Node

> **原型链和原型：**
> 先找原型对象的自身-> 先找自身.*proto* ->构造函数.prototype （对象）-> 构造函数.prototype.*proto* -> Object.prototype

node

node 之前主要是为了会用。
现在都安装了node。
简单来说，node.js就是运行在服务端的 JavaScript
Node.js是一个基于Chrome JavaScript 运行时建立的一个平台。

Node.js是一个基于Chrome V8引擎的JavaScript 的运行环境。
特征：（缺一不可）

- **单线程**
  服务器都有多个线程池，有线程池占用，就可以用其他的。->多线程
  单线程：就是只有一个办事路径。
- **非阻塞I/O**（I/O -> 输入输出 就相当于一来一回。）
  非阻塞 -> 输入后不需要在队列中等待。-> 执行任务时不堵塞通道。
- **事件驱动**
  类似回调函数，达到某个条件后执行。

node的应用场景：聊天室、爬虫、hexo、论坛
适用于：高并发、I/O密集、少量业务逻辑的场景。
node的应用场景及实践：https://blog.csdn.net/levin217/article/details/78735094

node达到了前后端统一。都用的JS。

小例子：
新建服务-> 按下shift右键打开管理员窗口， 输入node 和要运行的js 文件 回车就行了。

打印都是在服务器下打印。主要看命令行。
文件中的代码写法就是JS的写法。（不需要添加其他的，直接写代码就行）

require（引入）和module.exports（导出）

引入http，到默认文件夹下找一个叫 http.js 的文件

http是没有加路径的，因为它是自带的。如果要用一个自己定义的js文件，那么基本上都要写上路径（自定义文件的路径）。

require：引入

- require（指定文件）指定JS文件为js文件时，可不用写.js

```
let a = require('./3_module');
console.log(a)  // -> 服务器里面会打印  a  引入模块的导出内容
```

module.exports：导出 –> 导到前面被引入的文件

`module:英文意思：模块；exports英文意思：输出。 -> 模块输出/导出`

一般一个文件里面只有一次导出，否则后面的导出会覆盖前面的。

```
// 这个为导出模块，把这个文件引入到需要用的文件中。
module.exports.a = 10;
```

语法就是JS，

创建服务：

```
// 创建新服务的格式
// 1、先引入  http（创建新服务用）、fs(对文件进行操作)、
http.createServer( ( req, res)=>{
    req.url   // -> 接受到浏览器发送的地址栏信息
    res.write( "xxx" );  // 发送东西。
    res.end()            // 发送结束。
    } )
```

response：（响应 -> 就是）给客户端（浏览器）发送东西；
request：（请求->就是）接受客户端（浏览器）发送的信息。

```
// 此文件为服务器连接的文件。
let http = require('http');   // 此处引入http文件，是为了创建一个新服务。
http.createServer((request, response)=>{  // 创建一个服务
    console.log(request)
    response.write("xyz")  // 服务器给浏览器反馈的东西
    response.end();       //  end() 代表结束，必须写
})listen(80);             // 监听-> 保持开启状态。括号内是端口号。一般是80，可自己设置。
```

```
// 小例子：放到js文件中，连上服务器就能直接在浏览器中打开使用。
let http = require('http')  // 引入http  为创建新服务做准备
```

```
http.createServer((requst, responer)=>{ // 创建一个新服务。箭头函数内有两个形参：请求，发送。
    let name = request.url.split('=')[1]; // 创建新变量，name并为它赋值请求的路径=号后面的内容
    switch(name){                          // 创建一个条件语句，如果变量name  等于 ？
        case 'xyz':                        // 如果变量name等于 'xyz'
            response.write('{code:0, msg:chy}'); //那么发送''里的内容;
        break;
        case 'hongdan':
            response.write('{code:0, msg:cmy}');
        break;
        default:                          // 如果都不等于，
            response.write('{conde:1, msg:cyy}'); 那么发送：''内的内容
        break;
    }
    response.end();         // 发送结束
}).listen(80);              // 事件监听->保持服务器常开，端口为80，可改
```

> 模块化：分开写，便于管理。

fs模块（操作数据）

读取文件：

- fs模块：这个模块专门是用来操作（增删改查）文件的。
- fs模块是node自带的，和http一样。

fs.readFile(path,callback); 读取信息，第一个参数为读取文件的路径，第二个为回调函数。
callback(error, data) 回调函数里面有两个参数，第一个参数为读取报错的意思；第二个函数为当读取成功时，读取的信息。（当有数据返回，data就是读取的文件信息。）

```
const fa = require('fs');      // 引入fs 为读取信息做准备。
fs.readFile('./www/1.txt', (error, data)=>{ // 读取路径中的文件。并设置一个回调函数。

    // 第一种写法
    if (error) {                // 如果读取失败
        console.log('404');    // 打印404
    }else{                          // 如果读取成功就运行下面代码
        console.log(data.toString()) // 此处是把data的内容转成字符串。
    }
```

```
    console.log（111）    // 当前面报错时，此处的打印也打印不出来。   代表下面的代码都不能被执行，如果想让下面的代码执行，需要换种写法。

    // 第二种想法
    try{              // 运行里面的代码
        let data = fs.readFileSync('./www/1.txt');
        console.log(data.toString());
    }catch(error){

    }
    console.log(111) // 这种写法不会造成代码阻塞。   上面还是在读取，不管成功与否不影响后面的代码。

})

// let data = fs.readFileSync
```

是请求接口还是请求页面？
如果是”?” 走接口；不是？ 就走页面（文件）

```
let http = require('http');  // 准备创建新服务
let fs = require('fs');      // 引入fs 准备读取数据
http.createServer((req,res)=>{  // 创建新服务
    let url = req.url           // 定义变量，赋值为请求的路径
    if(url.includes('?')){      // 判断url 里面有没有'?'，有的话走的接口。
        res.writeHead(200, {'Content-Type':'text/html;charset=utf-8'});// 有这句下面的中文可以正常显示，不加的话是乱码。
        res.write('后天去玩');  // 页面显示：后天去玩
        res.end();    //  发送结束
     }else{                  // 如果里面没有'? ' 那就请求页面
         if(url === '/')url = '/index.html'; // 如果里面只有'/' 就请求后面路径上的文件，生成页面
         fs.readFile('./www'+url, (err, data)=>{ // 读取路径上的文件
            if(err){    // 读取失败后
                res.write('404');
            }else{      // 读取成功后
                res.write('./1.txt');
            }
            res.end()   // 发送结束
        });
    }
}).listen(80);
```

写文件：fs-writefile

```
let fs = require('fs')    // 引入'fs'   准备读写数据

//  创建新文件。
fs.writeFile('2.txt', 'dsnadkjsjdksa', (err)=>{  // 在同级创建2.TXT的文件，    第一个参数是文件的路径，内容为第二个参数。
    if(error){
        console.log('失败')
    }else{
        console.log('成功')
    }
})

// 删除指定的文件
fs.unlinkSync('./www/2.txt');
```

```
const http = require('http');       // 引入 http，准备创建新服务。
const fs = require('fs');           // 引入fs ，准备读取数据
const qs = require('querystring'); //a=b&c=d -> {a:b,c:d}  把路径上的字符串变成对象。
http.createServer((req,res)=>{      // 创建新服务
    res.writeHead(200,{'Content-Type':'text/html; charset=utf-8'});
//可使用中文
    let url = req.url;              // 创建变量url 并赋值请求的路径。
    /*
        rm:路径
        http://localhost?rm=地址
    */
    if(url.includes('?')){        // 如果url 里面有'?'  就运行下面的代码
        let obj = qs.parse(url.split('?')[1]);  // 创建变量obj 并赋值为 url中? 后面的部分->数据类型转为对象

        if(obj.rm){               // 如果obj里面的rm 不为空。那就运行下面的代码
            try {
                fs.unlinkSync(obj.rm);
                res.write('{"code":0,"msg":"删除成功"}');
            } catch (error) {
                console.log('删除失败');
                console.log(obj.rm);
                res.write('{"code":1,"msg":"删除失败"}');
            }
            res.end();           // 发送结束
        }

    }else{                      // 如果请求路径里面没有 '?'  就运行下面代码
```

```
        if(url === '/')url = '/index.html';  // 如果url后面的内容为'/'
那么请求地址为。。。

        fs.readFile('./www'+url,(error,data)=>{  // 读取信息。
            if(error){
                res.write(fs.readFileSync('./www/404.html'));
            }else{
                res.write(data);
            }
            res.end();        // 发送结束
        });

    }

}).listen(80);
```