Koa实战 - 鉴权

session-cookie方式

cookie原理解析

```
// cookie.js
const http = require("http")
http
    .createServer((req, res) => {
            if(req.url === '/favicon.ico'){
          res.end('')
          return
            }
        // 观察cookie存在
        console.log('cookie:', req.headers.cookie)
        // 设置cookie
        res.setHeader('Set-Cookie', 'cookie1=abc;')
        res.end('hello cookie!!')
    })
    .listen(3000)
```

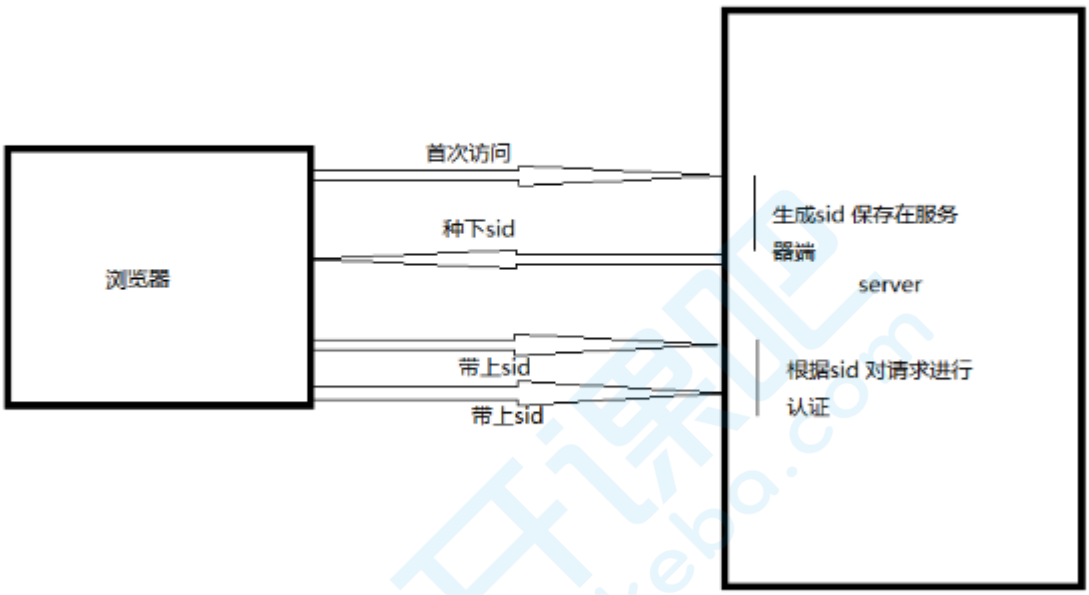
- Header Set-Cookie负责设置cookie
- 请求传递Cookie

session的原理解释

```
// cookie.js
const http = require("http")
const session = {}
http
    .createServer((req, res) => {
        // 观察cookie存在
        console.log('cookie:', req.headers.cookie)

        const sessionKey = 'sid'
        const cookie = req.headers.cookie
        if(cookie && cookie.indexOf(sessionKey) > -1 ){
            res.end('Come Back ')
            // 简略写法未必具有通用性
            const pattern = new RegExp(`${sessionKey}=([^;]+);?\s*`)
            const sid = pattern.exec(cookie)[1]
            console.log('session:',sid ,session ,session[sid])
        } else {
            const sid = (Math.random() * 99999999).toFixed()
            // 设置cookie
            res.setHeader('Set-Cookie', `${sessionKey}=${sid};`)
```

```
            session[sid] = {name : 'laowang'}
            res.end('Hello')
        }
        res.end('hello cookie!!')
    })
    .listen(3000)
```

- 原理



> 实现原理：
>
> 1. 服务器在接受客户端首次访问时在服务器端创建seesion，然后保存seesion(我们可以将seesion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。这一步通过秘钥对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。

哈希Hash - SHA MD5

```
- 把一个不定长摘要定长结果
- 摘要 xialaoshi -> x4sdfdsafsdafl3s3 - 防篡改
- 雪崩效应
```

摘要

对称DES

非对称 - RSA

- koa中的session使用：`npm i koa-session -S`

```
// app.js
const koa = require('koa')
const app = new koa()
const session = require('koa-session')

// 签名key keys作用 用来对cookie进行签名
app.keys = ['some secret'];

// 配置项
const SESS_CONFIG = {
  key: 'kkb:sess', // cookie键名
  maxAge: 86400000, // 有效期，默认一天
  httpOnly: true, // 仅服务器修改
  signed: true, // 签名cookie
};

// 注册
app.use(session(SESS_CONFIG, app));

// 测试
app.use(ctx => {
  if (ctx.path === '/favicon.ico') return;
  // 获取
  let n = ctx.session.count || 0;
  // 设置
  ctx.session.count = ++n;
  ctx.body = '第' + n + '次访问';
});

app.listen(3000)
```

使用redis存储session

HMAC

```
// redis.js
const redis = require('redis');

const client = redis.createClient(6379,'localhost');

client.set('hello','This is a value');

client.get('hello',function (err,v) {
  console.log("redis get ",v);
})
```

- 安装：`npm i -S koa-redis`
- 配置使用：

```
// koa-redis
const redisStore = require('koa-redis');
const redis = require('redis')
const redisClient = redis.createClient(6379, "localhost");

const wrapper = require('co-redis');
const client = wrapper(redisClient);

app.use(session({
  key:'kkb:sess',
  store: redisStore({client}) // 此处可以不必指定client
}, app));

app.use(async (ctx,next) => {
    const keys = await client.keys('*')
    keys.forEach(async key =>
        console.log(await client.get(key))
    )
    await next()
})
```

- session-cookie方式

```
// index.html
<html>

<head>
  <script src="https://cdn.jsdelivr.net/npm/vue/dist/vue.js"></script>
  <script src="https://unpkg.com/axios/dist/axios.min.js"></script>
</head>

<body>
  <div id="app">
    <div>
      <input v-model="username">
      <input v-model="password">
    </div>
    <div>
      <button v-on:click="login">Login</button>
      <button v-on:click="logout">Logout</button>
      <button v-on:click="getUser">GetUser</button>
    </div>
    <div>
      <button onclick="document.getElementById('log').innerHTML = ''">Clear
Log</button>
```

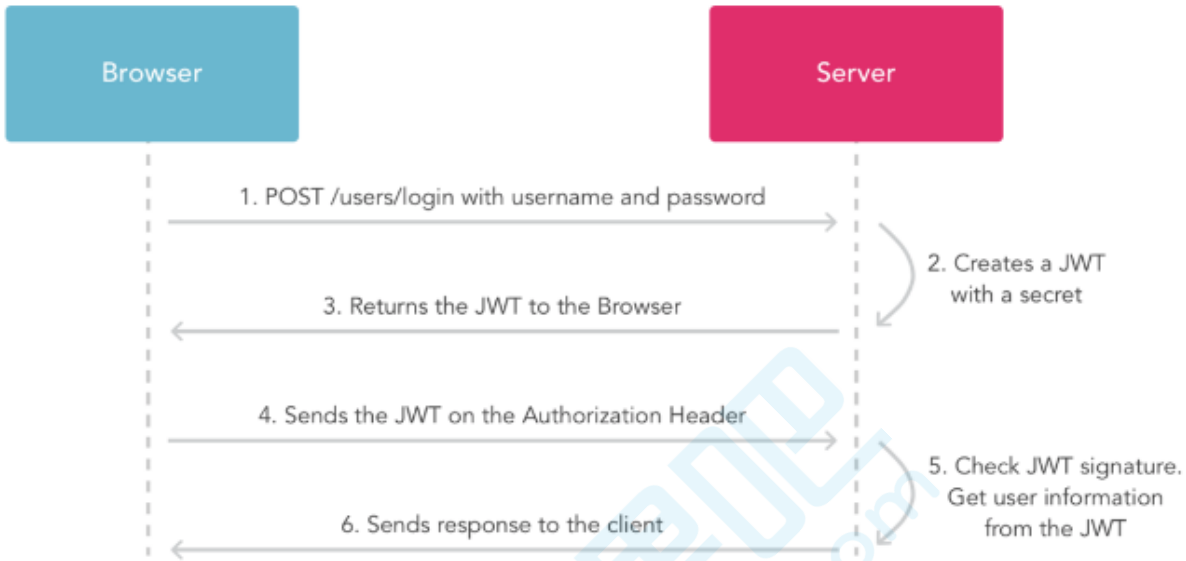
```
      </div>
    </div>
    <h6 id="log"></h6>
    </div>
    <script>
      // axios.defaults.baseURL = 'http://localhost:3000'
      axios.defaults.withCredentials = true
      axios.interceptors.response.use(
        response => {
          document.getElementById('log').append(JSON.stringify(response.data))
          return response;
        }
      );
      var app = new Vue({
        el: '#app',
        data: {
          username: 'test',
          password: 'test'
        },
        methods: {
          async login() {
            await axios.post('/users/login', {
              username: this.username,
              password: this.password
            })
          },
          async logout() {
            await axios.post('/users/logout')
          },
          async getUser() {
            await axios.get('/users/getUser')
          }
        }
      });
    </script>
  </body>
  </html>
```

```
const Koa = require('koa')
const router = require('koa-router')()
const session = require('koa-session')
const cors = require('koa2-cors')
const bodyParser = require('koa-bodyparser')
const static = require('koa-static')
const app = new Koa();

//配置session的中间件
app.use(cors({
    credentials: true
}))
app.keys = ['some secret'];

app.use(static(__dirname + '/'));
```

```
app.use(bodyParser())
app.use(session(app));

app.use((ctx, next) => {
    if (ctx.url.indexOf('login') > -1) {
        next()
    } else {
        console.log('session', ctx.session.userinfo)
        if (!ctx.session.userinfo) {
            ctx.body = {
                message: "登录失败"
            }
        } else {
            next()
        }
    }
})

router.post('/users/login', async (ctx) => {
    const {
        body
    } = ctx.request
    console.log('body',body)
    //设置session
    ctx.session.userinfo = body.username;
    ctx.body = {
        message: "登录成功"
    }
})
router.post('/users/logout', async (ctx) => {
    //设置session
    delete ctx.session.userinfo
    ctx.body = {
        message: "登出系统"
    }
})
router.get('/users/getUser', async (ctx) => {
    ctx.body = {
        message: "获取数据成功",
        userinfo: ctx.session.userinfo
    }
})

app.use(router.routes());
app.use(router.allowedMethods());
app.listen(3000);
```

Token 验证

- session不足
  - 服务器有状态
- 不灵活如果APP该怎么办 跨域怎么办
- 原理




```
1. 客户端使用用户名跟密码请求登录
2. 服务端收到请求，去验证用户名与密码
3. 验证成功后，服务端会签发一个令牌(Token)，再把这个 Token 发送给客户端
4. 客户端收到 Token 以后可以把它存储起来，比如放在 Cookie 里或者 Local Storage 里
5. 客户端每次向服务端请求资源的时候需要带着服务端签发的 Token
6. 服务端收到请求，然后去验证客户端请求里面带着的 Token，如果验证成功，就向客户端返回请求的数据
```

- 案例：令牌认证
  - 登录页，index.html

```
<html>
  <head>
    <script src="https://cdn.jsdelivr.net/npm/vue/dist/vue.js"></script>
    <script src="https://unpkg.com/axios/dist/axios.min.js"></script>
  </head>

  <body>
    <div id="app">
      <div>
        <input v-model="username" />
        <input v-model="password" />
      </div>
      <div>
        <button v-on:click="login">Login</button>
        <button v-on:click="logout">Logout</button>
        <button v-on:click="getUser">GetUser</button>
      </div>
```

```
    <div>
      <button @click="logs=[]">Clear Log</button>
    </div>
    <!-- 日志 -->
    <ul>
      <li v-for="(log,idx) in logs" :key="idx">
        {{ log }}
      </li>
    </ul>
  </div>
  <script>
    axios.interceptors.request.use(
      config => {
        const token = window.localStorage.getItem("token");
        if (token) {
          // 判断是否存在token，如果存在的话，则每个http header都加上token
          // Bearer是JWT的认证头部信息
          config.headers.common["Authorization"] = "Bearer " + token;
        }
        return config;
      },
      err => {
        return Promise.reject(err);
      }
    );

    axios.interceptors.response.use(
      response => {
        app.logs.push(JSON.stringify(response.data));
        return response;
      },
      err => {
        app.logs.push(JSON.stringify(response.data));
        return Promise.reject(err);
      }
    );
    var app = new Vue({
      el: "#app",
      data: {
        username: "test",
        password: "test",
        logs: []
      },
      methods: {
        login: async function() {
          const res = await axios.post("/users/login-token", {
            username: this.username,
            password: this.password
          });
          localStorage.setItem("token", res.data.token);
        },
        logout: async function() {
          localStorage.removeItem("token");
```

```
          },
          getUser: async function() {
            await axios.get("/users/getUser-token");
          }
        }
      });
    </script>
  </body>
</html>
```

- 登录接口
  - 安装依赖：`npm i jsonwebtoken koa-jwt -S`
  - 接口编写，index.js

```
const Koa = require('koa')
const router = require('koa-router')()

const jwt = require("jsonwebtoken")
const jwtAuth = require("koa-jwt")
const secret = "it's a secret"
const cors = require('koa2-cors')
const bodyParser = require('koa-bodyparser')
const static = require('koa-static')
const app = new Koa();
app.keys = ['some secret'];

app.use(static(__dirname + '/'));
app.use(bodyParser())

router.post("/users/login-token", async ctx => {
  const { body } = ctx.request;
  //登录逻辑，略
  //设置session
  const userinfo = body.username;
  ctx.body = {
    message: "登录成功",
    user: userinfo,
    // 生成 token 返回给客户端
    token: jwt.sign(
      {
        data: userinfo,
        // 设置 token 过期时间，一小时后，秒为单位
        exp: Math.floor(Date.now() / 1000) + 60 * 60
      },
      secret
    )
  };
});

router.get(
  "/users/getUser-token",
```

```
  jwtAuth({
    secret
  }),
  async ctx => {
// 验证通过，state.user
    console.log(ctx.state.user);

    //获取session
    ctx.body = {
      message: "获取数据成功",
      userinfo: ctx.state.user.data
    };
  }
);


app.use(router.routes());
app.use(router.allowedMethods());
app.listen(3000);
```

JWT(JSON WEB TOKEN)原理解析

1. Bearer Token包含三个组成部分：令牌头、payload、哈希

eyJhbGciOiJIUzI1NiIsInR5cCI6IkpXVCJ9.eyJkYXRhIjp7InVzZXJuYW1lIjoiYWJjIiwicGFzc3dvcmQiOiIxMTExMTEifSwiZXhwIjoxNTU3MTU1NzMwLCJpYXQiOjE1NTcxNTIxMzB9.pjGaxzX2srG_MEZizzmFEy7JM3t8tjkiu3yULgzFwUk

1. 签名：默认使用base64对payload编码，使用hs256算法对令牌头、payload和密钥进行签名生成哈希
2. 验证：默认使用hs256算法对hs256算法对令牌中数据签名并将结果和令牌中哈希比对

```
// jsonwebtoken.js
const jsonwebtoken = require('jsonwebtoken')
const secret = '12345678'
const opt = {
  secret: 'jwt_secret',
  key: 'user'
}
const user = {
  username: 'abc',
  password: '111111'
}

const token = jsonwebtoken.sign({
  data: user,
  // 设置 token 过期时间
  exp: Math.floor(Date.now() / 1000) + (60 * 60),
}, secret)

console.log('生成token:' + token)
```

```
// 生成
token:eyJhbGciOiJIUzI1NiIsInR5cCI6IkpXVCJ9.eyJkYXRhIjp7InVzZXJuYW1lIjoiYWJjIiwicGFzc3dvcmQiOiIxMTExMTEifSwiZXhwIjoxNTQ2OTQyMzk1LCJpYXQiOjE1NDY5Mzg3OTV9.VPBCQgLB7XPBq3RdHK9WQMkPp3dw65JzEKm_LZZjP9Y
console.log('解码:', jsonwebtoken.verify(token, secret, opt))
// 解码: { data: { username: 'abc', password: '111111' },
//   exp: 1546942395,
//   iat: 1546938795 }
```

> HMAC SHA256 HMAC(Hash Message Authentication Code，散列消息鉴别码，基于密钥的Hash算法的认证协议。消息鉴别码实现鉴别的原理是，用公开函数和密钥产生一个固定长度的值作为认证标识，用这个标识鉴别消息的完整性。使用一个密钥生成一个固定大小的小数据块，即MAC，并将其加入到消息中，然后传输。接收方利用与发送方共享的密钥进行鉴别认证等。
>
> **BASE64** 按照RFC2045的定义，Base64被定义为：Base64内容传送编码被设计用来把任意序列的8位字节描述为一种不易被人直接识别的形式。（The Base64 Content-Transfer-Encoding is designed to represent arbitrary sequences of octets in a form that need not be humanly readable.）常见于邮件、http加密，截取http信息，你就会发现登录操作的用户名、密码字段通过BASE64编码的
>
> **Beare**
>
> Beare作为一种认证类型(基于OAuth 2.0)，使用"Bearer"关键词进行定义
>
> **参考文档:**
>
> jsonwebtoken、koa-jwt
>
> 阮一峰 JWT解释
>
> http://www.ruanyifeng.com/blog/2018/07/json_web_token-tutorial.html

OAuth(开放授权)

- 概述：三方登入主要基于OAuth 2.0。OAuth协议为用户资源的授权提供了一个安全的、开放而又简易的标准。与以往的授权方式不同之处是OAUTH的授权不会使第三方触及到用户的帐号信息（如用户名与密码），即第三方无需使用用户的用户名与密码就可以申请获得该用户资源的授权，因此OAUTH是安全的。
- OAuth登录
  - 登录页面 index.html

```
<html>
<head>
  <script src="https://cdn.jsdelivr.net/npm/vue/dist/vue.js"></script>
  <script src="https://unpkg.com/axios/dist/axios.min.js"></script>
</head>
<body>
  <div id="app">
      <a href='/github/login'>login with github</a>
  </div>
</body>
</html>
```

- 登录接口 index.js

```
const Koa = require('koa')
const router = require('koa-router')()
const static = require('koa-static')
const app = new Koa();
const axios = require('axios')
const querystring = require('querystring')

app.use(static(__dirname + '/'));
const config = {
    client_id: '73a4f730f2e8cf7d5fcf',
    client_secret: '74bde1aec977bd93ac4eb8f7ab63352dbe03ce48'
}

router.get('/github/login', async (ctx) => {
    var dataStr = (new Date()).valueOf();
    //重定向到认证接口,并配置参数
    var path = "https://github.com/login/oauth/authorize";
    path += '?client_id=' + config.client_id;

    //转发到授权服务器
    ctx.redirect(path);
})
router.get('/github/callback', async (ctx) => {
    console.log('callback..')
    const code = ctx.query.code;
    const params = {
        client_id: config.client_id,
        client_secret: config.client_secret,
        code: code
    }
    let res = await axios.post('https://github.com/login/oauth/access_token',
params)
    const access_token = querystring.parse(res.data).access_token
    res = await axios.get('https://api.github.com/user?access_token=' +
access_token)
    console.log('userAccess:', res.data)
    ctx.body = `
        <h1>Hello ${res.data.login}</h1>
```

```
            <img src="${res.data.avatar_url}" alt=""/>
        `


})

app.use(router.routes()); /*启动路由*/
app.use(router.allowedMethods());
app.listen(3000);
```

- 单点登录

```
cd passport
node app.js
cd ../system
PORT=8081 SERVER_NAME=a node app.js
PORT=8082 SERVER_NAME=b node app.js


#user test
#password 123456
```