一、身份验证Token说明

1. Token的作用

Token是在客户端频繁向服务端请求数据，服务端频繁的去数据库查询用户名和密码并进行对比，判断用户名和密码正确与否，并作出相应提示，在这样的背景下，Token便应运而生。

Token是服务端生成的一串字符串，以作客户端进行请求的一个令牌，当第一次登录后，服务器生成一个Token便将此Token返回给客户端，以后客户端只需带上这个Token前来请求数据即可，无需再次带上用户名和密码。

Token的目的是为了减轻服务器的压力，减少频繁的查询数据库，使服务器更加健壮。

2. Token使用步骤

使用基于 Token 的身份验证方法，在服务端不需要存储用户的登录记录，大致的流程如下：

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 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。

二、交易所项目中的应用

jsonwebtoken

nodejs中jsonwebtoken库封装了token的功能。实现了web token令牌，默认签名算法是HMAC SHA256，也支持RSA SHA256签名等，expiresIn定义了到期的声明，使用方法如下：

```
jwt.sign({
  data: 'foobar'
}, 'secret', { expiresIn: '1h' });
```

验证可以使用下面两种方式：

```
// 同步
var decoded = jwt.verify(token, 'secret');
console.log(decoded.data) // foobar

// 异步
jwt.verify(token, 'secret', function(err, decoded) {
  console.log(decoded.data) // foobar
});
```

签名

在验证用户传来的用户名与密码成功后，将传来的数据进行签名，可以选择需要签名的字段，在验签时将可以全部解码出来。

```
const jwt = require('jsonwebtoken');

module.exports = {

    login: async(ctx) => {
        let params = ctx.request.body
        console.log(JSON.stringify(params))
        let {username, password} = params
        //1.在数据库中验证用户名与密码
        let {error, data} = await user.checkUser(username, password)
        if (error) {
            ctx.body = fail(error)
            return
        }

        //2.验证成功后进行签名
        let token = jwt.sign({
            id: data.id,
            phone: data.phone,
            password: data.password
        }, secret, {expiresIn: '24h'});

        //3.将签名后的token返回给客户端
        ctx.body = success(token)
    },
}
```

验签

当用户登录后进行的所有操作都需带上后台签名后返给前端的token。因此需要写一个中间件拦截所有请求，但至少还需过滤掉登录、注册，因为这两个接口都还未生成token，通过`jwt.verify(token, secret);`可以解密客户端请求头获取到的cookie为token的值，若成功获取加密的字典数据，这里解密后通过`decoded.id`把用户id附带到了请求体或者请求路径中，用于在业务功能控制器中容易获取到访问者的id。否则返回“token错误”，拒绝访问。

```
const jwt = require('jsonwebtoken');

//拦截所有请求
app.use(async(ctx, next) => {
    urlpath = ctx.request.path

    //1. 过滤掉不验证token的接口
    if(urlpath.indexOf("/login") == 0
    || urlpath.indexOf("/register") == 0
    || urlpath.indexOf("/css/") == 0
    || urlpath.indexOf("/html/") == 0
    || urlpath.indexOf("/js/") == 0){
        await next();
        return;
    }

    //2.从客户端的请求头中获取cookie为token的数据
    var token = ctx.cookies.get("token")
    console.log("token:"+token)

    //3.未附带token则拒绝访问
    if (!token) {
        ctx.body = fail("请登录！");
        return;
    }

    //4.通过秘钥secret解密token
    var decoded = jwt.verify(token, secret);

    //5.验证成功后能获取到用户id、用户名等数据。
    if (decoded) {
        //6. 根据请求类型不同，将用户数据设置到不同的位置
        if (ctx.request.method == "POST") {
            ctx.request.body.userid = decoded.id;
            ctx.request.body.phone = decoded.phone;
            console.log(JSON.stringify(ctx.request.body))
            // ctx.request.body.userinfo = decoded;
        } else {
            ctx.request.query.userid = decoded.id;

            ctx.request.query.phone = decoded.phone;
```

```
            console.log(JSON.stringify(ctx.request.query))
            // ctx.request.query.userinfo = decoded;
        }
        await next();
    } else {
        ctx.body = fail("token错误");
        return;
    }
})
```