

# koa鉴权

---

## 课堂目标

- 掌握三种常见鉴权方式：Session/Cookie、Token+jwt、OAuth

## 前言

前后端未分离以前，页面都是通过后台来渲染的，能不能访问到页面直接由后台逻辑判断。前后端分离以后，页面的元素由页面本身来控制，所以页面间的路由是由前端来控制了。当然，仅有前端做权限控制是远远不够的，后台还需要对每个接口做验证。为什么前端做权限控制是不够的？因为前端的路由控制仅仅是视觉上的控制，前端可以隐藏某个页面或者某个按钮，但是发送请求的方式还是有很多，完全可以跳过操作页面来发送某个请求。所以就算前端的权限控制做的非常严密，后台依旧需要验证每个接口。前端的权限控制主要有三种：路由控制（路由的跳转）、视图控制（按钮级别）和请求控制（请求拦截器）。这几种方式之后再详谈，前端做完权限控制，后台还是需要验证每一个接口，这就是鉴权。现在前后端配合鉴权的方式主要有以下几种：

- session-cookie
- Token验证(JWT)
- OAuth(开放授权)

# Session/Cookie

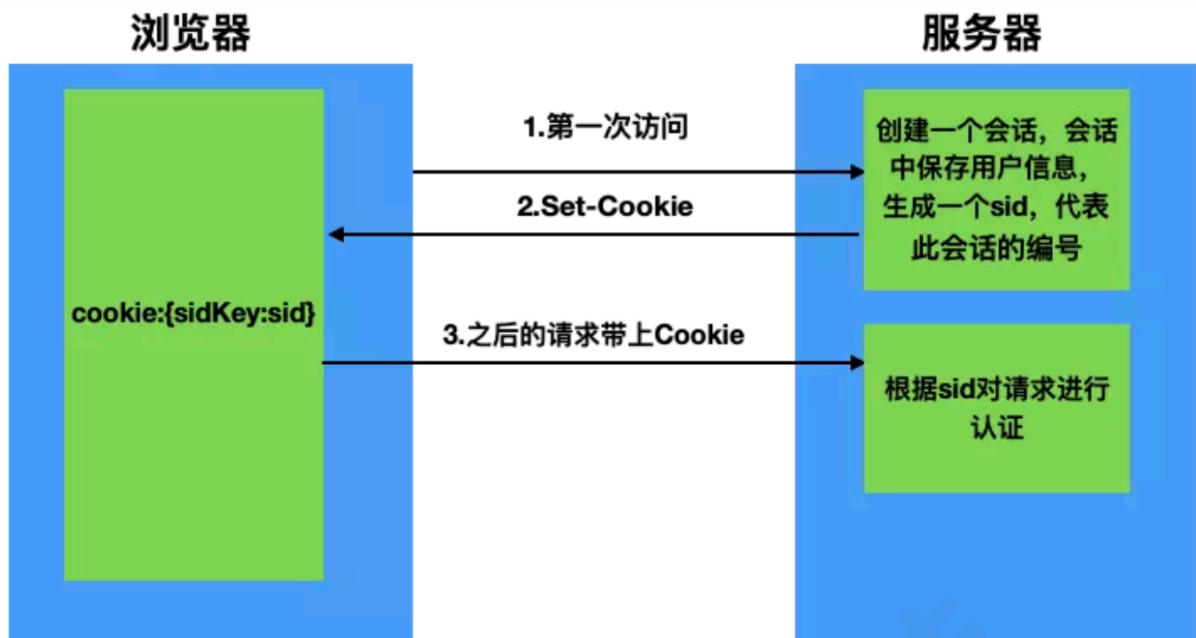
## cookie

Http协议是一个无状态的协议，服务器不会知道到底是哪一台浏览器访问了它，因此需要一个**标识**用来让服务器区分不同的浏览器。cookie就是这个管理服务器与客户端之间状态的标识。 cookie的原理是，浏览器第一次向服务器发送请求时，服务器在response头部设置Set-Cookie字段，浏览器收到响应就会设置cookie并存储，在下一次该浏览器向服务器发送请求时，就会在request头部自动带上Cookie字段，服务器端收到该cookie用以区分不同的浏览器。当然，这个cookie与某个用户的对应关系应该在第一次访问时就存在服务器端，这时就需要session了。

## session

session是会话的意思，浏览器第一次访问服务端，服务端就会创建一次会话，在会话中保存标识该浏览器的信息。它与cookie的区别就是session是缓存在服务端的，cookie 则是缓存在客户端，他们都由服务端生成，为了弥补Http协议无状态的缺陷。

## session-cookie认证



原理:

1. 服务器在接受客户端首次访问时在服务器端创建 session, 然后保存 session(我们可以将 session 保存在内存中, 也可以保存在 redis 中, 推荐使用后者), 然后给这个 session 生成一个唯一的标识字符串, 然后在响应头中种下这个唯一标识字符串。
2. 签名。这一步通过密钥对 sid 进行签名处理, 避免客户端修改 sid。(非必需步骤)
3. 浏览器中收到请求响应的时候会解析响应头, 然后将 sid 保存在本地 cookie 中, 浏览器在下次 http 请求的请求头中会带上该域名下的 cookie 信息。
4. 服务器在接受客户端请求时会去解析请求头 cookie 中的 sid, 然后根据这个 sid 去找服务器端保存的该客户端的 session, 然后判断该请求是否合法。

- koa 中 session 的使用: `npm i koa-session -S`

```
1 // 加密sessionid
```

```

2 // keys作用：用来对cookie进行签名
3 app.keys = ['session secret'];
4
5 // session配置
6 const SESS_CONFIG = {
7   key: 'xiaomage:sess', //设置cookie的key名字
8   maxAge: 86400000, //有效期：默认一天
9   httpOnly: true, // 仅服务端修改
10  signed: true, //签名cookie
11 }
12 app.use(session(SESS_CONFIG, app));
13 app.use(async ctx=> {
14   // ignore favicon
15   if (ctx.path === '/favicon.ico') return;
16
17   let n = ctx.session.count || 0;
18   ctx.session.count = ++n;
19   ctx.body = '第' + n + '次访问';
20 });

```

- 使用redis存储session

- 安装: `npm i koa-redis -S`
- 配置使用

```

1 const redisStore = require('koa-redis')
2 const redis = require('redis');
3 const client =
  redis.createClient(6379, 'localhost');
4

```

```

5 // 加密sessionid
6 // keys作用：用来对cookie进行签名
7 app.keys = ['session secret'];
8
9 // session配置
10 const SESS_CONFIG = {
11   key: 'xiaomage:sess', //设置cookie的key名字
12   maxAge: 86400000, //有效期：默认一天
13   httpOnly: true, // 仅服务端修改
14   signed: true, //签名cookie
15   store: redisStore({...client})
16 }
17 app.use(session(SESS_CONFIG, app));
18
19 app.use(async ctx=> {
20   //...
21   client.keys('*', (err, keys)=>{
22     console.log(keys);
23     keys.forEach(key => {
24       client.get(key, (err, val)=>{
25         console.log(val);
26       })
27     });
28   })
29 });

```

- 案例: 通过session实现用户鉴权

./public/login.html

```
1  <!DOCTYPE html>
2  <html lang="en">
3
4      <head>
5          <meta charset="UTF-8">
6          <meta name="viewport"
7  content="width=device-width, initial-scale=1.0">
8          <meta http-equiv="X-UA-Compatible"
9  content="ie=edge">
10         <title>Document</title>
11     </head>
12
13     <body>
14         <div id="app">
15             <input type="text" v-model='user'>
16             <input type="password" v-
17  model='pwd'>
18             <button @click='handleLogin'>登
19 录</button>
20             <button @click='handleLogout'>退出
21             </button>
22             <button @click='getUser'>获取用
23 户</button>
24         </div>
25         <script
26  src="https://cdn.bootcss.com/vue/2.6.10/vue.js">
27     </script>
28         <script
29  src="https://cdn.bootcss.com/axios/0.19.0/axios.
30  js"></script>
31         <script>
```

```
22         axios.defaults.withCredentials =
true; //允许客户端携带cookie
23         // 添加响应拦截器
24
25         axios.interceptors.response.use((response) =>{
26             // 对响应数据做点什么
27
28             console.log(JSON.stringify(response.data));
29             return response;
30         }, (error) => {
31             // 对响应错误做点什么
32             return Promise.reject(error);
33         });
34     new Vue({
35         el: '#app',
36         data() {
37             return {
38                 user: 'xiaomage',
39                 pwd: '123'
40             }
41         },
42         methods: {
43             async handleLogin() {
44                 await
45                 axios.post('/users/login',{
46                     user: this.user,
47                     pwd: this.pwd
48                 })
49             },
50             async handleLogout() {
```

```

48             await
    axios.post('/users/logout')
49         },
50         async getUser() {
51             await
    axios.get('/users/getUser')
52         }
53     },
54 })
55     </script>
56     </body>
57
58 </html>

```

- 登录/注销接口, ./routes/users.js

```

1  router.post('/login', async (ctx, next) => {
2      const { body } = ctx.request;
3      console.log('body', body);
4      ctx.session.userinfo = body.user;
5      ctx.body = {
6          ok: 1,
7          message: '登录成功'
8      }
9  })
10 router.post('/logout', async (ctx, next) => {
11     delete ctx.session.userinfo;
12     ctx.body = {
13         ok: 1,
14         message: '退出系统'
15     }

```



```

16 })
17 router.get('/getUser', async (ctx, next) => {
18   ctx.body = {
19     ok: 1,
20     message: '获取数据成功',
21     userinfo: ctx.session.userinfo
22   }
23 })

```

- 当访问/getUser路由的时候需要守卫中间件 ./middleware/auth.js

```

1 module.exports = async (ctx, next) => {
2   if(!ctx.session.userinfo){
3     ctx.body = {
4       ok: 0,
5       message: '用户未登录'
6     }
7   }else{
8     await next();
9   }
10 }

```

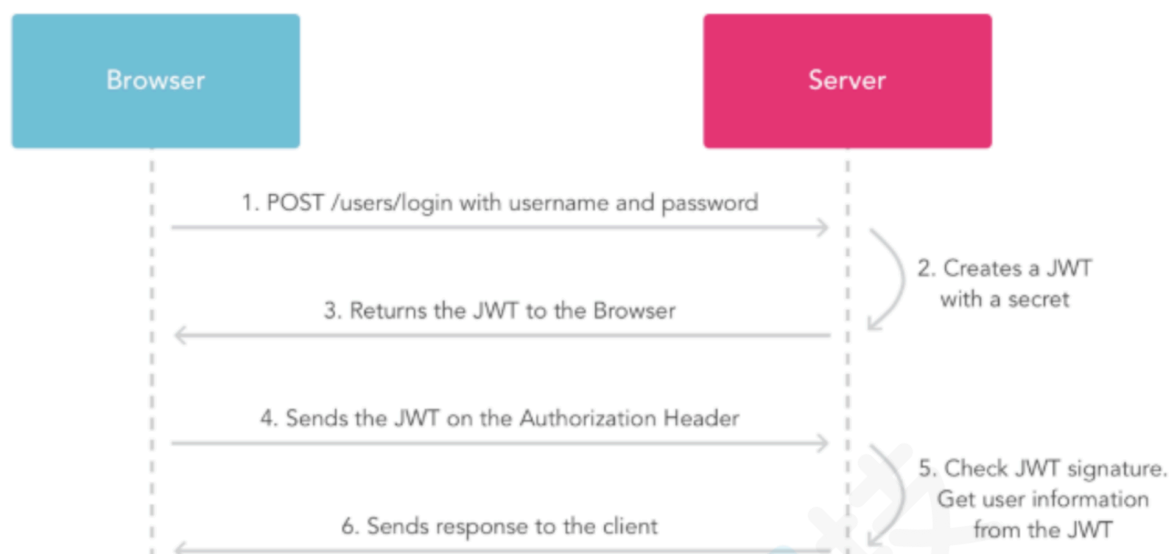
- 应用守卫 ./routes/users.js

```

1 router.get('/getUser', require('../middleware/auth'), async (ctx, next) => {...})

```

## Token+JWT认证



## 所需库

- koa-jwt: `npm i koa-jwt -S` jwt中间件
- jsonwebtoken: `npm i jsonwebtoken -S` 用于生成token下发给浏览器,在koa2以后的版本不再提供jsonwebtoken的方法,所以需要另外安装

./public/login-token.html

```
1 <!DOCTYPE html>
2 <html lang="en">
3
4 <head>
5     <meta charset="UTF-8">
6     <meta name="viewport" content="width=device-
width, initial-scale=1.0">
7     <meta http-equiv="X-UA-Compatible"
content="ie=edge">
8     <title>Document</title>
9 </head>
```

```
10
11 <body>
12     <div id="app">
13         <input type="text" v-model='user'>
14         <input type="password" v-model='pwd'>
15         <button @click='handleLogin'>登
16         录</button>
17         <button @click='handleLogout'>退出
18     </button>
19     <button @click='getUser'>获取用
20     户</button>
21     </div>
22     <script
23     src="https://cdn.bootcss.com/vue/2.6.10/vue.js">
24     </script>
25     <script
26     src="https://cdn.bootcss.com/axios/0.19.0/axios.
27     js"></script>
28     <script>
29         axios.defaults.withCredentials = true;
30         //允许客户端携带cookie
31         axios.interceptors.request.use((config)
32         => {
33             // 对请求数据做点什么
34             const token =
35             localStorage.getItem('token');
36             if (token) {
37                 // 判断是否存在token,如果存在的话,
38                 则每个http header都加上token
39                 // Bearer是jwt的认证头部信息
```

```
29     config.headers.common['authorization'] =  
    'Bearer ' + token;  
30     }  
31  
32     return config;  
33   }, (error) => {  
34     // 对请求错误做点什么  
35     return Promise.reject(error);  
36   });  
37   // 添加响应拦截器  
38  
39   axios.interceptors.response.use((response) => {  
40     // 对响应数据做点什么  
41     console.log(JSON.stringify(response.data));  
42     return response;  
43   }, (error) => {  
44     // 对响应错误做点什么  
45     return Promise.reject(error);  
46   });  
47   new Vue({  
48     el: '#app',  
49     data() {  
50       return {  
51         user: 'xiaomage',  
52         pwd: '123'  
53       }  
54     },  
55     methods: {  
      async handleLogin() {
```

```

56         const res = await
axios.post('/users/login-token', {
57             user: this.user,
58             pwd: this.pwd
59         })
60
        localStorage.setItem('token', res.data.token);
61    },
62    async handleLogout() {
63        await
64        axios.post('/users/logout')
65        },
66        async getUser() {
67            await
68            axios.get('/users/getUser-token')
69        }
70    },
71    })
72    </script>
73    </body>
74    </html>

```

接口配置:users.js

```

1  const jwt = require('jsonwebtoken');
2  const jwtAuth = require('koa-jwt')
3  const secret = 'this is a scret';
4
5  router.post('/login-token', async (ctx, next) =>
{

```

```

6   const { body } = ctx.request;
7   const userinfo = body.user;
8   ctx.body = {
9     ok: 1,
10    message: '登录成功',
11    user: userinfo,
12    // 使用jwt模块签名一个令牌,生成 token 返回给客
    户端
13    token: jwt.sign(
14      {
15        data: userinfo, //由于签名不是加密,令牌不要
        存放敏感数据
16        exp: Math.floor(Date.now() / 1000) + 60
        * 60 //过期时间一分钟
17      }, secret
18    )
19  }
20 })
21 router.get('/getUser-token', jwtAuth({ //对传入令
    牌进行校验
22   secret
23 })), async (ctx, next) => {
24   ctx.body = {
25     message: '获取数据成功',
26     userinfo: ctx.state.user.data
27   }
28 })

```

## OAuth(开放授权)

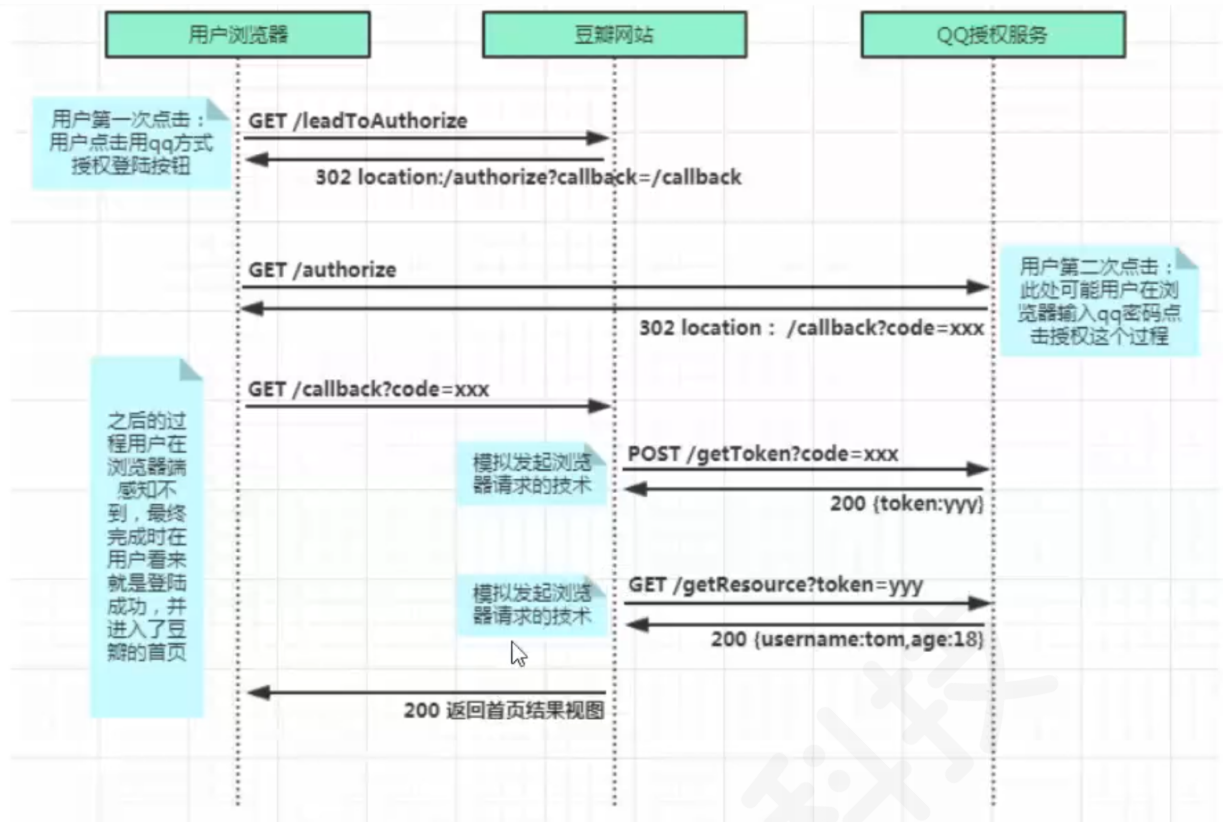
有的应用会提供第三方应用登录，比如掘金 web 客户端提供了微信、QQ账号登录，我们可以不用注册掘金账号，而可以用已有的微信账号登录掘金。看看用微信登录掘金的过程：

step1: 打开掘金，未登录状态，点击登录，掘金给我们弹出一个登录框，上面有微信、QQ登录选项，我们选择微信登录； step2: 之后掘金会将我们重定向到微信的登录页面，这个页面给出一个二维码供我们扫描，扫描之后； step3: 我们打开微信，扫描微信给的二维码之后，微信询问我们是否同意掘金使用我们的微信账号信息，我们点击同意； step4: 掘金刚才重定向到微信的二维码页面，现在我们同意掘金使用我们的微信账号信息之后，又重定向回掘金的页面，同时我们可以看到现在掘金的页面上显示我们已经处于登录状态，所以我们已经完成了用微信登录掘金的过程

这个过程比我们注册掘金后才能登录要快捷多了。这归功于 OAuth2.0，它允许客户端应用（掘金）可以访问我们的资源服务器（微信），我们就是资源的拥有者，这需要我们允许客户端（掘金）能够通过认证服务器（在这里指微信，认证服务器和资源服务器可以分开也可以是部署在同一个服务上）的认证。很明显，OAuth 2.0 提供了4种角色，资源服务器、资源的拥有者、客户端应用 和 认证服务器，它们之间的交流实现了 OAuth 2.0 整个认证授权的过程。

OAuth 2.0 登录的原理，根据4中不同的模式有所不同。本文使用授权码模式，所以只讲授权码模式下 OAuth2.0 的登录过程，其他模式可以自行搜索学习

原理图：

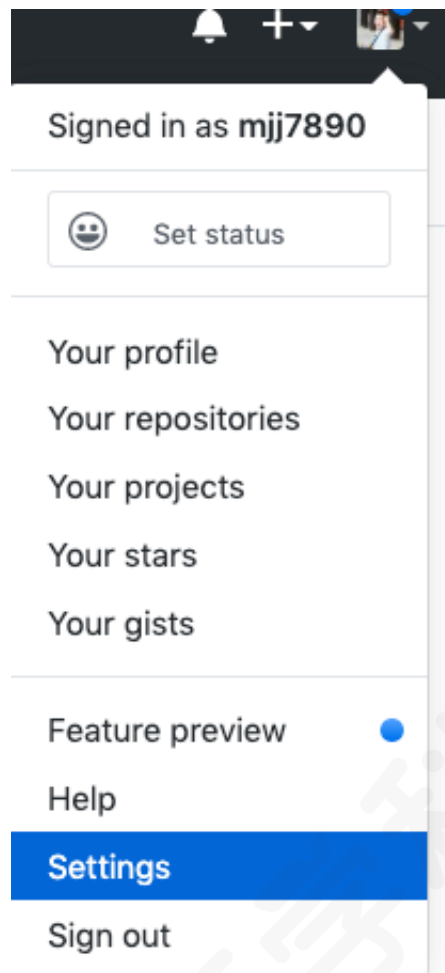


先注册

(1)

.





(2).

Personal settings

Profile

Account

Security

Emails

Notifications

Billing

SSH and GPG keys

Blocked users

Repositories

Organizations

Saved replies

Applications

Developer settings

## Public profile

Name

Your name may appear around GitHub where you contribute or are mentioned. You can remove it at any time.

Public email

Select a verified email to display

You can manage verified email addresses in your [email settings](#).

Bio

Tell us a little bit about yourself

You can @mention other users and organizations to link to them.

URL

Company

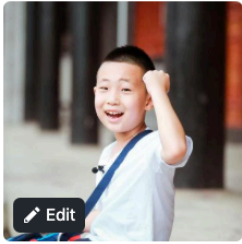
You can @mention your company's GitHub organization to link it.

Location

All of the fields on this page are optional and can be deleted at any time, and by filling them out, you're giving us consent to share this data wherever your user profile appears. Please see our [privacy statement](#) to learn more about how we use this information.

Update profile

Profile picture



Edit

(3).

Settings / Developer settings

GitHub Apps

OAuth Apps

Personal access tokens

## GitHub Apps

New GitHub App

Want to build something that integrates with and extends GitHub? [Register a new GitHub App](#) to get started developing on the GitHub API. You can also read more about building GitHub Apps in our [developer documentation](#).

(4)

**Application name \*****应用名称**

Something users will recognize and trust.

**Homepage URL \*****主页地址**

The full URL to your application homepage.

**Application description****应用描述**

This is displayed to all users of your application.

**Authorization callback URL \*****认证回调地址**

Your application's callback URL. Read our [OAuth documentation](#) for more information.

./public/login.auth.html

```
1 <!DOCTYPE html>
2 <html lang="en">
3 <head>
4   <meta charset="UTF-8">
5   <meta name="viewport" content="width=device-
width, initial-scale=1.0">
6   <meta http-equiv="X-UA-Compatible"
content="ie=edge">
7   <title>Document</title>
8 </head>
9 <body>
10   <a href="/users/login-github">Github登录</a>
11 </body>
12 </html>
```

./routes/users.js

```
1  const axios = require('axios');
2  const querystring = require('querystring');
3
4  router.get('/login-github', async (ctx, next) =>
5  {
6    // 重定向到认证接口,并配置参数
7    const path =
8    `https://github.com/login/oauth/authorize?
9    client_id=${config.client_id}`;
10    // 转发到授权服务器
11    ctx.redirect(path);
12  })
13  // OAuth 认证
14  // 用户的id和密钥
15  const config = {
16    client_id: 'baf67d3bf7e4d6662d03',
17    client_secret:
18    '8f9d964a884077c9b42bcfe91776e9adc27c906e'
19  }
20
21  router.get('/oauth/github/callback', async ctx
22  => { //这是一个授权回调, 用于获取授权码 code
23    const code = ctx.query.code; //// GitHub 回调传
24    回 code 授权码
25    // 带着 授权码code、client_id、client_secret 向
26    GitHub 认证服务器请求 token
27    const params = {
28      client_id: config.client_id,
29      client_secret: config.client_secret,
```

```
23     code: code
24   }
25
26   let res = await
  axios.post('https://github.com/login/oauth/acces
s_token', params)
27   const access_token =
  querystring.parse(res.data).access_token;
28   // 带着 token 从 GitHub 获取用户信息
29   res = await
  axios.get('https://api.github.com/user?
access_token=' + access_token);
30   console.log('userAccess', res.data);
31   ctx.redirect('/hello.html')
32 })
```

效果动态图:

