34 使用文件形式直接集成

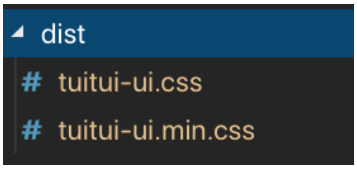
更新时间：2019-08-19 09:43:30

> 对自己不满是任何真正有才能的人的根本特征之一。
>
> ——契诃夫

我们这一节来做 UI 样式库的集成，在上一节的开篇中也提到了，现在这个样式库还存在以下三个问题：

1. 使用 @import 引用了太多的文件，会拖慢加载速度，不可能直接用在生产环境。
2. 没有处理兼容性的问题，要添加兼容性写法。
3. 文件没有压缩，体积会比较大，同样会影响加载速度。

我们这一节的目的就是要解决这几个比较关键的问题。最后，我们会得到一个解决掉这些问题后的最终文件，就可以直接在生产环境使用了。最终会生成下面这两个目标文件：



其中 /dist/tuitui-ui.min.css 就是我们可以在生产环境使用的最终文件。此外还生成了一个非压缩的 tuitui-ui.css，这个文件和压缩后的文件内容是一样的，只不过没有进行压缩。生成这个文件是为了可以看到带有格式的最终文件，如果出现问题可以快速定位出问题的地方。

环境的搭建

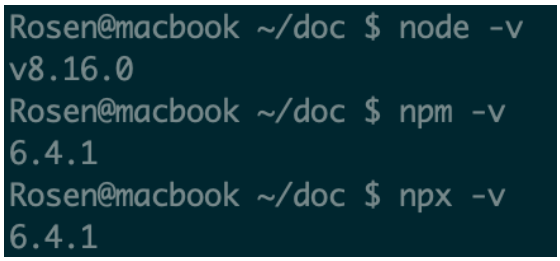
在做打包的时候，我们需要使用一些工具来辅助，我们先把这些工具介绍一下。

一、**Node.js + npm + npx**

首先是 Node.js + npm + npx 这个套装，下来介绍这三个都是做什么的：

> **@ Tips:**
>
> - Node.js 是运行在服务端的 JS 语言，可以用来处理前端的文件，它是我们这一节要用到的工具的环境基础。
> - npm (Node.js Package Manager)，是 Node.js 的包管理工具，主要用来管理 Node.js 项目的依赖包，包括对包的安装、查看和卸载等。
> - npx，这个工具是 Node.js 插件的执行工具，它可以执行 npm 包中包含的命令。

然后我们安装 Node.js，Windows 和 Mac 用户都可以在Node.js官网 下载对应的安装包（Widnows 下载 .msi 格式的文件，Mac 下载 .pkg 格式的文件），然后直接安装就可以了。我这里用的是 8.16.0 的版本，里面会包含 Node.js、npm 和 npx 三个工具。安装好以后，在命令行里执行下面命令来查看 Node.js、npm 和 npx 的版本：



如果发现 npx 没有的话，可以执行下面命令手动安装：

```
npm i npx@6.4.1 -g
```

最后一步，就是要把我们之前的项目，使用 npm 初始化一下，这样才可以使用 npm 来安装需要的插件。在项目目录里执行下面的命令：

```
npm init
```

随后输入项目相关的信息，包括名称、版本、关键词等，这里什么都不输入，直接回车也不会影响使用，以后也可以再改。

```
Rosen@macbook ~/doc/tuitui-ui (master)$ npm init
This utility will walk you through creating a package.json file.
It only covers the most common items, and tries to guess sensible defaults.

See `npm help json` for definitive documentation on these fields
and exactly what they do.

Use `npm install <pkg>` afterwards to install a package and
save it as a dependency in the package.json file.

Press ^C at any time to quit.
package name: (tuitui-ui)
version: (1.0.0)
description: tuitui-ui
entry point: (index.js) -
test command: -
git repository: (git@gitee.com:tuituitech/tuitui-ui.git)
keywords: tuitui-ui tuituitech ui
author: tuituitech
license: (ISC)
About to write to /Users/Rosen/doc/tuitui-ui/package.json:
```

输入完这些信息以后，最后会让你输入“yes”来确认，完成后就会在项目根目录中，生成一个 package.json 文件：

```
{
  "name": "tuitui-ui",
  "version": "1.0.0",
  "description": "tuitui-ui",
  "main": "-",
  "scripts": {
    "test": "-"
  },
  "repository": {
    "type": "git",
    "url": "git@gitee.com:tuituitech/tuitui-ui.git"
  },
  "keywords": [
    "tuitui-ui",
    "tuituitech",
    "ui"
  ],
  "author": "tuituitech",
  "license": "ISC"
}
```

这个文件就是 npm 的配置文件了，这样，有关 Node.js 的工作就完成了。

二、**postcss**

这里要介绍的第二个内容是，基于 Node.js 的postcss工具。这是一个 CSS 处理工具，在使用的时候还需要搭配其他的插件来配合。我们先使用 npm 把这个插件安装上，这里我们直接使用命令行工具来安装这个插件，所以安装的插件名称应该是“postcss-cli”。

```
npm i postcss-cli@6.1.3 -D
```

这条命令里 -D 的含义是把这个包的版本信息记录在 package.json 里的 devDependencies 字段里，表示这是在开发过程会使用到的插件。安装完成以后会在 package.json 里出现这个插件的信息：

这样这个插件就装好了，我们可以用下面的命令来测试一下：

```
npx postcss src/tuitui-ui.css -o dist/tuitui-ui.css
```

这条命令的含义就是使用 npx 调用了 postcss-cli 提供的 postcss 命令，后面跟着的就是一些参数。这条命令实际的格式是：

```
npx postcss 要处理的 CSS 文件位置 -o 生成目标文件的位置
```

执行完这一条命令后，就会在根目录下生成一个 dist 目录，里面就是刚生成的 tuitui-ui.css。但是命令行里会提醒你，没有使用任何插件，最后生成的文件基本没什么处理，只是在主文件的最后加了个 sourceMapping，我们后面再加上其他插件来处理。走到这里没有问题的话，就说明我们的环境就配置好了。

对 @import 的处理

接下来处理 @import 的问题，这里要使用到“postcss-import”这个插件。这个插件可以把 @import 方式引入的本地路径转变成这个路径里的 CSS 语句。经过这样处理以后，所有的样式文件就会被集中到目标文件里，@import 只剩下一个远程的路径，项目本地的 @import 就没有了。

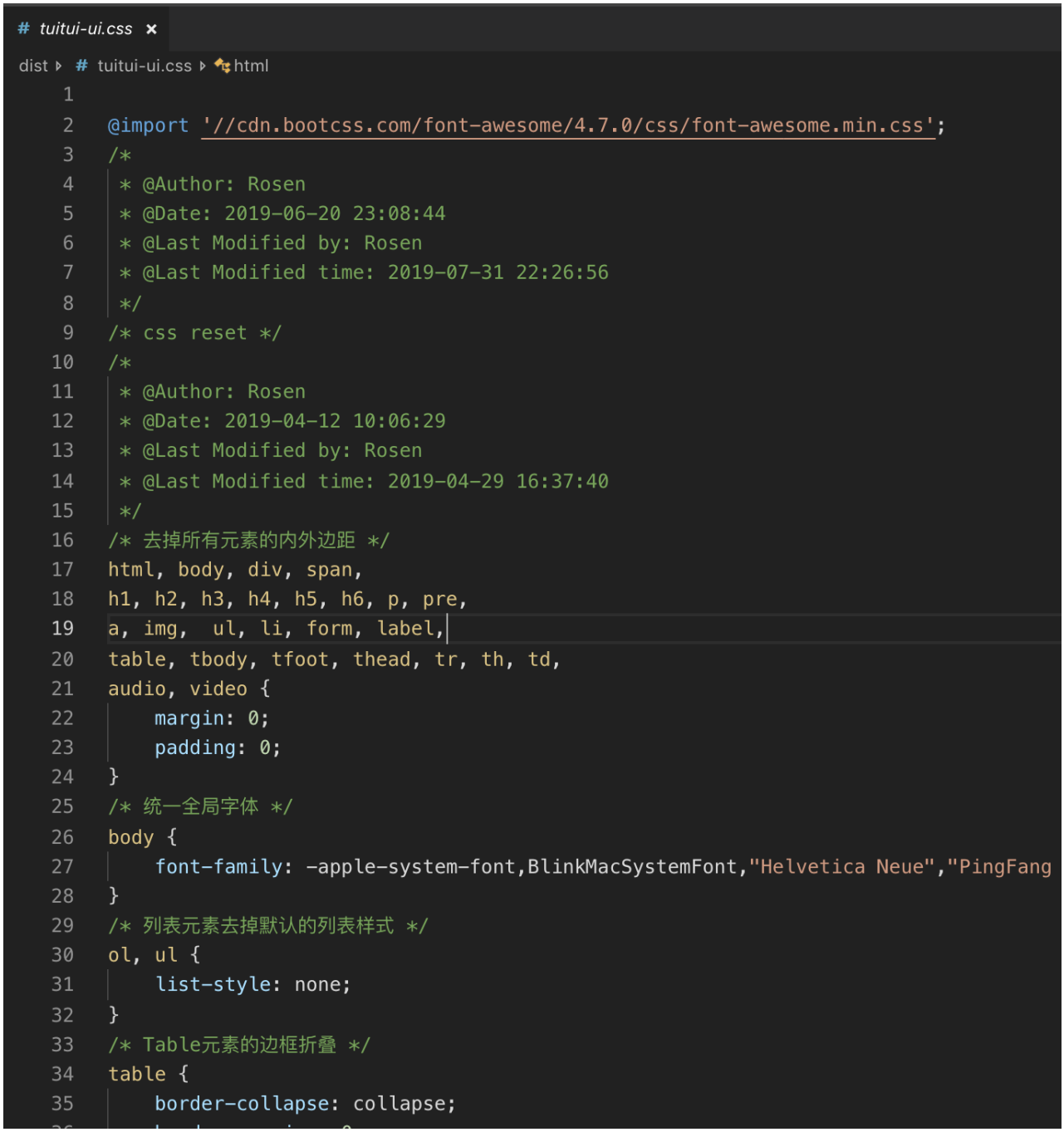
我们先来安装一下这个插件：

```
npm i postcss-import@12.0.1 -D
```

安装好以后，我们就可以在刚才的打包命令里添加这个插件来完成对@import的处理：

```
npx postcss src/tuitui-ui.css -o dist/tuitui-ui.css -u postcss-import --no-map
```

这里我们在之前的命令后面又加了些东西，“-u postcss-import”是表示在使用 postcss 处理 CSS 文件时要使用“postcss-import”插件。最后多的“–no-map”是为了去掉生成文件最后的sourceMapping信息。经过这样处理，会发现目标文件里的内容就变多了，各个文件的内容都被集成到这一个文件里了：



> **@ Tips:**
>
> 这里要注意两个问题：
>
> 1、文件中引入字体图标库用的远程 @import 并不会被替换，但会被提到文件的最前面。
>
> 2、我们的项目里没有涉及图片，如果需要处理图片的项目可以使用“postcss-url”来处理图片的路径问题。

对兼容性的处理

接下来要处理兼容性问题。目前来看，大部分 CSS3 的样式已经可以在新的浏览器上运行了，但是，有些属性或者属性值是要添加浏览器前缀后才可以用，所以我们还是要对代码进行兼容处理。在处理移动端样式的问题时，我们不再手动添加浏览器前缀，这里要介绍一个叫做“Autoprefixer”的插件。这个插件会分析代码，并根据Can I use这个网站提供的兼容性数据来自动添加兼容写法。下来我们先安装一下这个插件：

```
npm i autoprefixer@9.6.1 -D
```

安装好以后就可以直接在命令中使用这个插件了：

```
npx postcss src/tuitui-ui.css -o dist/tuitui-ui.css -u postcss-import autoprefixer --no-map
```

这条命令执行后，就可以去文件中找到一些样式被添加上了兼容写法，以我们之前开发的图标，上下振动的样式为例。原有代码：

```
/* 垂直方向上振动 */
.fa-vibrate-y{
    animation: fa-vibrate-y 1.5s infinite ease-in;
}
/* 振动轨迹 */
@keyframes fa-vibrate-y{
    0% {
        transform: translateY(-10%);
    }
    50% {
        transform: translateY(10%);
    }
    100% {
        transform: translateY(-10%);
    }
}
```

处理后的代码：

```
/* 垂直方向上振动 */
.fa-vibrate-y{
    -webkit-animation: fa-vibrate-y 1.5s infinite ease-in;
            animation: fa-vibrate-y 1.5s infinite ease-in;
}
/* 振动轨迹 */
@-webkit-keyframes fa-vibrate-y{
    0% {
        transform: translateY(-10%);
    }
    50% {
        transform: translateY(10%);
    }
    100% {
        transform: translateY(-10%);
    }
}
@keyframes fa-vibrate-y{
    0% {
        transform: translateY(-10%);
    }
    50% {
        transform: translateY(10%);
    }
    100% {
        transform: translateY(-10%);
    }
}
```

经过前后对比，就可以发现，原有代码中的 animation 和 @keyframes 都被添加了兼容写法，但是 transform 并没有添加兼容写法。这是因为默认情况下， autoprefixer 会给没有废弃的且占有率 > 0.5% 或者是最新发布的两个版本以内的浏览器提供兼容支持，但是这种配置不一定安全，所以我们也可以自己去调节这个兼容范围的配置。

autoprefixer 的兼容性配置方式使用的是browserslist工具提供的规则，所以我们按着 browserslist 的规则来配置需要的规则就可以了，这个工具需要在项目的根目录上建立“.browserslistrc”这个文件：

```
> 0.5%
last 2 version
not ie <= 10
not ie_mob <= 10
```

我们使用的是这样的配置，指定了对覆盖率大于 0.5% 或者是最新 2 个版本内的浏览器提供支持，并且可以不支持旧版本的 IE 和 IE_Mob。和默认配置相比，是去掉了not dead这个选项，表示对已经废弃的浏览器版本也提供前面规则的支持，这样会更安全一点。经过这样的配置再重新执行前面的命令，就会发现像 transform 这种属性也会被添加兼容写法：

```
/* 垂直方向上振动 */
.fa-vibrate-y{
    -webkit-animation: fa-vibrate-y 1.5s infinite ease-in;
            animation: fa-vibrate-y 1.5s infinite ease-in;
}
/* 振动轨迹 */
@-webkit-keyframes fa-vibrate-y{
    0% {
        -webkit-transform: translateY(-10%);
                transform: translateY(-10%);
    }
    50% {
        -webkit-transform: translateY(10%);
                transform: translateY(10%);
    }
    100% {
        -webkit-transform: translateY(-10%);
                transform: translateY(-10%);
    }
}
@keyframes fa-vibrate-y{
    0% {
        -webkit-transform: translateY(-10%);
                transform: translateY(-10%);
    }
    50% {
        -webkit-transform: translateY(10%);
                transform: translateY(10%);
    }
    100% {
        -webkit-transform: translateY(-10%);
                transform: translateY(-10%);
    }
}
```

我们使用的这个配置方式比较笼统，而 browserslist 可以支持区分浏览器的设置，同学们有兴趣可以去研究下这个配置。到这里我们对兼容性的处理也完成了。

压缩CSS文件

最后一个要处理的问题就是压缩了，这里要使用的插件是“cssnano”。这个工具会把CSS文件里的注释和空格都去掉，经过处理后就可以生成我们最终需要的压缩文件了。我们还是先来安装这个插件：

```
npm i cssnano@4.1.10 -D
```

这个插件也可以直接按着默认配置来使用，可以按下面的命令进行执行：

```
npx postcss src/tuitui-ui.css -o dist/tuitui-ui.min.css -u postcss-import autoprefixer cssnano --no-map
```

这里我们把输出的文件名改成了 dist/tuitui-ui.min.css，这表示是压缩后的文件。执行完这条命令以后，原来文件中的注释和空格就都被去掉了，最后会生成一个新的 /dist/tuitui-ui.min.css，如下图：

这样我们把文件压缩的问题也解决了，最后生成的这个文件我们就可以直接在别的项目使用了。

shell脚本的编写

最后一步，我们来写一个 Shell 脚本来记录下前面的命令，免得每次打包都要输入那么一大串的命令。Shell 脚本通常是在Linux 或者 Mac 系统上才能使用的一种命令式语言，但 Windows 上使用 gitbash 这类工具的话也能支持一部分 Shell 命令。这一块的内容完全是作为了解，同学们有兴趣就试着玩一玩，不感兴趣的话直接执行之前的命令也是一样的。我们先在根目录下建一个名为“shell”的目录，然后在里面放一个“build.sh”的文件：

```
# 清空dist目录中的旧文件
echo '正在清除原有dist文件...'
rm -rf dist/*.css

# 打包出不压缩的CSS文件tuitui-ui.css
echo '正在生成tuitui-ui.css文件...'
npx postcss src/tuitui-ui.css -o dist/tuitui-ui.css -u postcss-import autoprefixer --no-map

# 打包出被压缩的CSS文件tuitui-ui.min.css
echo '正在生成tuitui-ui.min.css文件...'
npx postcss src/tuitui-ui.css -o dist/tuitui-ui.min.css -u postcss-import autoprefixer cssnano --no-map
```

这个脚本很简单，就是顺序执行几条命令：

- 首先是清空原有的 dist 目录，来避免一些不必要的错误。
- 然后执行生成 /dist/tuitui-ui.css 文件的命令，就是刚才在解决兼容性问题时用的那条命令。
- 最后执行生成 /dist/tuitui-ui.min.css 这个压缩文件的命令，也就是上一步我们用过的那条。

制作完这个脚本以后，再想打包项目，就可以直接在项目里调用这个脚本了。可以在项目根目录执行：

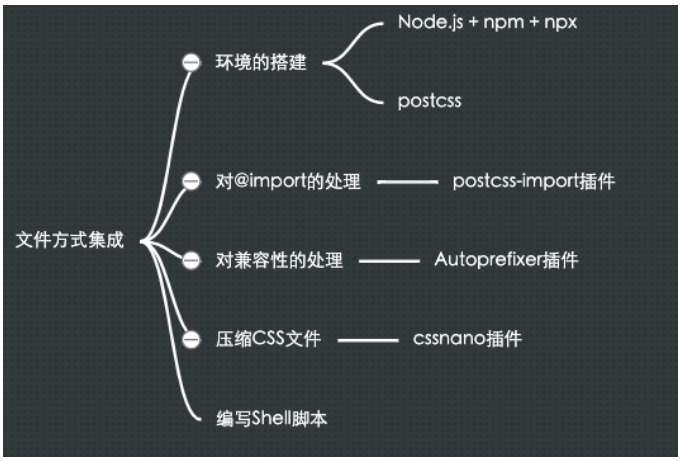
```
./shell/build.sh
```

执行后，就会在 dist 目录里生成新的文件：

```
Rosen@macbook ~/doc/tuitui-ui (master)*$ ./shell/build.sh
正在清除原有dist文件...
正在生成tuitui-ui.css文件...
正在生成tuitui-ui.min.css文件...
Rosen@macbook ~/doc/tuitui-ui (master)*$ ls -al dist/
total 88
drwxr-xr-x   4 Rosen  staff    128  8 13 23:55 .
drwxr-xr-x  18 Rosen  staff    576  8 13 23:43 ..
-rw-r--r--   1 Rosen  staff  26907  8 13 23:55 tuitui-ui.css
-rw-r--r--   1 Rosen  staff  14999  8 13 23:55 tuitui-ui.min.css
```

结语

到这里我们这一节的内容就完成了，这一节有涉及到一些和 CSS 没有太大关系的技术栈，比如 npm、npx、shell 等。对于没有这些知识基础的同学，可能会感觉有点困惑，这里建议同学们下去，把 npm 和 npx 相关的用法都学习一下，作为前端开发工程师，这些工具肯定是必不可少的；而对于 shell 脚本，可以作为了解，不做硬性要求。这一节我们使用的是比较简单的命令式打包，用的也基本都是插件的默认配置。如果有更复杂的需求，这些工具都是需要做配置的。这里给同学们留个作业，下去可以试着把这个打包的过程用 webpack 来完成一下，这样就可以对插件做更丰富的配置了。这一节的内容结构如下：



经过对代码的发布，同学们在其他项目中也可以直接引用推推UI的在线地址tuitui-ui.min.css了。我们这一节的内容就到这。

}

33 UI 框架的集成

35 使用 npm 形式集成