**全栈之路：Node.js + express 项目中常用的中间件**

## 注意:对于新版本的nodejs来说CommonJS语法有点过时了,建议使用ESModule中的import语法,需要在package.json里面添加type:”module”

这篇文章介绍了 Node.js + express 项目中常用的中间件，包括 dotenv 加载环境变量、chalk 改变输出样式、mount-routes 自动挂载路由、cors 解决跨域、body-parser 处理 post 请求数据、express-jwt 和 jsonwebtoken 处理用户信息、express-swagger 生成文档、crypto-js 加密算法、express-session 处理用户状态等，还列举了各中间件的安装、用法及相关代码示例。

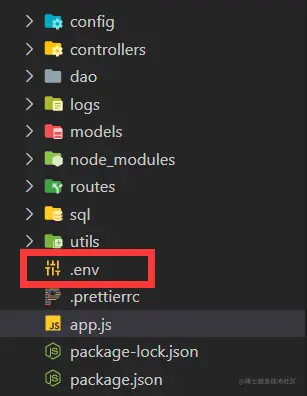
关联问题:如何优化dotenv配置cors如何处理特殊请求express-jwt使用注意啥

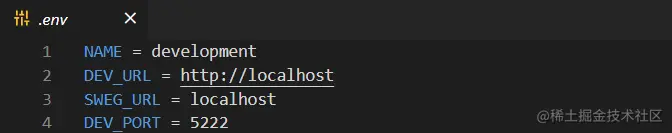
**1. dotenv**

dotenv 用于按需加载不同的环境变量文件，用法如下：

npm install dotenv

在项目根目录新建 .env 文件





在 app.js 文件中使用：process.env.DEV\_PORT，默认读取项目根目录下的.env文件

js

代码解读

复制代码

**const** express = require('express')

**const** dotenv = require('dotenv')

dotenv.**config**()

**const** app = **express**()

app.**listen**(process.env.DEV\_PORT, () => {

console.**log**(`项目启动成功: ${process.env.DEV\_URL}:${process.env.DEV\_PORT}`)

})

如果包含环境变量的文件位于其他位置，请指定自定义路径：

js

代码解读

复制代码

**const** dotenv = require('dotenv')

dotenv.**config**({ path: '/custom/path/to/.env' })

**2. chalk**

chalk 用于改变console输出的样式，用法如下：

js

代码解读

复制代码

npm install chalk

js

代码解读

复制代码

**const** chalk = require("chalk")

**log**(chalk.**red**("sea")) // 字体颜色

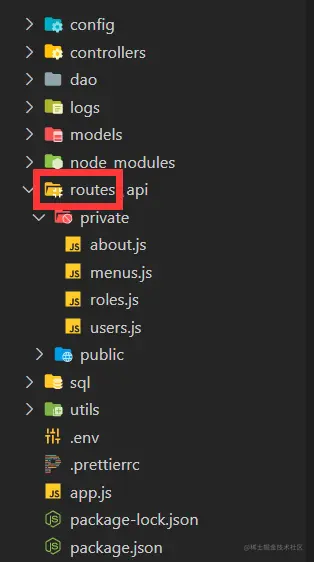
**log**(chalk.**bgGreen**("Forest")) // 背景色

**log**(chalk.bold.underline.bgYellow.**red**("WARNING!"))



**3. mount-routes**

mount-routes 可以自动挂载 routes 目录的所有路由，以文件名称作为路由的根，也可以指定具体的路径（使用第二个参数）



npm install mount-routes

**const** express = require('express')

**const** mount = require('mount-routes') // 路由加载

**const** app = **express**()

// 1.简单用法

**mount**(app)

// 2.带路径的用法

// 可以打印出路由表，true代表展示路由表在打印台

**mount**(app, path.**join**(process.**cwd**(), '/routes'), **true**)

**4. cors**

cors 主要用于解决跨域问题，原理：CORS 中间件配置在服务端，由一系列 HTTP 响应头组成，这个响应头可以决定浏览器是否阻止前端 JS 代码跨域获取资源，当接口服务器配置了 CORS 中间件之后，发起请求时就会自动配置相应的请求头，进而解除浏览器的跨域访问限制。

当浏览器进行跨域请求的时候，会在请求里添加头部 origin，表明自己协议，主机，端口。当服务器收到这个客户端发送的请求之后，如果需要允许能够访问，就需要添加头部信息 Access-Control-Arrow-Origin 到响应里面，浏览器收到传回来的这个头部信息就知道能不能进行跨域请求了。

js

代码解读

复制代码

npm install cors

js

代码解读

复制代码

**const** express = require('express')

**const** cors = require('cors')

**const** app = **express**()

app.**use**(**cors**())

app.**all**('/api/\*', **function** (req, res, next) {

res.**header**('Access-Control-Allow-Origin', '\*') // 配置跨域

res.**header**('Access-Control-Allow-Methods', 'PUT,POST,GET,DELETE,OPTIONS') // 允许请求的方法

res.**header**('Access-Control-Allow-Headers', 'X-Requested-With, token')

res.**header**('Access-Control-Allow-Headers', 'X-Requested-With, Authorization')

res.**header**('Content-Type', 'application/json;charset=UTF-8')

res.**header**('Access-Control-Allow-Headers', 'Content-Type,Content-Length, Authorization, Accept,X-Requested-With')

**if** (req.method == 'OPTIONS') res.**send**(200)

/\*让options请求快速返回\*/ **else** **next**()

})

**5. body-parser**

body-parser 用于处理 post 请求提交的数据，把数据保存在 req.body 中，以一个对象的形式提供给服务器，方便进行后续的处理。由于无论用户提交什么都会接受，所以需要在使用数据前进行验证来提高安全性。

此中间件已经被 express 集成，无需调用安装 body-parser，可以直接采用 express.json() 和 express.urlencoded() 实现相同功能，东西都是一样的，所以这里还是使用 body-parser 来介绍，下面我们来看一下 body-parser 常见的 API：

**（1）bodyParser.json([options])**：解析并返回 json格式的数据，只有 content-type: application/json 才进入这个中间件解析处理



**（2）bodyParser.urlencoded([options])**：表单 post 提交、axios、fetch 等库的 post 请求都需要这个中间件进行解析，返回json的格式数据，当请求的数据类型是application/x-www-form-urlencoded时才会进入这个中间件进行处理。



js

代码解读

复制代码

npm install body-parser

js

代码解读

复制代码

**const** express = require('express')

**const** bodyParser = require('body-parser') // 对http请求体进行解析

**const** app = **express**()

//处理请求参数解析

app.**use**(bodyParser.**json**())

app.**use**(bodyParser.**urlencoded**({ extended: **false** }))

**6. express-jwt + jsonwebtoken**

jsonwebtoken：生成 JWT 字符串（在服务器中将用户的信息转成 JWT 字符串） express-jwt：将 JWT 字符串解析还原成 JSON 对象（使用客户端保留的 JWT 加密字段，还原成真正的用户信息）

js

代码解读

复制代码

npm install express-jwt

npm install jsonwebtoken

**（1）定义 secret 密钥**：为了保证 JWT 字符串的安全性，防止 JWT 字符串传输的过程中被人破解，我们需要专门定义一个用于加密和解密的 secret 密钥。当生成 JWT 字符串的时候，需要使用 secret 密钥对用户的信息进行加密，最终得到加密好的 JWT 字符串；当把 JWT 字符串解析还原成 JSON 对象的时候，需要对 secret 密钥进行解密（使用和前面加密相同的密钥），密钥定义成字符串即可：



js

SIGN\_KEY = 'gLR+JUuKR/R5KrA1gr4ukg=='

**（2）登录成功后生成JWT字符串--生成Token**：

js

**const** jwt = require('jsonwebtoken')

exports.login = **function** (pm, cb) {

//登录逻辑

**Users**.**findOne**({where: {username: pm.username,}}).**then**(data => {

//判断用户是否正常

**if** (!data.state) {

**cb**(**null**, '账户已被禁用！请联系管理员')

logger.**error**('账户已被禁用！请联系管理员')

**return**;

}

**if** (pm.password === data.password) {

// 生成token

**let** token = 'Bearer ' + jwt.**sign**(

{

username: pm.username,

password: pm.password,

admin: **true**

},

process.env["SIGN\_KEY"],

{

expiresIn: 3600 \* 24 \* 3 //3天

// expiresIn: 30 //30s

}

)

**let** userInfo = data

//登录成功

**cb**({userInfo, token})

**return**

}

logger.**error**('密码错误')

**cb**(**null**, '密码错误！')

}).**catch**(err => {

logger.**error**(**JSON**.**stringify**(err))

**cb**(**null**, '用户不存在！请联系管理员添加')

})

}

**（3）将JWT字符串还原成为JSON对象--解析Token**：

js

**const** expressJwt = require('express-jwt')

/\*\*

\* token验证函数

\*

\* **@param** {**[type]**} req 请求对象

\* **@param** {**[type]**} res 响应对象

\* **@param** {**Function**} next 传递事件函数

\*/

exports.tokenAuth =**expressJwt**({

secret: process.env["SIGN\_KEY"],

algorithms: ['HS256'],

credentialsRequired: **true**, //对没有携带token的 接口不抛出错误

})

**（4）捕获解析JWT失败后产生的错误——错误中间件在最后 进行捕获错误**：当使用express-jwt 解析 Token 字符串时，如果客户端发送过来的 Token 字符串过期或不合法，会产生一个解析失败的错误，影响项目的正常运行，可以通过 Express 的错误中间件，捕获这个错误并进行相关的处理。

js

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

**if** (err.name === 'UnauthorizedError') {

logger.**error**(`${req.method} ${req.baseUrl + req.path} \*\*\* 响应：${JSON.stringify({

data: **null**,

code: err.status || 401,

message: err.message || 'token错误'

})}`);

res.**status**(401).**send**({data: **null**, code: err.status || 401, message: err.message || 'token错误'})

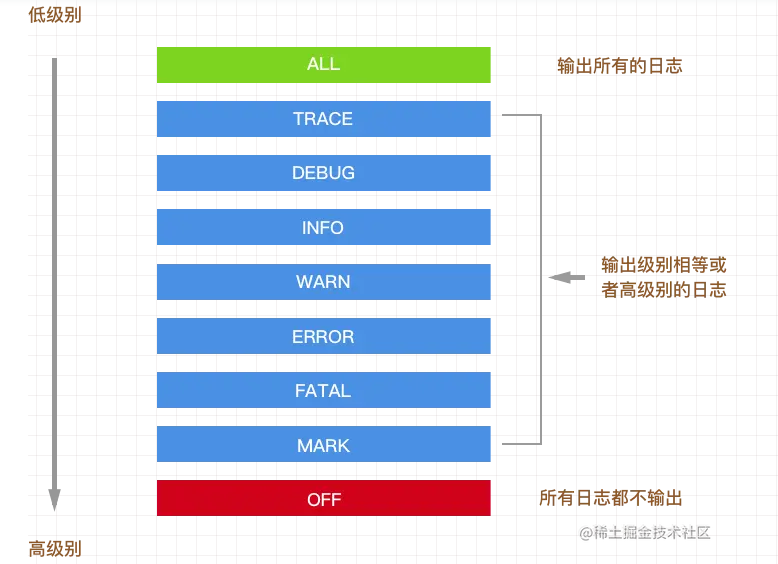
}

})

**7. log4js**

log4js 是一种 Node 日志管理工具，可以将自定义格式的日志输出到各种渠道。对于控制台的日志输出可以呈现彩色日志，对于文件方式的日志输出，可以根据文件大小或者日期进行日志切割。我们先来熟悉一下几个 log4js 中的概念。

**（1）Level**：日志的分级，更好地为展示日志（不同级别的日志在控制台中采用不同的颜色，比如 error 通常是红色的）



**（2）catetory**：日志的类型，在通过 getLogger 获取 Logger 实例时，唯一可以传的一个参数就是 loggerCategory（如'example'），通过这个参数来指定 Logger 实例属于哪个类别。

js

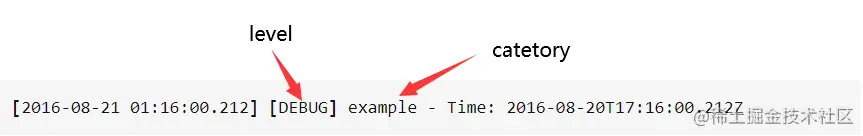
代码解读

复制代码

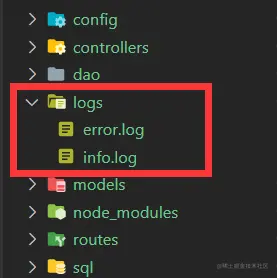
**var** log4js = require('log4js')

**var** logger = log4js.**getLogger**('example')

logger.**debug**("Time:", **new** **Date**())



**（3）appenders**：日志输出到哪里，默认将日志都输出到了控制台，在项目的根目录新建 logs 文件，作为日志的出口。



js

代码解读

复制代码

**var** log4js = require('log4js');

log4js.**configure**({

appenders: {

console: { type: 'console' },

info: {

type: 'file',

filename: 'logs/info.log'

},

error: {

type: 'file',

filename: 'logs/error.log'

}

},

categories: {

default: {

appenders: [ 'console','info' ],

level: 'debug'

},

info: {

appenders: ['info'],

level: 'info'

},

error: {

appenders: [ 'error', 'console' ],

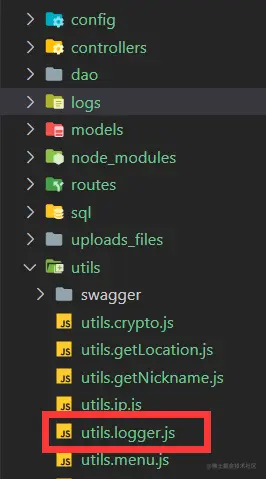
level: 'error'

}

}

});

**封装logger.js工具函数**：在项目根目录的 utils 文件夹中新建 utils.logger.js 文件



js

npm install log4js

**var** log4js = require('log4js');

log4js.**configure**({

appenders: {

console: { type: 'console' },

info: {

type: 'file',

filename: 'logs/info.log'

},

error: {

type: 'file',

filename: 'logs/error.log'

}

},

categories: {

default: {

appenders: [ 'console','info' ],

level: 'debug'

},

info: {

appenders: ['info'],

level: 'info'

},

error: {

appenders: [ 'error', 'console' ],

level: 'error'

}

}

});

/\*\*

\* 日志输出 level为bug

\* **@param** { **string** } content

\*/

exports.debug = ( content ) => {

**let** logger = log4js.**getLogger**('debug')

logger.level = 'debug'

logger.**debug**(content)

}

/\*\*

\* 日志输出 level为info

\* **@param** { **string** } content

\*/

exports.info = ( content ) => {

**let** logger = log4js.**getLogger**('info')

logger.level = 'info'

logger.**info**(content)

}

/\*\*

\* 日志输出 level为error

\* **@param** { **string** } content

\*/

exports.error = ( content ) => {

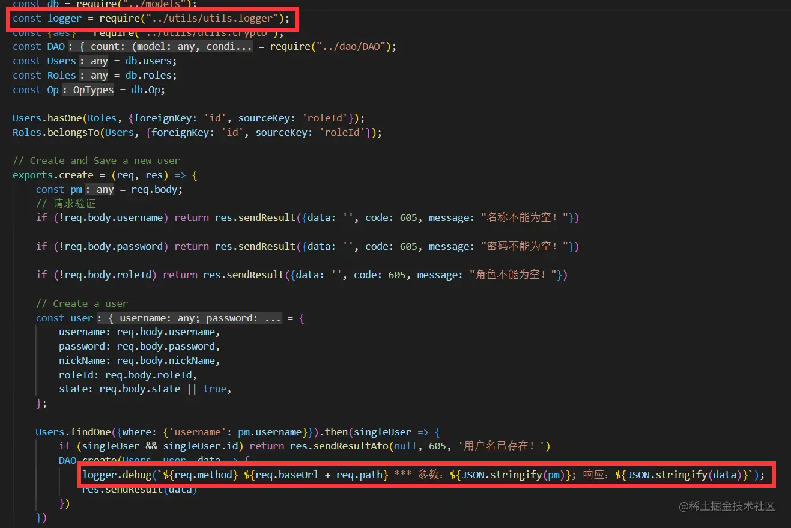
**let** logger = log4js.**getLogger**('error')

logger.level = 'error'

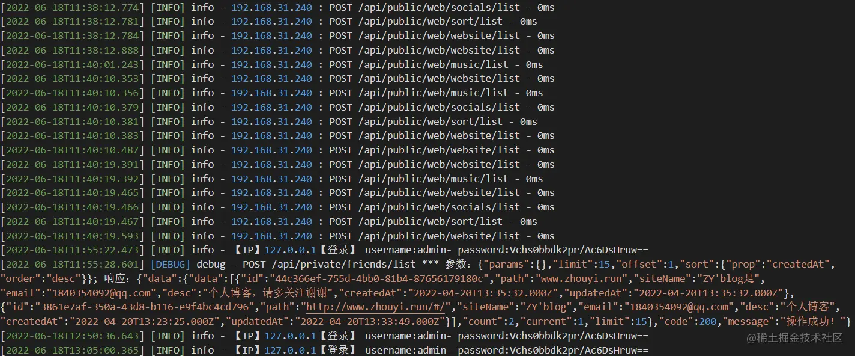
logger.**error**(content)

}

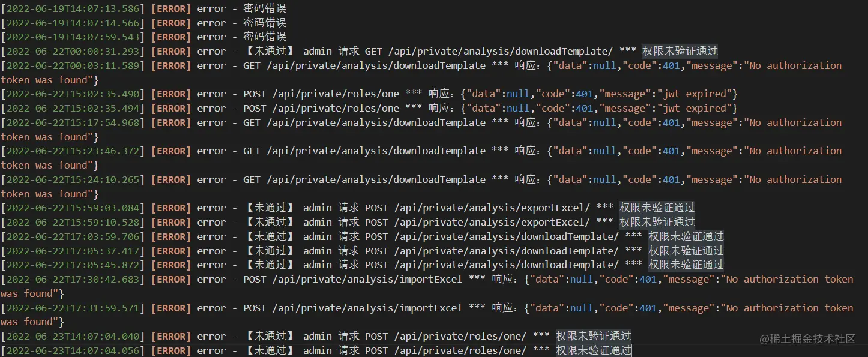
**在项目中使用**



**logs/info.log**



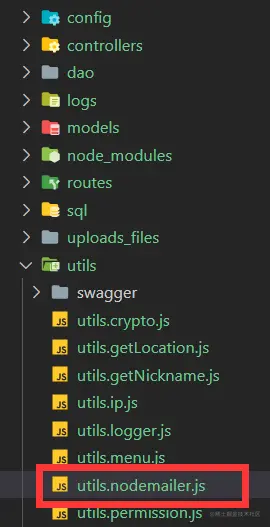
**logs/error.log**



**8. nodemailer**

Nodemailer 是一个简单易用的 Node.JS 邮件发送模块（通过 SMTP，sendmail，或者 Amazon SES），官方提供了一套固定的模板。

**封装nodemailer.js工具函数**



js

代码解读

复制代码

**const** nodemailer = require("nodemailer");

/\*\*

\* 邮箱发送

\*

\* **@param** {**Object**} pm 对方信息

\*/

exports.sendMailer = (pm) => {

**return** **new** **Promise**((resolve, reject) => {

// 创建Nodemailer传输器 SMTP 或者 其他 运输机制

**let** transporter = nodemailer.**createTransport**(

{

service: 'QQ', // 使用内置传输发送邮件 查看支持列表：https://nodemailer.com/smtp/well-known/

port: 465, // SMTP 端口

secureConnection: **true**, // 使用 SSL

auth: {

user: '1840354092@qq.com', // 发送方邮箱的账号

pass: '\*\*\*\*\*\*', // 邮箱授权密码

}

}

);

// 定义transport对象并发送邮件

transporter.**sendMail**({

from: `"MG'Blog" <1840354092@qq.com>`, // 发送方邮箱的账号

to: pm.email, // 邮箱接受者的账号

subject: "MG'Blog", // Subject line

// text: '"MG'Blog 👻"', // 文本内容

html: `<img src="https://encrypted-tbn0.gstatic.com/images?q=tbn:ANd9GcSy\_130iiorSSjF1RWgNBX7qy3evKv2HKsH0g&usqp=CAU">

<p style="text-indent: 2em;">您好！ "${pm.email}" </p>

<p style="text-indent: 2em;">您在<a href="http://www.zhouyi.run/#/">MG'Blog</a>上的留言博主已收到🎈 感谢您的支持！</p>

<p >✨回复内容：</p>

<p style="text-indent: 2em;">${pm.content}</p>

<p style="text-indent: 2em;">祝您工作顺利，心想事成🎉🎉🎉</p>

<p style="text-align: right;">—— <a href="http://www.zhouyi.run/#/">MG'Blog</a></p>`,

}, (error, info) => {

**if** (error) {

**reject**(error)

}

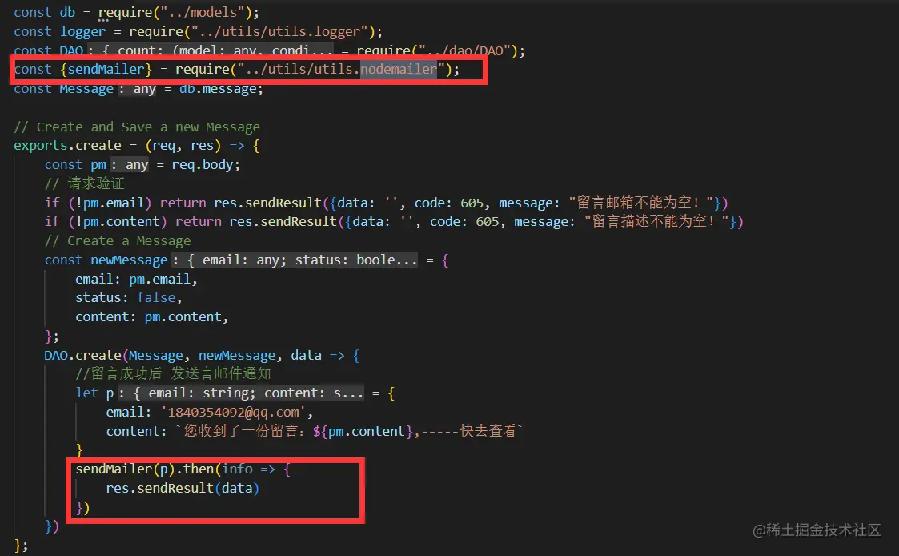
**resolve**(info)

});

})

}

**在项目中使用**

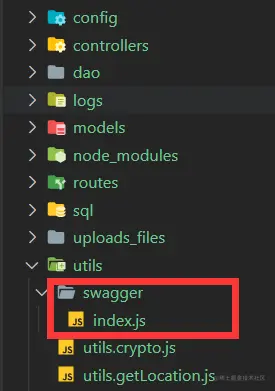


**9. multer**

multer 是一个 node.js 中间件，用于处理 multipart/form-data 类型的表单数据，它主要用于上传文件。

**10. express-swagger-generator**

express-swagger-generator 可以自动生成 api 文档



js

**const** options = {

swaggerDefinition: {

info: {

title: 'mg-api',

version: '1.0.0',

description: `芒果快熟’接口api`

},

host: `${process.env.SWEG\_URL}:${process.env.DEV\_PORT}`,

basePath: '/',

produces: ['application/json', 'application/xml'],

schemes: ['http', 'https'],

securityDefinitions: {

JWT: {

type: 'apiKey',

in: 'header',

name: 'Authorization',

description: ''

}

}

},

route: {

url: '/swagger',//打开swagger文档页面地址

docs: '/swagger.json' //swagger文件 api

},

basedir: \_\_dirname, //app absolute path

files: [ //在那个文件夹下面收集注释

'../../routes/api/private/\*.js',

'../../routes/api/public/\*\*/\*.js',

]

}

module.exports = options

app.js

js

**const** express = require('express')

**const** app = **express**()

**const** expressSwagger = require('express-swagger-generator')(app)

**const** options = require('./utils/swagger') //配置信息

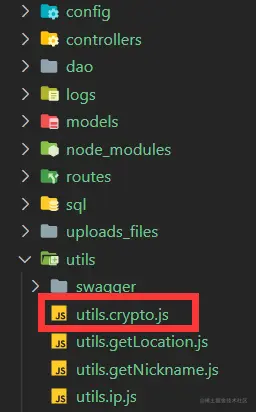
**expressSwagger**(options)

**11. crypto-js**

crypto-js 是谷歌开发的一个纯 JavaScript 的加密算法类库，可以非常方便的在前端进行其所支持的加解密操作。目前已支持的算法包括：

* MD5
* SHA-1
* SHA-256
* AES
* HMAC
  + HMAC-MD5
  + HMAC-SHA1
  + HMAC-SHA256

**封装加密/解密工具函数**：



js

npm install crypto-js

/\*\*

\* 通过crypto-js实现 加解密工具

\* AES、HASH(MD5、SHA256)、base64

\* **@author**: hzd

\*/

**const** **CryptoJS** = require('crypto-js') // 加密

**let** KP = {

key: '90268d3dc304f5f3', //process.env.VUE\_APP\_AES\_KEY, // 秘钥 16\*n:

iv: 'b894f52b46104ab2', //process.env.VUE\_APP\_AES\_IV // 偏移量

}

**function** **getAesString**(data, key, iv) {

// 加密

key = **CryptoJS**.enc.Utf8.**parse**(key)

// alert(key）;

iv = **CryptoJS**.enc.Utf8.**parse**(iv)

**let** encrypted = **CryptoJS**.AES.**encrypt**(data, key, {

iv,

mode: **CryptoJS**.mode.CBC,

padding: **CryptoJS**.pad.Pkcs7,

})

**return** encrypted.**toString**() // 返回的是base64格式的密文

}

**function** **getDAesString**(encrypted, key, iv) {

// 解密

key = **CryptoJS**.enc.Utf8.**parse**(key)

iv = **CryptoJS**.enc.Utf8.**parse**(iv)

**let** decrypted = **CryptoJS**.AES.**decrypt**(encrypted, key, {

iv,

mode: **CryptoJS**.mode.CBC,

padding: **CryptoJS**.pad.Pkcs7,

})

**return** decrypted.**toString**(**CryptoJS**.enc.Utf8)

}

// AES 对称秘钥加密

**const** aes = {

//加密

en: data => **getAesString**(data, KP.key, KP.iv),

//解密

de: data => **getDAesString**(data, KP.key, KP.iv),

}

// BASE64

**const** base64 = {

en: data => **CryptoJS**.enc.Base64.**stringify**(**CryptoJS**.enc.Utf8.**parse**(data)),

de: data => **CryptoJS**.enc.Base64.**parse**(data).**toString**(**CryptoJS**.enc.Utf8),

}

// SHA256

**const** **sha256** = data => {

**return** **CryptoJS**.**SHA256**(data).**toString**()

}

// MD5 '12654987' --> 60f6c0f4991073bdb49b56b3d38f2645

**const** **md5** = data => {

**return** **CryptoJS**.**MD5**(data).**toString**()

}

/\*\*

\* 签名

\* **@param** token 身份令牌

\* **@param** timestamp 签名时间戳

\* **@param** data 签名数据

\*/

**const** **sign** = (token, timestamp, data) => {

// 签名格式： timestamp + token + data(字典升序)

**let** ret = []

**for** (**let** it **in** data) {

**let** val = data[it]

**if** (

**typeof** val === 'object' && //

(!(val **instanceof** **Array**) || (val.length > 0 && **typeof** val[0] === 'object'))

) {

val = **JSON**.**stringify**(val)

}

ret.**push**(it + val)

}

// 字典升序

ret.**sort**()

**let** signsrc = timestamp + token + ret.**join**('')

**return** **md5**(signsrc)

}

module.exports = {

aes,

md5,

sha256,

base64,

sign,

}

**12. express-session**

express-session 是针对 express 框架提供的一套 session 扩展，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。express-session 这个中间件替代 cookie-parser 和 cookie-session 中间件成为处理用户状态的首选。

**session 的工作流程**：当浏览器访问服务器时，服务器创建一个 session 对象(该对象有一个唯一的id号 sessionId)，服务器会将 sessionId 以 cookie 的方式(set-cookie消息头)发送给浏览器，浏览器会将sessionId 保存到内存；当浏览器再次访问服务器时，会将 sessionId 发送给服务器，服务器依据sessionId 就可找到之前创建的 session 对象。

**express-session的使用**

js

npm install express-session

**const** session = require('express-session')

**const** express = require('express')

**const** fs = require('fs')

**const** app = **express**()

app.**use**(**session**({

secret: 'keyboard cat',

resave: **false**,

saveUninitialized: **true**,

cookie: { secure: **false**, maxAge: 800000 },

name: 'ivan'

}))

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

**const** html = fs.**readFileSync**('./views/index.html', 'utf-8')

**const** session = req.session // 获得session

session['key'] = 'value' // 设置session

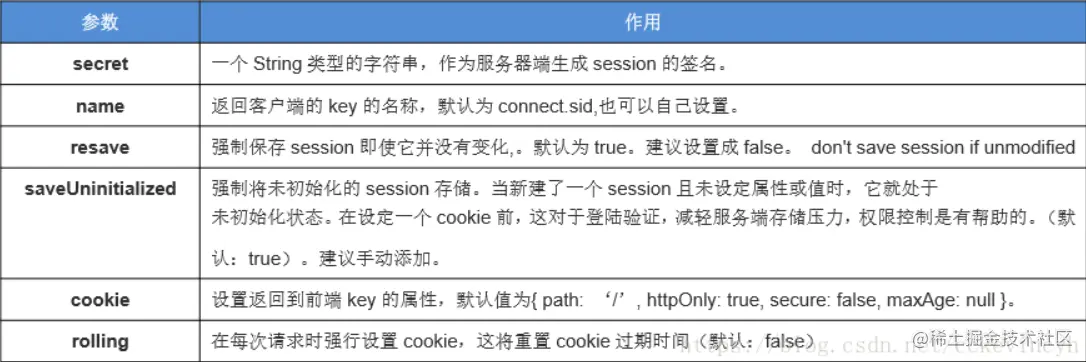
res.**setHeader**('set-cookies', session['key']) // 保存cookie在headers中

res.**end**(html) // 发送给客户端

})

app.**listen**(3000)

**express-session的常用参数**



*[Node.js:express中间件](https://juejin.cn/post/7067830987937284109?from=search-suggest" \t "_blank)*[2](https://juejin.cn/post/7067830987937284109?from=search-suggest" \t "_blank)

*[Node.js全栈](https://juejin.cn/post/7374719076695932955?from=search-suggest" \t "_blank)*[基础：从菜鸟到大神的极速飞跃](https://juejin.cn/post/7374719076695932955?from=search-suggest" \t "_blank)*[之路](https://juejin.cn/post/7374719076695932955?from=search-suggest" \t "_blank)*[！](https://juejin.cn/post/7374719076695932955?from=search-suggest" \t "_blank)

[一名](https://juejin.cn/post/6844903428779425800?from=search-suggest" \t "_blank)*[全栈](https://juejin.cn/post/6844903428779425800?from=search-suggest" \t "_blank)*[工程师](https://juejin.cn/post/6844903428779425800?from=search-suggest" \t "_blank)*[Node.js](https://juejin.cn/post/6844903428779425800?from=search-suggest" \t "_blank)**[之路](https://juejin.cn/post/6844903428779425800?from=search-suggest" \t "_blank)*

[后端实战手把手教你写文](https://juejin.cn/post/7119407620623974430?from=search-suggest" \t "_blank)*[件](https://juejin.cn/post/7119407620623974430?from=search-suggest" \t "_blank)*[上传接口：如何使用](https://juejin.cn/post/7119407620623974430?from=search-suggest" \t "_blank)*[Node.js](https://juejin.cn/post/7119407620623974430?from=search-suggest" \t "_blank)**[+](https://juejin.cn/post/7119407620623974430?from=search-suggest" \t "_blank)*[MongoDB 开发 RESTful API 接口（](https://juejin.cn/post/7119407620623974430?from=search-suggest" \t "_blank)*[Node.js](https://juejin.cn/post/7119407620623974430?from=search-suggest" \t "_blank)**[+](https://juejin.cn/post/7119407620623974430?from=search-suggest" \t "_blank)**[Express](https://juejin.cn/post/7119407620623974430?from=search-suggest" \t "_blank)**[+](https://juejin.cn/post/7119407620623974430?from=search-suggest" \t "_blank)*[Mo](https://juejin.cn/post/7119407620623974430?from=search-suggest" \t "_blank)

*[Node.js](https://juejin.cn/post/6856295843243229197?from=search-suggest" \t "_blank)*[从零开发Web Server博客](https://juejin.cn/post/6856295843243229197?from=search-suggest" \t "_blank)*[项目](https://juejin.cn/post/6856295843243229197?from=search-suggest" \t "_blank)*[前端晋升](https://juejin.cn/post/6856295843243229197?from=search-suggest" \t "_blank)*[全栈](https://juejin.cn/post/6856295843243229197?from=search-suggest" \t "_blank)*[工程师必备](https://juejin.cn/post/6856295843243229197?from=search-suggest" \t "_blank)

*[Node.js+Express](https://juejin.cn/post/7066748541917462542?from=search-suggest" \t "_blank)*

[盘点](https://juejin.cn/post/7348647534393475126?from=search-suggest" \t "_blank)*[node常用](https://juejin.cn/post/7348647534393475126?from=search-suggest" \t "_blank)*[框架](https://juejin.cn/post/7348647534393475126?from=search-suggest" \t "_blank)

*[Node.js](https://juejin.cn/post/6854573208247468039?from=search-suggest" \t "_blank)*[从零开发Web Server博客](https://juejin.cn/post/6854573208247468039?from=search-suggest" \t "_blank)*[项目](https://juejin.cn/post/6854573208247468039?from=search-suggest" \t "_blank)*[前端晋升](https://juejin.cn/post/6854573208247468039?from=search-suggest" \t "_blank)*[全栈](https://juejin.cn/post/6854573208247468039?from=search-suggest" \t "_blank)*[工程师必备](https://juejin.cn/post/6854573208247468039?from=search-suggest" \t "_blank)

[后端实战教程:如何使用](https://juejin.cn/post/7119077094897745957?from=search-suggest" \t "_blank)*[Node.js](https://juejin.cn/post/7119077094897745957?from=search-suggest" \t "_blank)*[开发 RESTful API 接口（](https://juejin.cn/post/7119077094897745957?from=search-suggest" \t "_blank)*[Node.js](https://juejin.cn/post/7119077094897745957?from=search-suggest" \t "_blank)**[+](https://juejin.cn/post/7119077094897745957?from=search-suggest" \t "_blank)**[Express](https://juejin.cn/post/7119077094897745957?from=search-suggest" \t "_blank)**[+](https://juejin.cn/post/7119077094897745957?from=search-suggest" \t "_blank)*[Sequelize + MySQL）](https://juejin.cn/post/7119077094897745957?from=search-suggest" \t "_blank)

[10 款](https://juejin.cn/post/6914561878916071431?from=search-suggest" \t "_blank)*[Node.js](https://juejin.cn/post/6914561878916071431?from=search-suggest" \t "_blank)*[框架，可用于你的下一个](https://juejin.cn/post/6914561878916071431?from=search-suggest" \t "_blank)*[项目](https://juejin.cn/post/6914561878916071431?from=search-suggest" \t "_blank)*

 [**Express.js CORS跨域配置和静态访问服务**](https://juejin.cn/post/7222484074482040887)

[前端](https://juejin.cn/tag/%E5%89%8D%E7%AB%AF)[Node.js](https://juejin.cn/tag/Node.js)[Express](https://juejin.cn/tag/Express)

 [**nodejs-day03-中间件、路由**](https://juejin.cn/post/6970244141447905311)

[前端](https://juejin.cn/tag/%E5%89%8D%E7%AB%AF)

 [**学习Node.js：解析express body-parser中间件源码**](https://juejin.cn/post/7276257954298183732)

[JavaScript](https://juejin.cn/tag/JavaScript)[前端](https://juejin.cn/tag/%E5%89%8D%E7%AB%AF)

 [**计算机网络http|面试中🍕🍕常问的几种处理跨域的方法**](https://juejin.cn/post/7123855528975056903)

[HTTP](https://juejin.cn/tag/HTTP)

 [**ajax可以跨域吗?**](https://juejin.cn/post/7102057260070731789)

[前端](https://juejin.cn/tag/%E5%89%8D%E7%AB%AF)

 [**用express搭建 静态托管服务器及简易接口**](https://juejin.cn/post/7091983986829819911)

[Node.js](https://juejin.cn/tag/Node.js)

 [**环境变量和开发环境中的跨域处理**](https://juejin.cn/post/6976244649979543583)

[前端](https://juejin.cn/tag/%E5%89%8D%E7%AB%AF)

 [**Express中间件之入门**](https://juejin.cn/post/7350111452952838154)

[前端](https://juejin.cn/tag/%E5%89%8D%E7%AB%AF)[Node.js](https://juejin.cn/tag/Node.js)[Express](https://juejin.cn/tag/Express)

 [**Express 入门指南（详细）**](https://juejin.cn/post/7189164529430265912)

[Express](https://juejin.cn/tag/Express)[前端](https://juejin.cn/tag/%E5%89%8D%E7%AB%AF)[Node.js](https://juejin.cn/tag/Node.js)

 [**node 基础包 chalk、cross-spawn、yargs-parser、which、fs-extra、Buffer、net 学习**](https://juejin.cn/post/7427477721377161251)

[前端](https://juejin.cn/tag/%E5%89%8D%E7%AB%AF)[Node.js](https://juejin.cn/tag/Node.js)

 [**Express的基本使用**](https://juejin.cn/post/6997008381852532767)

[前端](https://juejin.cn/tag/%E5%89%8D%E7%AB%AF)

 [**前端node.js-get和post接口的简单教学**](https://juejin.cn/post/7091989460614643742)

[Node.js](https://juejin.cn/tag/Node.js)

 [**Node提供的web服务**](https://juejin.cn/post/7387601870296662070)

[前端](https://juejin.cn/tag/%E5%89%8D%E7%AB%AF)[后端](https://juejin.cn/tag/%E5%90%8E%E7%AB%AF)[Node.js](https://juejin.cn/tag/Node.js)

 [**vite中环境变量的使用与配置（非常详细）**](https://juejin.cn/post/7172012247852515335)

[掘金·日新计划](https://juejin.cn/tag/%E6%8E%98%E9%87%91%C2%B7%E6%97%A5%E6%96%B0%E8%AE%A1%E5%88%92)

 [**小白NestJS官网文档祥读之中间件**](https://juejin.cn/post/7348999610205896723)