Node.js学习笔记

## 一、创建一个简易程序

1、在项目里创建一个js文件（server.js），添加Node.js的基本请求语句。

var http = require("http");//引入http模块

http.createServer(function (request, response) {//创建服务器

//发送HTTP头部信息

// HTTP 状态值: 200 : OK

// 内容类型: text/plain

response.writeHead(200, { 'Content-Type': 'text/plain' });

// 发送响应数据 "Hello World"

response.end('Hello World\n');//页面响应的结果

}).listen(8888);// 打开页面的端口

console.log('12343546');//管理器中显示的结果

2、验证：

（1、）cmd=》找到项目所在硬盘（如F:）

（2）、F:\>创建的js（server.js）所在项目的位置

（3）、js所在位置>node server.js(运行检测js文件是否成功运行，成功运行则显示console.log()中的内容)

## 二、Node.js REPL（交互式解释器）

node.exe，类似一个终端，可实现读取，执行，打印，循环

## 三、File System 文件系统模块

（一）fs模块用来对于本地文件系统进行操作。使用require('fs')访问这个模块。所有的方法都提供了异步和同步两种方式。

（二）异步操作的版本都会使用一个回调方法作为最后的一个参数。当操作完成的时候，该回调方法会被调用。而回调方法的第一个参数总是保留为操作时可能出现的异常。如果操作正确成功，则第一个参数的值是null或undefined。

（三）同步操作的版本方法名称则是在对应的异步方法之后加上一个Sync。如异步的rename()方法的同步版本是renameSync()。

1、fs.rename(p1,p2,[callback])，fs.renameSync(p1,p2)：

p1=oldName，p2=newName。

2、fs.truncate(fd,len,[callback])，截断某个文件。如果出现异常，回调函数会接收一个参数（err），否则不传递任何参数。

3、fs.chmod(path,mode,[callback]),修改文件权限。主要是第二个参数，实际上是：-rwxrwxrwx，三个一组，r:4，w:2，x:1，-:0。

4、fs.stat(path,[callback]),fs.lstat(path, [callback]),fs.fstat(fd, [callback]) 和 fs.statSync(path)、fs.lstatSync(path)、fs.fstatSync(fd)。读取文件元信息，回调函数将返回两个参数(err, stats)，其中stats是fs.Stats的一个对象.者的区别在于：  
第一个是返回文件的信息，参数是路径  
第二个是和第一个一样，当路径是文件链接时，返回这个链接文件的信息  
第三个是传递参数为fd文件描述符。

5、fs.link(srcpath, dstpath, [callback])、fs.linkSync(srcpath, dstpath) 和 fs.symlink(linkdata, path, [callback])、fs.symlinkSync(linkdata, path)   
建立文件链接，除非回调函数执行过程出现了异常，否则不会传递任何参数。link和symlink的区别是： link 创建的是hard link 所谓硬链接 symlink创建的是symbolic link 所谓符号链接。硬链接就是备份，软连接就是快捷方式

6、fs.readlink(path,[callback])、fs.realpath(path,[callback])、fs.unlink(path,[callback]) 和 fs.readlinkSync(path)、fs.realpathSync(path)、fs.unlinkSync(path)。readlink，回调函数返回两个参数（err,resolvedPath）,resolvedPath为解析后的文件路径。这3个函数分别是：1、readlink：读取链接源地址。2、realpath：根据相对地址转换为绝对地址。3、unlink：删除某一个文件链接。

7、fs.rmdir(path,[callback]),fs.mkdir(path,[callpath]),fs.readdir(path,[callback])（异步方法）。rmdir：删除目录。mkdir：建立目录。readdir：读取目录中内容。回调函数接受两个参数（err，files），其中files参数是保存了目录中所有文件名的数组（'.'和'..'除外）。

8、fs.close(fd,[callback])、fs.open(path,flags,[mode],[callback])(异步方法)。fs.open与C语言标准库中的fopen函数类似。它接受两个必选参数，path为文件的路径，flags可以是以下值：

r（读取模式打开），

r+（读写模式打开），

w（写入模式打开，如不存在则创建），

w+（读写模式，不存在则创建），

a（追加模式，如不存在则创建），

a+（读取追加模式，如不存在则创建）

mode参数用于创建文件时给文件指定权限，默认是0666.回调函数将会传递一个文件描述符fd。

9、fs.write(fd,buffer,offset,length,position,[callback])（异步方法）。将buffer缓冲器内容写入fd文件描述符，offset和length决定了将缓冲器中的哪部分写入文件。position指明将数据写入文件从头部算起的偏移位置，若position为null，数据将从当前位置开始写入。

var fs = require('fs');  
 fs.open('./ex.js', 'w', function(err , fd){  
     var buf = new Buffer(256);  
     var len = buf.write('12321321321321', 0)  
     fs.write(fd, buf, 0, len, 0, function(err, w){  
        console.log(w)  
    })  
})  
**注意：buffer尺寸的大小设置最好是8的倍数，效率较高。**

注意：写完后要关闭它！

10、fs.read(fd,buffer,offset,length,position,[callback])和fs.readFile(filename,[encoding],[callback])，前者是读取文件描述符和buffer的方法，后者是读取文件全部内容，比如输出html模板，或者css文件等。fs.read回电函数接受两个参数（err,bytesRead），bytesRead返回读取到的字节数。fs.readFile的回调函数将传入两个参数（err，data），其中data为文件内容，如果没有设置编码，那么将返回原始内容格式的缓冲器。

11、fs.writeFile(filename,data,encoding='utf8',[callback])，写入文件功能。

12、fs.Stats：fs.stat()和fs.lstat()方法返回的对象为此类型。

（1）stats.isFile()，（2）stats.isDirectory()，（3）stats.isBlockDevice()，

（4）stats.isCharacterDevice()，（5）stats.isSymbolicLink()(仅对fs.lstat()有效)，

（6）stats.isFIFO()（先入先出队列），（7）stats.isSocket()

13、fs.createReadStream(path,[options])，返回一个新的可读流对象options是包含如下默认值的对象：{flags:'r',encoding:null,fd:null,mode:0666,bufferSize:64\*1024}，如果不想读取文件的全部内容，可以在options参数中设置start和end属性值以读取文件中指定范围的内容。start和end包含在范围中（闭合集），取值从0开始。这两个参数需要同时设置。

14、fs.createWriteStream(path,[options])，包含默认值对象：{flags:'w',encoding:null,mode:0666}

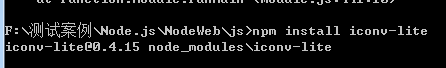
（四）解决中文文字乱码问题：

第一种情况：

var iconv = require('iconv-lite');//获取编解码模块（Iconv是node的编码插件模块）

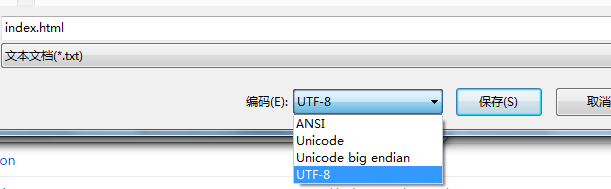
iconv.decode(data, 'gb2312')

如出报错问题：Cannot find module 'iconv-lite' 则说明为安装这个模块包，进行安装即可：cmd=>任务管理器=》cd 项目目录=》npm install iconv-lite。安装成功会出现。



第二种情况：

文件内容全文是乱码，可将文件用txt记事本打开，另存为时将编码选择为“utf-8”即可；



## 四、Node的模块概念

（一）、不同的功能组件被划分为不同的木块，可根据自己的需求引用不同的模块。每个模块都会暴露一些公共的方法或属性，需要有文件引用。

（二）、加载方法：模块名称以“./”开始，来加载与当前js文件同一目录的其他模块（如：./myModule.js）。Node.js提供了exports和require两个对象，其中exports是模块公开的接口，require用于从外部获取一个模块接口，即所获取模块的exports对象。可实现一个功能一个js模块。

对象封装有两种方法：1、将对象作为模块访问的接口：

//exports是模块公开接口，将world作为模块访问的接口

exports.world = function () {

var name = "私有属性输出：Darren";

this.location = "共有属性输出：Beijing";

this.showLog = function () {

console.log('共有方法showLog输出信息：Hi Darren!');

};

console.log("world作为模块接口被访问！");

}

2、将对象封装到模块中(module.js)

function showInfo(){

......

}

module.exports = showInfo;//将对象封装到模块中

外部引用：var module=require('./module')

var showInfo=new module();

注：外部调用的方法名必须与服务器封装的方法名一致！(下例中runn是服务器封装名称，start是server.js的方法名字，这是将server的start方法封装到runn中，所以外部引用需要写的是server.runn,而不是server.start,其中server是var server=require（'./server.js'）)



（三）、使用模块内部功能方法：代码中封装了模块的内部处理逻辑，一个模块一般都会暴露一些公开的方法或属性给其他的人使用。模块的内部代码需要把这些方法或属性给暴露出来。

例：var name='Darren';

this.location="BeiJing";

this.showLog=function (){

console.log('Hello Darren!');

}

这三类是私用属性，共有属性和共有方法。而私有属性和方法在模块以外是获取不到的，要变为共有属性和方法，需要在前面加this关键字。

## 五、Node.js EventEmitter

Node.js所有的异步I/O操作在完成时都会发送一个事件到事件队列。所有产生事件的对象都是events.EventEmitter的实例。

events模块只是提供了一个对象:events.EventEmitter。EventEmitter的核心是事件触发和事件监听功能的封装。

var events = require('events');

var emitter =new events.EventEmitter();//创建事件触发器

emitter.on("myEvents", function (msg) {//添加监听事件

console.log(msg);

})

setTimeout(function () {

emitter.emit('myEvents', 'myModule输出信息：Hello World!');//传参，触发事件

},1000);

## 六、Node.js Buffer(缓冲区)

（一）、Buffer类是随Node内核一起发布的核心库，为Node.js带来了一种存储原始数据的方法，可以让Node.js处理二进制数据。当需要在Node.js中处理I/O操作移动的数据时，就有可能使用Buffer库。

（二）、创建Buffer实例：

（1）var buf=new Buffer(10)(指定长度为10字节)；

（2）var buf=new Buffer([10,20,30,40])(通过给定的数组创建实例)；

（3）var buf=new Buffer("15456454","utf-8")(通过一个字符串创建实例)

（三）、写入缓冲区语法：buf.write(string,offset,length,encoding)

string : 写入缓冲区的字符串；offset : 缓冲区开始写入的索引值，默认为0；

length : 写入的字节数，默认为buf.length；encoding : 默认使用utf-8。

（四）、读取缓冲区数据的语法：buf.toString(encoding,start,end)

encoding : 使用的编码，默认utf8；start : 指定开始读取的索引位置(默认0)；

end : 结束位置，默认为缓冲区的末尾。（可直接在node.exe（REPL）交互解释器中运行查看结果）。

buf = new Buffer(26);

for (var i = 0 ; i < 26 ; i++) {

buf[i] = i + 97;

}

console.log(buf.toString('ascii')); // 输出: abcdefghijklmnopqrstuvwxyz

console.log(buf.toString('ascii', 0, 5)); // 输出: abcde

console.log(buf.toString('utf8', 0, 5)); // 输出: abcde

console.log(buf.toString(undefined, 0, 5)); // 使用 'utf8' 编码, 并输出: abcde

转化为JSON对象：buf.toJSON()。

JSON格式内容和String格式内容互相转换：

（1）、JSON转换为String：JSON.stringify(json)

//JSON.stringify(obj)将JSON转为字符串。

var json = { aa: ['sdddssd'],

bb: [ '892394829342394792399', '23894723984729374932874' ],

cc: ['11111111111111'] }

var string = JSON.stringify(json);

//结果string: {"aa":["sdddssd"],"bb":["892394829342394792399","23894723984729374932874"],"cc":["11111111111111"]}

（2）、String转换为JSON：JSON.parse(string)

//JSON.parse(string)将字符串转为JSON格式

JSON.parse(string);

//结果JSON:{ aa: ['sdddssd'],bb: [ '892394829342394792399', '23894723984729374932874' ],cc: ['11111111111111'] }

## 七、Node.js Stream(流)

（一）、Stream是一个抽象接口，Node中很多对象实现了这个接口，例如对http服务器发起请求的request对象就是Stream，还有stdout（标准输出）。

（二）、Node.js Stream有四种流类型：Readable（可读），Writable（可写），Duplex（可读可写），Transform（操作被写入数据，然后读出结果）。

（三）、所有的Stream对象都是EventEmitter实例。常用的事件有：

data：有数据可读时触发；end：没有更多数据可读时触发；

error：在接收和写入过程中发生错误时触发；finish：所有数据已被写入底层系统时触发。

（四）、从流中读取数据及将读取的数据写入流中：

var serv = http.createServer(function (request, response) {

//创建可读流

var readStream = fs.createReadStream('../doc/testTxt.txt');

//创建可写入流

var writeStream = fs.createWriteStream('../doc/writeStream.txt');

//添加监听事件

readStream.on('data', function (chunk) {

if (!response.write(chunk)) {//判断写缓冲区是否写满(node的官方文档有对write方法返回值的说明)

readStream.pause();//如果写缓冲区不可用，暂停读取数据

}

chunks.push(chunk);//将data保存到数组中

data += chunk;

size += chunk.length;//获取数组长度，方便将缓冲区数据合并

});

readStream.on('end', function () {

console.log("data content:" + data);

var buf = Buffer.concat(chunks, size);//合并缓冲区数据

var result = iconv.decode(buf,"gb2312");//转换中文信息

**writeStream.write(result)**;//将读出的数据写入新的文件流

response.end();

});

readStream.on('error', function (err){

console.log(err.stack);

});

response.on("drain", function () {//写缓冲区可用，会触发"drain"事件

readStream.resume();//重新启动读取数据

});

});

serv.listen(8188);

（五）、管道流（pipe）：将A文件的内容通过管道流（pipe）复制到B文件，与上面的方法类似，pipe是Stream类中的方法。

var readStream = fs.createReadStream('../doc/testTxt.txt');

//创建可写入流

var writeStream = fs.createWriteStream('../doc/writeStream.txt');

//管道操作,读取testTxt文件内容，并将内容写入到writeStream文件

readStream..pipe(writeStream);

（六）、链式流：链式是通过连接输出流到另外一个流并创建多个对个流操作链的机制。链式流一般用于管道操作。

//获取解压缩文件模块请求

var zlib = require("zlib");

//压缩文件方法

var gzip=zlib.createGzip();

fs.createReadStream("../doc/testTxt.txt").pipe(gzip).pipe(fs.createWriteStream("../doc/testTxt.txt.gz"));

//解压文件方法

var gunzip = zlib.createGunzip();

var readFile = fs.createReadStream("zlib\_file.gz");

var writeFile = fs.createWriteStream("zipFile.txt");

readFile.pipe(gunzip).pipe(writeFile);

console.log("解压文件成功！");

（七）、解压缩文件模块

require（“zlib”），这是获取解压缩文件功能库，压缩缓冲区Zlib中有几个函数方便解压缓冲区：gzip/gunzip:标准gzip压缩。deflate/inflate：基于Huffman编码的标准deflate压缩算法。deflateRaw/inflateRaw：针对原始缓冲区的deflate压缩算法。

## 八、Node.js函数

在javascript中，一个函数可以作为另一个函数的接收参数。

例如：

//把say函数作为person函数的第一个变量进行了传递。这里返回的不是say的返回值，而是say本身！

//say 就变成了person 中的本地变量activity，person可以通过调用activity（）带括号形式的来使用say 函数。

//因为say有一个变量，person 在调用activity（）时可以传递这样一个变量。

function say(str) {

console.log(str);

}

function person(activity, somthing) {

activity(somthing);

}

person(say,"Hello World!");

匿名函数：person(function (str){console.log(str)},"Hello World!");

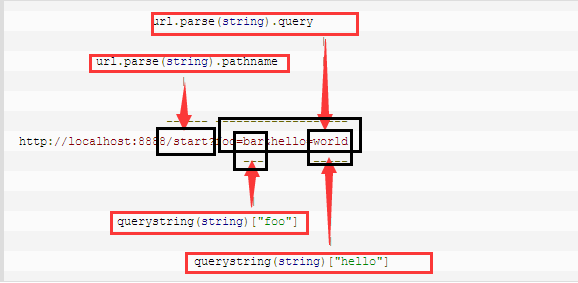
## 九、Node.js 路由

我们要为路由提供请求的URL和其他需要的GET及POST参数，随后路由需要根据这些数据来执行相应的代码。

因此，我们需要查看HTTP请求，从中提取出请求的URL以及GET/POST参数。这一功能应当属于路由还是服务器（甚至作为一个模块自身的功能）确实值得探讨，但这里暂定其为我们的HTTP服务器的功能。

我们需要的所有数据都会包含在request对象中，该对象作为onRequest()回调函数的第一个参数传递。但是为了解析这些数据，我们需要额外的Node.JS模块，它们分别是url和querystring模块。

例：http://localhost:8888/start?foo=bar&hello=world



var url = require("url");

var pathname = url.parse(request.url).pathname;

console.log("Request for " + pathname + " received.");

## 十、Node.js常用工具

util是Node.js的核心模块，提供常用函数的集合，用于弥补javascript核心功能过于精简的不足。

（一）、util.inherits是一个实现对象间原型继承的函数。JavaScript的面向对象特性是基于原型的，与常见的基于类的不同。Javascript没有提供对象继承的语言级别特性，而是通过原型复制来实现的。

//原型方法

Base.prototype.showName = function () {

console.log(this.name);};

（1）prototype方法。原型法的主要思想是，现在有1个类A，我想要创建一个类B，这个类以A为原型，并能进行扩展，我们称B的原型为A。

## 十一、Node.js的多进程

Node.js是以单线程模式运行的，但它使用的是事件驱动处理并发，这有助于我们在多核cpu的系统上创建多个子进程，从而提高性能。

## 十二、Node.js的工具模块

## 十三、连接MongoDB数据库

（一）、Web程序连接数据库基本配置

创建setting.js文件：module.exports = {

"cookie\_screct": "NodeWebCookie",//设置缓存信息

"ip": "localhost",

"db": "NodeExprsDB",//数据库名字

"host": 27071//端口号};

创建db.js 文件(用于连接数据库): var settings = require("../settings");

var mongoose = require('mongoose');

//var DB\_CONN\_STR = "mongodb://" + settings.ip + "/" + settings.db;

mongoose.connect("mongodb://" + settings.ip + "/" + settings.db);

module.exports = mongoose;

（二）、Web程序引用数据库

app.js文件添加引用：

var mongoose = require('mongoose');

var session = require("express-session");//加入缓存，保存用户登录信息

var MongoStore = require('connect-mongo')(session);//该模块用于将session存入mongo中

var db = require('./database/db');

var settings = require("./settings");

//新增session 将session存入mongo中

app.use(session({

secret: 'NodeWebCookie', //需要与setting中设置的cookie\_screct名字一致

key: settings.db,//cookie name

cookie: {maxAge: 1000 \* 60 \* 60 \* 24 \* 30},//30 days

resave: false,

saveUninitialized: true,

store: new MongoStore({

url: 'mongodb://localhost/NodeExprsDB'

})

}));

注：(1)当报错“Connection strategy not found”时，需要将app.js中设置session的secret的名字与setting.js中设置的cookies\_screct一致。

## 十四、Express.js

Express.js中有jade和ejs两种模型，ejs可以修改为html。

ejs有三种标签系统：

1、<% code%>：显示JavaScript代码；

2、<%= code%>：显示替换过HTML特殊字符的内容；

3、<%- code%>：显示原始HTML内容。

## 十五、Express.js+Node.js+MongoDB项目中联合操作

一、查询操作：

类名.find():

1、find():查询所有数据。

2、find({name:"Joke"}):查询name的值是Joke 的数据，条件查询。

如果是要and 查询，则是find({name:"Joke",age:"19"})，在同一个花括号{}中添加查询条件。

如果是or查询,需要添加关键字 '$or' ，find({$or: [{age: 22}, {age: 25}]})

3、模糊查询用正则表达式new RegExp()。需要注意的是在MongoDB中与在nodejs中编写的方法不一样。在MongoDB中可直接使用'/..../'斜杠：find({name:/Joke/})匹配所有name中包含Joke的数据，在Nodejs中则是find({name:new RegExp(Joke))。

4、