目 录

[第1章 Node.js](#_Toc32717_WPSOffice_Level1) [2](#_Toc32717_WPSOffice_Level1)

[1.1 基本认识](#_Toc21688_WPSOffice_Level2) [2](#_Toc21688_WPSOffice_Level2)

[1.1.1 认识](#_Toc21688_WPSOffice_Level3) [2](#_Toc21688_WPSOffice_Level3)

[1.1.2 特点](#_Toc26357_WPSOffice_Level3) [2](#_Toc26357_WPSOffice_Level3)

[1.1.3 基础](#_Toc15582_WPSOffice_Level3) [3](#_Toc15582_WPSOffice_Level3)

[1.2 安装](#_Toc26357_WPSOffice_Level2) [3](#_Toc26357_WPSOffice_Level2)

[1.2.1 Windows](#_Toc14131_WPSOffice_Level3) [3](#_Toc14131_WPSOffice_Level3)

[1.2.2 Linux安装](#_Toc11826_WPSOffice_Level3) [4](#_Toc11826_WPSOffice_Level3)

[1.3 Node.js基本使用](#_Toc15582_WPSOffice_Level2) [4](#_Toc15582_WPSOffice_Level2)

[1.3.1 Node.js中http服务](#_Toc18604_WPSOffice_Level3) [5](#_Toc18604_WPSOffice_Level3)

[1.3.2 npm](#_Toc9866_WPSOffice_Level3) [5](#_Toc9866_WPSOffice_Level3)

[1.3.3 node.js的模板应用](#_Toc16896_WPSOffice_Level3) [6](#_Toc16896_WPSOffice_Level3)

[1.3.4 ejs模板引擎](#_Toc22940_WPSOffice_Level3) [8](#_Toc22940_WPSOffice_Level3)

[1.3.5 express框架](#_Toc31038_WPSOffice_Level3) [9](#_Toc31038_WPSOffice_Level3)

[1.3.6 exp框架+ejs模板](#_Toc14759_WPSOffice_Level3) [11](#_Toc14759_WPSOffice_Level3)

[1.3.7 静态目录](#_Toc25193_WPSOffice_Level3) [11](#_Toc25193_WPSOffice_Level3)

[1.4 socket](#_Toc14131_WPSOffice_Level2) [13](#_Toc14131_WPSOffice_Level2)

[1.4.1 认识](#_Toc4703_WPSOffice_Level3) [13](#_Toc4703_WPSOffice_Level3)

[1.4.2 socket.io](#_Toc2876_WPSOffice_Level3) [13](#_Toc2876_WPSOffice_Level3)

[1.4.3 exp框架+ejs模板+sockect.io](#_Toc12313_WPSOffice_Level3) [14](#_Toc12313_WPSOffice_Level3)

# Node.js

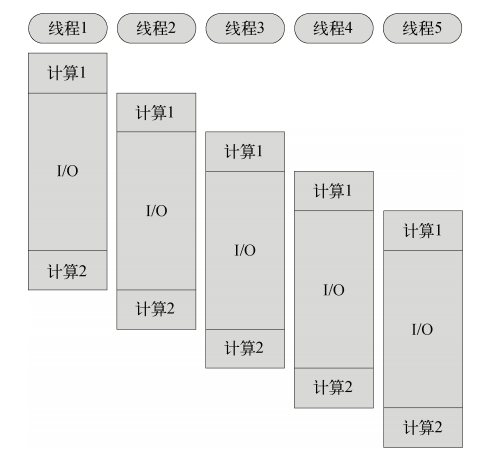
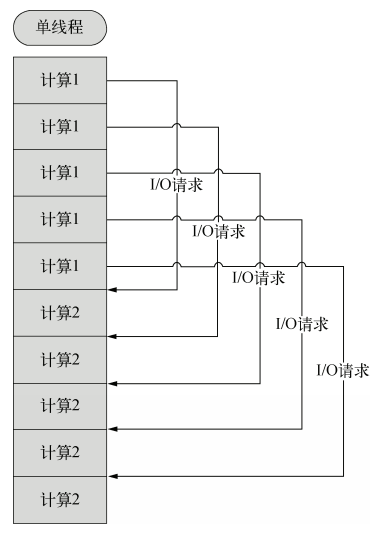
## 1.1 基本认识

### 1.1.1 认识

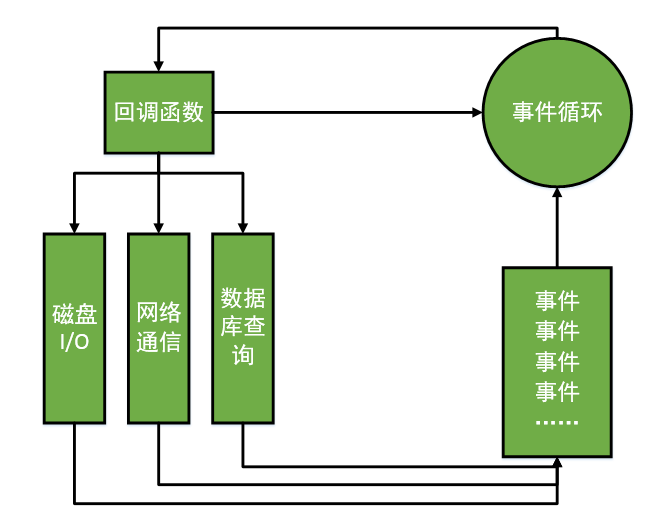
Node.js是一个让JavaScript运行在服务端的开发平台，它让JavaScript的触角伸到了服务器端。但Node.js不是一种独立的语言，不是Js框架，不是浏览器 的库(可以理解为运行在浏览器上的软件)，Node.js使用JavaScript进行编程，运行在JavaScript引擎上(Chrome V8)。Node.js跳过Apache、Maginx等HTTP服务器，它自己不用建设在任何服务器软件之上。Node.js的许多设计理念与经典架构LAMP有着很大的不同

### 1.1.2 特点

* 【单线程】通过非阻塞I/O、事件驱动机制，让Node.js程序宏观上也是并发的。另外，使操作系统完全不再有线程创建、销毁的时间开销。则坏处，就是一个用户造成了线程的崩溃，整个服务崩溃了，其他也崩溃了。

* 【异步非阻塞I/O】non-blocking I/O 例如：当访问数据库取得数据的时候，需要一段时间，在传统的单线程处理机制中，在执行了访问数据库代码之后，整体线程都将暂停下来，等待数据库的返回结果，才执行 后面的代码。而非阻塞I/O则是在访问数据库代码之后，将立即转而执行后面的代码，把数据库返回结果处理代码放在回调函数中。
* 【事件驱动】 event-driven 在Node中，客户端请求建立连接，提交数据等行为，会触发相应的事件。在Node中，在一个时刻，只能执行一个事件回调函数，但是在执行一个事件回调函数的中途，可以转而处理其他事件，然后返回继续执行原事件的回调函数，这种处理机制就是事件环机制



* 【整体】

单线程：减少内存开销，操作系统的内存换页。

非阻塞I/O：不会傻等I/O语句结束，而会执行后面的语句

事件机制、事件环：不管是新用户请求还是老用户的I/O完成，都将以事件方式加入事件环，等待调度。

* 【适合开发】Node.js善于I/O，不善于计算，因此Node.js最擅长的就是任务调度。当应用程序需要处理大量并发的I/O，而在向客户端发出响应之前，应用程序内部并不需要进行非常复杂的处理的时候，Node.js就非常适合。Node.js也非常适合于web socket配合，开发长连接的实时交互应用程序。

### 1.1.3 基础

* 【I/O】

I是指Input,输入；O是指Output，输出。I/O是指输入/输出接口。

* 【请求和响应】

request Headers：请求头信息

response Headers：响应头信息

## 1.2 安装

### 1.2.1 Windows

* 【安装】

官网下载：http://nodejs.org /dist/ 选择LTS版本的 双击安装

* 【测试】 检测是否安装成功

1. cmd 命令下输入node -v ,显示版本号 表示成功
2. 继续输入npm -v，x显示版本号 表示成功

### 1.2.2 Linux安装

* 【准备】

1. 查看自己linux(服务器)是64位系统还是32系统 通过uname -a命令来查看(x86\_64表示64位系统，i686 i386表示32位系统)

64

64bit

32

32bit

1. 下载相应系统的包 官网上找。

* 【解压】

在linux系统下解压包 (32位系统包来举例)

tar -xf node.v6.10.1-linux-x86.tar.xz

* 【移动】

将解压好的文件夹移动到 /usr/local/node

mv node-v6.10.1-linux-x86 /usr/local/node

* 【配置】配置环境变量

1. 打开 vim ~/.bash\_profile
2. 添加node的路径 PATH=$PATH:$HOME/bin:/usr/local/node/bin

* 【启动】启动配置环境变量的文件

source ~/.bash\_profile

* 【测试】测试Linux系统下是否成功

1.node -v

2.npm -v

在没有配置环境变量下需要进入到 /usr/local/node/bin 目录下测试

## 1.3 Node.js基本使用

node可以允许js文件，可以在js文件中搭建http服务作为服务器使用(跟python一样)。但同时需要注意：

node.js文件中不能出现alert(),DOM,BOM相关信息，在命令行中是会出现错误的

### 1.3.1 Node.js中http服务

* 【搭建】

//引入http模块

var http = require(‘http’);

//创建http服务

var server = http.createServer(function(req,res){

//结束服务器

res.end(‘hello world’);

});

server.listen(8000) //最好别与其他端口冲突

端口号的范围从0到65535

* 【req请求属性】 创建http中的第一个参数 request 常见属性

req.method 获取请求的方法

req.headers 获取请求头信息

req.url 获取请求的URL地址信息  
req.httpVersion 获取http协议的版本号

* 【res响应方法】 创建http中的第二个参数 response 常见方法

res.statusCode 获取响应码

res.write() 往文档中写入信息

res.writeHead() 往响应头里面写入信息

res.writeHead(200,{‘content-type’:’text/html;charset=utf8’});

res.setHeader() 设置响应头信息

res.setHeader(‘content-type’,’text/html;charset=utf8’);

res.end() 结束响应

### 1.3.2 npm

npm是node packaged modules的简称，它的作用是基于node.js管理扩展包之间的依赖关系。(使用方法和linux中的yum类似)

* 【下载】
* 单个下载

1. npm install jquery 下载最新版本的jquery
2. npm install jquery@1.11.3 下载指定版本的jquery
3. 了解一下npm淘宝镜像

* 多个下载

1. 创建项目的包信息 cmd命令行 npm init (或者当前目录下新建package.json)
2. 修改package.json的内容

{

“name” : “myapp”, //package名字(自定义)

“version”: “1.0.0”, //package的版本(自定义)

“desription”: “”, //描述(自定义)

“main”: “main.js”, //主入口文件(自定义)

“dependencies”: { //package的应用依赖模块，需要下载的东西

“jquery”: “1.8.3”,

“bootstrap”: “\\*”

},

“author”: “张三”, //作者

“license”: “isc” //授权

} //严格双引号就是双引号 用逗号做间隔

1. 运行 npm install

* 【卸载】

npm uninstall jquery //卸载jquery

* 【查看包信息】

1.npm ls //查看安装的包

2.npm ls jquery //查看jquery包的特性

3.npm info jquery //查看jquery包的详细信息

* 【更新包】

npm update jquery

* 【搜索包】

npm search jquery

* 【查看npm配置】

npm config list

* 【实例】 自动启动js文件(启动js文件 需要在安装过node的目录下，在命令行模式下输入 node js文件名)

如果你的js文件有改动的话，就不用使用node 文件名，再重新启动，直接在浏览器运行就可以看到效果。

1. 安装

npm install supervisor -g //全局的安装

1. 使用

supervisor 要启动的文件 //启动js文件

### 1.3.3 node.js的模块应用

根据目前所学将其分为四个模块 path、url、querystring、fs

* 【path】 操作url地址的信息

1.引入

var paths = require(‘path’);

2.应用

paths.extname(‘index.html’); //返回文件的后缀

paths.dirname(‘/foo/bar/baz/qunx’); //返回一个path的目录名 /foo/bar/baz

paths.basename(‘/foo/bar/baz/index.html’) //返回path最后一部分 index.html

paths.basename(‘/foo/bar/baz/index.html’,’.html’)//加上第二个参数则返回 index

* 【url】用于URL处理与解析

1. 引入

var ulrs = require(‘url’);

1. 应用

parse() //把url字符串解析成js对象

pathname //获取url参数路径信息

format() //把js对象解析成字符串

* 【querystring】 一般是对http请求所带的数据进行解析

1. 引入

var que = require(‘querystring’);

1. 应用

var str = ‘name=www&age=20’;

que.parse() //把普通的字符串装换成js对象

que.stringify() //把对象转换成普通的字符串

* 【fs】 文件系统模块 文件夹和文件

1. 引入

var fs = require(‘fs’);

1. 应用

fs.mkdir() //创建文件夹

fs.readdir() //读取文件夹

fs.rmdir() //删除文件夹

fs.writeFile() //写入文件

fs.readFile() //读取文件

在对文件的操作中buffer是一个存储二进制数据的字节流

* 【实例】node.js链接mysql数据库

1. 首先要在cmd命令行下安装mysql模块  
   npm install mysql   
   2.链接  
   var http = require(‘http’);

var mysql = require(‘mysql’);

var server = http.createServer(function(req,res){

//创建连接数据库信息

var connection = mysql.createConnection({

host : ‘localhost’,

user: ‘root’,

password :’123456’,

database :’’

});

//链接数据库

connection.connect();

//执行sql语句

connection.query(‘sql语句’,function(error,results,fields){

//判断链接错误

if(error) throw error;

//打印结果

console.log(results);

});

//断开连接数据库

connection.end();

});

server.listen(3000);

### 1.3.4 ejs模板引擎

EJS是一套简单的模板语言，帮你利用普通的JavaScript代码生成HTML页面

* 【安装/引用】

npm install ejs

var ejs = require(‘ejs’);

* 【遍历规则】在html模板中使用ejs的时候

1. 模板中要替换的变量需要使用<%= %>方式进行解析变量。注意ejs模板解析本事是<%=%> ，只有在变量值的时候加个=号。其他都是<%%>
2. 模板中使用#进行注释

* 【forEach】循环遍历 （前台遍历）

//arr是从服务器传过来的数组

//注意这里面 value和key名字可以自定义，但是位置不能进行改变

<% arr.forEach(function(value,key){%>

//第一种

<li><a><%= value%></a></li>

//第二种

<li><a><%= arr[key] %></a></li>

<%})%>

* 【if】 流程控制

<% if(true){%>

<h1>wwwww</h1>

<%} else {%>

<h2>yyyyyy</he>

<% }%>

* 【for】

<% for(var i=1;i<=10;i++){%>

<li><a><%= i%></a></li>

<%}%>

### 1.3.5 express框架

Express是一个基于Node.js 平台的极简、灵活的web应用开发框架，它提供一系列强大的特性，帮助你创建各种web和移动设备应用

* 【安装】

1. 安装express 并将其中保存到依赖列表中:

npm install express --save

1. 如果不需要安装到依赖列表中只是临时使用的话，只需把--save参数去掉即可

npm install express

注意：如果在安装Node模块时，如果指定了 --save参数，那么此模块将被添加到package.json文件中dependencies依赖列表中。然后通过npm install命令即可自动安装依赖列表中所列出的所有模板。

* 【应用】

//引入express

var express = require(‘express’);

//创建引用

var app = express();

//get路由规则

1.get路由不带参数

app.get(‘/’,function(req,res){

//使用send进行发送 返回到浏览器中显示的信息

1.res.send(‘hello’): //返回给参数1页面一个hello

2.res.sendFile(\_\_dirname+’1.html’); //解析HTML文件

});

2.get路由带参数

// /100 -> /:id 方式传参接收

app.get(‘/tiezi/:id’,function(req,res){

//该方式传参用req.params.id接收与获取

res.send(‘www’+req.params.id);

})

// ?id=100 该方式传参

app.get(‘/tiezi’,function(req,res){

//该方式传参用req.query.id接收与获取

res.send(‘www’+req.query.id);

})

//请求信息

app.get(‘/request’,function(req,res){

//直接调用 req.url等等

req.url

req.httpVersion

req.hostname

req.ip

req.params.name

req.path

req.query

})

//响应信息

app.get(‘/respones’,function(req,res){

//设置响应体 浏览器解析html

res.send(‘<body><li><a href="">美女</a></li></body>’);

//跳转

res.redirect(‘http://www.baidu.com’);

res.redirect(‘/form’) //路由规则

//下载

res.download(‘./form.html’); //可以使用相对路径

//返回json字符串 在配合ajax使用

res.json({name:’www’,age:20});

//设置404状态码

res.status(404).send(‘404 Not Found’);

res.status(404).sendFile(\_\_dirname+’/404.html’);

//cookie 需要引入cookie-parse （npm cookie 先按照再引用）

var cookieParser = require(‘cookie-parse’);

app.use(cookiePaeser());

//1.设置cookie

res.cookie(‘name’,’value’);

res.cookie(‘name’,’value’,{maxAge:1000}); //有效期以毫秒为单位

//2.获取cookie

console.log(req.cookies); //加s

console.log(req.cookies.name)

//3.删除cookie

res.clearCookie(‘name’);

});

//post路由规则 与get相似

app.post(‘/’,function(req,res){

//接受表单传过来的数据 (转下静态目录)

});

app.listen(3000); //个人总忘记这个端口报错很难找

### 1.3.6 exp框架+ejs模板

Express框架有默认的模板jade，但我们这里一般使用ejs模板与之搭配

* 【实例】

var express = require('express');

var app = express();

//引入ejs模板，将ejs更换成我们熟悉html

app.set('views', './views'); //参数1固定 2根据文件夹来考虑

app.engine('.html', require('ejs').\_\_express);

app.set('view engine', 'html');

//这是路由规则

app.get('/', function(req, res) {

//这里面的index是对应views(参数2)文件里面的index.html文件,后面的{}是往模板里面传递的数据。相当于Tp5框架里面的return view()。没有引入ejs模板时使用的是res.send()返回给前台数据，引入后是res.render()往指定页面中传递

res.render('index', { title: '这是ejs模板演示' });

});

//端口号是3000

app.listen(3000);

### 1.3.7 静态目录

静态目录可以理解为省去前面的相同的目录，个人觉得和框架中的绑定路由相似

* 【实例】 在框架中使用需要引入框架创建框架 app.use(express.static(目录))

var express = require('express');

var app = express();

//设置静态文件的托管目录

app.use(express.static('目录名'));

* 【formidable】 获取表传过来的数据。需要引入formidable

//获取表单的数据需要引入formidable

var formidable = require('formidable');

var form = new formidable.IncomingForm();

form.parse(req,function(err,fields,files){

//err 错误

//fields 返回的结果是对象

});

* 【实例】

//引入express 框架模板  
var express = require('express');  
//创建框架  
var app = express();  
//更改模板  
app.set('views', './user');  
app.engine('.html', require('ejs').\_\_express);  
app.set('view engine', 'html');  
//设置静态文件的托管目录  
app.use(express.static('node\_modules'));

//引入form  
var formidable = require('formidable');  
//创建formidable  
var form = new formidable.IncomingForm();  
//引入数据库  
var mysql = require('mysql');  
//创建数据库  
var connection = mysql.createConnection({  
 host : 'localhost',  
 database : 'ajax',  
 user : 'root',  
 password : '123456'  
});  
//链接数据库  
connection.connect();

//处理用户的添加  
app.post('/user/insert',function(req,res){  
 //判断cookie  
 if(!req.cookies.uname){  
 res.send('<script> alert("请登录");location.href="/login";</script>');  
 }  
 //获取表单信息 引入 创建  
 form.parse(req,function(err,fields,files){  
 if(err) throw err;  
 //获取 fields 返回的结果是对象  
 var uname = fields.username;  
 var pass = fields.password;  
 var phone = fields.phone;  
 //链接数据库  
 var sql = 'insert into usr(username,password,phone) values("'+uname+'","'+pass+'","'+phone+'")';  
 connection.query(sql,function(err,results,fields){  
 if(err) throw err;  
 // console.log(results);  
 if(results.affectedRows > 0){  
 res.redirect('/user/init');  
 res.send();  
 }else{  
 es.redirect('/user/add');  
 res.send();  
 }  
 });  
 //断开连接数据库  
 // connection.end();  
 });  
});

## 1.4 socket

### 1.4.1 认识

socket是通信的基础

建立Socket连接至少需要一对套接字，其中一个运行于客户端，称为ClientSocket ，另一个运行于服务器端，称为ServerSocket 。

套接字之间的连接过程分为三个步骤：服务器监听，客户端请求，连接确认。

### 1.4.2 socket.io

sockert.io是基于socket连接后对网络输入输出的处理

io是输入和输出 I/O输入/输出(Input/Output),分为IO设备和IO接口两个部分。

* 【安装】网址：<https://socket.io>

npm install socket.io --save

* 【服务器端】 fs文件举例

var app = require('http').createServer(handler);

var io = require('socket.io')(app);

var fs = require('fs');

function handler (req, res) {

fs.readFile(\_\_dirname + '/index.html', function (err, data) {

if (err) {

res.writeHead(500);

return res.end('Error loading index.html');

}

res.writeHead(200);

res.end(data);

});

}

io.on(‘connection’,function (socket){ // ‘connection’ 不能更改

//激活一个事件emit发送 参数1：套接字自定义。参数2向客户端发信息

socket.emit(‘news’,{helllo:’www’})； //服务器有一个发送那客服端有有一个接收。

//接收客户端发过来的信息

socket.on(‘my’,function(data){ //data客服端发动过来的数据

console.log(data);

});

});

app.listen(8000); //服务器端定义端口则客户端链接时也需要加上端口

* 【客户端】

<script src="/socket.io/socket.io.js"></script> //目前不知道为啥要加，但要加。

<script>

//和服务器端口进行链接

var socket = io('http://localhost:8000');

//接受服务器传过来的信息

socket.on('news', function (data) {

console.log(data);

//往服务器端发送

socket.emit('my', { my: 'data' });

});

</script>

### 1.4.3 exp框架+ejs模板+sockect.io

* 【服务器端】

var express = require('express'); //引入exp   
 var app = express(); //创建exp  
 var server = require('http').createServer(app); //搭建http服务器  
 var io = require('socket.io')(server); //引入socket.io

//引入ejs模板  
 app.set('views','./views');  
 app.engine('.html',require('ejs').\_\_express);  
 app.set('view engine','html');

//静态目录  
 app.use(express.static('node\_modules'));  
 app.use(express.static('public'));

//引入页面

app.get('/', function (req, res) {

res.render(‘index’,{title:’测试’}),

});

//监听客户端链接,回调函数会传递本次链接的socket

io.on('connection', function (socket) {

//发送消息到单独的客户端

socket.emit('news', { hello: 'world' });

//给所有的客户端发消息消息(包括自己)

io.emit('xxoo','souga');

//监听客户端发送的消息

socket.on('my', function (data) {

console.log(data);

});

});

server.listen(8000);

* 【客户端】

<script src="/socket.io/socket.io.js"></script>

<script>

//建立一个socket链接

var socket = io.connect('http://localhost:8000');

//往服务器发送消息

socket.emit('my', { my: 'data' });

//监听服务端发送的消息

socket.on('news', function (data) {

console.log(data);

});

</script>

* 【总结】

socket.io中需要注意以下几点：

1. 分清socket.emit 与io.emit 两者不一样 (socket 和io拿以上实例来说)
2. 找到关系一一对应。发送emit，接收on
3. 端口要查看是否一致
4. 个人总是忘记端口