Node.js基础

node知识图谱

课程目标

- 了解nodejs特点和应用场景
- 掌握node模块系统使用
- 掌握基础api使用
  - global
  - process
  - path
  - buffer
  - event
  - fs
  - http
- 实战一个cli工具

NodeJS是什么

node.js是一个异步的事件驱动的JavaScript运行时

> https://nodejs.org/en/
>
> 类比学习运行时这个概念
>
> - JRE java 运行时环境
> - C Runtime
> - .NET Common Language Runtime
>
> 运行时rumtime就是**程序运行的时候**。
> 运行时库就是**程序运行的时候**所需要依赖的库。
>
> 运行的时候指的是指令加载到内存并由CPU执行的时候。
> C代码编译成可执行文件的时候，指令没有被CPU执行，这个时候算是编译时，就是**编译的时候**。

node.js特性其实是JS的特性：

- 非阻塞I/O
- 事件驱动

node历史 — 为性能而生

> 话说有个叫Ryan Dahl(Google Brain) 里安 达尔 / raɪən dɑːl/，他的工作是用C/C++写高性能Web服务。对于高性能，异步IO、事件驱动是基本原则，但是用C/C++写就太痛苦了。于是这位仁兄开始设想用高级语言开发Web服务。他评估了很多种高级语言，发现很多语言虽然同时提供了同步IO和异步IO，但是开发人员一旦用了同步IO，他们就再也懒得写异步IO了，所以，最终，Ryan瞄向了JavaScript。
>
> 因为JavaScript是单线程执行，根本不能进行同步IO操作，所以，JavaScript的这一“缺陷”导致了它只能使用异步IO。

> 选定了开发语言，还要有运行时引擎。这位仁兄曾考虑过自己写一个，不过明智地放弃了，因为V8就是开源的JavaScript引擎。让Google投资去优化V8，咱只负责改造一下拿来用，还不用付钱，这个买卖很划算。
>
> 于是在2009年，Ryan正式推出了基于JavaScript语言和V8引擎的开源Web服务器项目，命名为Node.js。虽然名字很土，但是，Node第一次把JavaScript带入到后端服务器开发，加上世界上已经有无数的JavaScript开发人员，所以Node一下子就火了起来。
>
> 并发处理

- 多进程 - LinuxC Apache
- 多线程 - java
- 异步IO - js
- 协程 - lua openresty go deno- go TS

> 下一代Node deno
>
> https://studygolang.com/articles/13101

与前端的不同

- JS核心语法不变
- 前端 BOM DOM
- 后端 fs http buffer event os

- 运行node程序

```
// 01-run.js
console.log('hello,node.js!');
console.log('run me use: node 01-runnode!');
```

运行：`node 01-runnode.js`

> 每次修改js文件需重新执行才能生效，安装nodemon可以监视文件改动，自动重启:
>
> `npm i -g nodemon`

- 调试node程序：Debug - Start Debugging

> https://nodejs.org/en/

模块(module)

- 使用模块(module)
  - node内建模块

```
// 02-useModule.js
// 内建模块直接引入
const os = require('os')
const mem = os.freemem() / os.totalmem() * 100
console.log(`内存占用率${mem.toFixed(2)}%`)
```

- 第三方模块

> https://www.npmjs.com/

```
// 同级CPU占用率，先安装
npm i download-git-repo -s
```

```
// 导入并使用
const download = require('download-git-repo')
download('github:su37josephxia/vue-template', 'test', err => {
    console.log(err ? 'Error' : 'Success')
})
```

- 完善代码

```
const download = require('download-git-repo')
const ora = require('ora')
const process = ora(`下载.....项目`)
process.start()
download('github:su37josephxia/vue-template', 'test', err => {
    if(err){
        process.fail()
    }else{
        process.succeed()
    }
})
```

- promisefy

> 如何让异步任务串行化

```
const repo = 'github:su37josephxia/vue-template'
const desc = '../test'
clone(repo,desc)

async function clone(repo,desc) {
    const { promisify } = require('util');
    const download = promisify(require('download-git-repo'));
    const ora = require('ora');
    const process = ora(`下载项目.....`);
    process.start();
    try {
        await download(repo, desc);
    } catch (error) {
        process.fail()
    }
    process.succeed()
}
```

- 自定义模块：代码分割、复用手段

```
module.exports.clone = async function clone(repo, desc) {
    const ora = require('ora');
    const process = ora(`下载项目 ${repo}`);
```

```
    process.start();
    const { promisify } = require('util');
    const download = promisify(require('download-git-repo'));
    try {
        await download(repo, desc)
    } catch (error) {
        process.fail()
    }
    process.succeed()
}

// run
const {clone} = require('./download')
clone()
```

```
 导出内容可以是导出对象的属性
// download.js
  module.exports.clone = async function()

  // 01_run
  const {clone} = require('./down')
  clone()
```

核心API

- fs - 文件系统

```
// 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);
});
console.log('其他操作');

// fs常常搭配path api使用
const path = require('path')
fs.readFile(path.resolve(path.resolve(__dirname,'./app.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')
```

> 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
// UFT-8：一种变长的编码方案，使用 1~6 个字节来存储；
// UFT-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：用于创建web服务的模块

  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 - 是用于与node中流数据交互的接口

```
//创建输入输出流,06-stream.js
const rs = fs.createReadStream('./conf.js')
const ws = fs.createWriteStream('./conf2.js')
rs.pipe(ws);

//二进制友好，图片操作,06-stream.js
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。

- 工具链

```
mkdir vue-auto-router-cli
cd vue-auto-router-cli
npm init -y
```

```
# bin/kkb
console.log('cli.....')

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

```
npm link

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

// 引入commander

kkb文件

```
#!/usr/bin/env node
const program = require('commander')
program.version(require('../package').version, '-v', '--version')
    .command('init <name>', 'init project')
    .command('refresh','refresh routers...')
program.parse(process.argv)
```

kkb-init

```
#!/usr/bin/env node
const program = require('commander')
program
    .action(name => {
            console.log('init ' + name)
    })
program.parse(process.argv)
```

/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()
}
```

kkb-init

```
const {clone} = require('../lib/download')
            console.log('🚀创建项目: ' + name)
            await clone('github:su37josephxia/vue-template',name)
```

kkb-refresh

```
#!/usr/bin/env node

const program = require('commander')
const symbols = require('log-symbols')
const chalk = require('chalk')
// console.log(process.argv)
program
  .action(() => {
    console.log('refresh .... ')
  })
program.parse(process.argv)

const fs = require('fs')
const handlebars = require('handlebars')

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

compile({
  list
}, './src/router.js', './template/router.js.hbs')

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

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(symbols.success, chalk.green(`🚀${filePath} 创建成功`))
}
```

- 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
```