day 7.17

1.ES6

(1)函数增强

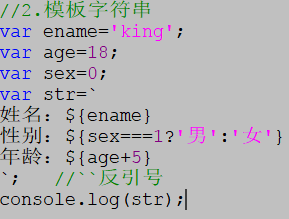
ES6 中允许给形参设置默认值，如果没有传递实参，则自动调用形参的默认值。

注意事项：默认值通常是从最后一个形参开始赋值。

|  |
| --- |
| function add(a,b,c=0) |

(2)模板字符串

|  |
| --- |
| `在模板字符串之间可以放任何形式的字符串 ${js表达式}`; |



练习：创建图书对象，包含有图书的编号，名称，价格，上架时间，是否在售（1/0）；使用模板字符串打印图书信息

2.Node.js概述

基于谷歌V8引擎（解释器），运行在服务器端的

对比JS和node.js

(1)js运行在浏览器端，存在多个浏览器，有兼容性问题；nodeJS运行在服务器端，只有一种解释器，不存在兼容性问题。

(2)两者都有内置（ES,原生）对象,自定义对象,宿主对象

(3)JS用于浏览器端的交互效果，nodeJS用于服务器端技术，

nodejs执行方式

脚本模式 node c:/xampp/…/1.js 回车

交互模式 node 回车 进入交互模式

退出 两次Ctrl+C 或者.exit

3.全局对象

nodeJS：global

在交互模式下，声明的变量和创建的函数都是global下的。 访问global.a/global.fn()

在脚本模式下，声明的变量和创建的函数都不是global下，访问形式不能使用global

JS：window

在JS下，脚本声明的变量和创建的函数，都是全局window下，访问window.a/window.fn()

4.console对象

控制台标准输出

console.log() 向控制台输出，打印字符串

console.info() 输出消息

console.warn() 输出警告

console.error() 输出错误

console.time(‘自定义字符串’) 开始计时

console.timeEnd(‘自定义字符串’) 结束计时

注意事项：开始计时的字符和结束计时字符串保持一致。

练习：分别使用while，do-while，for计算1~100之间所有整数和，计时查看

// C:\Users\web\Desktop\lijun\NODEJS\05\_console.js

5.process对象(在命令行中执行)

进程：执行程序的时候，操作系统分配一定的内存空间

process.arch 查看当前的CPU架构（x64 x86）

process.platform 查看当前的操作系统（‘win32’）

process.evn 查看当前的环境变量有哪些，软件添加到环境变量，可以在命令行中启动，速度更快

process.version 查看nodejs的版本号

process.pid 查看当前进程的PID（编号）

process.kill(PID) 结束某个编号的进程

6.Buffer对象

缓存区：在内存中存储数据的区域，常用于网络传输时的资源。

创建Buffer

|  |
| --- |
| var buf=Buffer.alloc(6,'一二三四五'); //Buffer首字母必须大写  var buf1=Buffer.alloc(6,'a');  console.log(buf);  //转成字符串  console.log(String(buf)); //一个中文占3个空间  console.log(buf1.toString()); //内存将被重复赋值 aaaaaa |

设置Buffer的空间大小为6，一个中文占3个空间，一个字母占一个空间，默认使用16进制存储

转成字符串: String(\*\*)/\*\*.toString()

7.全局函数

parseInt/parseFloat/isNaN/isFinite/eval/encodeURI/decodeURI

都是global下的，可以使用global访问 global.parseInt

(1)一次性定时器

|  |
| --- |
| 开启  var timer=setTimeout(回调函数,间隔时间);  当间隔时间到了，会执行一次回调函数，单位是毫秒  清除  clearTimeout(timer); |

(2)周期性定时器

|  |
| --- |
| 开启  var timer=setInterval(回调函数,间隔时间);  每个一段时间，会执行一次回调函数，单位是毫秒  清除  clearInterval(timer); |

练习：每隔3秒钟，打印一次‘hello’，打印完第3次，清除定时器。

(3)立即执行定时器(事件队列)

1)setImmediate(回调函数)/clearImmediate(timer) 放在事件队列中，等待程序执行完再执行

2) process.nextTick 放在主程序的末尾执行

|  |
| --- |
| console.log(5);  //立即执行  //开启  var timer=setImmediate(function(){  console.log(2);  });//放在事件队列中，等待程序执行完再执行  //清除  //clearImmediate(timer);  //立即执行  process.nextTick(function(){  console.log(4);  });//放在主程序的末尾执行  console.log(3);  //输出结果为5,3,4,2 |

C:\Users\web\Desktop\lijun\NODEJS\day7.18\09\_function.js

Day7.18

8.模块

模块是一个独立的功能体

在nodeJS下模块分为：核心模块，自定义模块，第三方模块

(1)自定义模块

在nodeJS下，每一个文件都是一个模块；文件中的代码自动被构造函数所包含

以下红色代码是nodeJS自动添加的

|  |
| --- |
| (function (exports, require, module, \_\_filename, \_\_dirname) {  程序员写的代码  }) |

C:\Users\web\Desktop\lijun\NODEJS\08\_1.js

|  |
| --- |
| require:引用另一个模块  module:指代当前所在的模块  module.exports:当前模块导出的对象，可以设置导出对象的名字  exports:指向module.exports ,  1.最好别分别定义module.exports和exports  2.NodeJs开发者建议导出对象用module.exports,导出多个方法和变量用exports  \_\_filename:当前模块的绝对路径和文件名称(有两个下划线)  \_\_dirname:当前模块的绝对路径(有两个下划线) |

(2)模块分类 （引用 文件/目录模块）

|  |  |  |
| --- | --- | --- |
|  | 以路径开头 | 不以路径开头 |
| 文件模块 | Require(‘./circle.js’) / Require(‘./circle’)为js文件可以省略后缀名  常用语引入用户自定义的模块 | require(‘querystring’)  常用于引入官方提供的核心模块 |
| 目录模块 | Require(‘./02\_2’);  (1)到02\_2目录下找package.json文件中main属性对应文件，属性名和属性值都要用双引号。  (2)如果找不到就引入index.js | require(‘04\_2’)  (1)到当前目录下的node\_modules中寻找目录04\_2文件夹;  (2)如果找不到会到上一级目录寻找，直到顶级目录。常用于引入第三方模块。 |

练习：

创建两个模块main.js(主模块)和circle.js(功能模块)；

在功能模块下创建两个函数，分别传递1个参数，计算圆的面积和周长，导出这两个函数；在主模块中引入功能模块，并调用两个函数。

练习：创建模块03\_1.js，引入当前目录下的03\_2目录，在03\_2目录含有文件fun.js，导出一个函数add（计算两

个数字相加）；在03\_1.js中调用该函数。

练习：在05目录下创建模块05\_1.js，引入不带路径的05\_2目录，含有文件foo.js，打印‘hello’；

(3)包(第三方模块)和npm

CommonJS是一种规范，JS项服务器端发展就是遵循了这种规范，才出现模块概念。

包（package）：就是一个目录模块

npm：用于管理包的工具（npm官网： <https://www.npmjs.com/>）

**切换目录**

cd 完整目录 回车

在指定目录下的空白区域，按住shift，单击鼠标右键，选择’在此处打开powershell窗口’.

**使用npm安装**

npm install 包名称 回车

自动从npm官网下载包到当前目录下的node\_modules中

npm init –y 初始化一个package.json文件，项目说明文件，会记录安装的所有包

npm install 自动安装package.json中记录的包

C:\Users\web\Desktop\lijun\课后笔记\_老师版

(4)核心模块

1)querystring（查询字符串模块）解析和重组

查询字符串：浏览器向web服务器发请求，传递数据的一种方式，位于URL中？后边的部分

<https://search.jd.com/Search?keyword=电脑&enc=utf-8>

parse() 将查询字符串解析为对象

stringify() 将对象格式化为查询字符串

C:\Users\web\Desktop\lijun\NODEJS\day7.18\07\_querystring.js

练习：获取查询字符串中商品的价格和名称

https://search.jd.com/Search?keyword=手机&enc=utf-8&pvid=2031

C:\Users\web\Desktop\lijun\NODEJS\day7.18\07\_querystring.js

2)url模块

用于对url解析和重组

|  |
| --- |
| <https://search.jd.com:80/Search?enc=utf-8&wq=1423> |

parse() 将url解析为对象

format() 将对象格式化为url

练习：获取以下url中查询字符串的数据

<https://www.tmooc.cn:443/course.heml?cid=18&cname=javascript>

第一步：将url解析为对象，获取到查询字符串

第二步：把获取的查询字符串解析为对象

C:\Users\web\Desktop\lijun\NODEJS\day7.18\08\_url.js

3)fs模块 (这个模块方法分为同步 fs.statSync(path)和异步 fs.stats(path,callback);)

一：异步方法：在事件队列中，通过回调函数执行（异步没有返回值）。

文件系统模块

用于文件的操作，文件包含目录形式和文件形式

1.查看文件的状态

fs.stat(path,callback); fs.stat()也是事件队列

path 文件的路径

callback 回调函数（function(err,stats){}）

err 可能产生的错误

stats.isDirectory() 是否为文件

stats.isFile() 是否为目录

C:\Users\web\Desktop\lijun\NODEJS\day7.18\10\_fs.js

|  |
| --- |
| //创建目录  fs.mkdir('mydir',(err)=>{  if(err) throw err;  }); |

二：同步方法：在主程序中执行，通过返回值获取结果 (同步有返回值)

|  |
| --- |
| //同步方法：在主程序中执行  const fs=require('fs');  var s=fs.statSync('.//02\_1.js');  console.log(s);  console.log(2); |

**同步和异步**

同步：会阻止代码往后直行，只有前边的代码执行完才能执行后边的，通过返回值获取结果。

异步：不会阻止代码往后执行，后边的代码可以先执行，不需要等待前边代码，通过回调函数执行。

课后任务

(1)复习今天的内容，整理思维导图

(2)练习

创建main.js模块，引入目录模块home，在home中含有文件fun.js，导出一个函数，传递文件的路径，返回文件的状态，在main.js中调用该函数。

get(‘02\_2’);

(2)预习http协议（请求和响应），http模块

day7.19

1.fs模块

(1)创建目录

fs.mkdir(path,callback)/fs.mkdirSync(path);

(2)移除目录

fs.rmdir(path,callback)/fs.rmdirSync(path);

(3)读取目录

fs.readdir(path,callback)/ fs.readdirSync(path)

callback err result:读取的结果，格式为数组

(4)写入文件

fs.writeFile(path,data,callback)/fs.writeFileSync(path,data)

文件不存在则会自动创建，清空写入

(5)追加写入

fs.appendFile(path,data,callback)/ fs.appendFileSync(path,data)

文件不存在则会自动创建，末尾追加写入

(6)读取文件

fs.readFile(path,callback)/ fs.readFileSync(path)

callback err data:读取的数据，格式为buffer

(7) 删除文件

fs.unlink(path,callback)/fs.unlinkSync(path);

(8) 检测文件是否存在

fs.existsSync(path);

存在返回true,不存在返回false

练习：如果data2.txt存在，同步删除它

(9)文件流

fs.createReadStream(path) 创建可读取的流

|  |
| --- |
| const fs=require('fs');  //1.创建可读取的流  var readStream=fs.createReadStream('stu.txt');  //设置编码为utf-8  readStream.setEncoding('utf8');  //监听是否有数据流  //一旦有数据进入，自动触发回调函数，来获取数据  readStream.on('data',(chunk)=>{ //chunk 获取的数据，分段的  console.log(chunk.toString());  });  //监听是否读取结束,自动触发回调函数  readStream.on('end',function(){  console.log('读取结束');  }); |

fs.createWriteStream(path) 创建可写入的流

pipe() 管道，可将数据从读取的流添加到写入的流

|  |
| --- |
| //2.创建可写入的流  //把读取的流放入到写入的流，完成文件的拷贝  //创建可读取流  var readStream=fs.createReadStream('stu.txt');  var writeStream=fs.createWriteStream('2.txt');  //把读取流放入到写入流  readStream.pipe(writeStream); |

2.http协议

浏览器和Web服务器之间的通信协议

(1)通用头信息

Request URL:向服务器获取哪些内容，对应浏览器地址栏

Request Method：get/post 请求的方法（打开网页默认都是get）

status: 响应的状态码

1\*\*：正在请求,

2\*\*:成功的响应

3\*\*：响应的重定向，跳转到另一个url

4\*\*：客户端请求错误

5\*\*：服务器端错误

Remote Address: 111.206.231.1:443 请求服务器的IP地址和端口号

(2)响应头信息

connection:建立的连接方式，keep-alive 持久连接

connection-Encoding:服务端内容的压缩形式 gzip

content-type: text/html 文件类型

transfer-Encoding:传输方式 chunked 分段传输

location：跳转的URL，常结合这状态码3\*\*使用

(3)请求头信息

accept：浏览器接收的文件类型有哪些

accept-encoding:接收的压缩类型有哪些

user-agent：客户端使用的浏览器

(4)请求主体:可有可无，用于浏览器向服务器发送的数据

3.http模块

既可以创建web服务器，，也可以模拟浏览器向其他的服务器发请求

http.get(url, callback) 向服务器发请求

url 请求的url callback 回调函数，获取响应的数据

res响应的对象

res.statusCode 获取响应的状态码

res.on('data',(chunk)=>{}) 监听服务器是否有数据传输

chunk:获取的数据，格式为buffer.

[www.weather.com.cn/data/sk/101010100.html](http://www.weather.com.cn/data/sk/101010100.html)

(2)创建web服务器

<http://127.0.0.1:8080>

|  |
| --- |
| var app=http.createServer(); //创建web服务器  app.listen(8080); //设置监听的端口，  //监听浏览器的请求，使用回调函数处理请求  app.on('request',(req,res)=>{  //req:请求对象  console.log(req.url,req.method); //请求的url，请求的方法  //res 响应对象  res.writeHead(状态码，头信息对象) 设置响应的头信息和状态码  res.write() 设置响应的内容  res.end() 结束并发送响应  }) |

7.19作业

|  |
| --- |
| const http=require('http');  const fs=require('fs');  //引入压缩模块  const zlib=require('zlib');  var app=http.createServer();  app.listen(8080);  app.on('request',(req,res)=>{  if(req==='/list'){  res.write('this is product list');  res.end();  }  else if(req.url==='/index'){  res.write('this is homepage');  res.end();  }  else if(req.url==='/'){  res.writeHead(302,{Location:'/index'});  res.end();  }  else if(req.url==='/study'){  /\*  //res.writeHead(302,{Location:'http://www.tmooc.cn'});  //响应文件  //读取文件  var data=fs.readFileSync('./demo.html')  res.write(data);  //把读取的文件作为响应的内容  \*/  //前提：设置响应的压缩类型为gzip这时候浏览器会对压缩进行解压  res.writeHead(200,{ContentEncoding:'gzip'});  //创建gzip压缩  var gzip=zlib.createGzip();  //创建读取文件流，添加到压缩包中  //通过管道（流）的形式响应到浏览器  fs.createReadStream('./demo.html').pipe(gzip).pipe(res);  }  else{  res.writeHead(404);  res.write('Not found');  res.end();  }  }) |

day7.22

1.使用zlib模块对文件进行压缩

|  |
| --- |
| //引入zlib压缩模块  const zlib=require('zlib');  //前提：设置响应的压缩形式  res.writeHead(200,{ContentEncoding:'gzip'});  //创建gzip压缩  var gzip=zlib.createGzip();  //创建读取文件流，添加到压缩包中  //通过管道（流）的形式响应到浏览器  fs.createReadStream('./demo.html').pipe(gzip).pipe(res); |

2.express

基于nodeJS平台，快速，开发，极简的web开发框架

安装 npm install express

网址：[www.expressjs.com.cn](http://www.expressjs.com.cn) /www.npmjs.com

1.创建web服务器

|  |
| --- |
| //引入第三方模块  const express=require('express');  //创建web服务器  var app=express();  //监听端口  app.listen(8080); |

(1)路由

浏览器向web服务器发请求，web服务器根据请求的url和请求的方法来做出响应

res 响应对象

1) res .send(content) 设置响应内容并发送 路由中只能使用一次

2) res .sendFile(\_\_dirname+'demo.html') 设置响应文件,需要使用绝对路径

3) res .redirect() 响应的重定向，跳转到另一个url网页

method 请求的方法 / url 请求的url

query 获取请求的查询字符串的数据，格式为对象。

params 获取路由传参的数据，格式是对象

|  |
| --- |
| 路由传参  app.get('/order/:did',(req,res)=>{  //使用req对象，获取路由传参的数据  console.log(req.params);  res.send('这是订单详情');  });  浏览器传递  <http://127.0.0.1:8080/order/10>  10就是实参，也就是传递的数据 |

练习：创建路由，请求的方法：get，请求的url：/shopping，使用路由传参传递上平的价格和规格，然后把价格和规格显示到了浏览器中

价格：4999

规格：总师版

|  |
| --- |
| app.get('/shopping/:price/:lei',(req,res)=>{  console.log(req.params);  res.send(`  价格:${req.params.price}<br>  规格:${req.params.lei}  `);  }); |

(2)获取post请求

|  |
| --- |
| /\*练习：使用express创建web服务器，创建登录的路由（获取登录的HTML），  请求的方法：get,请求的url：/login，响应文件login.html \*/  const express=require('express');  //引入查询字符串模块  const querystring=require('querystring');  //创建web服务器，并监听端口  var app=express();  app.listen(8080);  //创建一个登录路由  app.get('/login/:login',(req,res)=>{  console.log(req.params);  res.sendFile(\_\_dirname+'/'+req.params.login+'.html');  });  //根据表单提交来创建对应的路由，请求方法：post，请求的url：/mylogin,响应'登录成功'  app.post('/mylogin',(req,res)=>{  //获取post请求传递的数据  //监听是否有数据传递  req.on('data',(chunk)=>{  //chunk就是传递的数据  var str=chunk.toString();  //把查询字符串解析为对象  var obj=querystring.parse(str);  console.log(obj);  });  //console.log(res.query);  res.send('登录成功');  }); |

练习：向服务器请求方法：get /， url：/reg , 响应的一个注册的文件reg.html(表单：用户，密码邮箱，电话)，点击提交

(3)路由器

路由在使用中，不同模块可能出现相同的url，路由器可能将有同一模块下的所有路由管理起来，路由器最终经要挂载到web服务器才能使用。

(1)创建路由器模块

|  |
| --- |
| //引入express  const express=require('express');  //创建路由器  var router=express.Router();  //往路由器中添加路由  //请求方法get，url：/list  router.get('/list',(req,res)=>{  res.send('这是用户列表');  });  //添加诸多路由后，导出路由器对象  module.exports=router; |

(2)在服务器中引入并使用

|  |
| --- |
| //引入express模块  const express=require('express');  //引入路由器模块（自定义模块）  const user=require('./user.js');  //创建服务器  var app=express();  //监听端口  app.listen(8080);  //挂载到路由器：把路由器交给服务器使用  //同事要给所有的url1添加/user ->/user/list (在所有url前面添加/user前缀)  app.use('/user',user); |

课后任务

(1)复习今天的内容，整理思维导图

(2)练习

创建商品路由器product.js，添加路由，导出路由器；在服务器中引入并使用，给所有url添加/product.

(3)预习中间件（处理数据），复习MySQL数据库

day7.23

1.中间件

用户处理（过滤）请求，最终是要为路由提供服务

(1)应用级中间件(自定义); (2)路由级中间件; (3)内置中间件; (4)第三方中间件;

(1) 应用级中间件

app.use(url,(req,res,next)=>{})

url 表示中间件要给哪一个路由的url使用 ，对应路由的url

在中间中可以拦截到请求，也可以做出响应；

next是一个函数，表示执行下一个中间件或者路由

练习：创建路由，方法get，<url:/shopping> ,传递商品的价格price，要求在中间件中商品价格加100

，最后响应’商品价格为：\*\*\*’。

(2) 路由级中间件

路由器使用

(3)内置中间件

express当前只有一个内置中间件

把所有的静态资源托管到指定的目录，如果浏览器请求文件，自动到该目录下查找，

不需要创建路由来响应文件。

app.use(express.static('目录'));

练习：托管静态资源到files目录，如果files目录和public目录有相同名称的文件，查看会显示哪一个

(4)第三方中间件body-parser使用

练习：创建web服务器，托管静态资源到public目录，创建文件login.html,点击提交，向服务器发送post，请求的<url:/mylogin>, ,在路由中获取post请求的数据. 04\_three.js

|  |
| --- |
| const bodyparser=require('body-parser'); //1.引入body-parser中间件  app.use(bodyparser.urlencoded({  extended:false })); //2.使用body-parser中间件，获取post请求的数据，并解析为对象。  不使用第三方模块qs,而使用官方querystring模块解析对象  console.log(req.body); //3.req.body获取提交的表单数据 |

2.MySQL模块

mysql使用

mysql.exe –h127.0.0.1 –p3306 –uroot -p

INSERT INTO emp VALUES(…);

DELETE FROM emp WHERE lid=5;

UPDATE emp SET order=5,ename=’root’ WHERE lid=5;

SELECT \*FROM emp;

|  |
| --- |
| //创建连接对象  var connection=mysql.createConnection({ });  //执行连接  connection.connect();  //执行SQL语句  connection.query(sql,callback);  sql 要执行的SQL语句  callback 获取执行的结果  err 可能产生的错误  result SQL语句的结果  //防止SQL注入  //增添一个用户的整个数据  connection.query(`INSERT INTO emp VALUES(?,?,?,?  )`,[null,'wenjie','chongqing',6000],function(err,result){  //err 可能产生的错误  //result SQL语句执行的结果  if(err) throw err;  console.log(result);  });  /\*  var person={  eid:null,  ename:'zhangjie',  addr:'beijing',  salary:5800  };  //插入对象形式的数据  connection.query('INSERT INTO emp SET?',[person],function(err,result){  //err 可能产生的错误  //result SQL语句执行的结果  if(err) throw err;  console.log(result);  }); \*/  //关闭连接  connection.end(); |

查询结果返回一个数组

插入：对象 affectRows 大于0说明插入成功了

修改：对象 affectRows 大于0说明修改成功了

删除：对象 affectRows 大于0说明删除成功了

SQL注入（一个漏洞）

SELECT \*FORM emp WHERE eid=4 or 1=1; //查看所有数据

DELETE FROM emp WHERE eid=1 or1=1; //删除所有数据

使用占用符(?),对数据进行过滤

|  |
| --- |
| 使用连接池  var pool=mysql.createPool({ }); //创建连接池对象  // connectionLimit:20 设置连接池大小  pool.query(sql,arr,callback); 执行SQL语句 arr:防SQL注入插入的数据数组 |

练习：创建web服务器，托管静态资源到public目录，在public目录下创建user\_reg.html(包含用户名，密码，邮箱，手机，提交按钮) 提交的路由 post /reg

（）

课后任务：

（1）复习今天内容，整理思维导图

（2）课后练习

创建web服务器，托管静态资源到public目录，创建文件add.html,点击提交，向服务器请求（/add get）,获取数据，把数据插入到tedu下的dept表中，响应‘部门添加成功’

|  |  |
| --- | --- |
| 传递方式 | 获取方式 |
| get(查询字符串) | req.query |
| 路由传参 | req.params |
| post | Req.body(前提：引用body-parser中间件) |

day7.24

|  |  |
| --- | --- |
| 传递方式 | 获取方式 |
| get(查询字符串) | req.query |
| 路由传参 | req.params |
| post | req.body(前提：引用body-parser中间件) |

|  |  |
| --- | --- |
| mysql操作 | 返回结果result |
| INSERT（增） 无 | affectedRows |
| DELETE(删) 有 | affectedRows |
| UPDATE(改) 无 | affectedRows |
| SELECT（查） 有 | num(rows) |

1.注册页面

练习：创建用户路由器，添加注册路由（post ,/reg）,导出路由器，在服务器下引入并挂载到/user -> /user/reg

2.登录页面

3.检索用户

4.修改用户

5.分页查询

select \* from emp limit start,count

用户列表 user\_list.html

页码:pno 大小：size 提交按钮

路由：get /list

1.获取数据

2.如果页码为空，默认为1；如果大小为空，默认为3

3.把值转为整型

4.计算开始start

课后任务

完成用户模块下的删除路由，按照编号删除（get/delete）

完成商品模块路由器下的路由(列表) xz\_laptop

预习 git