Cookie 和 Session

会话控制技术

1. Cookies

Cookie 曲奇饼干，但是在这里控制会话

什么是会话?

在 http 协议中，**一次请求和一次响应就是一次会话**。

多次会话中，第二次会话，并不会知道上一次会话处理了些什么内容?

HTTP是短连接，无状态的；

所谓的Cookie技术，就是在用户发起请求之后，对于用户如果有特别需要标记的内容，**服务器端**就生成出一些字符，最多也就4kb，返回给客户端进行自动存储。

下一次会话的时候，**客户端** 会自动把Cookie通过 **请求头** 的方式自动携带。

服务器端进行解析，即可。

代码演示：

需要用到第三方中间件

```
$ npm i cookie-parser
```

注册cookie的代码 JavaScript 复制代码

```javascript
// 1. 引入模块
const cookieParser = require('cookie-parser');

// 3. 注册中间件
// 注册cookie服务
app.use(cookieParser());
```

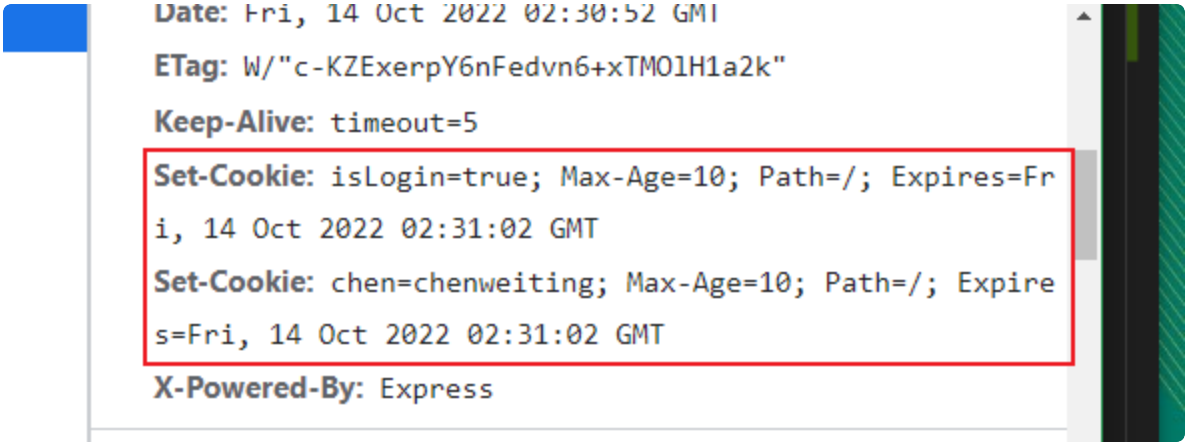
处理登录成功之后的cookie设置 JavaScript 复制代码

```javascript
// 2.1.2 POST 请求  登录
home.post('/login', (req, res) => {
    // post 请求的请求体参数，在 req.body 中
    // uname=amdin&password=abc123
    // { uname: "admin", password: "abc123" }
    const { uname, password } = req.body;

    if (uname !== 'admin') {
        res.send('用户名输入错误')
    } else if (password !== 'abc123') {
        res.send('密码错误')
    } else {
        // 登录成功后，使用res设置cookie，返回给客户端进行存储
        // maxAge: cookie 的有效时长，单位是毫秒
        // 一天的时长 1000 * 60 * 60 * 24
        res.cookie('isLogin', 'true', { maxAge: 10 * 1000 });
        res.cookie('chen', 'chenweiting', { maxAge: 10 * 1000 });

        res.send('登录成功')
    }
})
```

在响应头中，返回cookie的字段

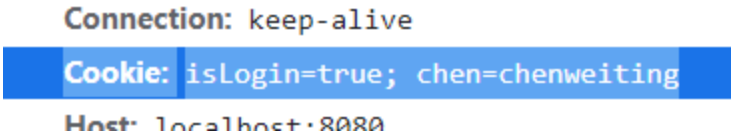


可以手动跳转到首页，请求头中会自动携带设置的cookie

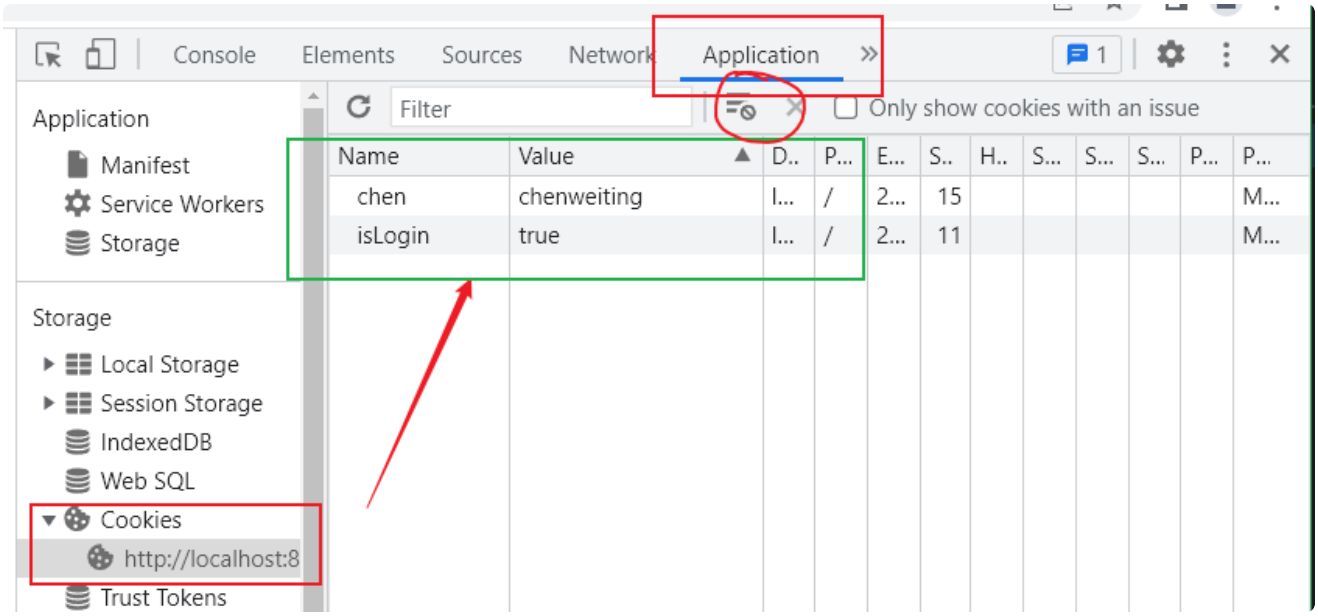
根据cookie字段判断是否登录过 JavaScript 复制代码

```
// 2.1.3 GET 请求    跳转到首页
home.get('/', (req, res) => {
    // 从req中获取cookies
    // console.log(req.cookies);
    const { isLogin, chen } = req.cookies;
    if (isLogin === 'true' || chen === 'chenweiting') {
        // 已经登陆过的
        res.render('index', {});
    } else {
        // 跳转到登录页
        // redirect 重定向到登录页面
        res.redirect('/login');
    }
})
```

如果本地有cookie，那么会自动携带进行判断



cookie存储在浏览器的内存里面，可以通过自带的工具查看



cookie 的缺点：

数据量不能太大，单条最多4kb，最多50条，数据量越大，请求加载越慢；

明文显示，不太安全，如果涉及到敏感数据，则不好处理，容易被劫持

2. Session

Session就是会话技术，它的特点是相对于 Cookie 存储量更大，存储在服务器端。

Session 基于 Cookie 进行工作的；

使用 Session 一般会使用第三方中间件

```
$ npm i express-session
```

注册session的代码 JavaScript 复制代码

```javascript
const session = require('express-session');

// 注册session
app.use(session({
    // 返回到客户端cookie的键 - 随便写
    name: 'sid',
    // secret 一般作为参与加密的字符串(签名)
    secret: 'helloword',
    // 是否每次为请求设置cookie
    saveUninitialized: false,
    // 是否在每次请求的时候重新保存
    resave: true,
    // 关于客户端cookie的描述
    cookie: {
        // 只能在http请求中完成
        httpOnly: true,
        // 最大有效时长
        maxAge: 20 * 1000,
        // 过期时间
        // expires
        // 有效范围
        path: '/'
    },
    // 存储位置：数据库，指定的文件，或者其他方式
    // store
}))
```

处理后端接口， 成功之后给 **请求头** 设置session

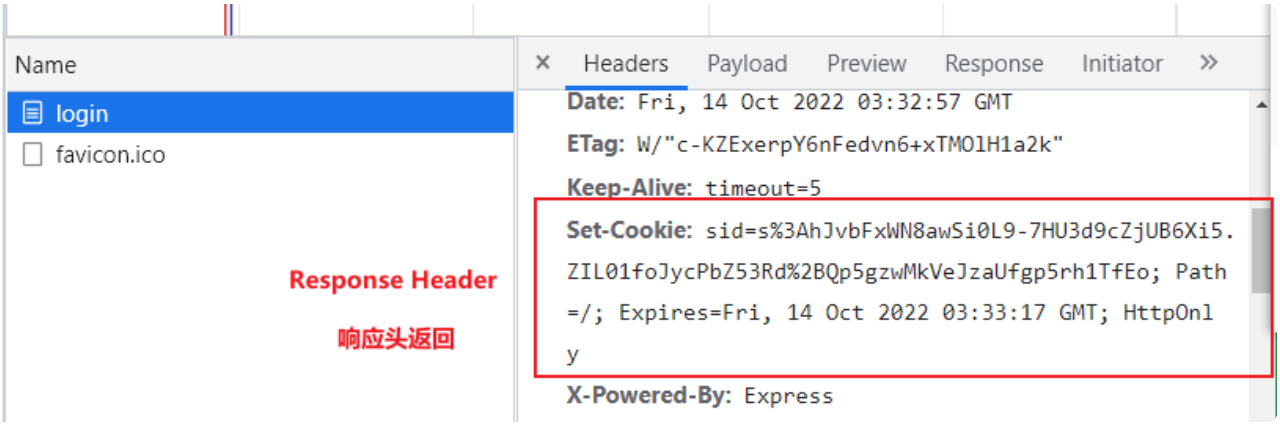
登录成功之后设置session　　JavaScript　复制代码

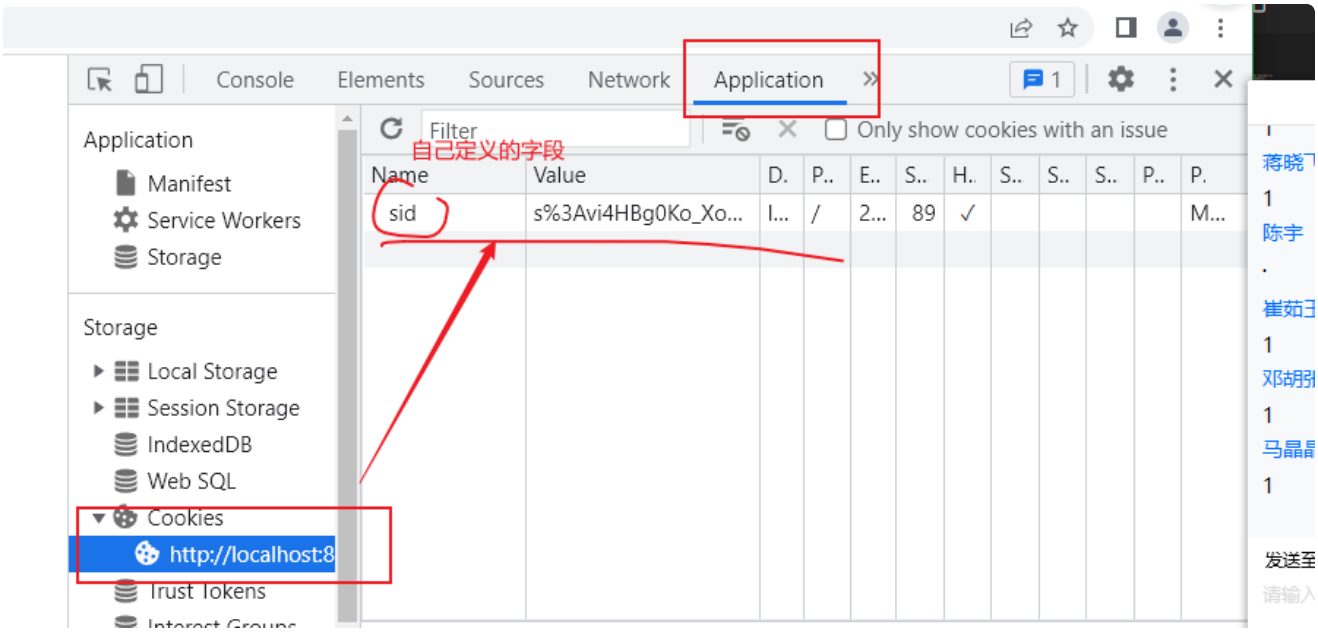
```
// 2.1.2 POST 请求  登录
home.post('/login', (req, res) => {
    const { uname, password } = req.body;

    if (uname !== 'admin') {
        res.send('用户名输入错误')
    } else if (password !== 'abc123') {
        res.send('密码错误')
    } else {

        // 给 请求头 设置session
        req.session.isLogin = true;
        req.session.users = req.body;

        res.send('登录成功')
    }
})
```





判断用户是否登录

通过请求头获取session字段 JavaScript 复制代码

```
// 2.1.3 GET 请求    跳转到首页
home.get('/', (req, res) => {
    // 数据是存在服务器端的，这次请求会自动携带cookie的密钥
    // 在session中间件解析完毕之后，返回给req.session
    // console.log(req.session);
    const { isLogin, users } = req.session;
    if (isLogin) {
        // 已经登陆过的
        res.render('index', users);
    } else {
        // 跳转到登录页
        res.redirect('/login');
    }
})
```

Session 的问题：

Session 确实可以存储大量的数据在服务器上，最终通过cookie返回加密的字段

客户端在发起后续的请求时，通过自动携带的cookie可以获取到服务器对应的数据

那么，Session 就一定安全吗?

相对来说，session会比cookie要安全一些，但是如果要获取了加密字段

依然可以根据有用的登录信息获取用户的数据，只是说大部分场景没有那么隐私，所以这个可以忽略不计。

3. 数据安全的其他方案

如果使用的是 http 协议，那么不管是 Cookie 或者是 Session 都差不多

如果使用的是 https 协议，那么不管是 Cookie 或者是 Session 也挺安全

每一次发送的请求，接收的响应，https都会进行加密处理的

一般在同级的服务器中，客户端给服务器端发送，基本上使用 Cookie 或者 Session就可以了(就是前端的页面或者代码，工作区在后端的服务器中)

大部分的业务场景中，后端比如是Java，他们的服务器在别的地方启动

http://192.168.97.126:5566 => 服务器端运行的地址

一般客户端的代码也会另起一台服务器，托管前端代码

http://192.168.97.118:8080 => 客户端运行项目的地址

如果是跨域了，跨服务器了，cookie是没法用的，需要特殊处理。

使用JWT的方式完成验证

Json Web Token， 是不同服务器之间解决验证的新方式。

需要下载验证的第三方 中间件

```
$ npm i express-jwt jsonwebtoken
```

参考链接： https://www.jianshu.com/p/6347fc14c185

生成一个 Token 的加密字段，返回给客户端，后续请求需要手动设置请求头