑主题
    dark() {
      setTheme("dark");
    },
    // 更改为自定义主题
    custom() {
      let newColor = {
        r: 12,
        g: 33,
        b: 234,
      };
      let newPrimaryColor = `${newColor.r},${newColor.g},${newColor.b}`;
      localStorage.setItem("primaryColor", newPrimaryColor); // 将新的主题色存入本地
      setTheme();
    },
  },
};
</script>
<style scoped lang="less">
.hello {
  display: flex;
  flex-direction: column;
```

```
    align-items: center;
    .box-1 {
      width: 50px;
      height: 50px;
      margin-bottom: 30px;
      background: rgba(@primaryColor, 1);
    }
    .box-2 {
      width: 50px;
      height: 50px;
      margin-bottom: 30px;
      background: rgba(@primaryColor, 0.3);
    }
    p {
      color: @primaryTextColor;
    }
  }
  </style>
```

当我们进入页面是，会采用默认主题样式，然后用户可以点击按钮更改自定义的样式，并且会保存到本地。

5 总结

利用less和css变量动态修改主题，我们主要新建了3个样式文件，作用分别是默认主题、自定义的几套主题以及修改主题的工具函数。本篇文章只是一个简单的入门，通常自定义主题我们会提供给用户颜色选择面板，大家可以结合使用。