Node.js再识与异步精讲

🤩Node.js简介

官网解释

https://link.zhihu.com/?target=https%3A//developers.google.com/v8/

NodeJS 是基于Chrome V8引擎的 JavaScript 运行环境。NodeJS使用事件驱动的、非阻塞的I/O模型，这使它变得轻量级和高效。NodeJS的包管理生态是 NPM，是现在世界上最大的开源程序包库。

创始

创始人Ryan Dahl，是一位专注于实现高性能Web服务器优化的专家，一直想要解决服务器高并发问题，在Chrome V8 引擎（高性能js引擎）发布之后，基于V8引擎开发出了一个可以让js运行在服务器端的环境。

V8运行在客户端，既然在客户端是可以发请求接收返回信息，那么把它放到服务器端应该也可以，所以他改了一些V8引擎的内核，收发请求之类的，然后将V8搬到了服务器端。

nodejs不是一种独立的语言，是使用js来进行编程，运行在V8引擎上。（底层代码是C++）

NodeJS之前，JavaScript运行环境是浏览器，NodeJS之后JavaScript又多了一个运行环境，就是NodeJS。

为什么要学NodeJs

- 性能好、部署容易，能够轻松处理高并发问题。
- 它让我们可以用js写后端程序，顶层路由设计等。
- Node.js 是前端工程化的重要支柱之一。（目前前端开发环境离不开nodejs，是我们使用很多工具和提高开发效率的基础）

👀Node.js和Javascript的关系

javascript的组成

- ECMAScript（语言基础，如：语法、数据类型结构以及一些内置对象）
- DOM（一些操作页面元素的方法）
- BOM（一些操作浏览器的方法）

node.js的组成

- ECMAScript（语言基础，如：语法、数据类型结构以及一些内置对象）
- OS（操作系统）
- file（文件系统）
- net（网络系统）
- database（数据库）

总结

- node.js是平台，javascript是编程语言；
- node.js应用于后端，javascript应用于前端
- javascript是客户端编程语言，需要浏览器的javascript解释器进行解释执行；
- node.js是一个基于Chrome JavaScript运行时建立的平台，它是对Google V8引擎进行了封装的运行环境；
- 简单的说node.js就是把浏览器的解释器封装起来作为服务器运行平台，用ECMAScript语法进行编程，在node.js上运行。因此，可以将node.js看成是运行在服务端的 javascript。

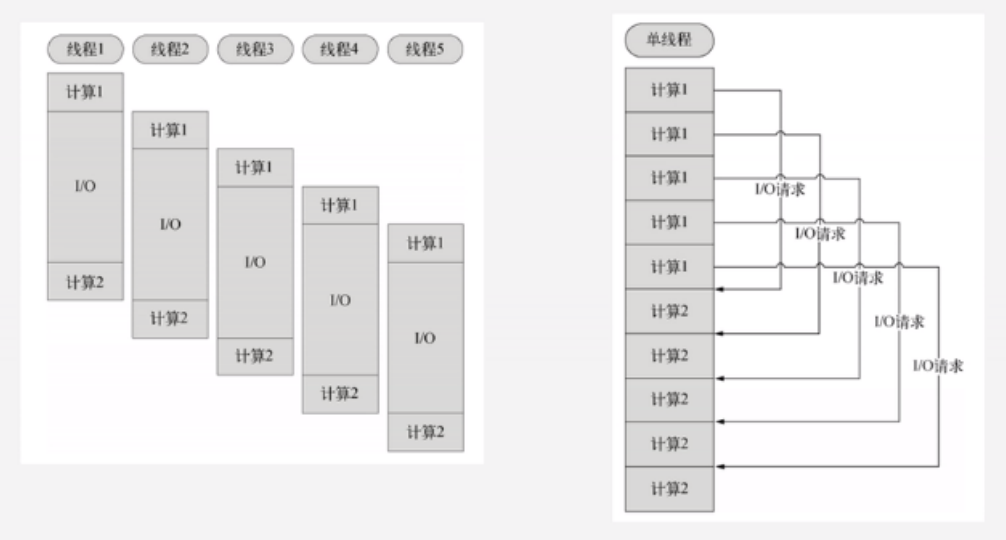
🤔NodeJs是怎么提高服务器性能的?

单线程

java/php/.net服务器语言中，为每个客户端创建一个新的线程，要让Web应用程序支持更多的用户，就需要增加服务器的数量，这样硬件成本就变高了。

nodejs只使用一个线程，用户连接就触发一个内部事件，通过非阻塞I/O、事件驱动机制，让程序宏观上是并行的。

减少了内存开销，省去了操作系统创建/销毁线程的时间开销



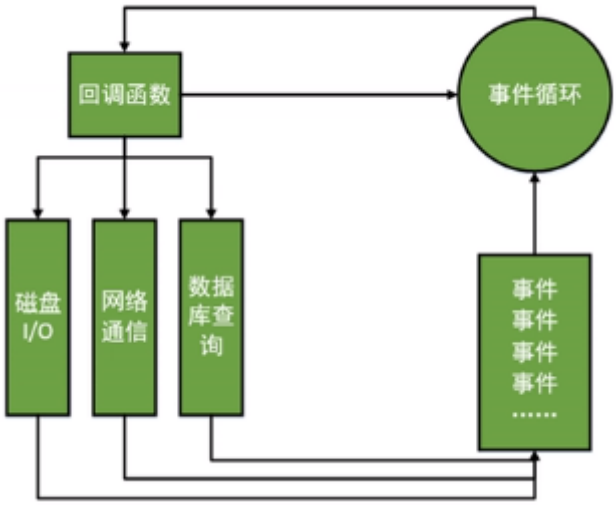
非阻塞I/O

传统单线程处理机制中，一旦有I/O操作，比如访问数据库，整个线程都会停下来，等待数据库的响应，才会执行后面的代码。也就是说I/O阻塞了代码的执行，极大地降低了代码的运行效率。

非阻塞I/O，在遇到I/O操作时，比如访问了数据库的代码后，会立即转而执行后面的代码，把对数据库返回结果的处理代码放在回调函数中。当某个I/O执行完毕，会以事件的方式通知这个线程，然后线程执行这个事件的回调函数。这个线程一直在进行计算操作，其CPU核心利用率一直是100%。

事件驱动

不管是新用户的请求还是老用户的I/O完成，都将以事件的方式加入事件循环，等待调度（注意不是排队，会优先执行老用户的回调函数）。



底层代码中近半数都用于事件队列，回调函数队列的构建。

其实三个特点是一回事。

nodejs适合做什么

善于I/O，不善于计算，即善于任务调度，不善于大量CPU计算。也非常适合与websocket配合，开发长连接的实时交互应用程序。

- 用户表单收集（百度表单填写就是用的node）
- 考试系统
- 聊天室

同时它只适用于小型开发，无法挑战PHP等老牌后台语言感觉其更重要的意义在于是一个强大的工具。但是也不要小瞧了它，他也是许多公司都在用的中间件哦。

NPM

也就是node包管理器，也用特别安装，在安装node的时候，就已经安装了。

node包就是封装了特定功能的模块，有人将一些经常使用的、成熟的功能封装成node包并将其放在了npm上。通过npm我们可以免费使用其他人做好的模块，只需下载后引入即可。当然你也可以自己造轮子为社区做贡献😁。

初始化项目时，会用到`npm init`命令，其中会生成package.json文件和node_moudules文件夹

package.json

用来定义包的属性，也是模块的描述文件。

比如

```
//简单一点的项目
{
  "name": "nodejs",
  "version": "1.0.0",
  "description": "one demo",
  "main": "demo.js",
  "scripts": {
    "test": "echo \"Error: no test specified\" && exit 1",
    "start": "node demo.js"
  },
  "author": "hxy",
  "license": "ISC"
}
```

- main 入口文件
- version 版本号 ('^'表示固定版本号)
- dependencies 项目运行所依赖的模块
- devDependencies 项目开发所需要的模块
- scripts 用于指定脚本命令(npm run + 命令)

1. dependencies是生产环境，devDependencies是开发环境。在开发项目时，npm install 会自动下载devDependencies和dependencies下面的模块。当你只是使用某个包时不会下载devDependencies下面的模块，而当你修改某个包/项目时所有依赖都会下载。
2. `npm run 为每条命令提供了pre-和post-两个钩子（hook）。以npm run lint为例，执行这条命令之前，npm会先查看有没有定义prelint和postlint两个钩子，如果有的话，就会先执行npm run prelint，然后执行npm run lint，最后执行npm run postlint。

```
//复杂一点的项目
{
  "name": "mydemo",
  "version": "0.1.0",
  "private": true,
  "scripts": {
    "serve": "vue-cli-service serve",
    "build": "vue-cli-service build",
    "lint": "eslint --ext .js,.vue src test/unit/specs"
  },
  "dependencies": {
    "core-js": "^3.6.4",
    "vue": "^2.6.11",
    "vue-router": "^3.1.5"
  },
  "devDependencies": {
    "@vue/cli-plugin-babel": "^4.2.0",
    "@vue/cli-plugin-router": "^4.2.0",
    "@vue/cli-service": "^4.2.0",
```

```
    "node-sass": "^4.12.0",
    "sass-loader": "^8.0.2",
    "vue-template-compiler": "^2.6.11"
  }
}
```

package-lock.json 只是为了确保依赖包的正常使用，比如锁定依赖树，包的版本等等。

下载包`npm install`

1.使用命令`npm install <package>`安装模块。

2.--save/ - S 模块名被添加到dependencies

--save -dev / -D 模块名被添加到devDependencies

-g 把模块安装到全局环境中

@版本号 下载相应的版本包

😎实操

nodejs官网 https://nodejs.org/en/

检查是否安装成功

· node -v

一个启动端口的小栗子💩

```
const http = require('http')
const server = http.createServer((req,res)=>{
    res.writeHead(200,{'Content-type':'text/html;charset=UTF-8'});
    res.end("嘿嘿嘿，这是我的第一个Node页面")
});
server.listen(8080,'127.0.0.1');
console.log('Server is running at http://127.0.0.1:8080/');
```

基本模块

`fs`

`fs模块`就是文件系统模块，负责读写文件。`nodeJs`内置的`fs模块`提供了异步和同步的方法

异步

```
//异步读取文件
fs.readFile("./demo.txt", function (err, data) {
  if (err) {
    console.log(err);
```

```
  } else {
    console.log("data:" + data.toString());
  }
});


//异步写入文件
const data = {
  'name' : 'hy',
  'age'  : '16',
  'id'   : '2022'
}

const dataText = JSON.stringify(data)

const option = {
  encoding : 'utf8' ,
  flag     : 'w'
}

fs.writeFile('demo.txt' ,dataText, option , function(err){
  if(err){
    console.log('写文件失败')
  }else{
    console.log('写文件成功')
  }
} )
```

同步

```
//同步读文件
file = fs.readFileSync(filePath , 'utf-8')

//同步写入文件
fs.writeFileSync(filename, data[, options])
```

相关方法

```
//列出文件
fs.readdir(path , callback)
//删除文件
fs.unlink(path , callback)
//截断文件
fs.truncate(path , len ,  callback) 返回true和flase
//创建目录
 fs.mkdir(path , [mode] , callback)
//删除目录
fs.rmkdir(path , callback)
```

`url`

即用来处理和解析url的模块

```
//parse  用来解析url,返回url属性对象
const url = require("url")

const myURLA = url.parse('https://www.baidu.com/p/a/t/h?query=string#hash');
console.log(myURLA);
/**
Url {
  protocol: 'https:',                协议
  slashes: true,
  auth: null,                        用户密码
  host: 'www.baidu.com',             主机名
  port: null,                        端口号
  hostname: 'www.baidu.com',         主机名（不带端口号）
  hash: '#hash',                     哈希值
  search: '?query=string',           查询字符串
  query: 'query=string',             请求参数
  pathname: '/p/a/t/h',              路径名
  path: '/p/a/t/h?query=string',     带查询的路径名
  href: 'https://www.baidu.com/p/a/t/h?query=string#hash' }
*/
```

`resolve`

```
//以一种 Web 浏览器解析超链接的方式把一个目标 URL 解析成相对于一个基础 URL
url.resolve('http://example.com/', '/one');    // 'http://example.com/one'
url.resolve('http://example.com/one', '/two')  // 'http://example.com/two'
```

`format`

```
let urlobj =  url.format({
    protocol: 'https',
    hostname: 'example.com',
    pathname: '/some/path',
    query: {
      page: 1,
      format: 'json'
    }
  });  //https://example.com/some/path?page=1&format=json
```

`path`

`path.join`

用于连接路径。该方法的主要用途在于，会正确使用当前系统的路径分隔符，Unix系统是"/"，Windows系统是"\"。

```
console.log(path.join("/path" , "/user"));  //\path\user
```

`path.resolve`

```
//path.resolve([from ...], to)
//将 to 参数解析为绝对路径，给定的路径的序列是从右往左被处理的，后面每个 path 被依次解析，直到构造完成一个绝对路径

console.log(path.resolve("" , "../"))   //C:\Users\Administrator\Desktop
console.log(path.resolve("" , "./"))   //C:\Users\Administrator\Desktop\node-learn
```

`http`

```
//server.js
//引入模块
const http = require('http')

//创建http服务器
const server = http.createServer((req , res)=>{
    const {url , method } = req
    console.log("url:%s,method:%s" , url , method)
    //监听客户连接事件
    req.on("connection" , (socket)=>{
        console.log("客户端连接" , socket)
    })
    //接受传送数据
    req.on("data", data=>{
        console.log("recive data" , data)
    })
    req.on("end" ,()=>{
        //返回数据
        res.end('hello world')
    })
})

//监听8000端口,等待连接
server.listen(8000 , ()=>{
    console.log("server start")
})


//client.js
const http = require("http")

const postData = "test"
  const options = {
    hostname: 'localhost',
    port: 8000,
    path: '/',
    method: 'POST',
    headers: {
```

```
      'Content-Type': 'application/x-www-form-urlencoded',
      'Content-Length': Buffer.byteLength(postData)
    }
  };

  const req = http.request(options, (res) => {
    console.log(`状态码: ${res.statusCode}`);
    console.log(`响应头: ${JSON.stringify(res.headers)}`);
    res.setEncoding('utf8');
    res.on('data', (chunk) => {
      console.log(`响应主体: ${chunk}`);
    });
    res.on('end', () => {
      console.log('响应中已无数据');
    });
  });

  req.on('error', (e) => {
    console.log(`请求遇到问题: ${e.message}`);
  });

  // 将数据写入请求主体。
  req.write(postData);
  req.end();
```

作业

lv1：用nodejs做一个简易版的网络爬虫，只用获取页面的简单元素即可。

lv2: 用node的基本模块做一个简易版的服务器，前端传什么返回什么，注意跨域问题。