# MintUI-- Unit06

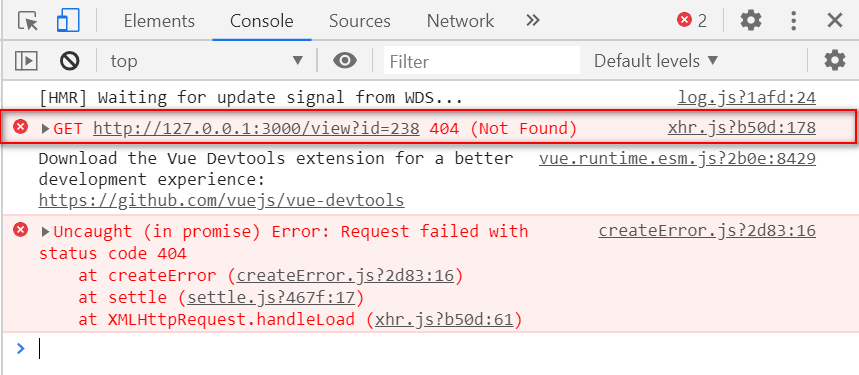
# 1.文章详细的实现

现在需要获取出地址栏中的参数值，并且在mounted钩子函数中发送异步请求，去获取该篇文章的详细信息，所以：  
第一步：获取地址栏中的参数值

this.$route.params.id

第二步：在mounted钩子函数中发送请求，示例代码如下：

mounted(){  
 let id = this.$route.params.id;  
 this.axios.get('/view?id=' + id).then(res=>{  
 //....   
 });  
}



其根本原因是：服务器根本没有对应的路由地址！！！所以：

A.在app.js中创建/view的 GET类型的路由，示例代码如下：

// 获取指定文章详细信息的接口
  
server.get('/view',(req,res)=>{
  
 //获取地址栏中的id参数
  
 let id = req.query.id;
  
 //以id为条件进行文章相关信息的查找操作(一会儿还要进行调整)
  
 let sql = 'SELECT id,subject,content FROM xzqa\_article WHERE id=?';
  
 pool.query(sql,[id],(err,results)=>{
  
 if(err) throw err;
  
 res.send({message:'查询成功',code:1,results:results[0]});
  
 });
  
});

B.在脚手架中进行服务器数据的接收，示例代码如下：

data(){
  
 return {
  
 info:{}
  
 }
  
},
  
mounted(){
  
 //获取动态参数
  
 let id = this.$route.params.id;
  
 //发送请求
  
 this.axios.get('/view?id=' + id).then(res=>{
  
 this.info = res.data.results;
  
 });
  
 }

C.直接在页面组件中输出内容，示例代码如下：

<h1 style="border:2px solid #f00;">{{info.subject}}</h1>
  
<div v-html="info.content"></div>

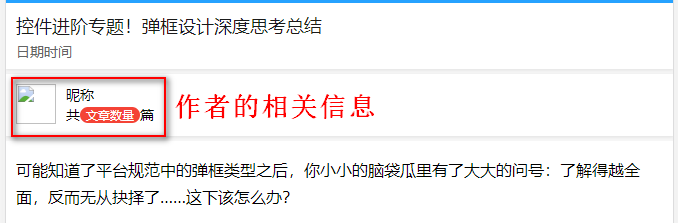
采用v-html指令的原因是：在数据库中文章的正文存储了大量的HTML标签，而默认情况下，Vue为防止XSS攻击，将对内容的特殊符号，如>、<等进行转换操作 -- HTML实体，而现在本案例中依然要输出正常的HTML 标签，所以需要通过v-html指令。

<style scoped>内的样式不会对v-html指令内的内容生效。如果要修饰v-html指令内的内容的话，应该使用<style>定义全局样式，示例代码如下：

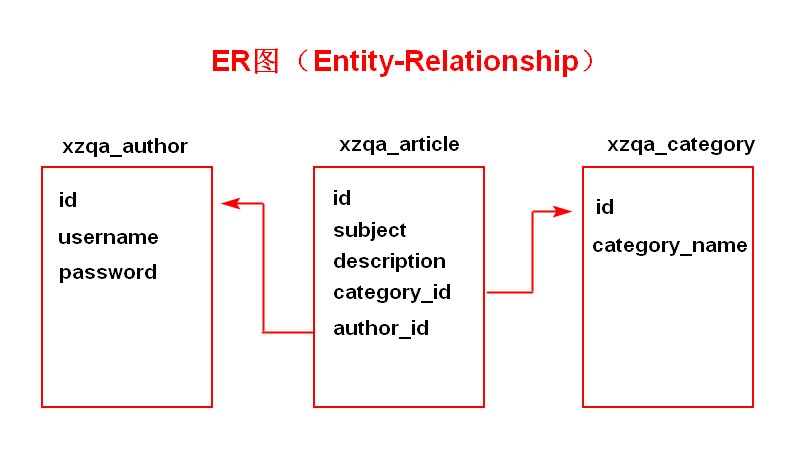
<style>
  
 .article-content p{
  
 padding:5px 0;
  
 font-size:16px;
  
 }
  
 .article-content img{
  
 max-width: 100%;
  
 display: block;
  
 }
  
</style>

https://cn.vuejs.org/v2/api/#v-html

文章详细信息中还包括作者的相关信息，如下图所示：



回顾一下原来的数据表ER图



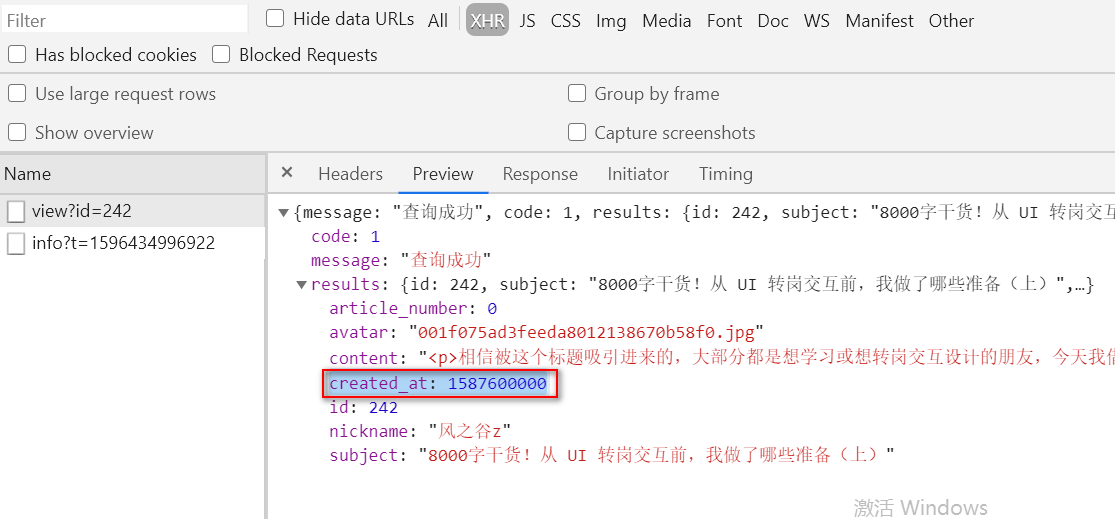
根据上述的ER图可以发现：数据表xzqa\_article与xzqa\_author之间存在外键关联。所以现在需要修改app.js 中的/view路由，以进行连接的查找操作，示例代码如下：

// 获取指定文章详细信息的接口
  
server.get('/view',(req,res)=>{
  
 //获取地址栏中的id参数
  
 let id = req.query.id;
  
 //以id为条件进行文章相关信息的查找操作(一会儿还要进行调整)
  
 let sql = 'SELECT a.id,subject,content,nickname,avatar,article\_number FROM xzqa\_article AS a INNER JOIN xzqa\_author AS u ON author\_id = u.id WHERE a.id=?';
  
 pool.query(sql,[id],(err,results)=>{
  
 if(err) throw err;
  
 res.send({message:'查询成功',code:1,results:results[0]});
  
 });
  
});

现在在文章详细信息中，除发表日期外，其余都是完整的。于是还需要修改app.js，并且/view的SQL中添加该字段才可以。

let sql = 'SELECT a.id,subject,content,nickname,avatar,article\_number,created\_at FROM xzqa\_article AS a INNER JOIN xzqa\_author AS u ON author\_id = u.id WHERE a.id=?'

此时脚手架的运行结果如下：



现在问题是：如何将类似1587600000的数字转换成可识别的日期时间格式？ -- Moment

# 2.Moment

Moment是一个JavaScript日期时间处理类库。

http://momentjs.cn/

安装

npm install --save moment

解析 -- 将不同的日期格式转换成Moment对象

//将当前日期和时间转换成Moment对象
  
moment();
  
//将指定的日期转换成Moment对象
  
moment('2020-8-3');
  
//将Unix时间戳转换成Moment对象
  
//时间戳(Timestamp)，指公元1970-01-01 00:00:00到现在经历的秒数
  
moment.unix(Unix时间戳)

moment对象的format()方法

该方法用于格式化moment 对象，其语法结构是：

moment.format("格式")

Y四位的年份

MM两位的月份

DD两位的日期

HH两位的小时

mm两位的分钟

ss两位的秒

在Vue使用moment：需要在main.js完成以下代码：

import moment from 'moment';
  
Vue.prototype.moment = moment;

Badge组件

Badge组件用于实现徽章，其语法结构是：

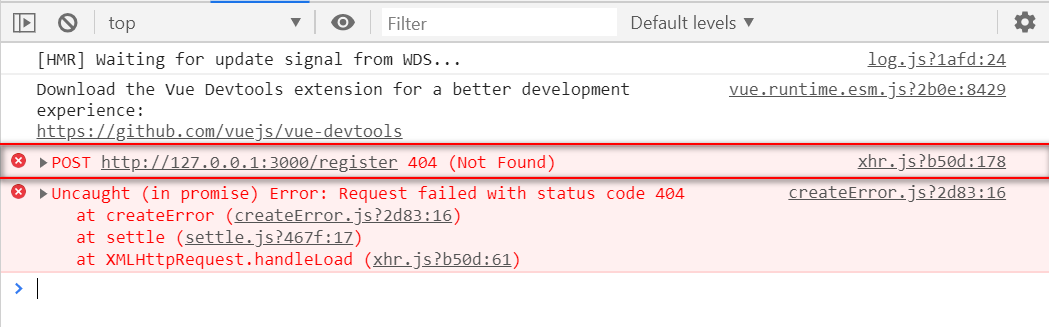
<mt-badge type="primary|success|error|warning" size="small|large|normal" color="自定义颜色">
  
 ...
  
</mt-badge>

# 3.用户注册

在前期已经实现了用户注册的基本的页面结构及客户端的业务，但是在所有信息合法的情况下，必须去服务器上校验该用户是否已经存在，如果存在，则返回错误信息给客户端；如果不存在，则直接在用户上注册成功。所以现在：

this.axios.post('/register','username=' + this.username + '&password=' + this.password).then(res=>{
  
 //...
  
});

如果在客户端所有的信息都合法的情况下，单击“免费注册”按钮的话，运行结果如下：



其原因是：在服务器上没有相应的API,所以:

第一步：在app.js中创建/register的POST类型的路由，示例代码如下:

//用户注册的接口
  
server.post('/register',(req,res)=>{
  
   
});

第二步：在获取以POST方式提交的相关信息 -- BodyParser模块

npm install --save body-parser

进行相关的配置操作

//引入body-parser模块
  
const bodyParser = require('body-parser');
  
   
//将bodyParser作为Server中间件使用
  
server.use(bodyParser.urlencoded({
  
 extended:false
  
}));

现在可以通过req.body.xx来获取以POST方式提交的数据了，示例代码如下：

//用户注册的接口
  
server.post('/register',(req,res)=>{
  
 //获取用户提交的用户名等信息
  
 let username = req.body.username;
  
 console.log(username);
  
});

接下来就应该以当前的用户名为条件，在xzqa\_author数据表中查找该用户是否存在，如果用户存在,则产生合法的错误信息；如果用户不存在,则将用户的相关信息写入数据表，示例代码如下：

//用户注册的接口
  
server.post('/register',(req,res)=>{
  
 //获取用户提交的用户名等信息
  
 let username = req.body.username;
  
 //以用户名为条件进行查找操作,如果用户存在,则产生合法的错误信息
  
 //如果用户不存在,则将用户的相关信息写入数据表 -- xzqa\_author
  
 let sql = 'SELECT COUNT(id) AS count FROM xzqa\_author WHERE username=?';
  
 pool.query(sql,[username],(err,results)=>{
  
 if(err) throw err;
  
 if(results[0].count){
  
 res.send({message:'注册失败',code:0});
  
 } else {
  
 //获取密码信息
  
 let password = req.body.password;
  
 console.log(password);
  
 }
  
 })
  
   
});

密码是在xzqa\_author表中是采用MD5进行加密的，所以在写入记录时，必须以密码字段通过MySQL数据库本身提供的MD5()函数进行加密，此时的SQL成了:

let sql = 'INSERT xzqa\_author(username,password) VALUES(?,MD5(?))';

综上，现在/register路由示例代码如下：

//用户注册的接口
  
server.post('/register',(req,res)=>{
  
 //获取用户提交的用户名等信息
  
 let username = req.body.username;
  
 //以用户名为条件进行查找操作,如果用户存在,则产生合法的错误信息
  
 //如果用户不存在,则将用户的相关信息写入数据表 -- xzqa\_author
  
 let sql = 'SELECT COUNT(id) AS count FROM xzqa\_author WHERE username=?';
  
 pool.query(sql,[username],(err,results)=>{
  
 if(err) throw err;
  
 if(results[0].count){
  
 res.send({message:'注册失败',code:0});
  
 } else {
  
 //获取密码信息
  
 let password = req.body.password;
  
 //完成数据写入操作
  
 let sql = 'INSERT INTO xzqa\_author(username,password) VALUES(?,MD5(?))';
  
 pool.query(sql,[username,password],(err,results)=>{
  
 if(err) throw err;
  
 res.send({message:'注册成功',code:1});
  
 });
  
 }
  
 })
  
   
});

最后在客户端接收服务器返回的信息，并且根据返回的信息，在页面组件中完成不同的功能，示例代码如下：

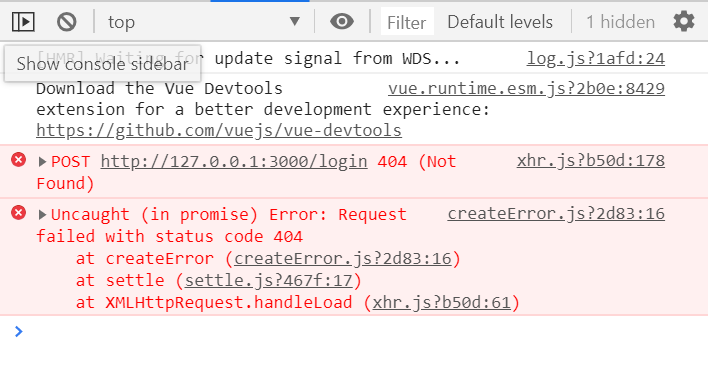
this.axios.post('/register','username=' + this.username + '&password=' + this.password).then(res=>{
  
 //注册失败
  
 if(res.data.code == 0){
  
 this.$messagebox("注册提示","对不起,用户已存在");
  
 } else {
  
 this.$router.push('/');
  
 }
  
});

# 4.用户登录

用户登录时仍然以发送POST类型的请求，请求地址为/login,所以现在脚手架中可以以下列示例代码发送请求了：

this.axios.post('/login','username=' + this.username + '&password=' + this.password).then(res=>{
  
   
});

此时运行后，结果如下图所示：



之所以产生错误：是因为服务器不存在指定的接口，所以：

第一步：在app.js创建/login的POST类型的接口，示例代码如下：

// 用户登录接口
  
server.post('/login',(req,res)=>{
  
   
});

第二步：获取POST提交的数据，示例代码如下：

//获取用户名和密码信息
  
let username = req.body.username;
  
let password = req.body.password;

现在要以获取到的用户名和密码为条件进行查询操作了，示例代码如下：

// 用户登录接口
  
server.post('/login',(req,res)=>{
  
 //获取用户名和密码信息
  
 let username = req.body.username;
  
 let password = req.body.password;
  
 //以用户名和密码为条件进行查找，如果找到，则代表用户登录成功;否则代表登录失败
  
 let sql = 'SELECT id,username FROM xzqa\_author WHERE username=? AND password=MD5(?)';
  
 pool.query(sql,[username,password],(err,results)=>{
  
 if(err) throw err;
  
 if(results.length == 1){
  
 res.send({message:'登录成功',code:1});
  
 } else {
  
 res.send({message:'登录失败',code:0});
  
 }
  
 });
  
});

最后在客户端接收服务器返回的信息，并且根据返回的信息进行不同的操作，示例代码如下：

this.axios.post('/login','username=' + this.username + '&password=' + this.password).then(res=>{
  
 //登录成功
  
 if(res.data.code == 1){
  
 this.$router.push('/');
  
 } else {
  
 this.$messagebox("登录提示","对不起,用户名或密码错误");
  
 }
  
});

用户登录成功之后，暂时没有任何变化 --- 要想看到变化，需要 Vuex与WebStorage的配合使用！！！