1. 什么是中间件？

中间件就是在接收请求之后返回响应之前要执行的函数，而这些函数有一定的业务处理能力。

1. 使用中间件语法： app.use([路径],中间件函数)
2. 使用中间件**注意事项**：
3. 使用中间件要注意代码的存放位置，通常使用中间件的代码放在所有路由的最前面
4. 当给next()方法传参时，会查找含有err、req、res、next四个参数的中间件

const express = require('express');

const app = express();

app.listen(3000, () => {

    console.log('server port 3000');

});

//设计路由：   app级别路由写法

// app.use((req, res) => {

//     console.log('use....');

// });

app.get('/teacher', (req, res) => {

    // res.writeHead(200, 'ok', { "content-type": "text/html;charset=utf-8" });

    res.end('这是老师相关信息');

});

//自定义中间件

function demo(req, res, next) {

    console.log('use demo.');

    res.writeHead(200, 'ok', { "content-type": "text/html;charset=utf-8" });

    // next(); //查找并执行对应的路由

    //注意：当给next()方法传参时，会查找含有err、req、res、next四个参数的中间件

    let y = test();

    console.log(y, 8888);

    if (y) {

        next();

    } else {

        next('error message...');

    }

}

function test() {

    return 'abc';

}

app.use(demo);

app.get('/student', (req, res) => {

    res.end('学生信息');

});

// app.get('/', (req, res) => {

//     res.end('首页');

// });

app.all('\*', (req, res) => {

    res.end('404错误');

});

app.use((err, req, res, next) => {

    console.log(err, 6666);

    res.end(err);

});

1. 使用express内置中间件：
2. 什么是express的静态资源托管：

就是让用户通过我们的web服务器能访问到静态资源文件(比如：.js、、css、图片文件)

1. Express静态资源托管语法：使用内置中间件express.static()

express.static('要托管的静态资源目录')

中文文档地址：

<https://www.expressjs.com.cn/en/4x/api.html#express.static>

1. 使用express内置中间件 express.urlencoded()接收post提交的参数：

express.urlencoded([options])

中文文档地址：

<https://www.expressjs.com.cn/en/4x/api.html#express.urlencoded>

const express = require('express');

const path = require('path');

const app = express();

app.listen(4000, () => {

    console.log('server port 4000');

});

//使用express.static()内置中间件来实现静态资源托管：

// app.use('/image', express.static(path.resolve(\_\_dirname, 'image')));

// app.use('/css', express.static(path.resolve(\_\_dirname, 'css')));

app.use(express.static(path.resolve(\_\_dirname, 'public')));

//使用内置中间件express.urlencoded()接收post提交的参数：

//解析表单编码格式为enctype="application/x-www-form-urlencoded"的表单

app.use(express.urlencoded({ extended: true }));

//设计路由：

//显示注册界面

app.get('/', (req, res) => {

    res.sendFile(path.resolve(\_\_dirname, 'reg.html'));

});

//处理注册

app.post('/loginact', (req, res) => {

    let { usr, pwd, repwd } = req.body;

    console.log(usr, pwd, repwd);

    res.send(req.body);

});

reg.html文件内容如下：

<!DOCTYPE html>

<html lang="en">

<head>

    <meta charset="UTF-8">

    <meta name="viewport" content="width=device-width, initial-scale=1.0">

    <title>Document</title>

    <link rel="stylesheet" href="css/common.css">

</head>

<body>

    <h3>注册用户</h3>

    <div><img src="image/3333.jpg"></div>

    <form action="/loginact" method="post">

        帐号：<input type="text" name="usr"><br> 密码：

        <input type="password" name="pwd"><br> 密码：

        <input type="password" name="repwd"><br>

        <input type="submit" value="注册"><br>

    </form>

</body>

</html>

1. 为什么要使用cookie或session?

因为http协议是状态协议(没有记忆功能)，当我们使用了cookie或session这些技术后就可以让我们的系统有记忆功能。

1. 设置cookie: res.cookie(‘cookie名称’,’cookie值’[,选项])

**常用选项**有：

httpOnly:为true则只能服务器访问cookie、客户端不能访问

maxAge:设置cookie有效时间，单位为毫秒

path:设置cookie只能在某个路径中使用

domain:设置cookie只能在某个域名中使用

1. 获取cookie：使用第三方的cookie-parser中间件获取cookie
2. 安装cookie-parser: npm i cookie-parser

1. 使用中间件cookie-parser:

中文文档地址：

<https://www.expressjs.com.cn/en/resources/middleware/cookie-parser.html>

var cookieParser = require('cookie-parser')

app.use(cookieParser())

1. express操作cookie的代码如下：

const path = require('path');

const express = require('express');

var cookieParser = require('cookie-parser');

const app = express();

app.listen(4000, () => {

    console.log('server port 4000');

});

//使用中间件cookie-parser获取cookie:

app.use(cookieParser());

//设置cookie :

//res.cookie('cookie名称','cookie的值'[,选项]);

app.get('/setcookie', (req, res) => {

    res.cookie('TST', 666, { httpOnly: true });

    //maxAge:设置cookie的有效时间，时间单位毫秒

    res.cookie('USR', 'Tom', { maxAge: 60 \* 1000, });

    res.cookie('AGE', 22, { httponly: true, path: "/cookies" });

    res.cookie('UID', 123, { domain: 'www.qq.com' });

    res.send("<script>alert('cookie设置成功');location.href='/getcookie';</script>");

});

//获取cookie:要使用第三方的中间件cookie-parser

app.get('/getcookie', (req, res) => {

    console.log(req.cookies);

    res.send(req.cookies);

});

//获取cookie

app.get('/cookies', (req, res) => {

    console.log(req.cookies);

    res.send(`获取cookie:`);

});

1. express操作session要使用第三方中间件cookie-session:
2. 安装cookie-session: npm i cookie-session
3. 使用cookie-session:

中文文档地址：

<https://www.expressjs.com.cn/en/resources/middleware/cookie-session.html>

const cookieSession = require('cookie-session');

//使用中间件cookie-session操作session:

app.use(cookieSession({

    name: 'cursession', //cookie名称

    keys: ['123^%$abd', '\*(&^%#876'] //密钥

}));

1. express操作session，如下：

const path = require('path');

const express = require('express');

const cookieSession = require('cookie-session');

const app = express();

app.listen(4000, () => {

    console.log('server port 4000');

});

//使用中间件cookie-session操作session:

app.use(cookieSession({

    name: 'cursession', //cookie名称

    keys: ['123^%$abd', '\*(&^%#876'] //密钥

    // Cookie Options

    //maxAge: 24 \* 60 \* 60 \* 1000 // 24 hours

}));

//设计路由

//设置session  :req.session.session名称 = 值

app.get('/setsession', (req, res) => {

    req.session.UID = 21;

    req.session.FENSHU = [89, 90];

    res.send("<script>alert('session设置成功');location.href='/getsession';</script>");

});

//获取session

app.get('/getsession', (req, res) => {

    console.log(req.session);

    let { UID, FENSHU } = req.session;

    res.send(req.session);

});

1. Cookie与Session的区别：

(1)、Session 会在浏览器关闭之后失效，Cookie 则可以在理论上永久有效【默认有效时间为整个会话期间】。

(2)、Cookie 数据存放在客户的浏览器上，

Session 数据存放在服务器上。

(3)、Cookie 不安全，黑客可以分析本地的 Cookie

并进行 Cookie 欺骗。而 Session 由于保存在远程

服务器上，相对安全。所以考虑到安全性时，

重要的信息不要从 Cookie 中存取而是应该存在 Session 中。

(4)、Cookie有大小限制，一般是4KB。

域名20-50个以内，Session 则没有这方面的限制。

(5)、浏览器的设置可能禁用 Cookie，

这时所有关于 Cookie 的应用都将失败，但是

Session 却永远不会有这个问题

1. session与cookie的联系:

Session 依赖 cookie，因为 sessionId[唯一的] 存在客户端cookie中。