# 上一节我们实现了csrf的防护代码，但是一个网站的很多功能都响应防护不仅仅是转账，如收藏，关注某个用户等等，总不能一个第三方的网站能够随随便便就关注了你网站的用户。

# 在实际开发中，凡是使用post提交的接口都需要添加csrf防护。只要关系到用户数据的接口都需要csrf防护。在我们的项目中，除了/transfer的post路由，/的post路由也就是登录路由有需要csrf防护，可以在get路由里面颁发csrf\_token,在post路由里面验证csrf\_token

# 还是webA\_anti\_csrf项目，注意webB只是攻击测试用的

|  |
| --- |
|  |

# 打开01-anti-csrf-app.js，我们来给登录路由添加csrf防护

## 1.找到/路由的get方法代码处，我们在这里颁发csrf\_token

|  |
| --- |
|  |

## 2.然后这个登录页面，也就是views/login.html需要改造一下，使用ajax提交，因为这样子可以设置header

|  |
| --- |
| <!DOCTYPE html>  <html>  <head>      <meta charset='UTF-8' />      <title>用户登录</title>  </head>  <body>      <h3>用户登录</h3>      <hr>      <form>          <p>账号：<input type="text" name="username" id="username"></p>          <p>密码：<input type="password" name="pwd" id="pwd"></p>          <p>              <input type="button" value="登录" id="btnLogin">          </p>      </form>      <script src="/js/jquery3.1.0.min.js"></script>      <script>          $('#btnLogin').click((e) => {              let username = $('#username').val()              let pwd = $('#pwd').val()              let params = { username, pwd }              $.ajax({                  url: '/',                  type: 'post',                  contentType: 'application/json',//这个不可少                  data: JSON.stringify(params),                  headers: { 'X-CSRFToken': getCookie('csrf\_token') },                  success: function (resp) {                      console.log(resp);                      let ret = JSON.parse(resp)                      if (ret.code === 200) {                          alert('登录成功，将跳转到转账页面')                          $(location.href = '/transfer')                      } else {                          alert(ret.errMsg)                      }                  }              })          })          //获取cookie函数：getCookie          function getCookie(name) {              let r = document.cookie.match("\\b" + name + "=([^;]\*)\\b")              return r ? r[1] : undefined          }      </script>  </body>  </html> |

### 注意：这里非常关键的一步是提交header设置，还有需要把Content-type设置为json格式

## 3.修改01-anti-csrf-app.js根路由的post处理代码，添加token验证功能，并且把响应数据改为json

|  |
| --- |
|  |

## 4.测试：

### 1运行程序，先把login页面的ajax请求代码中设置header里面的csrf\_token代码注释，

|  |
| --- |
|  |

### 效果如下

|  |
| --- |
|  |

### 2.打开login页面的ajax请求代码中设置header里面的csrf\_token代码

|  |
| --- |
|  |

### 效果：

|  |
| --- |
|  |
| **登录成功后进入转账页面，填写账号和金额，点击转账** |
|  |

### 在弹出的消息框中点击确定

|  |
| --- |
|  |

### 说明我们登录功能的token验证也是ok的

# 功能是实现了，但是代码有点重复，我们需要抽取可复用代码，我们可以使用路由中间件，抽取路由组件来实现

# 我们把webA\_anti\_csrf复制一份，改名webA\_anti\_csrf-middleware，然后把里面的01-anti-csrf-app.js复制一份改名：02-anti-csrf-app-middleware.js，然后我们来修改这个文件

## 1.利用express来创建路由对象，并且在app在全局注册这个利用对象

|  |
| --- |
|  |

## 2.把所有路由处理函数前面的app改为router

|  |
| --- |
|  |

## 3.先测试一下，报错了

|  |
| --- |
|  |

## 4.我们调整一下注册路由组件代码的位置，把它放的监听代码前面，工作正常

|  |
| --- |
|  |
|  |
|  |

### 说明注册路由代码的位置有有讲究的，位置不对代码就会报错

## 5.编写一个中间件函数叫做csrfProtect()，代码如下

|  |
| --- |
|  |

## 6.把csrf\_token验证的所有代码注释掉，

|  |
| --- |
|  |
|  |

## 7.然后在注册路由组件的代码中把我们的中间件作为第一个参数，router作为第二个参数

|  |
| --- |
|  |

## 8.测试一下，此时会有一个现象，页面不停的转圈圈，后台没有报错

|  |
| --- |
|  |
|  |

## 9.检查一下，原来我们的中间件没有给get方法放行，我们修改一下代码

|  |
| --- |
|  |

## 10.再测试一下，效果如图

|  |
| --- |
|  |
|  |
|  |
|  |
|  |

## 11.然后我们启动webB，看看防护效果

|  |
| --- |
|  |
|  |

## 是由防护效果的，

## 12.我们把login.html的ajax请求的设置header的代码注释一下，测试效果如下

|  |
| --- |
|  |

## 13.我们把logn.html的代码还原，它又正常了

|  |
| --- |
|  |

## 说明登录路由的csrf验证也是有些的

## 14.我们把上面的代码再优化一下，把在req的get方法里面颁发token也放到中间件里面

|  |
| --- |
|  |

## 15.然后把其他地方颁发token的代码删除，测试一下

|  |
| --- |
|  |
|  |
|  |

## 16. 我们把login.html的ajax请求的设置header的代码注释一下，测试效果如下

|  |
| --- |
|  |

## 17. 我们把logn.html的代码还原，它又正常了

|  |
| --- |
|  |
|  |
|  |

# 本节学习完成，02-anti-csrf-app-middleware.js的完整代码如下，注意如果项目很复杂我们需要把路由对象抽取到一个独立的文件

|  |
| --- |
| let express = require('express')  let cookieParser = require('cookie-parser')  let cookieSession = require('cookie-session')  let bodyParser = require('body-parser')  let path = require('path')  let port = 3500  //创建路由对象  let router = express.Router()  let app = express()  //全局注册bodyParser中间件  app.use(bodyParser.urlencoded({extended:false}))  app.use(bodyParser.json())  //全局注册cookieParser中间件,注意：有的中间件需要小括号，有的中间件不需要  app.use(cookieParser())  //全局注册cookieSession中间件  app.use(cookieSession({      name:'my\_session',      keys:['adefg%%78$$','o9l1##dd09=='] ,//这个其实就是加密用的盐      maxAge:2\*24\*1000\*60\*60 //过期时间为2天  }))  app.use(express.static(\_\_dirname+'/public'))//静态服务器，根据需要配置  //处理中文乱码  app.use(function (req, res, next) {      res.setHeader('Content-Type', 'text/html;charset=utf-8');      next();  });  //配置模板引擎,4步，一个engine和3个set  app.engine('html',require('express-art-template'))  app.set('view options',{      debug:process.env.NODE\_ENV!=='production'  })  app.set('views',path.join(\_\_dirname,'views'))  app.set('view engine','html')  //编写一个能够随机生成token的函数  function getRandomString(n){      let str = ''      while(str.length<n){          str += Math.random().toString(36).substring(2)      }      return str.substring(str.length-n) //注意这里其实就是从0开始截取到最后  }  router.all('/',(req,res)=>{       if(req.method=="GET"){          // console.log(getRandomString(48));          res.render('login')       }else if(req.method=="POST"){           //注意：csrf防护我们以及在中间件里面做了，这里就不需要写防护代码了           //获取用户名，密码           let {username,pwd} = req.body           console.log(username,pwd);           //模拟数据库数据验证           if(username === 'admin' && pwd === '12345'){                 //验证成功,需要设置状态保持，用来在其他路由验证用户是否登录                 req.session['username'] = username                 //转到transfer路由              //    res.redirect('/transfer')              res.json({code:200})           } else{              // res.send("用户名或者密码错误！！！")              res.json({code:20000,errMsg:'用户名或者密码错误！！！'})           }         }  })  //转账路由,需要设置csrf\_token  router.all('/transfer',(req,res)=>{       //验证用户是否登录,必须一上来就做，无论用户是get还是post请求都需要登录验证      let username = req.session['username']      if(!username){ //用户没有登录          //重定向到首页执行登录操作          res.redirect('/')      }      //功能进入这里，说明用户已经登录了      if(req.method=="GET"){          //产生csrf\_token并且设置到cookie          // res.cookie('csrf\_token',getRandomString(48))          res.render('transfer') //渲染转帐页面      }else if(req.method=="POST"){          // //模拟转账,          //注意：csrf防护我们以及在中间件里面做了，这里就不需要写防护代码了          let {account,money} = req.body            //模拟转换成功，在后输出信息          console.log(`成功给${account}转账${money}元`);          res.json({account,money})      }    })  router.get('/done',(req,res)=>{      res.render('done')  })  //csrf中间件函数  function csrfProtect(req,res,next){      if(req.method==='GET'){//在get方法颁发token          //颁发csrf\_token,使用res对象          res.cookie('csrf\_token',getRandomString(48))          next()      }      else if(req.method==='POST'){          let csrf\_token = req.cookies['csrf\_token'] //获取cookie中的token          // let x\_csrf\_token = req.get('X-CSRFToken') //获取请求头中的token          //获取请求头中的token写法2,cookie-parser会把heder里面的键全部转为小写          let x\_csrf\_token = req.headers['x-csrftoken']          // console.log(csrf\_token,x\_csrf\_token);          //然后进行比较验证他们是否是同一个token           if(csrf\_token!==x\_csrf\_token){              //如果不相等，说明是csrf伪造请求，必须终止代码继续往下执行              res.json('csrf\_token验证失败,转账无法进行,去检查你的token')              return           } else{              next()           }      }  }  //注册路由对象，注意需要先完成router所有功能的编写，然后再注册router，否则会报错  app.use(csrfProtect,router)  app.listen(port,()=>{       console.log('server is ready :http://localhost:'+port+'/');  }) |

# 这就是通用csrf代码。