Nodejs生成swagger后端API文档时：报错

Not all input has been taken into account at your final specification.

Here's the report:

Error in ./server.js :

YAMLSemanticError: Map keys must be unique; "properties" is repeated at line 17, column 13:

type: object

^^^^^^^^^^^^…

Server is running on http://localhost:3000

报错原因：swagger注释与其缩进没整对，另外就是swagger的注释所写的参数数量与js代码所需要的个数不一样（需要更新数据库的操作所需的变量）

--------------------------------------------------------------------------------------------------------------------------------

在export和require导入其他文件内容进来的时候不要形成闭环（就是不要A去导入B然后B又导入A的）这样会导致导入导出混乱，然后报错一大堆

在http响应时，若有静态资源和路由时，路由请求路径为/时，谁先匹配上就先响应谁（即谁在前就响应谁）

Eg:

const express = require( express");

//创建应用对象

const app = express();

//创建路由app.get('/，(req, res) => (

res.send(我才是首页~~~');

//静态资源中间件设骂

;app.use(express .static(dirname +'/public'))

//监听端口，启动服务

app.listen(3eee，() => f

console.log(服务已经启动，端口 3000

上述代码来看，发请求，但后面啥也不写，即请求http：//127.0.0.1：3000时，响应的是：我才是首页

在之前的express里，需要下载一个包：body-parser，这样可以解析请求体的数据

解析querystring：用法是：const urlencodeParser= bodyParser.urlencoded（{extened：false}）

解析json： const jsonParser = bodyParser.json（）

然后在路由里面嵌入中间件

Eg：app.post( /login', urlencodedParser, (req, res)=>

//获取 用户名 和 密码

console.log(req.body);

res.send("获取用户的数据)

)}

Ps:如果声明中间件没有起名字，直接是用的回调函数，那么路由中没有传中间函数名他也能执行中间件

在连接数据库的时候，一般要防止数据库断连，一般要弄一个检查数据库断了过后自动重连的机制，用数据库连接池来搞，涉及到连接之后要释放连接池。

详细代码：

const mysql = require("mysql");

const dbConfig = {

    host :"127.0.0.1",

    user :"root",

    password: "",

    // password :"",

    database: "competitionprotocol"

};

const pool = mysql.createPool({

    // 十次限制连接

    connectionLimit: 10,

    ...dbConfig

});

function connectToDatabase() {

    pool.getConnection((err, connection) => {

        if (err) {

            console.error('Database connection failed: ' + err.stack);

            // 连接失败，等待5秒后尝试重新连接

            setTimeout(connectToDatabase, 5000);

            return;

        }

        console.log('Connected to database');

        // 释放连接回连接池

        connection.release();

    });

}

// 处理连接丢失的事件

pool.on('error', (err) => {

    console.error('Database Pool Error: ' + err.stack);

    if (err.code === 'PROTOCOL\_CONNECTION\_LOST') {

        // 重新连接

        connectToDatabase();

    } else {

        throw err;

    }

});

function query(sql, values, callback) {

  pool.getConnection((err, connection) => {

      if (err) {

          console.error('Error acquiring connection: ' + err.stack);

          return callback(err);

      }

      connection.query(sql, values, (err, results) => {

          connection.release(); // 释放连接回连接池

          if (err) {

              console.error('Error executing query: ' + err.stack);

              return callback(err);

          }

          callback(null, results);

      });

  });

}

module.exports = {

  connectToDatabase,

  query

};

连接池本身是一个事件发射器对象，连接出错就会返回错误信息，本质是有一个类：名为EventEmitter

const EventEmitter = require('events');

// 创建一个新的事件发射器对象

const myEmitter = new EventEmitter();

// 注册事件监听器

myEmitter.on('myEvent', (arg) => {

console.log('Event triggered with argument:', arg);

});

// 触发事件

myEmitter.emit('myEvent', 'Hello, EventEmitter!');

//============服务器运行nodejs服务================================

用pm2工具:  
npm install -g pm2

pm2 start app.js // 启动

pm2 start app.js -i max //启动 使用所有CPU核心的集群

pm2 stop app.js // 停止

pm2 stop all // 停止所有

pm2 restart app.js // 重启

pm2 restart all // 重启所有

pm2 delete app.js // 关闭

pm2 delete <id or name> //删除

查看正在运行的pm2 : pm2 list

查看日志输出: pm2 logs app

默认情况下，PM2 会将日志存储在 ~/.pm2/logs/ 目录下

命令为:ls ~/.pm2/logs/