

NODEJS01

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}`

练习：创建一个员工对象，包含姓名、性别、生日、工资；使用模板字符串打印员工的信息。

'姓名: **，性别: **，生日: **，工资: **'

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。

<http://nodejs.cn> 中文

<http://nodejs.org> 英文

对比 JS 和 NODEJS

(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`

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属于是全局作用域下的，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。

练习：使用计时查看 `for`, `while`, `dowhile` 循环 10000 的耗时。

(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：在内存中存储数据区域，存储网络传输时的资源

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/encodeURIComponent/decodeURI/is`

`NaN/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);`

当间隔时间到了，执行回调函数；

清除

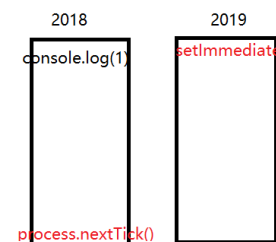
`clearInterval(timer);`

练习：使用周期性定时器每隔 3 秒打印 hello，打印三次后，清除定时器。

③立即执行(了解)

```
process.nextTick(回调函数)
//在当前事件循环的结尾(2018 年底)执行
```

```
var timer=setImmediate(回调函数)
clearImmediate(timer)
//在下一个事件循环的开头(2019 年初)执行
```



4. 模块

模块可以理解为一个功能体(积木块)

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。

以下代码中红色代码都是 NODEJS 自动为每一个文件添加的

```
(function(exports,require,module,__dirname,__filename){
  //程序员编码写的代码
})
```

__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)，可以供其它的模块使用的属性和方法
exports 等价于 **module.exports**

(2)练习：

创建两个模块 main.js(主模块), circle.js(功能模块)；在功能模块中创建两个函数，传递 1 个参数，分别获取圆的周长(getLength)和面积(getArea)，导出这两个函数；在主模块中引入功能模块，调用两个方法。

NODEJS02

1. 模块

	以路径开头	不以路径开头
文件模块	require('./circle.js') 常用于用户自定义的模块，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。(必须双引号，不加符号)	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。

练习：创建模块 03_1.js，引入当前目录下的 03_2 目录模块；在 03_2 下创建 test.js，导出一个函数 fn(打印两个数字相加)，在 03_1.js 中调用

练习：在 05 目录下创建模块 05_1.js，引入不带路径的目录模块 05_2，05_2 目录中含有 hello.js 文件(打印一句话)。

2. 包和 NPM

NPM: Node Package Manage

包(package): 就是一个目录模块，里边包含有多个文件，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，单击鼠标右键->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

练习：把百度搜索时的查询字符串解析为对象，获取关键词。

ie=utf-

8&f=8&rsv_bp=0&rsv_idx=1&tn=baidu&wd=电脑

(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

练习：浏览器请求的 URL

<https://www.tmooc.cn:3000/web/1810.html?id=10&name=tom>

获取 URL 中传递的 sid 和 name 的值

08_exercise.js

(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，里边有两个参数

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；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) 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，然后写入；

如果文件已经存在，则清空文件中的内容，然后写入。

练习

创建目录 mydir，在该目录下创建文件

1.txt，写入 1；创建文件 2.txt，写入 2；读取

mydir 下所有的文件；删除 mydir（自学删除文件 unlink）。

整个过程全部使用同步方法。

NODEJS03

1.文件系统模块——fs

①fs.unlink(path,callback)/fs.unlinkSync(path)

删除文件

②fs.existsSync(path) 判断文件是否存储

存在 true 不存在 false

练习：判断文件 num.txt 是否存在，如果不存在则创建，并初始化数字 0；在原来的数字上加 1。

③

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

读取文件

返回的数据是 buffer 形式。

④fs.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。

练习：使用文件操作来创建文件 user.txt，每次写入一个对象，

{uid:1, uname:'tom', upwd: '123456'}

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, 要向服务端请求哪个文件。

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

Remote 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

可以模拟浏览器向服务器端发请求, 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, 用来获取服务器端的响应

`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

`buf` 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

练习: 创建 web 服务器, 监听 3001 端口, 使用事件监听浏览器的请求, 打印请求的方法、URL;

`http://127.0.0.1:3001/admin/login.html`

`http://localhost:3001/member/shopping.html`
`05_server.js`

`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

练习: 创建 web 服务器, 监听 3000 端口; 接收浏览器端的请求; `06_server.js`

`/login` 响应文本 `this is login page`

`/member` 响应文本 `this is member`

`page`

`/` 跳转到 `/member`

如果以上都没有匹配的, 响应文本 `404 not found`

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`

```
const express=require('express');
```

```
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

练习

使用 express 创建 web 服务器, 创建以下路由

```
get  '/index'    发送文本"这是首页"
get  '/login'    发送文件 "login.html"
post '/register'  发送文本"注册成功"
get  '/'         跳转到  /index
```

NODEJS04

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。

练习: 创建文件 03_post.js, 创建 web 服务器, 新建路由(get /reg), 响应一个注册文件(reg.html)

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(现阶段)的方式传递数据, 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)=>{
  获取的结果是 buffer 数据, 需要使用查询字符串解析为对象
})
```

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',(req,res)=>{
  req.params //获取路由传递的数据, 格式为对象
})
```

浏览器请求方式

http://127.0.0.1/detail/5

5 就是传递的数据, 使用 lid 来接收

练习: 创建购物车的路由, 请求的 URL:

/shopping, 请求的方法 get, 传递商品的价格(price)和名称(pname)。

用户模块

用户列表 /list 详情 /detail 删除 /delete

/user/list /user/detail /user/delete

商品模块

列表/list 详情/detail 删除 /delete

/product/list /product/detail /product/delete

4.路由器

路由在使用过程中, 不同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。

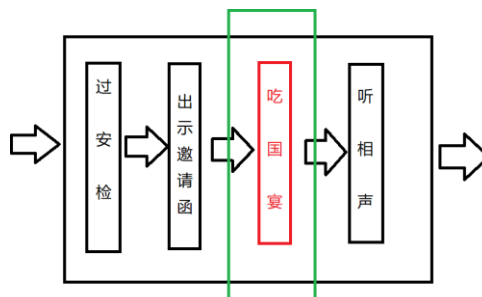
```
const express=require('express');
var router=express.Router(); //创建空的路由器对象
router.get('/list', (req,res)=>{ }); //往路由器中添加路由
module.exports=router;
```

在 web 服务器下使用路由器

```
const userRouter=require('./user.js'); //引用路由器模块
server.use('/user', userRouter); //把路由器挂载到/user 下, 访问形式 /user/list
```

练习: 创建商品模块路由器(product.js), 添加路由商品列表(list)、商品删除(delete)、商品添加(add), 在 web 服务器引入, 并挂载到/product

5.中间件



中间件的作用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

练习：创建路由(get, /view)响应当前的浏览次数，每次请求，响应的次数加 1。

在函数外初始化一个变量，设置值为 0；在中间件中实现变量加 1，在路由中响应变量。

(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'))`

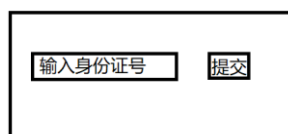
托管静态资源到某一个目录，如果浏览器端要请求静态资源，则自动到这个目录下查找。

静态资源：html、css、客户端 js、图像...

练习：将静态资源托管到 files 目录下，查看如果两个静态目录下有相同名称的文件，显示哪一个？

练习：

创建 web 服务器，托管静态文件（如下图），点击查询，服务器端获取输入的身份证号(中间件)，截取出生的年月日和性别；在路由中响应给浏览器。



The image shows a rectangular box containing a form. Inside the box, there is a text input field with the placeholder text '输入身份证号' (Enter ID Number) and a button labeled '提交' (Submit).