## **1. 什么是session [#](https://zhufeng-document.vercel.app/html/23.session.html" \l "t01. %E4%BB%80%E4%B9%88%E6%98%AFsession)**

* session是另一种记录客户状态的机制，不同的是Cookie保存在客户端浏览器中，而session保存在服务器上
* 客户端浏览器访问服务器的时候，服务器把客户端信息以某种形式记录在服务器上，这就是session。客户端浏览器再次访问时只需要从该Session中查找该客户的状态就可以了

## **2. cookie与session区别**

1. cookie数据存放在客户的浏览器上，session数据放在服务器上。
2. cookie不是很安全，别人可以分析存放在本地的COOKIE并进行COOKIE欺骗 考虑到安全应当使用session
3. session会在一定时间内保存在服务器上。当访问增多，会比较占用你服务器的性能 考虑到减轻服务器性能方面,应当使用COOKIE
4. 单个cookie保存的数据不能超过4K，很多浏览器都限制一个站点最多保存20个cookie

将登陆信息等重要信息存放为session、其他信息如果需要保留，可以放在cookie中

## **3. session实现**

1. 在服务器端生成全局唯一标识符session\_id
2. 在服务器内存里开辟此session\_id对应的数据存储空间
3. 将session\_id作为全局唯一标示符通过cookie发送给客户端
4. 以后客户端再次访问服务器时会把session\_id通过请求头中的cookie发送给服务器
5. 服务器再通过session\_id把此标识符在服务器端的数据取出

**var** express = require('express');**var** cookieParser = require('cookie-parser');**var** app = express();

app.use(cookieParser());//存放会话数据 key卡号 value就是卡号对应的数据对象**var** sessions = {};//与客户端约定的会话ID**var** SESSION\_KEY = 'connect.sid'//当用户访问根目录的时候 执行对应的回调函数

app.get('/',**function**(req,res){

res.setHeader('Content-Type','text/html;charset=utf-8');

// 1. 先取出cookie中的sessionId 卡号

**var** sessionId = req.cookies[SESSION\_KEY];

// 如果有卡号的，也就是有ID的话 老顾客

**if**(sessionId){

//取出此卡号对应的信息，余额

**var** sessionObj = sessions[sessionId];

**if**(sessionObj){

//扣掉10块钱

sessionObj.balance = sessionObj.balance -10;

res.send('欢迎你老顾客，你卡上还剩'+sessionObj.balance);

}**else**{

genId(res);

}

//如果没有的话就是新顾客

}**else**{

genId(res);

}

**function** **genId**(res){

//由店家生成一个唯一的卡号

**var** id = Date.now()+''+Math.random();

//要在店家的小本上记录一下此卡号对应的余额

sessions[id] = {balance:100};

//把这个卡发给顾客带回家

res.cookie(SESSION\_KEY,id);

//告诉 用户送他一张卡

res.send('欢迎你新顾客，送你一张价值100元的剪发卡');

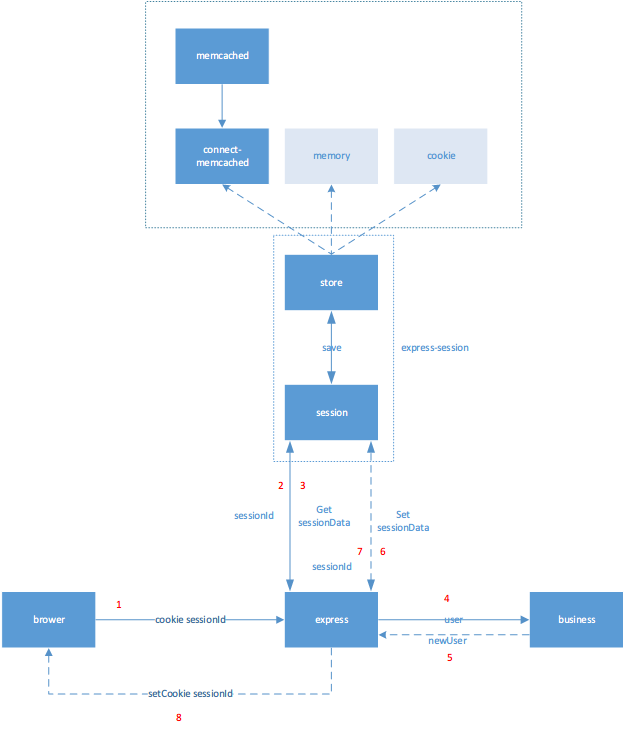
}

});

app.listen(9090);

## **4. session中间件**

[session](https://github.com/expressjs/session)



$ npm install express-session

| **参数** | **描述** |
| --- | --- |
| name | 设置 cookie 中，保存 session 的字段名称，默认为 connect.sid |
| store | session 的存储方式，默认存放在内存中，也可以使用 redis，mongodb 等 |
| secret | 通过设置的 secret 字符串，来计算 hash 值并放在 cookie 中，使产生的 signedCookie 防篡改 |
| cookie | 设置存放 session id 的 cookie 的相关选项，默认为 (default: { path: '/', httpOnly: true, secure: false, maxAge: null }) |
| genid | 产生一个新的 session\_id 时，所使用的函数， 默认使用 uid2 这个 npm 包 |
| rolling | 每个请求都重新设置一个 cookie，默认为 false |
| saveUninitialized | 是指无论有没有session cookie，每次请求都设置个session cookie ，默认给个标示为 connect.sid |
| resave | 是指每次请求都重新设置session cookie，假设你的cookie是10分钟过期，每次请求都会再设置10分钟 |

## **5. 实现sesion计数器**

**let** express = require('express');**let** session = require('express-session');**let** path = require('path');**let** FileStore = require('./filestore')(session);**let** app = express();

app.use(session({

secret: 'zfpx',

resave: true,

saveUninitialized: true,

store: **new** FileStore({

root: path.join(\_\_dirname, 'sessions'),

maxAge: 1000,

gc: 1

})

}));

app.get('/visit', **function** (req, res) {

**let** visit = req.session.visit;

**if** (visit) {

visit = visit + 1;

} **else** {

visit = 1;

}

req.session.visit = visit;

res.send(`欢迎你的第${visit}次光临`);

});

app.listen(8080);

## **6. session实现权限**

**var** express = require('express');**var** cookieParser = require('cookie-parser');**var** session = require('express-session');**var** uuid = require('uuid');**var** app = express();

app.set('view engine', 'html');

app.engine('html', require('ejs').\_\_express);

app.set('views', \_\_dirname);

app.use(require('cookie-parser')());

app.use(session({secret: 'zfpx',

resave: true,

saveUninitialized: true}));

/\*\*

\* curl -v -H "cookie: username=customer" http://localhost:8080/user

\*/

**function** **checkUser**(req, res, next) {

**if** (req.session && req.session.username)

next();

**else**

res.redirect('/');

}

//进入登录页

app.get('/', **function** (req, res) {

res.render('index');

});

//登录

app.get('/login', **function** (req, res) {

req.session.username = req.query.username;

res.redirect('/user');

});

//用户页面

app.get('/user', **function** (req, res) {

console.log(req.session);

res.render('user', {username: req.session.username});

});

//用户退出

app.get('/logout', **function** (req, res) {

req.session.usrename = null;

res.redirect('/');

});

app.listen(8080);

## **7. 自定义存储位置**

| **方法** | **含义** |
| --- | --- |
| get | 获取sessoin |
| set | 设置session |
| destroy | 销毁session |

### **7.1 保存到文件中**

**let** util = require('util');**let** mkdirp = require('mkdirp');**let** fs = require('fs');**const** path = require('path');

**function** **createFileStore**(session) {

**const** Store = session.Store;

util.inherits(FileStore, Store);

**function** **FileStore**(options) {

**let** { dir = path.resolve(\_\_dirname, 'sessions') } = options || {};

**this**.dir = dir;

mkdirp(**this**.dir);

}

FileStore.prototype.resolve = **function** (sessionId) {

**return** path.join(**this**.dir, `${sessionId}.json`);

}

FileStore.prototype.get = **function** (sessionId, callback) {

fs.readFile(**this**.resolve(sessionId), 'utf8', (err, data) => {

**if** (err) **return** callback(err);

callback(err, JSON.parse(data));

});

}

FileStore.prototype.set = **function** (sessionId, session, callback) {

fs.writeFile(**this**.resolve(sessionId), JSON.stringify(session), callback);

}

FileStore.prototype.destroy = **function** (sessionId, callback) {

fs.unlink(**this**.resolve(sessionId), callback);

}

**return** FileStore;

}

module.exports = createFileStore;//FileStore

### **7.2 保存到Redis数据库中**

**let** util = require('util');**var** redis = require("redis");**function** **createRedisStore**(session) {

**const** Store = session.Store;

util.inherits(RedisStore, Store);

**function** **RedisStore**(options = {}) {

**this**.client = redis.createClient(options.port || 6379, options.host || 'localhost');

}

RedisStore.prototype.get = **function** (sessionId, callback) {

**this**.client.get(String(sessionId), (err, data) => {

callback(err, JSON.parse(data));

});

}

RedisStore.prototype.set = **function** (sessionId, session, callback) {

**this**.client.set(sessionId, JSON.stringify(session), callback);

}

RedisStore.prototype.destroy = **function** (sessionId, callback) {

**this**.client.del(sessionId, callback);

}

**return** RedisStore;

}

module.exports = createRedisStore;//FileStore