

Node.js基础

课程目标

- 掌握异步I/O概念、promisify用法、流、buffer
- 掌握一个简单http服务（页面、json数据、静态资源）
- 实战一个cli工具(vue路由约定)

I/O处理

异步IO

EventLoop是什么

一个循环 每次循环叫tick 每次循环的代码叫task

- V8引擎单线程无法同时干两件事
- 文件读取、网络IO缓慢且具有不确定性
- 要通过异步回调方式处理又称为异步IO
- 先同步再异步 异步放入队列等同步完成后在执行 每次循环叫一个tick (process.nextTick())



```
while (eventLoop.waitForTask()) {  
  eventLoop.processNextTask()  
}
```

<https://www.processon.com/view/link/5e70b1c2e4b011fcce9b89b5>

事件循环深度版

```
// 03-fs.js  
const fs = require('fs');
```

```
// 同步调用
const data = fs.readFileSync('./conf.js'); //代码会阻塞在这里
console.log(data);

// 异步调用
fs.readFile('./conf.js', (err, data) => {
  if (err) throw err;
  console.log(data);
})

// promisify
const {promisify} = require('util')
const readFile = promisify(fs.readFile)
readFile('./conf.js').then(data=>console.log(data))

// fs Promises API node v10
const fsp = require("fs").promises;
fsp
  .readFile("./confs.js")
  .then(data => console.log(data))
  .catch(err => console.log(err));

// async/await
(async () => {
  const fs = require('fs')
  const { promisify } = require('util')
  const readFile = promisify(fs.readFile)
  const data = await readFile('./index.html')
  console.log('data',data)
})();

// 引用方式
Buffer.from(data).toString('utf-8')
```

Buffer缓冲区

读取数据类型为Buffer

- Buffer - 用于在 TCP 流、文件系统操作、以及其他上下文中与八位字节流进行交互。八位字节组成的数组，可以有效的在JS中存储二进制数据

```
// 04-buffer.js
// 创建一个长度为10字节以0填充的Buffer
const buf1 = Buffer.alloc(10);
console.log(buf1);

// 创建一个Buffer包含ascii.
// ascii 查询 http://ascii.911cha.com/
const buf2 = Buffer.from('a')
console.log(buf2,buf2.toString())

// 创建Buffer包含UTF-8字节
// UTF-8: 一种变长的编码方案，使用 1~6 个字节来存储;
```

```
// UTF-32: 一种固定长度的编码方案, 不管字符编号大小, 始终使用 4 个字节来存储;
// UTF-16: 介于 UTF-8 和 UTF-32 之间, 使用 2 个或者 4 个字节来存储, 长度既固定又可变。
const buf3 = Buffer.from('Buffer创建方法');
console.log(buf3);

// 写入Buffer数据
buf1.write('hello');
console.log(buf1);

// 读取Buffer数据
console.log(buf3.toString());

// 合并Buffer
const buf4 = Buffer.concat([buf1, buf3]);
console.log(buf4.toString());

// 可以尝试修改fs案例输出文件原始内容
```

Buffer类似数组, 所以很多数组方法它都有
GBK 转码 iconv-lite

http服务

创建一个http服务器, 05-http.js

```
const http = require('http');
const server = http.createServer((request, response) => {
  console.log('there is a request');
  response.end('a response from server');
});
server.listen(3000);
```

```
// 打印原型链
function getPrototypeChain(obj) {
  var protoChain = [];
  while (obj = Object.getPrototypeOf(obj)) { // 返回给定对象的原型。如果没有继承属性, 则返回 null 。
    protoChain.push(obj);
  }
  protoChain.push(null);
  return protoChain;
}
```

显示一个首页

```
const {url, method} = request;
if (url === '/' && method === 'GET') {
  fs.readFile('index.html', (err, data) => {
    if (err) {
      response.writeHead(500, { 'Content-Type':
        'text/plain; charset=utf-8' });
    }
  });
}
```

```

        response.end('500, 服务器错误');
        return ;
    }
    response.statusCode = 200;
    response.setHeader('Content-Type', 'text/html');
    response.end(data);
});
} else {
    response.statusCode = 404;
    response.setHeader('Content-Type', 'text/plain;charset=utf-8');
    response.end('404, 页面没有找到');
}
}

```

编写一个接口

```

else if (url === '/users' && method === 'GET') {
    response.writeHead(200, { 'Content-Type': 'application/json' });
    response.end(JSON.stringify([{name:'tom',age:20}]));
}

```

Stream流

stream - 是用于与node中流数据交互的接口

```

//二进制友好, 图片操作, 06-stream.js
const fs = require('fs')
const rs2 = fs.createReadStream('./01.jpg')
const ws2 = fs.createWriteStream('./02.jpg')
rs2.pipe(ws2);

//响应图片请求, 05-http.js
const {url, method, headers} = request;

else if (method === 'GET' && headers.accept.indexOf('image/*') !== -1) {
    fs.createReadStream('.'+url).pipe(response);
}

```

Accept代表发送端（客户端）希望接受的数据类型。比如：Accept: text/xml; 代表客户端希望接受的数据类型是xml类型。

Content-Type代表发送端（客户端|服务器）发送的实体数据的数据类型。比如：Content-Type: text/html; 代表发送端发送的数据格式是html。

二者合起来，Accept:text/xml; Content-Type:text/html，即代表希望接受的数据类型是xml格式，本次请求发送的数据的数据格式是html。

CLI工具

创建工程

```
mkdir vue-auto-router-cli
cd vue-auto-router-cli
npm init -y
npm i commander download-git-repo ora handlebars figlet clear chalk open -s
```

```
# bin/kkb.js
#指定脚本解释器为node
#!/usr/bin/env node
console.log('cli.....')

# package.json
"bin": {
  "kkb": "./bin/kkb.js"
},
```

```
# 将npm 模块链接到对应的运行项目中去
npm link

# 删除的情况
ls /usr/local/bin/
rm /usr/local/bin/kkb
```

定制命令行界面

commander.js

kkb.js文件

```
#!/usr/bin/env node
const program = require('commander')
program.version(require('../package').version)

program
  .command('init <name>')
  .description('init project')
  .action(name => {
    console.log('init ' + name)
  })

program.parse(process.argv)
```

打印欢迎界面

/lib/init.js

```
const {promisify} = require('util')
const figlet = promisify(require('figlet'))
const clear = require('clear')
const chalk = require('chalk')
const log = content => console.log(chalk.green(content))
module.exports = async name => {
  // 打印欢迎画面
  clear()
  const data = await figlet('KKB welcome')
  log(data)
}
```

```
// bin/kkb.js
program
  .command('init <name>')
  .description('init project')
  .action(require('../lib/init'))
```

克隆脚手架

/lib/download.js

```
const {promisify} = require('util')
module.exports.clone = async function(repo, desc) {
  const download = promisify(require('download-git-repo'))
  const ora = require('ora')
  const process = ora(`下载.....${repo}`)
  process.start()
  await download(repo, desc)
  process.succeed()
}
```

/lib/init.js

```
const {clone} = require('./download')
module.exports.init = async name => {
  // console.log('init ' + name)
  log('🚀 创建项目:' + name)
  // 从github克隆项目到指定文件夹
  await clone('github:su37josephxia/vue-template', name)
}
```

安装依赖

```
// promisiy化spawn
// 对接输出流
const spawn = async (...args) => {
  const { spawn } = require('child_process');
  web全栈架构师
```

```

    return new Promise(resolve => {
      const proc = spawn(...args)
      proc.stdout.pipe(process.stdout)
      proc.stderr.pipe(process.stderr)
      proc.on('close', () => {
        resolve()
      })
    })
  })
}

module.exports.init = async name => {

  // ....
  log('安装依赖')
  await spawn('cnpm', ['install'], { cwd: `./${name}` })
  log(chalk.green(`
  📦 安装完成:
  To get Start:
  =====
  cd ${name}
  npm run serve
  =====
  `)))
}

```

启动项目

```

const open = require("open")
module.exports.init = async name => {
  // ...
  // 打开浏览器
  open(`http://localhost:8080`);
  await spawn('npm', ['run', 'serve'], { cwd: `./${name}` })
}

```

约定路由功能

- loader 文件扫描
- 代码模板渲染 hbs Mustache风格模板

/lib/refresh.js

```

const fs = require('fs')
const handlebars = require('handlebars')
const chalk = require('chalk')
module.exports = async () => {

  // 获取页面列表
  const list =

```

```

    fs.readdirSync('./src/views')
      .filter(v => v !== 'Home.vue')
      .map(v => ({
        name: v.replace('.vue', '').toLowerCase(),
        file: v
      }))

// 生成路由定义
compile({
  list
}, './src/router.js', './template/router.js.hbs')

// 生成菜单
compile({
  list
}, './src/App.vue', './template/App.vue.hbs')

/**
 * 编译模板文件
 * @param meta 数据定义
 * @param filePath 目标文件路径
 * @param templatePath 模板文件路径
 */
function compile(meta, filePath, templatePath) {
  if (fs.existsSync(templatePath)) {
    const content = fs.readFileSync(templatePath).toString();
    const result = handlebars.compile(content)(meta);
    fs.writeFileSync(filePath, result);
  }
  console.log(chalk.green(`🚀 ${filePath} 创建成功`))
}
}

```

/bin/kkb

```

program
  .command('refresh')
  .description('refresh routers...')
  .action(require('./lib/refresh'))

```

发布npm

```

#!/usr/bin/env bash
npm config get registry # 检查仓库镜像库
npm config set registry=http://registry.npmjs.org
echo '请进行登录相关操作: '
npm login # 登陆
echo "-----publishing-----"
npm publish # 发布
npm config set registry=https://registry.npm.taobao.org # 设置为淘宝镜像
echo "发布完成"
exit

```


