2.学习一门编程语言的基本步骤

(1)了解背景知识：历史、现状、特点、应用场景。

(2)搭建开发环境，编写hello world

(3)变量和常量

(4)数据类型

(5)运算符

(6)逻辑结构

(7)通用的小程序

(8)函数和对象

(9)第三方库、框架

(10)实用的项目

3.JS概述

(1)历史

1995年，JS最早出现在Netscape的浏览器中

2009年，JS遵循CommonJS规范，开始向服务器端发展

(2)现状

既可以允许在客户端浏览器，也可以运行在服务器端

(3)特点

解释型语言，编译一行执行一行。

弱类型语言

基于对象

跨平台

4.JS的执行环境

(1)浏览器自带的JS解释器

(2)nodeJS下的JS解释器

在命令行下 node -v 查看系统中nodejs版本号

(3)执行JS代码

浏览器：

创建01.js和01.html

在01.html中引入01.js

<script src="./01.js"></script>

node.js下

node C:/xampp/.../01.js

5.js代码规范

(1)区分大小写

(2)每行代码结束的分号客家可不加，提倡都加

(3)分为当行注释(//......)和多行注释(/\*......\*/)

6.变量

(1)变量就是用于存放数据的容器

x=1 y=2

(2)变量的命名规则

变量名称是以字母、数字、美元符号($)、下划线组成的；不能以数字开头。

多个连词之间的命名方式（下划线命名法、驼峰命名法）

user\_name userName

不能使用关键字和保留字作为变量名

(3)变量的注意

变量声明后，可以重新赋值

变量声明后未赋值，此时的值是undefined

使用未声明的变量会报错

(4)一次性声明多个变量

var a=1,b=2,c;

多个变量之间用逗号隔开。

7.常量

一旦声明不能重新赋值。

8.数据类型

分为原始类型和引用类型

原始类型分为数值型、字符串型、布尔型、未定义型(undefined)、空(null)

(1)数值型

分为整型和浮点型

整型在内存中占4个字节，浮点型占8个字节

8进制，以0开头，例如：var num2=012; num2为10。

16进制，以0x开头，例如：var num3=0xb; num3为11，x不区分大小写

Day6

1.

10进制，1234.56->1.23456**E3**->123456**E-2**

typeof(数据) 检测数据类型

(2)字符串型

数据被引号包含就是字符串类型；不区分单双引号

console.log('健'.charCodeAt());\*/

(3)布尔型

在程序中**真**或者**假**的结果

true/false

常用语一些是否的结果，如是否登录，是否注册

(4)未定义型

声明了变量未赋值，结果就是undefined

(5)空

用于释放（销毁）一个引用类型的地址，只有一个值null

2.数据类型转换

(1)隐式转换

①数字+字符串：数字被转换成字符串

②数字+布尔型：布尔型被转成数字 true->1 false->0

③布尔型+字符串：布尔型转换成了字符串

JS中加号(+)的作用

执行加法运算

执行字符串的拼接

使用 -\*/ 执行运算

尝试将运算符的两端转成数值型，如果含有非数字则返回NaN(Not a Number)。

NaN 类型 number

(2)强制转换

①将任意类型转为整型

**parseInt()**

从开头查找数字，遇到非数字或者小数点，返回前边的数字

②将任意类型转为浮点型

**parseFloat()**

转换规则和parseInt类似，区别在于遇到小数点继续往后查找数字。

③将任意类型转为数值型

**Number()**

如果要转换的数据中含有非数字，则返回NaN

true->1 false->0

④数值型和布尔型转为字符串型

**toString()**

var num=10

num.toString(16) //a

如果要转换的数据是数值，可以设置进制(8,16,2)

3.运算符

表达式：由运算符连接操作的数据，所组成的形式就是表达式

(1)算术运算符

+ - \* / % ++ --

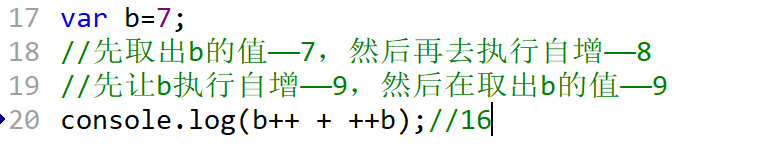
%取余

++ 自增，在原来的基础之上加1

-- 自减，在原来的基础之上减1

console.log(num++) 先打印num的值，再执行自增

console.log(++num) 先自增，再打印



(2)比较运算符

> < >= <= == != ===(全等于) !==(不全等于)

返回一个布尔型的结果

==只是比较两个值是否相同

===不仅比较值，还会比较类型是否相同

!==同理

3>'10' //false

数字和字符串比较，字符串转成数字

'3'>'10' //true

比较收个字符的Unicode码，如果首个字符相同，则比较第二个字符

3>'10a' //false

3<'10a' //false

3=='10a' //false

'10a' -> NaN

NaN和任何数比较(> >= < <= ==)都返回false

NaN==NaN 返回false

(3)逻辑运算符

&& 并且 || 或者 ! 非

返回一个布尔型的结果

&& 关联的两个条件都满足，结果是true，否则false

|| 关联的两个条件只需满足其一，结果是true，否则false

! 取反 !false -> true !true -> false

逻辑短路

&& 当第一个条件为false的时候，就不需要在执行第二个条件

|| 当第一个条件为true的时候，就不需要再执行第二个条件

(4)位运算符(了解)

在执行运算符的时候，会把数字转成二进制进行运算

按位与(&) 3&5 上下两位都是1，结果是1，否则是0

按位或(|) 3|5 上下两位含有1，结果是1，否则是0

按位异或(^) 5^7 上下两位不同为1，相同为0

按位右移(>>) 7>>1 删除二进制的最后一位，大概变小到原来的一半

按位左移(<<) 7<<1 在二进制的最后添加一个0，增加一倍

(5)赋值运算符

= += -= \*= /= %=

(6)三目运算符

单目运算符：只需要一个数据或者表达式

a++ a-- !false

双目运算符：需要两个数据或者表达式

+ - \* / % > < >= <= == != === !== && || & | ^ >> << = += -= \*= /= %=

三目运算符：需要三个数据或者表达式

条件表达式 **?** 表达式1 **:** 表达式2

age>=18 ? console.log('成年人') : console.log('未成年人');

如果条件表达式为true，执行表达式1；如果条件表达式为false，执行表达式2

Day7

1.浏览器端函数

alert() 弹出警示框(消息框)

prompt() 弹出提示框(输入框)，需要使用变量来接受输入的值；值得类型是字符串。

2.流程控制

程序=算法+数据

程序分为顺序执行、选择执行、循环执行

(1)if语句

语句1

if(条件表达式){

语句2

}

语句3;

执行流程：

①执行语句1

②执行条件表达式，如果为true，执行语句2，false跳过

③语句3

注意：如果if后的语句中只有一行，是可以省略大括号。

在if语句的条件表达式中，有一些值默认就是false

0、NaN、''、undefined、null

(2)if-else语句

语句1;

if(条件表达式)

{

语句2

}else

{

语句3

}

执行过程：

①执行语句1

②执行条件表达式，如果是true，执行语句2；如果是false，执行语句3

(3)if-else的嵌套

用于判断多钟情况

语句0;

if(条件表达式1){

语句1;

}else if(条件表达式2){

语句2;

}else ...... if(条件表达式n){

语句n

}else{

语句n+1} //以上所有的条件表达式都是false

执行流程：

①执行语句0

②执行条件表达式1

如果是true，执行语句1；

如果是false，执行条件表达式2

.......

③如果以上条件都是false，执行语句n+1

2.switch-case语句

是-一种特殊的分支语句，可以根据一个表达式的不同值，来选择执行不同的程序。

语句0;

switch(表达式){

case 1; //如果表达式的值是1

语句1;

break; //终止，不会再往后执行其他语句

...

case n;

语句n;

break;

default:

语句n+1;

}

注意：在case中表达式和值的比较实用的是全等于比较，要求值和类型都要满足结果才是true。

对比if-else嵌套和switch-case语句

相同点：两者都可以用于多项分支语句。

不同点：if-else可以判断相等或者不等的情况，实用范围更广；switch-case只实用全等的情况，结构上更为清晰，执行效率相对高。

3.循环执行

循环：就是一遍又一遍执行**相同或者相似**的代码

循环的两个要素：

循环的条件：重复的次数

循环体：重复执行的相同或者相似代码

(1)while循环

while(循环的条件){ //是一个布尔型的值

循环体

}

Day8

1.break关键字

break可以结束任何形式的循环

说明：''(空字符)在和数字比较的时候，转成0

2.do-while循环

do{

循环体

}while(循环条件);

不管循环条件是否为true，都会执行一次循环体

3.for循环

for(初始值;循环条件;i的变化){

循环体

}

①执行初始值

②判断循环条件

③如果循环条件是true执行循环体，是false结束循环

④如果执行了循环体，执行i的变化

⑤重新执行②

如果循环条件表达式中有多个条件，以最后一个条件为准

4.break和continue

break：结束循环，后续不会再执行其他的循环

continue：跳过本次循环，继续下一次循环

5.循环嵌套

while，do-while，for循环三者之间可以互相嵌套

Day9

1.函数

parseInt() parseFloat() typeof()

分为系统函数和自定义函数

自定义函数：

function：功能体、函数，可以接受若干个数据，返回处理的结果。用于封装反复执行的代码

(1)创建普通函数

Function 函数名称(){

函数体——要封装的反复执行的代码

}

调用

函数名称();

(2)创建带有参数的函数：

Function 函数名称(参数列表){ //形参

函数体

}

函数名称(参数列表) //实参

参数列表：可以是0个或者多个数据，之间用逗号隔开。穿件函数是的参数称为形参，调用函数是的参数称为实参，调用的时候，实参的值赋给形参。

形参本质上就是一个声明了的变量，但未赋值。

(3)带有返回值得函数

function 函数名(参数列表){

函数体

return 返回值

}

//调用

函数名称(参数列表)

return表示函数执行后，所返回的结果。

注意：

①如果没有return或者return后没有返回值，结果都是undefined。

②return可以返回任意类型的数据。

③return后的所有代码都不会被执行。

**对比return、break和continue**

return返回结果，用于终止函数的执行，常用于函数中。

break用于结束循环，结束switch语句。

continue跳过当前循环，继续下一次循环。

2.变量的作用域

(1)作用域

变量或者函数的可访问范围

分为两种：

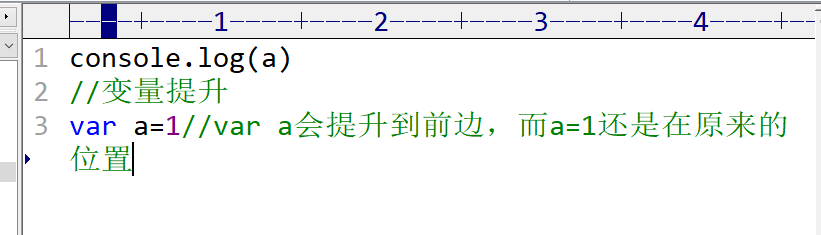
全局作用域：在全局作用域下声明的变量可以在任意位置访问到。

函数(局部)作用域：函数作用域下声明的变量只能在函数内部访问。

注意：在函数内部使用var关键字声明的变量是局部变量，不使用var关键字声明的变量是全局变量。

(2)变量声明提升

JS程序执行前，使用var关键字声明的变量会提升到所在作用域的最前边，但赋值还是在原来的位置



3.函数的作用域

函数和变量类似，也分为全局作用域和函数作用域

全局作用域：在全局作用域下创建的函数可以在任意位置调用。

函数作用域：在函数作用域下创建的函数只能在函数内部调用。

函数声明提升：和变量一样，在程序执行前，把使用function声明的函数提升到最前边

function fun(){

return 1

}

函数调用：fun()获取函数的返回值(return后的值)

函数名称：fun保存的是函数在堆内存中的地址

4.递归的调用

在函数的内部调用自身

递归要有跳出条件，需要结合return使用

Day10

1.匿名函数

没有名字的函数 function(){ }

创建函数——函数声明

function 函数名称(){}

(1)创建函数——函数表达式

var 函数名称=function(形参列表){

函数体

return 返回值

}

调用：函数名称(实参列表)

对比函数声明和函数表达式的区别

函数声明存在函数提升，在任何的位置都可以调用

函数表达式不存在函数提升，必须先创建再调用

(2)匿名函数自调用

创建函数(局部)作用域，防止造成全局污染

(function(){

函数体//就是局部作用域，创建的变量和函数都是不能被外部访问

})()

(3)作为回调函数

把匿名函数作为实参传递给形参，此时的形参就是函数名称

function fn(num){

//num就是函数的名称

num() //调用传递的匿名函数

}

fn(function(){......})

2.系统函数(全局函数)

encodeURI() 对一个url进行编码

decodeURI() 对一个已经编码的url进行解码

parseInt() 将数据转为整型

parseFloat() 将数据转为浮点型

isNaN() 检查一个数据是否为NaN，是->true 不是->false

isFinite() 检查一个数据是否为有限值，是->true 不是->false

1/0->Infinity

eval() 执行字符串中的表达式 eval('1+2') -> 3

3.对象

是一种引用类型数据，存储在堆内存中。

对象：是一组属性(property)和方法/功能(method)的集合

(1)JS中的对象

内置对象：JS提供的

宿主对象：根据不同的执行环境来划分

自定义对象：自己创建的对象

(2)创建自定义对象

①对象字面量(直接量)

②内置构造函数

③自定义构造函数

4.使用对象字面量创建对象

使用大括号{ }创建空对象

属性名和属性值之间用冒号隔开

多组属性之间用逗号隔开

属性名中引号可加可不加，如果出现特殊字符，必须添加引号

var phone={color:'red','made-in':'China'}

5.使用内置构造函数创建对象

var book=new Object() //创建一个空对象

book**.id**=103 //添加属性

book**['title']**='三国演义' //添加属性，属性名必须加引号，如果不加引号，会被认为是变量

6.访问对象中的属性

(1)获取属性值

emp.eid

emp['ename']

如果要获取的属性名不存在，返回undefined

(2)遍历对象中的属性(for-in)

获取到对象中每一个属性名，进而获取属性值

**for**(var key **in** emp){

//emp要遍历的对象

//key要遍历的每一个属性名

//emp[key]通过属性名获取对应的属性值

}

注意：for-in不能遍历预定义(JS默认为每一个对象添加的)的属性

7.检测对象中是否含有某个属性

'属性名' in 对象 // true ->存在 false->不存在

对象.hasOwnProperty('属性名') // true ->存在 false->不存在

对象.属性名 === undefined //true ->不存在 false->存在

8.对象中的方法

var person={

name:'Tom',

say:function(){

this.name //this指代当前的对象

}

}

person.say()

Day11

数组

数组是有多个元素组成的集合

每一个元素就代表一个数据

1.创建数组

(1)数组字面量

[元素1,元素2......]

数组中可以存放任意类型的数据

(2)内置构造函数

new Array(5) //初始化数组的长度为5，可以继续添加第6个元素

new Array(元素1,元素2,......)

访问数组中的元素

数组名称[下标]，下标是从0开始

2.获取数组元素的个数

数组名称.length

使用数组长度添加一个新的元素

数组[数组.length] = 值

3.数组的分类

数组分为**索引数组**和**关联数组**

索引数组：以整数作为下标

关联数组：以字符串作为下标

var arr=[]

arr['eid'] = 1

arr['ename'] = 'tom'

4.遍历数组中的元素

for循环，循环遍历数组的下标

for (var i=0; i<score.length; i++)

{

i 代表数组元素的下标

score[i] 下标对应的元素

}

注意：只能遍历索引数组

for-in

for (var key in arr)

{

key 数组的下标

arr[key] 下标对应的元素

}

注意：既可以遍历索引数组，也可以遍历关联数组。

5.数组中的方法(API)

API-应用程序编程接口：预先定义好的函数/方法

**toString()**  将数组中的元素按逗号分隔成字符串

**join('字符')**  将数组中的元素按照指定的字符分隔成字符串

**concat(arr1,arr2......)** 拼接两个或者更多的数组

**slice(start,end)** 截取数组中的元素，start开始的下标，end结尾的下标，不包括end本身；负数表示倒数

**splice(start,count,value1,value2......)** 删除数组中的元素；start开始的下标，count删除的个数，value1，value2......删除后补充的元素

reverse() 翻转数组中的元素

sort() 对数组中的元素进行排序，默认按照Unicode由小到大

对数字排序

sort(function(a,b){

return a-b //由小到大

return b-a //由大到小

})

Day12

1.数组API(方法)

**push()** 往数组的末尾添加元素，返回数组的长度

**pop()**  删除数组末尾的元素，返回删除的元素

**unshift()** 往数组的末尾添加元素，返回数组的长度

**shift()** 删除数组开头的元素，返回删除的元素

2.二维数组

数组中的每一个元素也是数组

**var arr=[ [],[],[]... ]**

访问二维数组中的元素 arr[下标][下标]

3.字符串操作

包装对象：目的是让原始类型的数据也可以像引用类型一样，具有属性和方法

JS提供了3种包装类型：String、Number、Boolean

将任意数据包装成字符串对象

**new String(true)** 返回object类型

**String(true)** 返回字符串 'true'

(1)转义字符——**\**

转换字符的意义

**\n** 将字符n转义成换行

**\t** 将字符t转义成制表符(tab键)

**\'** 将引号转成普通的字符

......

(2)字符串中常用方法

**toUpperCase()**  将英文字母转为大写

**toLowerCase()**  将英文字母转为小写

**length** 获取字符串的长度

**charAt()** 获取下标对应的字符

**charCodeAt()** 获取摸一个字符对应的Unicode码

**indexOf(value,start)** 查找某个字符串的下标，value是要查找的字符串，start开始查找的下标，默认是0，如果找不到返回-1

**lastIndexOf(value,start)** 查找摸一个字符串最后一次出现的下标，找不到返回-1

**slice(start,end)** 截取字符串，start开始的下标，end结束的下标，不包含end本身；如果end为空，截取到最后

**substr(start,count)** 截取字符串，start开始的小标，count截取的长度，如果count为空，截取到最后

**substring()**

**split(sep)**  按照指定的字符分隔为数组，sep是分隔符

4.匹配模式

作用：用于查找、替换字符串

**replace(value1,value2)** 查找并替换。value1要查找的字符串，value2要替换的字符串；value1可以使用字符串形式，也可以使用正则表达式形式 /china/ig

i -> ignore 忽略大小写

g -> global 全局查找

**match(value)** 用于查找匹配的字符串，返回一个数组

**search(value)** 用于查找满足条件的第一个字符的下标，如果找不到返回-1

5.Math对象

Math对象不需要使用new创建，可以直接使用。

**PI** 获取圆周率

**abs()** 去绝对值

**floor()** 向下取整

**ceil()** 向上取整

**round()** 四舍五入取整

**max()** 取一组数字的最大值

**min()** 取一组数字的最小值

**pow(x,y)** 取x的y次幂

**random()** 取随机 >=0 <1

Day13

1.字符串方法

**String.fromCharCode(index)** 获取任意Unicode码对应的字符

**substring(start,end)**  截取字符串，start开始的下标，end结尾的下标，不包含end本身；如果两个参数是负数，自动转成0

2.Date对象

用于对日期和时间进行存储和计算

(1)创建Date对象

new Date('2018/11/11 10:20:30')

new Date(2018,10,11,10,20,30)//第2个参数月份的范围0~11

new Date() 存储当前的系统时间

new Date(100\*60\*60\*24) 存储的是距离计算机元年的毫秒数对应的日期时间

(2)获取Date对象的信息

**getFullYear()/getMonth()/getDate()/getHours()/getMinutes()/getSeconds()/getMilliseconds()/getDay()(星期0~6)/getTime()(距离计算机元年毫秒数)**

(3)转为本地字符串

toLocaleString() // 年-月-日 时:分:秒

toLocaleDateString() // 年-月-日

toLocaleTimeString() // 时:分:秒

(4)设置Date对象的信息

setFullYear()/setMonth()/setDate()/setHours()/setMinutes()/setSeconds()/setMilliseconds()/setTime()

3.Number对象

new Number(值) 创建对象

将一个数据转为Number对象，本质还是数字

Number.MAX\_VALUE 获取计算机能存储的最大值

Number.MIN\_VALUE 获取计算机能存储的最小值

toFixed(n) 保留小数点后n位

toString() 将数字转为字符串类型

数字+'' //隐式转为字符串类型

4.Boolean对象

new Boolean(值) 创建布尔对象，本质上讲数据转为布尔型

Boolean(值) 转为布尔型

!!值 隐式讲数据转为布尔型

toString() 将布尔型数据转为字符串

**null的作用**：用于释放(销毁)一个引用类型的数据

5.错误处理

SyntaxError 语法错误，错误的使用了中文，缺少括号...程序不会执行

ReferenceError 引用错误，使用未声明的变量

TypeError 类型错误，错误的使用了数据的类型，错误的使用括号

RangeError 范围错误，参数的使用超出了范围

引用错误、类型错误、范围错误出现后，会影响后续代码的执行

**异常处理**

try{

尝试执行的代码，可能出现错误

}catch(err){

err:捕获的错误信息

处理错误的方案

}

6.ES6新特征

**E**CMA**S**cript6

(1)块级作用域——局部作用域

使用let关键字声明的变量，只能在块级作用域下访问，不能被外部访问

块级作用域：{ }、for、while、do-while都是块级作用域

(2)箭头函数

是回调函数的另一种写法，和匿名函数不完全一致

sort((a,b)=>{

return a-b

})

如果箭头函数的函数体中只有一行代码，并且是以return形式，可以简化为

sort((a,b)=>a-b)

day14

1.ES6新特征

(1)函数中的参数

ES6允许为形参设置默认值，如果没有传递实参，自动调用形参的默认值

(2)模板字符串

`在此之间可以写任意形式的代码 ${JS语法}`

2.nodejs概述

nodejs基于谷歌V8引擎(JS解释器)，运行在服务器端的语言，基于JS。

**对比JS和nodejs**

http://nodejs.cn

https://nodejs.org

(1)JS运行在浏览器端，存在多个浏览器，容易产生兼容性的问题；而nodejs在服务器端只有一个运行环境，不存在兼容性。

(2)两者都有内置(ES)对象、自定义对象、宿主对象(根据执行环境的不同)

(3)JS用于网页中的交互效果，而nodejs用于服务器的数据库操作、文件操作...

**nodejs的执行方式**

脚本模式 node C:/xampp/.../1.js

交互模式

node 回车

两次ctrl+c 或者输入.exit

3.全局对象

nodejs：global

在交互模式下，声明的变量或者创建的函数都属于全局对象下的，可以使用global访问；例如 var a = 1; global.a

在文件中声明的变量或者创建的函数都属于是局部作用域下的，不能使用global来访问

JS：window

在浏览器下，文件中声明的变量或者创建的函数都输域是全局作用域下的，可以使用全局对象访问；例如 var a = 1; window.a

(1)console对象

global.console.log() 打印消息

global.console.info() 打印消息

global.console.warn() 打印警告消息

global.console.error() 打印错误消息

global.console.time('自定义字符串') 开始计时

global.console.timeEnd('自定义字符串') 结束计时

自定义字符串前后要保持一致

(2)process对象

查看当前计算机的进程

process.arch 查看当前CPU架构 x64

process.platform 查看当前的操作系统 win32

process.env 查看当前计算机的环境变量

process.version 查看当前nodejs的版本号

process.pid 查看当前的进程编号

process.kill() 杀死某一个编号的进程

(3)Buffer对象

缓冲区：在内存中存储数据的区域，存储网络传输是的资源

创建buffer

var buf = Buffer.alloc(5,'abcde')

将buffer存储的数据转为普通字符

buf.toString()

(4)全局函数

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

①一次性定时器

开启

var timer = setTimeout(回调函数,间隔的时间)

当间隔的时间到了，执行回调函数；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,间隔的时间)

当间隔的时间到了，执行回调函数

清除

clearInterval(timer)

③立即执行

process.nextTick(回调函数)

在当前事件循环的结尾执行

var timer = setImmediate(回调函数)

clearImmediate

在下一个事件循环的开头执行

4.模块

模块可以理解为是一个功能体

在nodejs下模块分为自定义模块、核心模块(官方提供)、第三方模块

在nodejs下任意一个文件都是一个模块，文件中的代码默认是被一个构造函数所包含，

(function(exports,require,module,\_\_dirname,\_\_filename){

//程序员编写的代码

})

\_\_dirname 当前模块(文件)的完整路径

\_\_filename 当前模块(文件)的完整路径和文件名称

require() 引入一个模块

module 指代当前的模块对象

module.exports 当前模块的导出对象(公开)，可以供其他的模块使用的属性和方法

exports 等价于module.exports

day15

1.模块

|  |  |  |
| --- | --- | --- |
|  | 以路径开头 | 不以路径开头 |
| 文件模块 | require('./circle.js')  常用语用户自定义的模块，如果后缀名是js的话，可以省略后缀名 | require('querystring')  常用于引入官方提供的核心模块 |
| 目录模块 | require('./02\_2')  在02\_2目录下自动引入文件index.js；或者使用package.json文件声明main属性，来指定要引入的文件名称 | require('04\_2')  要求引入的目录放在当前目录下node\_modules中。如果查找不到，则到上一级目录下查找，直到顶层目录。常用于第三方模块 |

2.包和npm

npm:Node Package Manage

包(package)：就是一个目录模块，里面包含有多个文件，其中有一个文件命名为package.json的文件，是包说明文件

自动下载 http://www.npmjs.com

切换到下载的目录

①cd 完整路径

change directory

②在要下载的目录下，按住shift键，单机鼠标右键->在此处打开powershell窗口

使用npm安装第三方包

npm install 包的名称

3.核心模块

是nodejs官方提供的模块，可以直接引入，不需要创建

(1)查询字符串模块——querystring

浏览器向服务器发送请求，传递数据的一种方式

http://www.codeboy.com/product\_details.html?lid=5&name=dell

parse() 将查询字符串解析为对象

stringify() 将对象转换成查询字符串

(2)URL模块

parse() 将url解析为对象

protocol 协议

hostname 主机(域名/IP地址)

port 端口

pathname 文件在服务器上的路径

query 查询字符串

format() 将对象转换成url

query属性对应的是对象

(3)文件系统模块——fs

①fs.stat(path,callback)/fs.statSync(path) 查看文件的状态，通过回调函数来获取结果

path 要查看的文件的路径

callback 回调函数

err 如果查看失败的错误信息

stats 文件的状态信息

isDirectory() 是否为目录

isFile() 是否为文件

**对比同步和异步的区别**

同步会阻止后续代码的执行，只有方法执行完，才能继续执行后边的代码；是通过返回值来获取结果

异步不会阻止后续代码的执行，把执行的结果放到整个程序的最后执行；是通过回调函数来获取结果

②fs.mkdir(path,callback)/fs.mkdirSync(path)创建目录

path 要创建的目录的路径

callback 回调函数，只有一个参数

err 如果创建失败的错误信息

③fs.rmdir(path,callback)/fs.rmdirSync(path)删除目录

path 要删除的目录的路径

callback 回调函数，获取删除的结果

err 如果删除失败的错误信息

④fs.readdir(path,callback)/fs.readdirSync(path) 读取目录中的文件

callback

files 读取的文件，返回数组

⑤fs.writeFile(path,data,callback) 写入文件/创建文件

data 要写入的数据

如果文件不存在，则创建文件；如果文件已经存在，则清空文件中的内容，然后写入

day16

1.文件系统模块——fs

fs.unlink(path,callback)/fs.unlinkSync(path) 删除文件

fs.existsSync(path) 判断文件是否存储

存在true不存在false

fs.readFile(path,callback)/fs.readFileSync(path) 读取文件，返回的数据是buffer形式

fs.appendFile(path,data,callback)/fs.appendFileSync(path,data) 追加写入，如果文件不存在则创建文件，如果文件已经存在，则在末尾写入数据

2.http协议

是浏览器和web服务器之间的通信协议

(1)通用头信息

Request URL：请求的URL，要向服务器端请求那个文件

Request Method：请求的方法 get/post

Status Code：响应的状态码

2\*\* 服务器成功的响应

3\*\* 响应的重定向，跳转到另一个网址

4\*\* 客户端错误

5\*\* 服务器端错误

Romote Address：请求的远程服务器的IP地址和端口

(2)响应头信息

Connection：keep-alive；连接的方式：持续连接

**Content-Type**：text/html；响应的文件类型

Content-Encoding：响应的文件压缩形式

Transfer-Encoding：响应时的传输方式，chunked(分段传输)

**Location**：响应时跳转的URL，通常结合着300系列状态码

(3)请求头信息

Accept：客户端接受的文件类型有哪些

Accept-Encoding：客户端接受的文件压缩形式

Accept-Language：客户端接受的语言类型

Connection：客户端和服务器的连接方式，持续连接

(4)请求主体

可有可无，客户端向服务器端传递数据

3.http模块

可以模拟浏览器想服务器端发请求，也可以创建web服务器

(1)模拟浏览器

http.get(url,callback)

get 请求的方法

url 请求的网址

callback 回调函数，用来获取服务器短的响应

res 响应的对象

res.statusCode 获取响应的状态码

res.on('data',(buf)=>{ })

使用事件来获取服务器端响应的数据

bur是服务器端响应的数据，格式为buffer数据

(2)创建web服务器

var server = http.createServer() 创建web服务器

server.listen(3000) 分配端口，监听3000端口的变化

server.on('request',(req,res)=>{ })

//接收浏览器的请求，是一个事件，一旦有请求，自动执行

req 请求的对象

url 请求的路径，显示端口后的部分

method 请求的方法，直接通过地址栏默认使用get方法

headers 请求的头信息

res 响应的对象

write() 响应的内容为文本形式，向浏览器中写入文本

writeHead(302,{ }) 设置响应的状态码和响应的头信息；如果要跳转需要设置Location属性

end() 响应结束

3.express框架

基于nodejs，用于构建web服务器的框架

官网：www.expressjs.com.cn

安装：npm install express

//引入express第三方模块

const express = require('express')

//创建web服务器

var server = express()

server.listen(3000)

(1)路由

浏览器向web服务器发来请求，web服务器要根据**请求的方法**和**请求的URL**来**作出响应**

路由三要素：请求的方法、请求的URL、响应的内容

响应的对象(res)

res.send() 响应文本，只能响应一次send；如果是数字认为是状态码

res.sendFile() 响应文件，必须使用绝对路径(\_\_dirname)

res.redirect() 响应的重定向

day17

1.路由中的请求对象

req.method 获取请求的方法

req.url 获取请求的URL

req.headers 获取请求的头信息

req.query 获取请求时以查询字符串形式传递的数据，返回格式为对象

2.post和get请求

get请求以查询字符串的形式传递数据，服务器端使用**req.query**获取数据，结果是对象

post请求是通过表单提交(form)的方式来传递数据，服务器端通过事件形式获取数据(后期会有简单的方法)

req.on('data',(buf)=>{

//当有数据传输的时候，格式为buffer数据

var str = buf.toString()

//使用查询字符串讲数据解析为对象

var obj = querystring.parse(str)

console.log(obj)

//获取数据后，响应到浏览器

res.send('注册成功，用户名'+obj.uname)

})

3.使用路由传递数据——路由传参

设置路由中接收的名称

server.get('/detail/:lid/:pname',(req,res)=>{

//获取浏览器传递数据

console.log(req.params)//返回对象

res.send('这是商品详情')

})

浏览器请求方式

http://127.0.0.1/detail/5/dell

5就是传递的数据，使用lid来接收

4.路由器

路由在使用过程中，不同的模块可能出现相同的URL，把同一个模块下的路由挂载到特定的前缀

例如：商品模块下的路由挂载到/product，访问形式/product/list，用户模块下的路由挂载到/user，访问形式/user/list

路由器就是一个js文件，把同一模块下的路由放到一起

//引入express

const express = require('express')

//创建空的路由器

var router = express.Router()

//添加路由

//请求方法：get 请求URL：/list

router.get('/list',(req,res)=>{

res.send('这是用户列表')

})

//导出路由器router

module.exports = router

**在web服务器下使用路由器**

//引入user.js路由器模块

const userRouter = require('./user')

//把引入的用户路由器挂载到/user

server.use('/user',userRouter)

5.中间件

中间件的作用是为主要的业务逻辑所服务，分为5个

应用级中间件、路由级中间件、内置中间件、第三方中间件、错误级中间件

(1)应用级中间件

每一个中间件就是一个函数，需要配合其他的中间件或者路由使用。

server.use(回调函数) 拦截所有的路由

server.use('/detail',回调函数) 拦截特定的路由

(2)路由级中间件

用于在服务器中将路由器挂载到特定的URL

server.use('/user',userRouter)

(3)内置中间件

在express中只有一个内置的中间件

server.use(express.static('要托管的目录'))

托管静态资源到某一个目录，如果浏览器端要请求静态资源，则自动到这个目录下查找。

静态资源：html、css、客户端js、图像......

day18

1.第三方中间件body-parser

将post请求的数据解析为对象

使用中间件

//urlencoded()将post请求数据解析为对象

server.use(bodyParser.urlencoded({

extended:false//extended是否使用扩展的qs模块将查询字符串解析为对象；false不使用扩展的qs，使用核心模块querystring

}))

在路由中获取数据

req.body 返回一个数据的对象格式

浏览器向服务器端传递数据的方式

get req.query

post req.body

路由传参 req.params

2.MySQL模块

增 INSERT INTO emp VALUES(NULL,'tom'......);

删 DELETE FROM emp WHERE uid=5;

改 UPDATE emp SET uname='jerry',sex=1 WHERE uid=5;

查 SELECT \* FROM emp;

mysql.exe -h127.0.0.1 -P3306 -uroot -p

(1)普通连接MySQL

//创建连接对象

var connection = mysql.createConnection({

host:'127.0.0.1',

port:'3306',

user:'root',

password:'',

database:'tedu'

})

//执行连接

connection.connect()

//执行sql语句

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

if (err) throw err

console.log(result)

})

//sql语句执行完，关闭连接

connection.end()

(2)使用连接池

//使用连接池连接MySQL

//创建连接池对象

var pool = mysql.createPool({

host:'127.0.0.1',

port:'3306',

user:'root',

password:'',

database:'tedu',

connectionLimit:20 //设置连接池的数量

})

//执行sql语句

//取一个连接执行sql语句

pool.query('SELECT \* FROM emp WHERE eid=5',(err,result)=>{

if(err) throw err

console.log(result)

})

day19