# Vue中图片路径如何正确引入

如果是相对路径，在打包之后路径会发生变化，导致图片找不到，不能正常加载。

如果是绝对路径，在部署项目的时候，项目更换位置，绝对路径会相应的发生变化，例如：在A目录下的绝对路径/A/static/images/x.png，如果部署的目录发生变化，更换到B项目下，则为/A/static/images/x.png。

Vue中图片的加载一般使用别名，在build/webpack.base.conf.js中，做如下配置：

alias: {

 'src': path.resolve(\_\_dirname, '../src'),

 'assets': path.resolve(\_\_dirname, '../src/assets'),

 'myImg': path.resolve(\_\_dirname, '../static/images')

}

在引用路径的时候，只需要使用别名，注：要在别名前加~，告诉加载器它是一个模块，而不是相对路径

<style>

.logo {

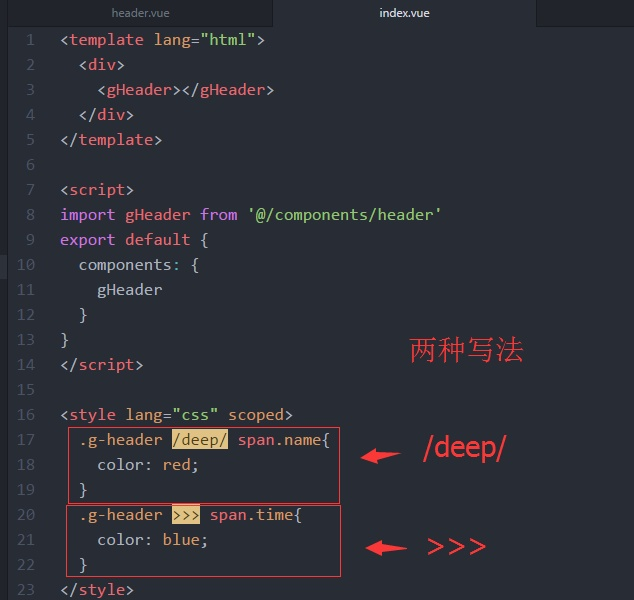
background: url(~myImg/bg.jpg)

}

</style>

# Vue中样式冲突、污染问题

scoped原理：Vue-loader 在编译的过程中会为组件每一个元素节点增加 scopeId 作为属性，同时为所有的样式类加上属性选择器 scopeId，从而达到隔离的效果，就是说加了scoped只能作用于本子页面，这样的话会导致父页面的样式不会作用于子组件



父组件



子组件

对于某些子页面，需要继承父页面样式的，一般使用深度选择器/deep/,如上图，父组件的样式可以作用于子组件。

<style scoped>

@import "../static/css/user.css";

.user-content{

background-color: #3982e5;

}

</style>

使用@import引入外部样式表作用域依然是全局的，在生产环境中不要使用@import引入css，因为在请求到的css中含有@import引入css的话，会发起请求把@import的css引进来，多次请求浪费不必要的资源。

@import并不是引入代码到<style></style>里面，而是发起新的请求获得样式资源，并且没有加scoped，所以作用域是全局的。

# Vue自定义全局变量或方法（表单验证-基于elementUI）

* 创建validate.js  
  exports.install = function (Vue, options) {

Vue.prototype.filter\_rules= function (item){

};

};

这样一个filter\_rules的方法就创建成功了，执行完下一步之后，就可以在任何页面的标签上使用filter\_rules方法

* 在main.js中使用validate.js

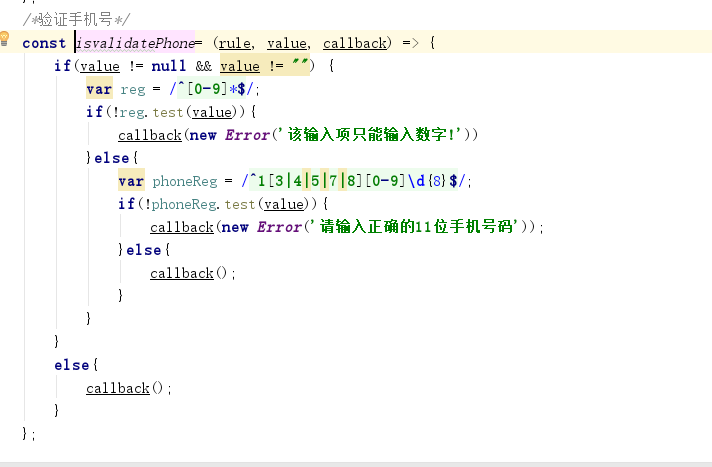
import Validate from "../static/js/validate";

Vue.use(Validate);

* 添加验证

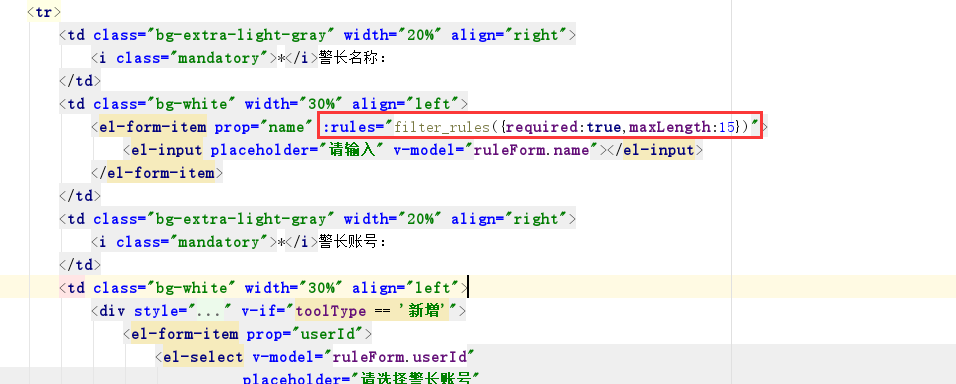


该图上部分是直接调用的elementUI现有的验证，下部分是自定义的验证



该图是自定义验证的具体验证方法，总的思路还是使用了elementUI的验证，但是elementUI的验证需要在每个页面都去写，所以在此处，封装一个全局的验证方法，进行验证。

* 使用验证



# Vue使用mock实现前后端分离

1. 对ajax请求的拦截以及数据造假（Mock）

* 安装依赖

执行命令：npm install mockjs --save-dev

* 创建mock.js（规则）对请求拦截

在src下创建一个mock.js文件，内容如下，具体步骤为引入mockjs，然后造假数据，最终调用Mock.mock(url,type,data)。url为拦截的请求地址，type为请求方式（可不填），data为返回的数据（也可以是造假规则）。注：造假数据=Mock.mock(造假规则)



* main.js使用mock

在main.js使用我们上边创建的mock.js，具体操作require('./mock.js')

* 数据造假

参照：http://www.cnblogs.com/yangguoe/p/9364017.html

1. mock-adapter对axios请求拦截以及数据造假（同上）

* 安装依赖

执行命令：npm install mockjs --save-dev

* 创建mock.js（规则）对请求拦截

import axios from 'axios';

import MockAdapter from 'axios-mock-adapter';

// 设置模拟调试器实例

var mock = new MockAdapter(axios);

// 模拟任意GET请求到 /users

//reply的参数为 (status, data, headers)

mock.onGet('/users').reply(200,data);

data还是使用mock的造假规则，var data = Mock.mock(造假规则)

* main.js使用mock

在main.js使用我们上边创建的mock.js，具体操作require('./mock.js')