CPU密集：压缩、解压、加密、解密

I/O密集：文件操作、网络操作、数据库

web常见场景：静态资源读取，数据库操作，渲染页面

高并发对应之道：增加机器数、增加每台机器的CPU数----多核

进程：是计算机中的程序关于某数据集合上的一次运行活动，是系统进行资源分配和调度的基本单位

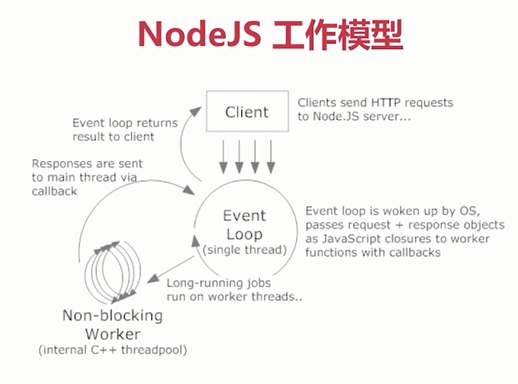
多进程：启动多个进程，多个进程可以一块执行多个任务

线程：进程内一个相对独立的、可调度的执行单元，与同属一个进程的线程共享进程的资源

多线程

多线程：启动一个进程，在一个进程内启动多个线程，这样，多个线程也可以一块执行多个任务

nodejs的单线程：1.单线程只针对主进程，I/O操作系统底层多线程调度2.不是单进程



常用场景：1.web server 2.本地代码构建 3.实用工具开发

环境：(function(exports(输出),require(函数),module(模块本身),\_filename(文件地址),\_dirname(文件夹地址)){})

1.CommomJS

每个文件是一个模块，有自己的作用域

在模块内部 module 变量代表模块本身

module.exports属性代表模块对外接口（缩写：exports,但是不能修改指向）

require规则：

/表示绝对路径，./表示相对于当前文件的

支持js、json、node拓展名，不写依次尝试

不写路径则认为是build-in 模块或者各级node\_modules内的第三方模块

require 特性：

module被加载的时候执行，加载后缓存

一旦出现某个模块被循环加载，就只输出已经执行的部分，还未执行的部分不会输出

2.global(转化为全局变量)

CommomJS Buffer、process、console timer

global.testVar2 = 200;

3.process

const { argv, argv0, execArgv, execPath } = process;

argv.forEach(*item* => {

*console*.log(item);

});

*console*.log(argv0);

*console*.log(execArgv);

*console*.log(execPath);

const { env } = process;

*console*.log(env);

setImmediate(() => {

*console*.log('setImmediate');

})

setTimeout(() => {

*console*.log('timeout');

}, 0)

process.nextTick(() => {

*console*.log('nextTick');

})

nextTick快于setTimeout 快于setImmediate（推荐）

基础API

path（路径）：

1.normalize（规范路径），join（拼接路径）resolve（相对路径→绝对路径）

2.basename(文件名),dirname(所在文件夹名),exyname(拓展名)

3.parse（返回路径的对象），format（与parse相反，从一个对象返回一个路径字符串）

优先级：dir(路径)>root(根目录) base>ext,name(base=name.ext)

4.sep(路径的分隔符)，delimiter（path的分隔符，引用path:process.env.PATH），

win12(在windows下面的情况：win32.sep,win32.delimiter),posix

path：\_\_dirname、\_\_filename总是返回文件的绝对路径，process.cwd()总是返回执行node命令所在文件夹

./(../)相对路径:在require方法中总是相对当前文件所在文件夹在其他地方和process.cwd()一样，相对node启动文件夹（path.resolve(./)）

path.relative(config.root, filePath): 一个路径相对于另一个路径的相对地址

Buffer（缓冲）:

1.Buffer用于处理二进制数据流

2.实例类似整数数组，大小固定

3.c++代码在V8堆外分配物理内存

Buffer.alloc(10,1)：创建10个1

Buffer.allocUnsafe(5,1):数组内容没有初始化

Buffer.from([1,2,3])

Buffer.from(‘test’,’默认utf8’)

静态属性方法：

Buffer.byteLength,Buffer.isBuffer(),Buffer.concat()

buf.length,buf.toString(‘默认utf8’),buf.fill(),buf.equals(),buf.indexOf(),buf.copy(b,I,j)

解决中文乱码：StringDecoder

const StringDecoder = require('string\_decoder').StringDecoder;

events（事件）：

eventEmitter监听器: eventEmitter.on(),eventEmitter.emit()(触发事件)，eventEmitter.once()（只调用一次），eventEmitter.removeListener(‘’,),eventEmitter.removeAllListeners(‘’);

const EventEmitter = require('events');

class CustomEvent extends *EventEmitter* {

}

const ce = **new** *CustomEvent*();

fs(文件系统)：

readFile:

fs.readFile('./25\_readfile.js', 'utf8', (*err*, *data*) => {}); /\* 异步操作 \*/

const data = fs.readFileSync('./01\_run.js', 'utf8');/\* 同步操作\*/

/\* ./读取当前所有文件的名字，../读取上一层所有文件的名字 \*/

fs.readdir('../'/\* './' \*/, (*err*, *files*) => {});

writeFile,rename,unlink,mkdir,rmdir:

/\* 添加文件 \*/

const content = *Buffer*.from('this is a test.');

fs.writeFile('./text', content/\* 'This is a test', 'utf8' \*//\* {

encoding: 'utf8'

} \*/, *err* => {

if (err) throw err;

*console*.log('done!'); });

/\* 文件重命名 \*/

fs.rename('./text', 'text.txt', *err* => {

if (err) throw err;

*console*.log('done!');});

/\* 删除文件 \*/

fs.unlink('./text.txt', *err* => {

if (err) throw err;

*console*.log('done!');});

/\* 添加文件夹 \*/

fs.mkdir('test', *err* => { });

/\* 删除文件夹 \*/

fs.rmdir('./test', *err* => { });

stat:

fs.stat('./26\_stat.js', (*err*, *stats*) => {

/\* if (err) throw err; \*/

if (err) {

*console*.log('文件不存在');

return; };

*console*.log(stats.isFile());/\* 判断是否为文件 \*/

*console*.log(stats.isDirectory());/\* 判断是否为文件夹 \*/

*console*.log(stats);/\* 文件信息 \*/})

watch:

/\* 监听文件变化 \*/

fs.watch('./', {

recursive: true/\* 是否循环，是否递归 \*/

}, (*eventType*, *filename*) => {

*console*.log(eventType, filename)

});

stream(流):

createReadStream:

/\* 读取文件 \*/

const rs = fs.createReadStream('./30\_readstream.js');

rs.pipe(process.stdout);/\* 读一点给一点（水管） stdout：输出到控制台 \*/

createWriteStream:

/\* 写入文件 \*/

const ws = fs.createWriteStream('./text.txt');

const tid = setInterval(() => {

const num = parseInt(Math.random() \* 10);

if (num < 8) {

ws.write(num + '')/\* stream只能接受字符串或buffer，所以要转换为字符串 \*/

} else {

clearInterval(tid);

ws.end();

}

}, 200);

ws.on('finish', () => {

*console*.log('done!');

});

promisify: try{ //正常执行 }catch(e/\*你感觉会出错的 错误类型\*/)

/\* 将异步改为类似同步的书写 \*/

const promisify = require('util').promisify;

const read = promisify(fs.readFile);

/\* 读取文件1 \*/

read('./31\_promisify.js').then(*data* => {

*console*.log(data.toString())

}).catch(*ex* => {

*console*.log(ex); /\* catch：若出错，则抛出异常提示 \*/

});

/\* 读取文件2 \*/

async function test() {

try {

const content = await read('./31\_promisify.js');

*console*.log(content.toString());

} catch (ex) {

*console*.log(ex);

}

}

test();

.gitignore,.npmignore:

1.匹配模式前/代表项目根目录 2.匹配模式最后加/代表是目录 3.匹配模式前加！代表取反

1.\*代表任意个字符 2.?代表任意一个字符 3.\*\*匹配多级目录（包括文件夹）4.[name],[!name] 5.{a1,a2,a3} 6.{num1..num2}(之间)

.editorconfig:规范编码方式

.eslintrc.js(ESLint):制定编码规则

http.creatSever((req,res)=>{})(req:客户端请求 res：服务器端响应)

const server = http.createServer((*req*, *res*) => {

res.statusCode = 200;

res.setHeader('Content-Type', 'text/plain' /\* text/html \*/

);

res.end('Hello HTTP!');

});

server.listen(conf.port, conf.hostname, () => {}

server.listen(conf.port, conf.hostname, () => {

const addr = `http://${conf.hostname}:${conf.port}`;

*console*.info(`Server started at ${chalk.green(addr)}`);

});

Handlebars.compile()：将模板编译为函数,生成的执行函数接受context作为参数,用来渲染模板。

supervisor:监听运行

{{#each files}}

<a *href*="{{../dir}}/{{this}}">{{this}}</a> <!-- ../dir dir的上一级目录 -->

{{/each}}

const dir = path.relative(config.root, filePath);/\* 一个路径相对于另一个路径的相对地址,若是相对于同样的路径，则返回空 \*/

const data = {

title: path.basename(filePath),

dir: dir ? `/${dir}`/\* 从根路径开始 \*/ : '',

files

};

res.end(template(data));

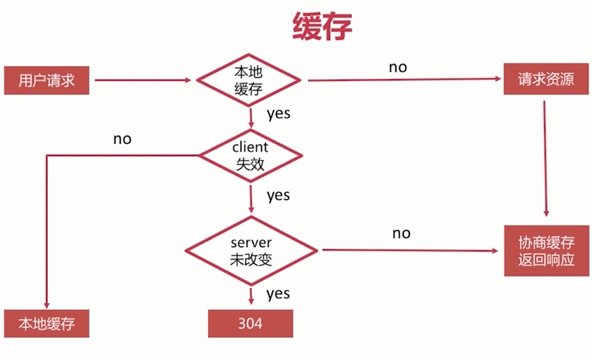
files.map：把每个元素通过函数传递到当前匹配集合中

match() ：可在字符串内检索指定的值，或找到一个或多个正则表达式的匹配。

range：

range：bytes=[start]-[end] Accept-Ranges:bytes Content-Range:bytes start-end/total

缓存：



缓存header：Expies（返回绝对时间）Cache-Control（返回相对时间） 判断本地是否失效

If-Modified-Since（从这个时间之后是否修改）/Last-Modified（返回最近修改的时间） 向服务校验时达到的结果

If-None-Match/Etag

cli：