

Mint UI Unit06

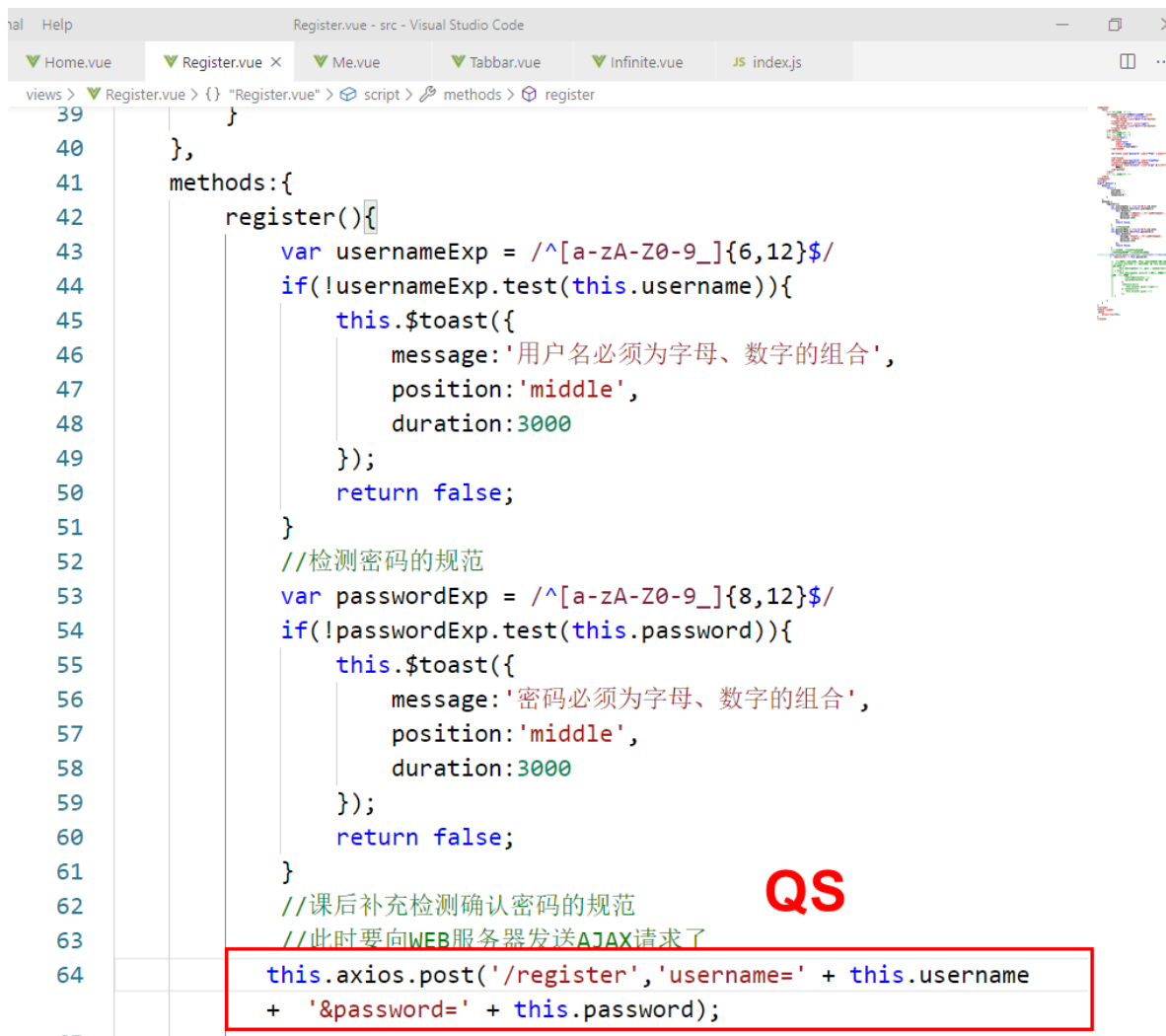
1.用户注册

当用户输入用户名和密码并且通过 JS 的检测后，此时要向 WEB 服务器发送 AJAX 请求，以完成用户的注册行为。

只要涉及隐私信息(如密码等)的表单，都必须通过 POST 方式提交

GET 提交适用于：（1）URL 地址栏传递（2）搜索引擎

代码截图如下：



```
39    },
40  },
41  methods: {
42    register() {
43      var usernameExp = /^[a-zA-Z0-9_]{6,12}$/
44      if (!usernameExp.test(this.username)) {
45        this.$toast({
46          message: '用户名必须为字母、数字的组合',
47          position: 'middle',
48          duration: 3000
49        });
50        return false;
51      }
52      //检测密码的规范
53      var passwordExp = /^[a-zA-Z0-9_]{8,12}$/
54      if (!passwordExp.test(this.password)) {
55        this.$toast({
56          message: '密码必须为字母、数字的组合',
57          position: 'middle',
58          duration: 3000
59        });
60        return false;
61      }
62      //课后补充检测确认密码的规范
63      //此时要向WEB服务器发送AJAX请求了
64      this.$axios.post('/register', 'username=' + this.username
65        + '&password=' + this.password);
--
```

但在测试时即使成功输入了用户名和密码，但运行结果截图如下：

```
<input type="password" class="mint-field-core">
```

✖

▶ POST http://127.0.0.1:4000/register 404 (Not Found) xhr.js?b50d:178

✖

▶ Uncaught (in promise) Error: Request failed with status code 404
at createError (createError.js?2d83:16)
at settle (settle.js?467f:17)
at XMLHttpRequest.handleLoad (xhr.js?b50d:61)

> |

此时是由于 WEB 服务器不存在指定的路由引起的，所以：

第一步：ctrl+c 停止WEB服务器

第二步：创建 /register 的 POST 方式的路由，代码如下：

```
server.post('/register', (req, res) => {  
  //1. 获取用户输入的用户名及密码  
  //2. 将获取到的用户名和密码写入数据表  
  res.send('OK');  
});
```

此时就要在 app.js 中获取以 POST 方式提交的用户名和密码等信息

因为 Express 框架默认情况下只能接收表单以 GET 方式提交的数据，如果要获取 POST 方式提交的数据，则需要安装 body-parser 中间件：

```
npm install --save body-parser
```

```
const bodyParser = require('body-parser');  
  
server.use(bodyParser.urlencoded({  
  extended: false  
}));
```

此时可通过 req.body.名称 来获取以 POST 方式提交的数据！代码如下：

```
server.post('/register', (req, res) => {

    //1. 获取用户输入的用户名及密码
    var username = req.body.username;
    var password = req.body.password;

    //2. 将获取到的用户名和密码写入数据表
    res.send('OK');
});
```

虽然在 /register 路由中可以获取出用户的相关信息，但是仍然不能进行数据表记录的插入工作，因为现在在 zhihu 数据库中不存在相应的数据表，所以，创建数据表：

```
CREATE TABLE zhihu_users(
    id INT UNSIGNED NOT NULL KEY AUTO_INCREMENT COMMENT '用户ID, 主键且自增',
    username VARCHAR(30) NOT NULL UNIQUE COMMENT '用户名, 唯一',
    password VARCHAR(32) NOT NULL COMMENT '用户密码, MD5'
) COMMENT '用户表';
```

执行操作时，需要将用户的密码并没有采用 md5 进行加密，解决方案有：

方案A：直接在书写 SQL 语句时采用 MySQL 提供的 md5 函数，示例代码如下：

```
INSERT users(username, password) VALUES('tom', MD5('tom'));
```

方案B：如果项目中所使用的并非 MySQL 数据库或者是数据库没有提供 md5 的函数，直接采用 md5 模块对指定的字符串进行加密

```
npm install --save md5
```

```
const md5 = require('md5');

md5('xxxx')
```

现在可以采用以上两种方式中的任何一种对密码进行 md5 操作，但是有可能发出现以下的错误信息：

```
Error: ER_DUP_ENTRY: Duplicate entry 'teduwuhua' for key 'username'
    at Query.Sequence._packetToError (E:\www\Material\WEBTN1912\14_VUEUI\Day06\demo\server\node_modules\mysql\lib\protocol\sequences\Sequence
:47:14)
    at Query.ErrorPacket (E:\www\Material\WEBTN1912\14_VUEUI\Day06\demo\server\node_modules\mysql\lib\protocol\sequences\Query.js:79:18)
    at Protocol._parsePacket (E:\www\Material\WEBTN1912\14_VUEUI\Day06\demo\server\node_modules\mysql\lib\protocol\Protocol.js:291:23)
    at Parser._parsePacket (E:\www\Material\WEBTN1912\14_VUEUI\Day06\demo\server\node_modules\mysql\lib\protocol\Parser.js:433:10)
    at Parser.write (E:\www\Material\WEBTN1912\14_VUEUI\Day06\demo\server\node_modules\mysql\lib\protocol\Parser.js:43:10)
    at Protocol.write (E:\www\Material\WEBTN1912\14_VUEUI\Day06\demo\server\node_modules\mysql\lib\protocol\Protocol.js:38:16)
    at Socket.<anonymous> (E:\www\Material\WEBTN1912\14_VUEUI\Day06\demo\server\node_modules\mysql\lib\Connection.js:88:28)
    at Socket.<anonymous> (E:\www\Material\WEBTN1912\14_VUEUI\Day06\demo\server\node_modules\mysql\lib\Connection.js:526:10)
    at Socket.emit (events.js:198:13)
```

出现该类型错误的根本原因是：数据表中的某个字段使用了唯一约束（也就是该字段中不能存在重复的记录），但是在插入或更新记录时，导致了该字段产生重复记录，所以提示该字段记录重复的错误信息。

解决方案就是：在进行插入/更新之前，查找字段的记录是否存在，如果不存在，则进行插入/更新操作
此时完整的代码截图如下：

```
Go Run Terminal Help app.js - server - Visual Studio Code
JS app.js x
JS app.js > ...
1 //加载Express模块
2 const express = require('express');
3
4 //加载CORS模块
5 const cors = require('cors');
6
7 //加载MySQL模块并且进行配置
8 const mysql = require('mysql');
9
10 //加载body-parser模块
11 const bodyParser = require('body-parser');
12
13 //加载MD5模块
14 const md5 = require('md5');
15
16 //创建MySQL连接池
17 const pool = mysql.createPool({
18     //MySQL数据库服务器的地址
19     host: '127.0.0.1',
20     //MySQL数据库服务器端口号
21     port: 3306,
22     //数据库用户的用户名
23     user: 'root',
24     //数据库用户的密码
25     password: '',
26     //数据库名称
27     database: 'zhihu',
28     //数据库编码方式
29     charset: 'utf8',
30     //最大连接数
31     connectionLimit: 15
32 });
33
34 //创建Express应用
35 const server = express();
36
37 //使用CORS模块
38 server.use(cors({
39     origin: ['http://127.0.0.1:8080', 'http://localhost:8080', 'http://196.
40     168.1.3:8080']
41 }));
42
43 //使用body-parser模块
44 server.use(bodyParser.urlencoded({
45     extended: false
46 }));
47
48 //categories的GET路由
49 server.get('/categories', (req, res) => {
50     var sql = 'select id,category_name,category_info from zhihu_category';
51     pool.query(sql, (err, results) => {
52         if (err) throw err;
53
54         res.send({
55             message: '查询成功',
56             code: 1,
57             results: results
58         });
59     });
60 });
```

```

60 });
61
62 //register的POST方式路由
63
64 server.post('/register', (req, res) => {
65
66     //1. 获取用户输入的用户名及密码
67     var username = req.body.username;
68     var password = md5(req.body.password);
69
70     //2. 以输入的用户名为条件进行查找，如果用户名不存在，则插入记录；否则提示错误信息
71     var sql = 'SELECT COUNT(id) AS count FROM zhihu_users WHERE username=?';
72     pool.query(sql, [username], (err, results) => {
73         if (err) throw err;
74         //获取指定用户名的数量，值只为0(代表用户不存在)或1(代表用户已存在)
75         var count = results[0].count;
76         if (count == 1) {
77             //错误信息
78             res.send({ message: '用户注册失败', code: 0 });
79         } else {
80             //3. 将获取到的用户名和密码等信息写入数据表
81             var sql = 'INSERT zhihu_users(username,password) VALUES(?,?)';
82             //通过MySQL连接池执行SQL语句
83             pool.query(sql, [username, password], (err, results) => {
84                 if (err) throw err;
85                 res.send({message: '用户注册成功',code:1});
86             });
87         }
88     });
89
90
91
92 });
93

```

代表用户已存在

代表用户不存在

第三步：输入 `node app.js` 启动 WEB 服务器

现在已经可以正常返回给 AJAX 请求相关的信息了，但是目前还没有接收并且处理这些返回信息，所以：(在 `src/views/Register.vue` 中)

代码如下：

```

this.axios.post('/register','username=' + this.username + '&password=' +
this.password).then(res=>{
    var code = res.data.code;
    //根据服务器的响应信息，显示不同的注册结果
    if(code == 0){
        this.$messagebox('注册提示','对不起，该用户已存在');
    } else {
        this.$messagebox.confirm('用户注册成功，是否立即登录？','注册成
功',{
            confirmButtonText:'是',
            cancelButtonText:'否'
        })
        .then(action=>{
            this.$router.push('/login');
        }).catch(err=>{
            this.$router.push('/');
        });
    }
});

```

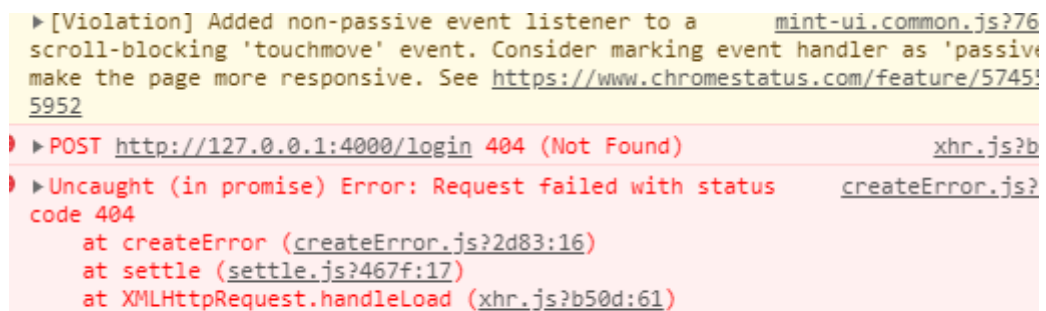
```
    }  
  });  
}
```

2. 用户登录

操作步骤如下：

1. 修改 `src/views/Login.vue`，当单击按钮时发送 `POST` 类型的 `AJAX` 请求

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

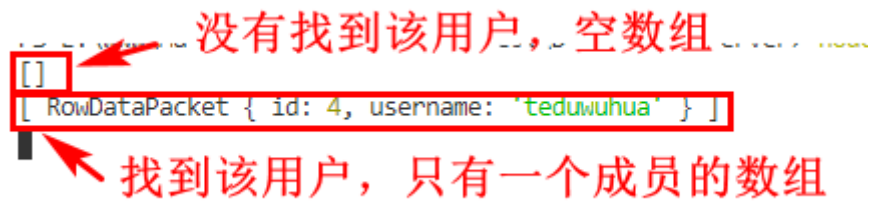


The screenshot shows the browser's developer console with the following messages:

- A yellow warning: "[Violation] Added non-passive event listener to a scroll-blocking 'touchmove' event. Consider marking event handler as 'passive' to make the page more responsive. See <https://www.chromestatus.com/feature/5745552>".
- A red error: "POST http://127.0.0.1:4000/login 404 (Not Found) xhr.js?b".
- A red error: "Uncaught (in promise) Error: Request failed with status code 404". The stack trace includes:
 - at `createError` (`createError.js?2d83:16`)
 - at `settle` (`settle.js?467f:17`)
 - at `XMLHttpRequest.handleError` (`xhr.js?b50d:61`)

2. 原因是 `WEB` 服务器不存在指定的路由，所以需要修改 `app.js` 代码如下：

```
server.post('/login', (req, res) => {  
    //获取用户名  
    var username = req.body.username;  
    //获取密码并且加密  
    var password = md5(req.body.password);  
    //以当前的用户名和密码为条件在数据表查询是否存在该用户  
    //如果存在则登录成功，否则登录失败  
    var sql = 'SELECT id,username FROM zhihu_users WHERE username=? AND  
password=?';  
    pool.query(sql, [username, password], (err, results) => {  
  
        if (err) throw err;  
        //如果用户名或密码存在错误的话,results将返回空数组  
        //如果用户名和密码都正确的话,results将返回只有一个成员的数组  
        if (results.length == 0) {  
            res.send({ message: '用户名或密码错误', code: 0 });  
        } else {  
            //下午还要改  
            res.send({ message: '用户登录成功', code: 1 });  
        }  
    });  
});  
});
```



3. 客户端接收服务器返回的信息，并且根据返回的结果以显示不同的信息，代码如下：

```
login(){
  this.axios.post('/login','username=' + this.username + '&password=' +
this.password).then(res=>{
  //获取服务器的响应代码
  var code = res.data.code;
  if(code == 0){ //登录失败
    this.$messagebox.alert('用户名或密码错误','登录失败');
  } else { //登录成功
    this.$router.push('/');
  }
  });
}
```

4. 用户登录已经可以实现了，但是当用户成功登录后，顶部信息仍然显示如下图所示结果：



此时应该显示其他信息 -- 如 退出 或 注销 ---- 引申出 Vuex

QQ登录接口

<https://wiki.open.qq.com/wiki>

微信登录接口

https://open.weixin.qq.com/cgi-bin/frame?t=home/web_tmpl&lang=zh_CN

手机短信登录

<https://www.163yun.com/product/sms>

3. Vuex

3.1 Vuex是什么?

`vuex` 是一个专为 `vue` 应用程序开发的状态管理模式。它采用集中式存储来管理应用中的所有组件的状态, 简单来说, 对 `vue` 应用中多个组件共享的状态进行集中式的管理。

状态 (`state`) 就是组件中共享的数据。

应用状态的案例:

- 登录信息
- 社交应用的中的通知
- 电商应用中的购物车

每一个 `vuex` 应用的核心就是 `store`, `store` 就是一个容器, 包含了应用中的所需要的状态。

创建 `store` 实例之所以能够在各个组件中共享的原因是:

`store` 实例是一个全局单例模式, 其提供了一种机制: 将状态从根组件注入到每一个子组件中。所以在子组件中可以通过 `this.$store` 访问

3.2 安装

```
npm install --save vuex
```

状态管理建议存储在 `src/store` 目录

`Vuex` 本质就是 `Vue` 的插件

3.3 核心概念

• `state`

`state` 定义了应用的数据, 其数据类型可以为 `string`、`number`、`boolean`、`array`、`object`

• `Getters`

`getters` 的作用是用于获取 `state` 中的数据信息, 也就代表 `getters` 依赖于 `state`

`vuex` 允许在 `store` 中定义 `getters` (可以认为是 `store` 的计算属性), `getters` 的返回值会根据它的依赖被缓存起来, 只有当它的依赖值发生变化时才会被重新计算。其语法结构是:

```
getters:{
  getNums(state){
    ...
  }
}
```

`getters` 中将 `state` 作为唯一参数进行传递, `state` 代表就是当前 `store` 的数据, 其会自动注入。

• `Mutations`

`Mutations` 是改变状态的的操作方法, 也是 `Vuex` 修改 `state` 唯一的方法, 其语法结构是:

```
mutations:{
  函数名称(state[,payload]){
    ...
  }
}
```

`state` 作为 `mutations` 的第一个参数，而且 `vuex` 中的 `state` 会自动注入

`payload` 译为提交载荷，也就是向 `store.commit` 传入的额外参数，在多数情况下，载荷应该是一个对象。

调用 `mutations` 时需要执行 `commit()` 方法，其语法结构是：

```
store.commit("函数名称", 参数)
```

• Actions

`Actions` 类似于 `Mutations`，不同于：

- `Actions` 可以包含任意异步操作，但 `Mutations` 不允许发生异步操作
- `Actions` 提交给 `Mutations`，而不是直接修改 `state`

语法结构是：

```
actions:{
  函数名称(context){
    ....
  }
}
```

`dispatch()` 方法用于分发 `Action`，其语法结构是：

```
$store.dispatch('函数名称'[,payload])
```

