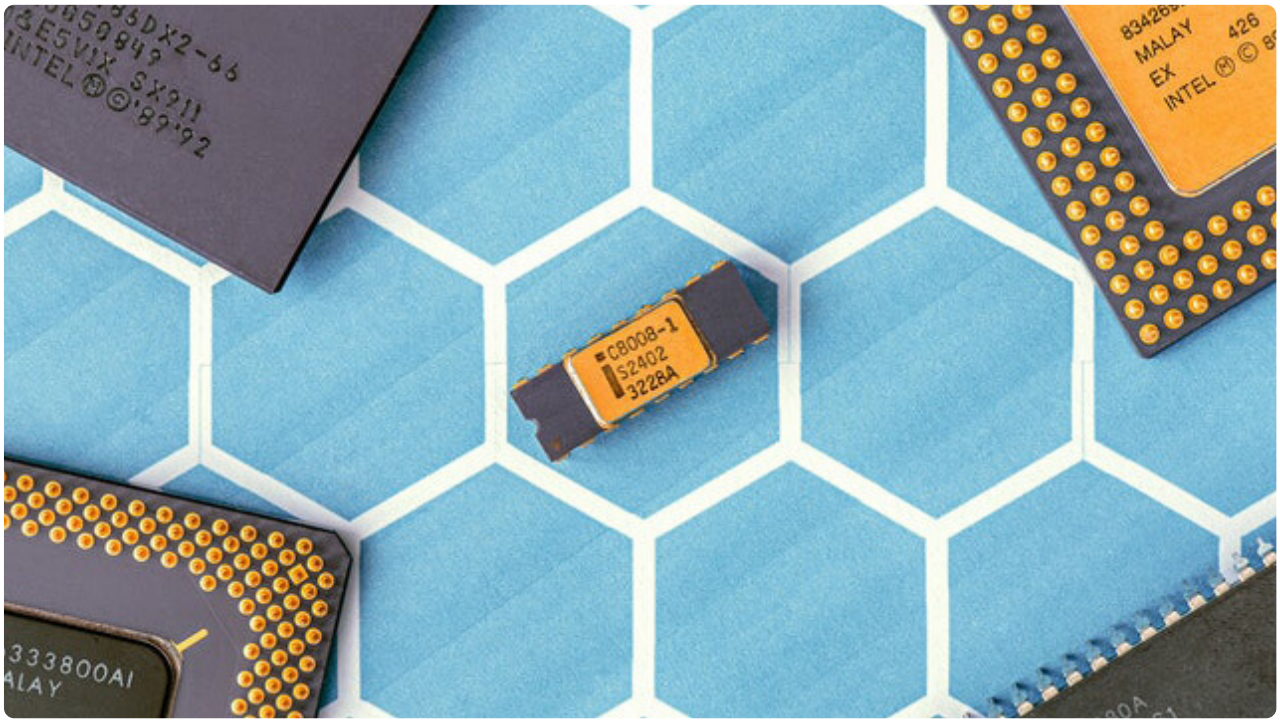
33 验证码和接口限频

更新时间：2019-10-09 09:35:03



> 上天赋予的生命，就是要为人类的繁荣和平和幸福而奉献。
>
> ——松下幸之助

一个真实的企业级业务，对安全性的考虑是必不可少的，除了我们前端常见的XSS跨站脚本攻击攻击和CSRF跨站请求伪造攻击之外，我们对日常的业务逻辑也要有安全意识，例如我们系统的短信验证码，如果被非法利用会对我们的资源造成严重隐患，下面就对我们这些有安全隐患的地方加以处理和巩固。

本章节完整源代码地址，大家可以事先浏览一下：

Github-app.js
Github-config.js
Github-index.vue

设置后台支持跨域

在我们之前的开发中，由于我们是前后端分离，使用Vue cli3生成的项目会自动为你生成一个devserver来供调试，所以大多数场景下，如果你需要请求本地的后台服务，由于端口不一致，会出现跨域的问题，关于跨域问题简单普及一下。

跨域定义：

同源策略/SOP（Same origin policy）是一种约定，由Netscape公司1995年引入浏览器，它是浏览器最核心也最基本的安全功能，如果缺少了同源策略，浏览器很容易受到XSS、CSFR等攻击。所谓同源是指"协议+域名+端口"三者相同，即便两个不同的域名指向同一个ip地址，也非同源。

在我们常见的ajax请求场景中：

```
http://www.123.com/index.html 页面调用 http://www.123.com/server.php （非跨域）
http://www.123.com/index.html 页面调用 http://api.123.com/server.php （跨域 域名不同）
http://www.123.com/index.html 页面调用 https://www.123.com/server.php （跨域 协议不同）
http://www.123.com/index.html 页面调用 http://www.123.com:8808/server.php （跨域 端口不同）
```

请注意：localhost和127.0.0.1虽然都指向本机，但也属于跨域。

解决跨域的方法：

1. JSONP：这里不再讲解，大家可以去网上搜索，大把资料。
2. 代理：代理的意思就是请求由代理发出，前端请求同域下的接口，这个接口负责代理转发，将数据获取到并返回。相当于绕过了浏览器端，请求由后端发出，自然就不存在跨域问题。
3. 修改response header，这个是我们本次项目采用的方案，代码如下：
   在后端项目的app.js中加入下面逻辑。

```
// 跨域配置 本地调试使用
app.use(function(req, res, next) {

  res.header("Access-Control-Allow-Origin", 'http://localhost:8080');//允许localhost来源访问
  res.header("Access-Control-Allow-Headers", "Origin, X-Requested-With, Content-Type, Accept, wec-access-token, Set-Cookie");//允许设置返回的修改设置的header值
  res.header('Access-Control-Allow-Method,'POST,GET');//允许访问的方式
  res.header("Access-Control-Allow-Credentials", "true");//允许在跨域请求中带上Cookie

  next();
});
```

如果你只需要跨域请求，可以单独设置`Access-Control-Allow-Origin`这个值即可，其他的配置是我们项目中需要的，另外如果想要在跨域请求中带上cookie(默认情况是不会带的)，需要设置`Access-Control-Allow-Credentials`为`true`，同时在前端的axios中设置`withCredentials: true`。

登录态接口限制

当接口支持跨域，就表示你的接口在浏览器端可以被其他业务调用，或者是别人采用POSTMAN这种API请求工具也可以直接调用你的某些接口，所以我们就需要对接口做一定的调用限制。
对一些接口做登录态校验：如果你的接口需要登录态，就表明当没有登录态的请求访问你的接口时，就需要返回失败，我们可以利用Express的拦截器去实现：
在后端项目的app.js中添加如下代码。

```
//app.use是Express拦截器的方法
app.use(function(req, res, next) {

 // 拿取token 数据 按照自己传递方式写
 var token = req.headers['wec-access-token']||'xx';
 // 检查token是否有效（过期和非法）
 var user = tokenUtil.checkToken({token});
 if (user) {
    //将当前用户的信息挂在req对象上，方便后面的路由方法使用
    req.user = user;

    // 续期
    tokenUtil.setToken({user,res});

    next(); //继续下一步路由
 } else {
    //需要登录态域名白名单
    if (config.tokenApi.join(',').indexOf(req.path) < 0) {
       next();
       return;
    }
    res.json({ code: 1000, message: '无效的token.' });
 }
});
```

没错，上面的代码在之前章节《使用JSON Web Token实现用户会话token存储和验证》中出现过，这里我们主要看一下我们的登录态校验白名单，即配置一些接口API路径，只要名单里的接口，就需要进行登录态校验，如果没有登录态，就要返回错误，阻止后面的接口逻辑执行。

我们在后端项目的根目录创建config.js，在这里添加白名单：

```
module.exports = {
 //需要登录态的接口
 tokenApi:[
 '/post/uploadimg',
 '/post/uploadimgaliyun',
 '/post/savepost',
 '/likecomment/addlike',
 '/likecomment/addcomment',
 '/user/update',
 '/likecomment/removelike',
 '/message/addmsg',
 '/message/getchatlist',
 '/message/getchathistory'
 ],
 uploadPath:'//app.nihaoshijie.com.cn/upload/'
};
```

这样配置之后，我们就在这里维护接口的登录态需求即可。

短信验证码接口安全加固

对于短信验证码接口，由于本身逻辑是不需要登录态的，因为本身用户就是登录用的接口，这个时机用户是不会有登录态的，所以我们采取下面的加固措施。

校验Referer：在`phonecode`这个接口中，增加校验`Referer`逻辑。

在后端项目的routes文件夹下的users.js路由文件中，找到`phonecode`路由，添加如下代码：

```
router.post('/phonecode', (req, res, next)=> {
  ...

  var refList = ['https://app.nihaoshijie.com.cn/index.html','http://localhost:8080/index.html']
  // console.log(req.headers.referer)
  if (refList.indexOf(req.headers.referer) < 0) {
    return
  }
  ...
})
```

虽然Referer在某些场景中也可以被模拟，但是增加了一层难度，使接口不那么容易被攻破。

限制调用频率：在 phonecode 这个接口中，增加对单一IP客户的限制调用频率逻辑。

首先我们需要安装express-rate-limit模块，是一个基于Express路由的接口限频率的组件模块，在后端项目的根目录执行：

```
npm install express-rate-limit --save
```

然后，在后端项目的routes文件夹下的users.js路由文件中，找到 phonecode 路由，添加如下代码：

```
var rateLimit = require("express-rate-limit");
/*
* 验证码请求限制调用频率同一个ip 1分钟调用最多1次
*/

var phonecodeLimiter = rateLimit({
  windowMs: 1 * 60 * 1000, // 分钟调用1次
  max: 1, // 1分钟调用1次
  handler:function(req, res, next){
    res.json({
      code: 1,
      msg: '请稍后请求'
    });
  }
});
```

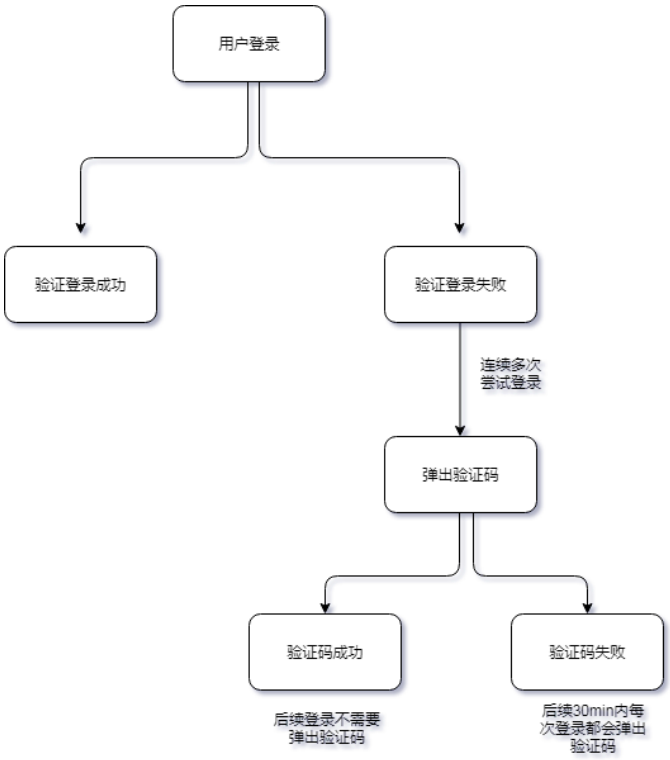
上面代码流程是针对 phonecode 路由，添加一个针对同一个客户端***1分钟调用1次***的频率限制,这是一个中间件，我们需要在 phonecode 路由配置到第二个参数，代码如下：

```
router.post('/phonecode',phonecodeLimiter, (req, res, next)=> {
  ...
})
```

当然，对于接口的限制还有其他的一些方法，在这里就不在赘述了，大家可以自由发挥。

登录接口添加验证码

对于验证码，我相信大家在熟悉不过了，验证码作为登录操作的一个有效防御武器，可以在一定程度上保护用户的账户安全和服务的流量攻击。接下来，就将验证码集成到我们的项目中。验证码逻辑流程图如下：



安装验证码模块：

```
npm install svg-captcha --save
```

svg-captcha是一个第三方验证码模块，提供基于SVG图片格式的验证码(相对于一般图片SVG格式更不容易被机器人识别)。

在后端项目的routes文件夹下的users.js路由里增加逻辑，引入验证码校验。

下面这段代码主要是创建了一个get方法的路由，路径是/captcha，当浏览器请求 http://xx.xx.xx/captcha 就会进入这个方法，这个方法返回一张验证码的图片，前端将值赋给 <img> 的 src 即可：

```
/*
* 获取验证码图片
*/
router.get('/captcha', (req, res)=> {
  var captcha = svgCaptcha.create({
    ignoreChars:'abcdefghijklmnopqrstuvwxyzABCDEFGHIJKLMNOPQRSTUVWXYZ',// 排除字母，只用数字
    noise: 2 // 干扰线条的数量
  });
  // console.log(captcha)
  res.cookie('captcha', captcha.text, {
      maxAge: 60*1000*30,// 设置到cookie里 时效1分钟
      httpOnly:true
  });

  //返回验证码图片
  res.type('svg');
  res.status(200).send(captcha.data);
});
```

上面代码展示了利用svg-captcha生成一张验证码图片，效果如下：

方法：`svgCaptcha.create(options)`参数配置如下:

```
size: 4 // 验证码长度
ignoreChars: '0o1i' // 验证码字符中排除 0o1i
noise: 1 // 干扰线条的数量
color: true // 验证码的字符是否有颜色，默认没有，如果设定了背景，则默认有
background: '#cc9966' // 验证码图片背景颜色
noise: 2 // 干扰线条的数量
```

获取验证码之后，需要将值设置到cookie里面，`httpOnly:true`表示cookie里的值不能被浏览器端修改，更加安全。

然后，我们将在前端项目的src目录下的components文件夹下的login文件夹的`index.vue`中将验证码嵌入到前端，代码如下：

```
<div v-if="needCaptcha" class="weui-cell weui-cell_vcode captcha-code">
    <div class="weui-cell__hd"><label class="weui-label">图形验证码</label></div>
    <div class="weui-cell__bd">
        <input v-model="captcha" class="weui-input" type="number" placeholder="请输入验证码"/>
    </div>
    <div class="weui-cell__ft">
        <img class="weui-vcode-img" :src="captchaUrl" @click="reloadCaptcha"/>
    </div>
</div>
```

上面代码中：

`needCaptcha`用来控制什么时候展示验证码。

展示效果如下：

完成了前端的逻辑后，在后端项目的routes文件夹下的users.js路由里增加逻辑，引入对登录接口添加限频逻辑：

```
/*
* 登录请求限制调用频率同一个ip 1分钟调用最多10次
*/
var signupLimiter = rateLimit({
  windowMs: 1 * 60 * 1000, // 分钟调用10次
  max: 10, // 1分钟调用10次
  skip: function (req, res) {
    return req.cookies.captcha ? true : false;
  },
  handler:function(req, res, next){
    res.json({
      code: 0,
      data: {
        code: 'needCaptcha'
      }
    });
  }
});
```

上面代码的逻辑是：

`req.cookies.captcha`：用来判断此次登录是否需要校验验证码，在`/captcha`路由中会去设置这个值。
`skip`：表示如果此次登录是需要弹验证码的，先绕过限频逻辑，去校验验证码。
`handler`：接收一个方法，表示命中限频逻辑时的处理方法，这里给前端返回一个标志位`needCaptcha`用来弹验证码。

接下来，在后端项目的routes文件夹下的users.js的`/signUp`路由里增加逻辑，引入校验验证码：

```
router.post('/signup', signupLimiter, async (req, res, next)=> {

  // 如果cookie里有验证码，证明此次登录请求是需要验证码
  if (req.cookies.captcha) {
    // 如果没有输入验证码，返回前端需要输入验证码
    if (!req.body.captcha) {
      res.json({
        code: 0,
        data: {
          code: 'needCaptcha'
        }
      });
      return
    }
    //验证验证码是否正确
    if (req.body.captcha.toLocaleLowerCase() !== req.cookies.captcha.toLocaleLowerCase()) {
      res.json({
        code:1,
        msg:'验证码错误'
      });
      return
    } else {

      //验证码校验正确，清除cookie，下次就不需要输入验证码登录
      res.clearCookie('captcha');
    }
  }

  ...
})
```

上面代码的流程基本和我们之前的流程图一致，需要注意的是验证码一般不区分大小写，所以校验验证码需要转换成小写去判断。另外在验证码输入正确时要清除cookie的标志位，下次就不弹了。

小节

本章节主要讲解了对项目中整个用户登录模块的安全巩固措施。

相关技术点：

1. 什么是跨域，如何在我们的项目中去解决跨域问题，以及相关的配置解释。
2. 在代码中，使用 express-rate-limit 对一些敏感接口进行限频，防止非法利用，另外对短信验证接口添加referer校验机制。
3. 验证码的整个实现逻辑和技术原理，使用 svg-captcha 来引入验证码，需要注意的是我们采用cookie的方式来记录是否需要弹出验证码。

本章节完整源代码地址：

Github-app.js

Github-config.js

Github-index.vue

}

32 模拟真实APP完成消息推送

34 小白的阿里云服务器申请之旅