中间件

中间件理解是一个过滤器，作用是为主要的业务逻辑服务

分为应用及中间价、路由及中间件、内置中间件、第三方中间件、错误处理中间件

（1）应用级（自定义）中间件

每个中间件都是一个函数

server.use(path,function(req,res,next){

})//过滤路由中URL的路由

server.use(function(req,res,next){})过滤所有的路由

//使用中间件验证注册

//参数1：给哪一个URL的路有使用；如果为空，会给所有的路由使用；

//参数2：中间件函数:可以获取请求，以及做出响应;给路由使用

//next：会执行下一个中间件或者路由

const express=require('express')

var server=express()

server.listen(8080)

//使用中间件验证注册

//参数1：给哪一个URL的路有使用；如果为空，会给所有的路由使用；

//参数2：中间件函数:可以获取请求，以及做出响应;给路由使用

server.use(function(req,res,next){

console.log('验证了数据是否为空')

//会执行下一个中间件或者路由

next()

})

server.get('/list',function(req,res){

res.send('这是商品的列表')

})

//主要的业务逻辑

练习给URL为/reg的 路由添加前置中间件，验证是否可用

练习：创建路由（get,/view）响应当期那的浏览次数；每次浏览次数加1.在中间件的外部声明变量初始化为0，在中间件中完成加1，在路由中把该变量响应到浏览器

const express=require('express')

var server=express()

server.listen(8080)

var n=0;

//添加中间件

server.use('/view',function(req,res,next){

n++;//++隐式将n转成数字

next()

})

server.get('/view',function(req,res){

res.send(n.toString())

})

（2）路由级中间件

路由器的使用

server.use(path,路由器);

(3)内置中间件

在express4下只保留一个内置的中间件

express.static('目录') const express=require('express')

var server=express()

server.listen(8080)

//静态资源：html,css，浏览器端js,img...

//把所有的静态资源托管到public目录

//使用内置中间件static(静态)

server.use(express.static('public'));

把金泰资源托管到指定的目录，如果浏览器请求静态资源，自动到该目录下查找

静态资源：html、css、浏览器js、图像…

（4）第四方中间件

npm install 中间件名称

//引入body-parser中间件

const bodyParser=require('body-parser')

server.use(bodyParser.urlencoded({

extended:false

//不使用扩展的qs模块，而使用querystring模块格式化为对象

}))

urlencodeed：将post请求的数据格式化为对象

extended：不是用第三方qs模块，而是用核心模块querystring将查询字符串格式化为对象

在路由中获取post请求数据

req.body格式化为对象

练习：创建文件40-post.js,创建web服务器，托管静态资源到public下，创建login.html,使用浏览器请求该文件，点击按钮，向服务器发请求。

post /login

const express=require('express')

//引入body-parser中间件

const bodyParser=require('body-parser')

var server=express()

server.listen(8080)

//使用bodyParser中间件将post请求数据格式化为对象

server.use(bodyParser.urlencoded({

extended:false

//不使用扩展的qs模块，而使用querystring模块格式化为对象

}))

server.post('/login',function(req,res){

//获取post请求的数据，格式化为对象

//前提：已经使用了中间件body-parser

console.log(req.body)

res.send('登陆成功')

})

server.use(express.static('public'))

<h1>用户登录</h1>

<form method='post' action="/login">

用户:<input type="text" name='uname'><br>

密码:<input type="text" name='updw'><br>

<input type="submit">

</form>

mysql模块

连接MySQL

增INSERT INTO emp VALUES(NULL,'TOM')

删DELETE FROM emp WHERE uid=5

该UPDATE emp SET uname='jarry'WHERE uid=5

查SELECT\*FROM emp

（1）连接MySQL

//引入MySQL模块

const mysql=require('mysql')

//创建连接对象，传递连接数据库的需要的服务器，端口，用户名，密码，要使用的数据库

var connection=mysql.createConnection({

host:'127.0.0.1',

port:3306,

user:'root',

passwrod:'',

database:'tedu'//连接后使用的数据库

})

connection.connect()//建立连接

connection.query('SELECT\*FROM emp',function(err,result){

//err可能产生的错误

if(err)throw err

//result执行的结果

console.log(result)

})//sql 是要执行的SQL语句，回调函数，用于获取SQL执行的语句的结果

//关闭连接

connection.end()

使用MySQL模块删除员工表emp中编号为5的数据，打印结果

const mysql=require('mysql')

//创建连接对象

var connection=mysql.createConnection({

host:'127.0.0.1',

port:3306,

user:'root',

passwrod:'',

database:'tedu'//连接后使用的数据库

})

//建立连接

connection.connect()

connection.query('DELETE FROM emp WHERE eid=4',function(err,result){

if(err)throw err

if(result.affectedRows>0){

console.log('删除成功')}

else{console.log('删除失败')}

})

//关闭连接

connection.end()

（2）连接池

//引入MySQL模块const

mysql=require('mysql')

//创建连接池连接对象，传递连接需要的地址、端口、用户名、密码、要使用数据库、设置连接池的大小connectionLimit,默认是15

var pool=mysql.createPool({

host:'127.0.0.1',

port:3306,

user:'root',

password:'',

database:'tedu',

connectionLimit:20//设置连接池的大小

})

//执行SQL语句

pool.query('select\*from emp',function(err,result){

if(err)throw err;

console.log(result)

})执行SQL语句，SQL要执行的SQL语句，回调函数，用于获取执行的结果

//往部门表dept插入数据,查看结果

//引入MySQL模块const

mysql=require('mysql')

//创建连接对象

var pool=mysql.createPool({

host:'127.0.0.1',

port:3306,

user:'root',

password:'',

database:'tedu',

connectionLimit:20//设置连接池的大小

})

//var did=18

//var dname='安保部'

var boj={

did:14,

dname:'后勤部'

}

//?占位符，防止SQL注入（攻击数据库）

pool.query('INSERT INTO dept VALUES(?,?)',[boj.did,boj.dname],function(err,result){

if(err)throw err;

console.log(result)

})

或者直接插入

var obj={

did:21,

dname:'财务部'

}

pool.query('Insert into dept set ?',[obj],function(err,result){

if(err)throw err

console.log(result)

})//只能在MySQL中使用

//将部门表dept中编号为40的数据，部门名称改为'测试部'

//引入MySQL模块const

mysql=require('mysql')

//创建连接对象

var pool=mysql.createPool({

host:'127.0.0.1',

port:3306,

user:'root',

password:'',

database:'tedu',

connectionLimit:20//设置连接池的大小

})

pool.query('update dept set dname=? where did=?',["测试1部",40],function(err,result){

if(err)throw err;

console.log(result)

})

创建web服务器，托管静态资源到public下，在该目录下创建add.html,点击提交按钮，想服务器发请求（get /add）,服务器获取数据，并将数据插入到tedu下的dept,如果插入成功，响应‘部门添加成功’

const express=require('express')

const mysql=require('mysql')

var server=express()

server.listen(8080)

//托管到静态资源到public

server.use(express.static('public'))

//创建连接池对象

var pool=mysql.createPool({

host:'127.0.0.1',

port:3306,

user:'root',

password:'',

database:'tedu',

conncetionLimit:20

})

//更具表单提交创建对应的路由

server.get('/add',function(req,res){

//获取浏览器传递的数据

var obj=req.query

console.log(obj)

//数据插入到数据库

pool.query('INSERT INTO dept SET ?',[obj],function(err,result){

if(err)throw err;

console.log(result)

if(result.affectedRows>0){

res.send({code:200,msg:'add 添加成功'})

}

})

})

<h1>哈哈哈</h1>

<form method="get" action="/add">

编号:<input type="text" name='did'><br>

部门:<input type="text" name='dname'><br>

<input type="submit">

</form>

往学子插入数据

[app.js](xz/app.js)  [user.js](xz/routes/user.js)  [user-rag.html](xz/public/user-rag.html) [pool.js](xz/pool.js)

1.[用户注册](xz/public/user-rag.html)

2[用户登录](xz/public/user_login.html)

3[用户登录](xz/public/user_detail.html)

4用户检索

5[用户列表](xz/public/user_list.html)

html在C:\Users\web\Desktop\wname\学子\xz\public

app.js

const express=require('express')

//n引入路由器

const userRouter=require('./routes/user.js')

//引入body-parser

const bodyParser=require('body-parser')

//创建web服务器

var server=express()

server.listen(8080)

//托管静态资源到public下

server.use(express.static('public'))

//使用body-parser中间件，将post请求的数据格式化为对象

server.use(bodyParser.urlencoded({

extended:false

}))

//使用路由器

//使用（挂载）的URL：/user

// /user/login

server.use('/user',userRouter)

user.js

//创建路由器对象

const express=require('express');

//引入连接池模块

const pool=require('../pool.js')

//创建路由器对象

var router=express.Router();

//添加路由

//1.用户注册

//get /login

/\*router.get('/rag',function(req,res){

//格式化为对象

var obj=req.query;

console.log(obj);

//检测是否为空

if(obj.uname==''){

res.send({code:401,msg:'uname required'});

return;

};

if(obj.upwd==''){

res.send({code:402,msg:'upwd required'});

return

}

if (obj.phone=='')

{res.send({code:403,msg:'phone required'})

return

}

if (obj.email=='')

{res.send({code:404,msg:'email required'})

return

}

// 执行SQL语句

pool.query('INSERT INTO xz\_user SET ?',[obj],function(err,result){

//判断错误

if(err) throw err;

console.log(result)

//判断是否成功

if(result.affectedRows>0){

res.send({code:200,msg:'res suc'})

}

})

//res.send('注册成功');

});\*/

//2用户登录

/\*router.post('/login',function(req,res){

//获取post请求的数据

var obj=req.body;

console.log(obj)

//验证是否为空401 402

if(!obj.uname){res.send({code:401,msg:'uname required'})

return

}

if(!obj.upwd){res.send({code:402,msg:'upwd required'})

return

}

//执行SQL语句

//查询用户表中是否含有用户名和密码同时匹配的数据

pool.query('select\*from xz\_user where uname=?and upwd=?',[obj.uname,obj.upwd],function(err,result){

if(err)throw err;

//console.log(result)

//判断数组长度是否大于0

if(result.length>0){

res.send({code:200,msg:'login suc'})

}else{res.send({code:300,msg:'login err'})}

})

//res.send('登陆成功')

})\*/

//3用户登录

/\*router.get('/update',function(req,res){

//获取get请求数据

var obj=req.query

//console.log(obj)

var n=400

//便利对象属性，获取所有的属性

for(var key in obj){

n++;

//console.log(key,obj [key])

//判断属性是否为空

if(!obj[key]){

res.send({code:n,msg:key + ' required'})

return

}

}

//执行SQL语句，修改数据

pool.query('update xz\_user set phone=?,email=?,user\_name=?,gender=? where uid=?',[

obj.phone,

obj.email,

obj.user\_name,

obj.gender,

obj.uid

],function(err,result){

if(err)throw err

//console.log(result)

//判断是否修改成功

if(result.affectedRows>0){

res.send({code:200,msg:'update suc'})

}else{res.send({code:301,msg:'update err'})}

})

})\*/

//4用户检索

/\*get

/user/detail

获取数据

验证是否为空

执行SQL语句（查询编号对应的用户） \*/

/\*router.get('/detail',function(req,res){

var obj=req.query

if(!obj.uid){

res.send({code:401,msg:'uid required'})

return

}

pool.query('select\*from xz\_user where uid=?',[obj.uid],function(err,result){

if(err)throw err

res.send(result)

})

})\*/

//5用户列表

/\*router.get('/list',function(req,res){

//get请求的数据

var obj=req.query

//验证是否为空

if(!obj.pno){

obj.pno=1

}

if(!obj.count){

obj.count=3}

//将字符串转数字整形

obj.pno=parseInt(obj.pno)

obj.count=parseInt(obj.count)

//页码开始值

var start=(obj.pno-1)\*obj.count

pool.query('select\*from xz\_user limit ?,?',[start,obj.count],function(err,result){

if(err)throw err;

res.send(result)

})

})\*/

//6删除

router.get('/delete',function(req,res){

var obj=req.query

//验证是否为空

if(!obj.uid){

res.send({code:401,msg:'uid required'})

return

}

pool.query('delete from xz\_user where uid=?',[obj.uid],function(err,result){if(err)throw err

if(result.affectedRows>0){

res.send({code:200,msg:'delete suc'})}else{res.send({code:301,msg:'delete err'})}

})

})

//导出路由器对象

module.exports=router;