# cookie与session

## 1 cookie简介

http是无状态协议，即使在同一个网站，从一个页面跳转到另外一个页面时，服务器和浏览器都没有任何内在的方法认识到他们的关联，所有的http请求都要包含所有必要的信息，服务器才能满足这个请求。

为了实现‘登录’、记忆用户喜好等功能，就有了cookie和session（会话）。cookie的想法很简单：服务器发送一点信息，浏览器在一段可配置的时期内保存它。

cookie和session都是基于服务器的，不同的是cookie存储在客户端，session存储在服务端。

cookie包括：名字、值、过期时间、路径、域，路径与域一起构成cookie的作用范围；cookie只有4k大小。

过期时间如果不设置，关闭浏览器，cookie就会消失，这种生命期为浏览器会话期间的cookie被称为会话cookie。会话cookie一般保存在内存中，这种行为是不规范的。若设置了过期时间，则保存信息到硬盘上。

第一次访问一个服务器，不可能携带cookie，必须是服务器得到这次请求，在下行响应报头中，携带cookie信息，此后每次浏览器往这个服务器发送的请求，都会携带这个cookie。

cookie特点：

cookie是不加密的，用户可以自由看到；

用户可以删除cookie，或者禁用它；

cookie可以被篡改、攻击；

cookie存储量很小，未来要被localStorage替代。

## 2 cookie在express中的应用

express中的cookie，你肯定能想到： res负责设置cookie， req负责识别cookie。

const express = require('express');  
const cookieParser = require('cookie-parser');  
  
let app = express();  
app.use(cookieParser());  
  
app.get('/',function(req,res){  
 res.cookie('name','lisi',{maxAge:900000,httpOnly:true});  
 res.send(req.cookies);  
});  
  
app.get('/other',function (req,res) {  
 *//当用户访问首页缓存了cookie后，再访问other，在这个页面我们就可以获取cookie* let cookieName = req.cookies.name;  
 res.send(cookieName);

});  
  
app.listen(3000);

## 3 session在express中的应用

Session（会话）不是一个天生就有的技术，而是依赖cookie。

当程序需要为某个客户的请求创建一个session时，服务器首先检查这个客户端请求里是否包含了一个session标 识session id，如果包含，则说明此前已经为此客户端创建了session，服务器就按照session id把这个session检索出来使用，检索不到会新建一个。

如果客户端请求中不包含session id，服务器则为客户端创建session且生成一个与此session关联的session id。session id 的值应该是一个既不会重复，也不会容易被发现的规律字符串。在本次响应中，session id 将会返回给客户端保存，客户端保存session id 的方式是cookie，cookie名一般是 SESSIONID。

基础案例：

var session = require("express-session");  
app.use(session({  
 secret: 'keyboard cat',  
 resave: false,  
 saveUninitialized: true  
}));  
  
app.get("/",function(req,res){  
 if(req.session.login == "1"){  
 res.send("欢迎" + req.session.username);  
 }else{  
 res.send("没有成功登陆");  
 }  
 });  
  
app.get("/login",function(req,res){  
 req.session.login = "1"; *//设置这个session* req.session.username = "考拉";  
 res.send("你已经成功登陆");  
});

## 4 综合案例

const express = require('express');  
const cookieParser = require('cookie-parser');  
const session = require('express-session');  
  
*//使用第三方库，使session在服务器关闭重启后，数据保持一致*const FileStore = require('session-file-store')(session);  
  
let app = express();  
  
app.use(cookieParser());  
  
app.use(session({  
 store: new FileStore(),  
 *//没有前两个选项会有警告* resave: false,  
 saveUninitialized: true,  
 secret: 'aF,.j)wBhq+E9n#aHHZ91Ba!VaoMfC', *// 建议使用 128 个字符的随机字符串* cookie: {  
 maxAge:20000 *//加入寿命可以确保 浏览器关闭再开启后数据仍然存在* },  
}));  
  
*//session的数据存储在服务器中，每次服务器关闭数据都会回到初始状态*app.get('/num',function (req,res) {  
 if (!req.session.num) {  
 req.session.num = 1;  
 }  
 req.session.num++;  
 res.send('num is ' + req.session.num);  
});  
  
app.listen(2000);

## 5 session模块实现

Node没有提供session模块，需要我们自己实现。

startSession方法：启动session管理

newSession方法：为客户端创建一个新的session id

cleanSession方法：清楚session数据。

本模块应用一个session数组来存储所有的session，当有session id时，直接返回session数据，当session id 不存在时，创建一个，并存储在客户端的cookie中。

let sessions = [];  
function startSession(req, res) {  
 let connect = {req: res, res: res};  
 let cookies = {};  
 if (typeof connect.req.headers.cookie != 'undefined') {  
 connect.req.headers.cookie.split(';').forEach(function (cookie) {  
 let parts = cookie.split('=');  
 cookies[parts[0].trim()] = (parts[1] || '').trim();  
 });  
 } else {  
 cookies.SESSIONID = 0;  
 }  
  
 let SESSIONID = cookies.SESSIONID;  
 if (typeof sessions[SESSIONID] != 'undefined') {  
 session = sessions[SESSIONID];  
 if(session.expires < Date()){  
 delete sessions[SESSIONID];  
 return newSession(connect.res);  
 } else {  
 let date = new Date();  
 session.expires = date;  
 return sessions[SESSIONID];  
 }  
 } else {  
 return newSession(connect.res);  
 }  
  
};  
  
function newSession(res) {  
 let chars = '0123456789ABCDEFGHIJKLMNOPQRSTUVWXYZabcdefghijklmnopqrstuvwxyz';  
 let SEESIONID = '';  
 *//随机产生一个长度为40的SESSIONID* for(let i = 0; i < 40; i++){  
 let rnum = Math.floor(Math.random() \* chars.length);  
 SEESIONID += chars.substring(rnum,rnum+1);  
 }  
 if (typeof sessions[SEESIONID] != 'undefined') {  
 return newSession(res);  
 }  
 let date = new Date();  
 date.setMinutes(date.getMinutes() + 30);  
 let session = {  
 SEESIONID:SEESIONID,  
 expires:date  
 };  
 sessions[SEESIONID] = session;  
 res.setHeader('Set-Cookie','SESSION=' + SEESIONID);  
 return session;  
}  
  
function cleanSeesion() {  
 for(session in sessions){  
 if(session.expires < Date()){  
 delete sessions[session.SEESIONID];  
 }  
 }  
}

const http = require('http');  
let app = http.createServer(function (req,res) {  
 global.sessionLib = lib.session.startSession(req,res);  
});  
  
*// 调用方式*if(!sessionLib['name']){  
 sessionLib['name'] = 'zs';  
}

将startSeesion返回的session对象作为全局变量存储。

## 6 带加密的登录注册

const express = require('express');  
const crypto = require('crypto');  
const session = require('express-session');  
const bodyParser = require('body-parser');  
  
let userdata = {};  
  
let app = express();  
app.set('view engine','ejs');  
app.use(express.static('./public'));  
app.use(session({  
 secret:'hihi',  
 resave:false,  
 saveUninitialized:true  
}));  
*//配置bodyparser*app.use(bodyParser.urlencoded({extended: true}));  
  
  
app.get('/',function (req,res) {  
 if(req.session.isLogin == '1'){  
 res.send('欢迎' + req.session.name);  
 } else {  
 res.send('请先登录');  
 }  
});  
  
app.get('/login',function (req,res) {  
 res.render('login');  
});  
app.post('/login',function (req,res) {  
 let username = req.body.username;  
 let password = req.body.password;  
 let md5password = md5PWD(md5PWD(password).substr(2,5) + md5PWD(password));  
 if(userdata.username == username && userdata.password == md5password){  
 req.session.name = username;  
 req.session.isLogin = '1';  
 res.send('登录成功');  
 } else {  
 res.send('登录失败');  
 }  
});  
  
app.get('/regist',function (req,res) {  
 res.render('regist');  
});  
app.post('/regist',function (req,res) {  
 let username = req.body.username;  
 let password = req.body.password;  
 *//各种加密* let md5password = md5PWD(md5PWD(password).substr(2,5) + md5PWD(password));  
 *//存储* userdata.username = username;  
 userdata.password = md5password;  
 res.send('注册成功');  
});  
  
app.listen(3000);  
  
function md5PWD(password) {  
 let md5 = crypto.createHash('md5');  
 let newPWD = md5.update(password).digest('base64');  
 return newPWD;  
}