Express

1. 概念
    1. 基于Nodejs平台，快速开放极简的web开发框架。
    2. 本质：npm上第三方包，提供了快速创建web服务器的边界方法
    3. 作用：帮助我们快速创建web服务器或API接口的服务器
2. 基本的使用
    1. 安装
        1. 控制台进入指定目录
        2. 先创建一个服务器对象 npm init -y
        3. 然后安装npm i express因为express需要js对象，所以不用安装全局的包
    2. 利用Express创建Web服务
        1. 导入需要使用的express包
            1. const express=require('express')
        2. 创建web实例
            1. const app=express()
        3. 定义允许访问的地址（路由）
            1. 请求的方式
                1. pp.get("uri",(req,res)=>{res.send("hello world");});
                2. uri一开始是写/
            2. 向客户端响应数据
                1. 原先的输出：res.end()
                2. 现在的输出：res.send();
        4. 启动服务（监听端口）
            1. pp.listen(8000,()=>{})
        5. 控制台观察端口是否启动
            1. windows平台
                1. netstat -ano | findStr 8080
            2. Unix系平台（macOS、linux）
                1. netstat -tnpl | grep 8080(想要查找的端口号)
            3. 符号|
                1. 管道，变量修饰符，过滤器
    3. 访问调试 restful规范（对请求方式的约束）
        1. GET：查询请求类型
            1. 浏览器中直接输地址是GET请求
            2. 取全部的数据（列表）
            3. 取单个的数据（详情）

    2. POST：新增请求类型
        1. 新增是不带条件的
        2. 需要请求体的
    3. PUT：修改请求类型
        1. 修改是要条件的
        2. 修改条件的传递是通过地址栏传递的
        3. 修改数据的主体是通过请求体传递的
    4. DELETE：删除请求类型
        1. 删除需要条件的
        2. 条件通过地址栏传递
        3. 没有请求体
    5. all:匹配所有的请求类型（一般不会用的）
    6. 访问地址匹配规则
        1. 从上往下执行，需要匹配上“请求类型”+请求路径，才会走对应的回调函数
4. 获取query字符串
    1. get请求，获取地址栏带?的，这样才是对象
    2. 通过req.query获取对象
    3. 获取的结果是端口号后面那一堆?user:chenwei&gender:male
5. 动态路由参数传递
    1. 获取地址栏后面直接是参数的
    2. app.put('/user/:id') :id就是动态参数，因为你不知道用户是谁
    3. req.params.id获取动态参数 .id是匹配:id位置的参数
6. 静态资源托管（express让你一行搭建静态资源服务器）
    1. app.use('虚拟前缀',express.static("public")
    2. public为你放静态资源的文件夹
    3. 虚拟前缀可以帮我们迷惑别人，或者让自己明白静态资源的功能
3. 路由
    1. 概述：按键与服务的映射关系
        1. 路由指的是客户端发送的请求和服务端处理方法对应的关系
    2. 定义路由
        1. 请求的方法
        2. 请求的地址（uri）
        3. 对应的处理函数（响应）
    3. 路由模块化
        1. 将一个文件的路由规则拆分到若干个路由文件（js文件）
        2. 核心思想：能拆就拆（拆到不能拆为止，解耦，高内聚，低耦合）
        3. 步骤
            1. 创建独立js空白文件（路由模块化文件）（放到一个目录上）
            2. 在js文件中使用express.Router()方法创建路由模块对象

3. 使用路由对象完成路由规则对应的业务编写
4. 使用模块化导出（module.exports=router）
5. 在主入口文件中能够使用app.use方法来注册定义的路由模块

4. 中间件
    1. 概念：业务的中间环节
        1. express中，当一个请求到达服务器后，可以在给客户响应之前连续调用多个中间1. 件，来对本次请求和返回响应数据进行处理
    2. 中间件的本质就是一个函数
    3. 中间件的分类
        1. 依据发布商分类
        2. 内置中间件，express本身自带的
            1. express.json
                1. 作用：接收JSON格式提交的数据,绑定到req.body
                2. 兼容性问题：express>=4.16.0
                3. app.use(express.json())
                4. 接收完数据后，会将数据的对象形式挂载到req请求对象的body属性上
            2. express.urlencoded
                1. 作用：处理post表单数据
                2. 兼容性问题：express>=4.16.0
                3. app.use(express.urlencoded({extended:false}))
                    1. post表单数据一般不带方法的，所以要用extended:false
                    2. extended:false,表示要求在解析数据的时候使用query string库,会剔除传递过来的方法和对象
                    3. extended:true,表示要求在解析数据的时候使用qs库,不会剔除传递过来的方法和对象（默认）
                4. 其在接收完数据后，会将数据的对象形式挂载到req请求对象的body属性上
        3. 第三方中间件，第三方开发的
            1. body-parser接收post数据
            2. 步骤
                1. 安装第三方中间件body-parser npm i -S body-
                2. 应用文件中导入body-parser
                3. 通过中间件调用app.use(body.urlencoded({extended:false}))
                4. 在匹配的路由中通过req.body获取post数据
            3. express内置的express.urlencoded中间件，基于body-parser这个第三方中间件进一步封装出来的。
            4. 一般以前的express版本使用第三方中间件
        4. 自定义中间件，开发者自己编写的
            1. 本质定义一个处理请求的函数
                1. 参数必须有req,res,next

    2. next参数让中间件能够让流程向下执行下去，直到匹配路由，发送响应给客户端
        1. 匹配路由就是app.post,delete,put,get
        2. next也是一个函数，继续传递下去
    3. 也可以给request对象添加属性来进行中间件的数据向下传递
    4. 整个请求链路中，所有中间件与最终路由共用一份req和res
2. req.on()监听data和end事件
    1. data事件将buffer入栈
    2. end事件 调用Buffer中的连接方法，将字节数组连接起来 Buffer.concat(arr)
    3. req.on本质是一个异步请求方法，应该将next放到监听end事件中，不然数据没有挂到req.body就下一步了
    4. 意思是req.on和在同一层的其他方法是同时执行的
3. 引入query string库解析对象 let post=querystring.parse(buffer.toString());
4. 将解析后的对象绑定到req.body上，req.body=post;
5. 中间件封装成模块就是JSON对象形式，方便其他js文件调用自定义中间件
    1. 创建一个导出模块用的js文件
    2. 将自定义中间件导出
    3. module.exports=csBodyParse;
    4. 然后其他js文件app.use('./路径')
    5. ./当前目录下的
5. 错误类型中间件(写到路由后面)
    1. 异常中间件
        1. 作用：专门用来捕获整个项目发生的异常错误，从而防止项目异常崩溃的问题产生
        2. 在路由中try catch中throw new Error("");将错误改成新的自己定义的err
        3. 然后用异常中间件，将错误信息返回回去
        4. 格式：错误级别中间件的函数参数中，必须有四个形参，分别是(err,req,res,next)
            1. err参数的作用：包含错误的信息err.message
    2. 404中间件(与异常中间件的顺序无所谓，只要路由后面)
        1. 所有路由都匹配不上，就走这个中间件，其实就类似一个什么请求方式都能进的路由
        2. 作用：定义404的页面需要输出的效果
        3. 参数req,res,next res.status(404).send("404");返回状态码
6. 局部中间件
    1. 在路由的第二个参数那里写上[express.urlencoded({extended:false}),express.json()]
7. 依据使用层面分类
8. 应用级别中间件
9. pp.use 全局使用中间件 所有路由都生效
10. pp.请求方法('地址',[中间件],回调函数) 当前路由生效

        11. 路由级别中间件
        12. outer.use
5. cookie和session（会话控制）