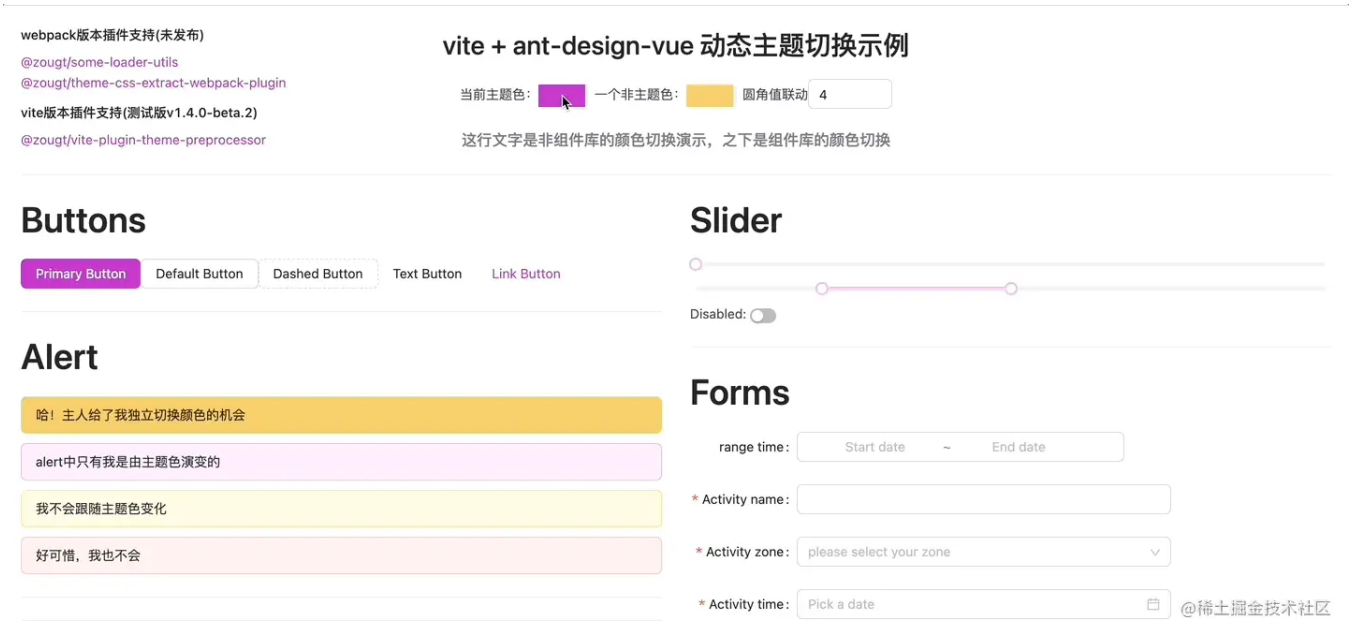
基于less和sass的在webpack或vite中的动态主题的实现方案

2021-12-26更新

在线动态主题的实现，比原文中提到的方案六的插件更加易用，具有如下特点：

- 使用成本很低
- 跟ui框架无关，Element-ui、iview、Ant-design 等等等（只要基于 less/sass）都可以
- 不依赖 css3 vars
- 浏览器兼容性良好(IE9+ ?)
- 一个主色带动所有梯度色



webpack插件支持

@zougt/some-loader-utils

@zougt/theme-css-extract-webpack-plugin

vite插件支持

@zougt/vite-plugin-theme-preprocessor

demo 仓库

需求背景

当使用 react + ant-design 或 vue + element-ui 的组合或者其他框架，在进行项目开发到一半或者已经完成开发时，客户方想要加入在线预设主题切换的效果，这时有如下的选择：

- 方案一：使用 css3 的 Variables(需要考虑浏览器支持情况)重新整理源码中的 less 或者 sass 变量，在线修改 css 变量达到切换效果，**但是组件库中使用了很多的 less 或者 sass 的颜色函数还只是预处理能力不支持 css 变量编译的，需要做很多的组件样式覆盖处理，这是需要不少的工作量的；**

- 方案二：预设多份 less 或者 sass 变量文件，使用 webpack 或 gulp 等构建能力提前将所有的样式(包括组件库的)编译出总的多份 css 文件，在线切换 css 文件到达目的，**但是需要对项目的所有的 less、sass 的引用模式作调整，对构建环境也需很大的调整，样式与 js 完全分离，如果有使用 css modules 是更大的麻烦，而且在开发模式下修改调试样式极其不友好，还不能友好地对组件库的的 less 或 sass 按需编译；**

- 方案三：(不现实的)采用 css in js 方案对页面和组件重构；

- 方案四：如果你需要的是在线用颜色面板选择任意的主题色切换，如果是 less 则可以采用 less.js 在线编译的能力(不考虑性能情况)，如果是 sass 则需要后台服务实时编译 sass，**但这些对使用 css modules 的不太友好处理，在原来的项目做改动也要不少的工作量；**

- 方案五：如果使用的是 ant-design,可以选择采用antd-theme-webpack-plugin、antd-theme-generator、umi-plugin-antd-theme等，**这也仅限于 antd；**

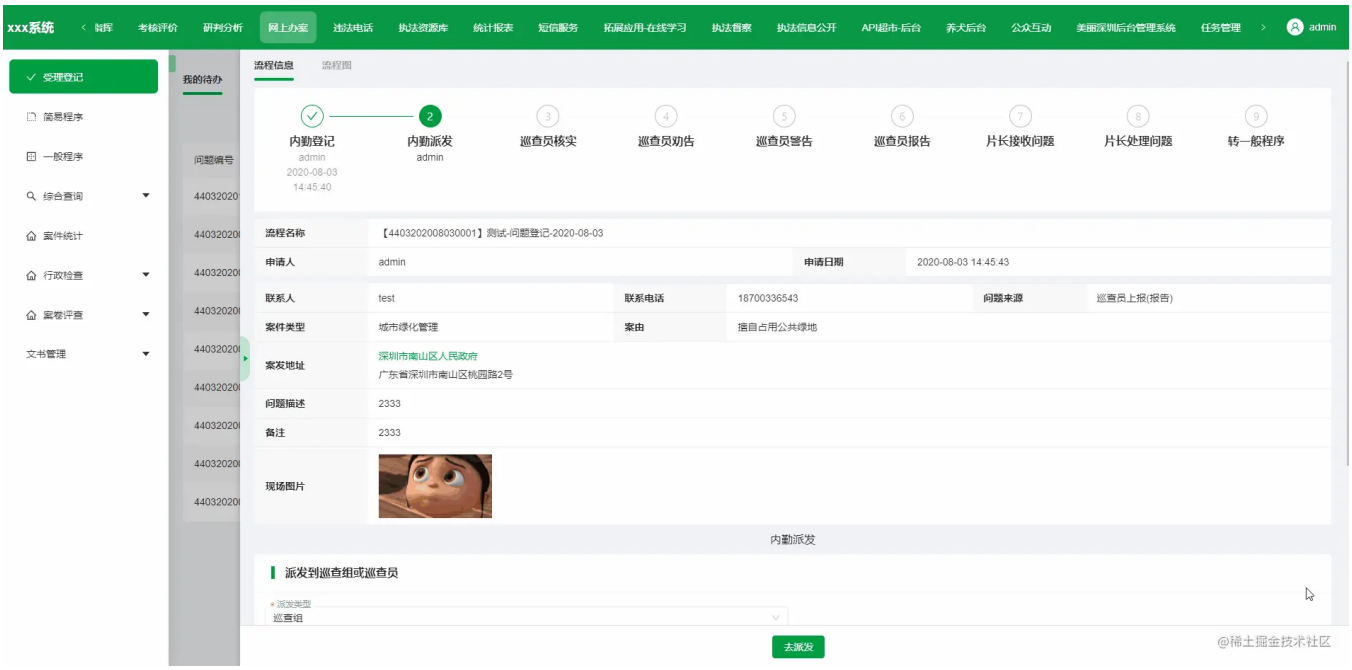
- 方案六：使用webpack-theme-color-replacer(vite版本对应的是vite-plugin-theme)，此方法是可以选任意主题色切换的，并且不需要实时编译的后台服务，**但是使用起来略显复杂**，像antd-design这类组件库提供了一个主题色生成其他梯度颜色的js方法的，能够清楚整个组件库的颜色梯度是如何与主题色关联的，这类就还好一点。还有就是这种做法只适用于颜色值，如果还需要包含border-radius、font-size等其他非颜色变量值，用这个方式可能无法做到，因为主题不只是颜色部分。

以上的方案如果都不适合你，不妨往下看看

为此还有一个方案，更简单，更优雅，更友好的适用于预设多主题的编译，几乎无需修改源码，并且无关框架组件库等，只要是基于 less 和 sass，不局限于 webpack、gulp、vite 等构建工具，就是本文重点介绍的内容：

基于 less 和 sass 的预设多主题编译方案

先看一个效果图



已封装的工具

- 主要的实现就是修改了 less 和 sass 的 render 的编译逻辑，请使用@zougt/some-loader-utils的 `getLess` 和 `getSass` 方法，替代当前构建环境中的less和sass编译器，目前在 webpack 和 vite 中使用测试过。
- 如需对编译后的主题 css 抽取成独立的文件请看 webpack 插件 `@zougt/theme-css-extract-webpack-plugin` 。
- 如需 vite 版本的只需 vite 插件@zougt/vite-plugin-theme-preprocessor。

多主题编译示例（以 sass + webpack为例）

在webpack中，只需简单配置sass-loader的属性implementation ，可以直接查看 @zougt/some-loader-utils

> 有多少个主题变量的文件，就会对经过编译器的 less/sass 文件进行编译多少次，所以 multipleScopeVars 项越多，必然会增加编译时间，建议在 开发模式 只提供一个变量文件，在 需要调试切换主题 或 生产模式 时就提供完整的变量文件个数进行打包。

webpack.config.js

```js
const path = require("path");
// const sass = require("sass");
const { getSass } = require("@zougt/some-loader-utils");
const multipleScopeVars = [
  {
    scopeName: "theme-default",
    path: path.resolve("src/theme/default-vars.scss"),
  },
  {
    scopeName: "theme-mauve",
    path: path.resolve("src/theme/mauve-vars.scss"),
  },
];
module.exports = {
  module: {
    rules: [
      {
        test: /\.scss$/i,
        loader: "sass-loader",
        options: {
          sassOptions: {
            // 不使用 getMultipleScopeVars 时，也可从这里传入 multipleScopeVars
            // multipleScopeVars
          },
          implementation: getSass({
            // getMultipleScopeVars优先于 sassOptions.multipleScopeVars
            getMultipleScopeVars: (sassOptions) => multipleScopeVars,
            // 可选项
            // implementation:sass
          }),
        },
      },
    ],
  },
};
```

主题包含的不只是颜色部分

假设目前有两种预设主题的 scss 变量文件

```scss
//src/theme/default-vars.scss

/**
*此scss变量文件作为multipleScopeVars去编译时，会自动移除!default以达到变量提升
*同时此scss变量文件作为默认主题变量文件，被其他.scss通过 @import 时，必需 !default
*/
```

```scss
$primary-color: #0081ff !default;
$--border-radius-base: 4px !default;
```

scss 复制代码

```scss
//src/theme/mauve-vars.scss

$primary-color: #9c26b0 !default;
$--border-radius-base: 8px !default;
```

一个组件的 scss

scss 复制代码

```scss
//src/components/Button/style.scss

@import "../../theme/default-vars";
.un-btn {
  position: relative;
  display: inline-block;
  font-weight: 400;
  white-space: nowrap;
  text-align: center;
  border: 1px solid transparent;
  background-color: $primary-color;
  border-radius: $--border-radius-base;
  .anticon {
    line-height: 1;
  }
}
```

编译之后

src/components/Button/style.css

css 复制代码

```css
.un-btn {
  position: relative;
  display: inline-block;
  font-weight: 400;
  white-space: nowrap;
  text-align: center;
  border: 1px solid transparent;
}
.theme-default .un-btn {
    background-color: #0081ff;
    border-radius: 4px;
}
.theme-mauve .un-btn {
```

```
    background-color: #9c26b0;
    border-radius: 8px;
}
.un-btn .anticon {
  line-height: 1;
}
```

在 `html` 中改变 classname 切换主题，只作用于 html 标签：

```html 复制代码
<!DOCTYPE html>
<html lang="zh" class="theme-default">
  <head>
    <meta charset="utf-8" />
    <title>title</title>
  </head>
  <body>
    <div id="app"></div>
    <!-- built files will be auto injected -->
  </body>
</html>
```

```js 复制代码
document.documentElement.className = "theme-mauve";
```

如果不想 html 的权重 classname,可以使用 `@zougt/theme-css-extract-webpack-plugin` 分离出独立的主题 css 文件，在线切换主题 css 文件即可：

```js 复制代码
const toggleTheme = (scopeName = "theme-default") => {
  let styleLink = document.getElementById("theme-link-tag");
  if (styleLink) {
    // 假如存在id为theme-link-tag 的link标签，直接修改其href
    styleLink.href = `/${scopeName}.css`;
    // document.documentElement.className = scopeName;
  } else {
    // 不存在的话，则新建一个
    styleLink = document.createElement("link");
    styleLink.type = "text/css";
    styleLink.rel = "stylesheet";
    styleLink.id = "theme-link-tag";
    styleLink.href = `/${scopeName}.css`;
    // document.documentElement.className = scopeName;
    document.head.append(styleLink);
  }
};
```

使用 Css Modules

如果是模块化的 scss，得到的 css 类似：

```css 复制代码
.src-components-Button-style_theme-default-3CPvz
  .src-components-Button-style_un-btn-1n85E {
  background-color: #0081ff;
}
.src-components-Button-style_theme-mauve-3yajX
  .src-components-Button-style_un-btn-1n85E {
  background-color: #9c26b0;
}
```

实际需要的结果应该是这样：

```css 复制代码
.theme-default .src-components-Button-style_un-btn-1n85E {
  background-color: #0081ff;
}
.theme-mauve .src-components-Button-style_un-btn-1n85E {
  background-color: #9c26b0;
}
```

如果是webpack，在 webpack.config.js 需要对 `css-loader` (v4.0+) 的 modules 属性添加 getLocalIdent:

```js 复制代码
const path = require("path");
// const sass = require("sass");
const { getSass } = require("@zougt/some-loader-utils");
const { interpolateName } = require("loader-utils");
function normalizePath(file) {
  return path.sep === "\\" ? file.replace(/\\/g, "/") : file;
}
const multipleScopeVars = [
  {
    scopeName: "theme-default",
    path: path.resolve("src/theme/default-vars.scss"),
  },
  {
    scopeName: "theme-mauve",
    path: path.resolve("src/theme/mauve-vars.scss"),
  },
];
module.exports = {
  module: {
```

```
        rules: [
          {
            test: /\.module.scss$/i,
            use: [
              {
                loader: "css-loader",
                options: {
                  importLoaders: 1,
                  modules: {
                    localIdentName:
                      process.env.NODE_ENV === "production"
                        ? "[hash:base64:5]"
                        : "[path][name]_[local]-[hash:base64:5]",
                    //使用 getLocalIdent 自定义模块化名称 ， css-loader v4.0+
                    getLocalIdent: (
                      loaderContext,
                      localIdentName,
                      localName,
                      options
                    ) => {
                      if (
                        multipleScopeVars.some(
                          (item) => item.scopeName === localName
                        )
                      ) {
                        //localName 属于 multipleScopeVars 的不用模块化
                        return localName;
                      }
                      const { context, hashPrefix } = options;
                      const { resourcePath } = loaderContext;
                      const request = normalizePath(
                        path.relative(context, resourcePath)
                      );
                      // eslint-disable-next-line no-param-reassign
                      options.content = `${hashPrefix + request}\x00${localName}`;
                      const inname = interpolateName(
                        loaderContext,
                        localIdentName,
                        options
                      );

                      return inname.replace(/\\?\[local\\?]/gi, localName);
                    },
                  },
                },
              },
              {
                loader: "sass-loader",
                options: {
```

```
                implementation: getSass({
                  // getMultipleScopeVars优先于 sassOptions.multipleScopeVars
                  getMultipleScopeVars: (sassOptions) => multipleScopeVars,
                  // 可选项
                  // implementation:sass
                }),
              },
            },
          ],
        },
      ],
    },
};
```

vue-cli创建的工程中使用

最近使用了vue-cli4创建开发工程（2021/8/6）

js 复制代码

```
// 在 vue.config.js 配置很简单 ， less版本只需把 getSass 改成 getLess 使用

const path = require('path');

const { getSass } = require('@zougt/some-loader-utils');

const ThemeCssExtractWebpackPlugin = require('@zougt/theme-css-extract-webpack-plugin');

const multipleScopeVars = [
    {
        scopeName: 'theme-mauve',
        name: '木槿',
        path: 'src/scss/theme-mauve.scss',
    },
    {
        scopeName: 'theme-cyan',
        name: '天青',
        path: 'src/scss/theme-cyan.scss',
    },
    {
        scopeName: 'theme-default',
        name: '墨黑',
        path: 'src/scss/theme-default.scss',
    },
];

module.exports = {
```

```
    css: {
        loaderOptions: {
            scss: {
                // 这里的选项会传递给 sass-loader
                implementation: getSass({
                    // getMultipleScopeVars优先于 sassOptions.multipleScopeVars
                    getMultipleScopeVars: (sassOptions) => multipleScopeVars.map((item) => {
                        return { ...item, path: path.resolve(item.path) };
                    }),
                }),
            },
        },
    },
      chainWebpack: (config) => {
        config
            .plugin('ThemeCssExtractWebpackPlugin')
            .use(ThemeCssExtractWebpackPlugin, [
                {
                    multipleScopeVars,
                    // extract: process.env.NODE_ENV === 'production',
                    extract: false,
                },
            ]);
    },
}
```

为此经过一段时间的研究，实现了基于less、sass的(新、旧)项目预设主题编译方案，使得此类需求变得很简单，并且兼容性很好，在此作一个分享。