# 一、node介绍

## 1.1 为什么要学习node.js

## 1.2 node.js是什么

## 1.3 node能做什么

# 二、node起步

## 2.1 安装node环境

## 2.2 解析执行JavaScript

1. 创建编写JavaScript脚本文件

2. 终端文件所在目录，运行： ‘node 文件名’

注：文件名不要‘ node.js ’ 命名

## 2.3 http

服务器：

```javascript

// 1. 加载http核心模块

var http = require('http')

// 2. 创建服务器

var server = http.createServer()

// 3. 服务器要做的事

// 提供服务：对数据服务

// 发请求：

//  接收请求

//  处理请求

//  反馈（发送响应）

//  当客户端请求过来，就会自动触发服务器的request请求事件，然后执行第二个参数：回调处理函数

server.on('request', () => {

    console.log('receive request from client');

})

// server.on('request', (request, response) => {

//     console.log('receive');

// })

// 4. 绑定端口号，启动服务

server.listen(3000, () => {

    console.log('sever is running...');

})

```

## 2.4 文件的读写

1. 浏览器中，JS没有文件的读写操作能力；在node中，js具有文件操作能力

2. js是file-system，即文件系统。Node中，文件的操作，需引这个核心模块（提供所有文件操作的API）

// 1. 加载fs核心模块

var fs = require('fs')

// 2. 读取文件

fs.readFile('./a.txt', (err, data) => {

    if(err) {

        console.log('read error');

    }

    // 这里的data是二进制数，使用tostring转成字符串

    console.log(data.toString());

})

// 3. 写入文件（注意，如果文件中已经有内容，使用以下操作会使新写入的内容覆盖原内容）

fs.writeFile('./a.txt', 'write something into this file, go', (err, data) => {

    if(err) {

        console.log('write error');

    }

    console.log(data.toString());

})

# 三、node中的模块系统

浏览器端模块化规范：AMD、CMD

服务器端：commonJS

兼并服务器和浏览器端规范：Es6

使用node编写应用程序时，主要就是使用：

-EcmaScript语言

    -和在浏览器中使用一样，只不过在node中，没有Bom和Dom

-核心模块

    -文件操作的fs

    -http服务器操作的模块http

    -url路劲操作模块

    -path路劲操作模块

    -os操作系统信息

-第三方模块：必须使用npm install xxx 下载才可使用

    -art-template

-自己写的模块

    -即自己创建的文件

## 3.1 什么是模块化

-自己写的模块

    -即自己创建的文件

    -一个文件即使一个模块，在node中，【没有全局作用域，它是文件作用域】，即模块之间是独立的，在不同的文件使用时必须要重新引用，并且文件内的变量没有exports时，不能被外界使用。

    -通信规则：

        -加载require

        -导出exports

## 3.2 CommonJS模块规范

node中，js还有一个很重要的概念，模块系统

    -模块作用域（而非全局作用域）

    -使用require方法加载模块

    -使用exports接口对象导出模块中的成员

### 3.2.1 加载require

// 加载：require

var 自定义名 = require('模块名')

这里require有两个作用；

    -执行被加载模块中的代码

    -得到被加载的模块中，exports导出的接口对象

### 3.2.2 导出exports

// 导出：exports

    -node是模块作用域，默认文件中所有的成员只在当前模块有效

    -对于希望可以被其他模块访问的成员，需要把这些成员都挂载到export接口对象中即可

    导出多个成员（必须在对象中）：

        exports.a = 123

        exports.b = function() {

            console.log('i am b');

        }

        exports.c = {

            foo: 'foo',

        }

        exports.c = 'hello'

导出单个成员

        module.exports = 'hello'

        以下情况会覆盖：(后者会覆盖前者)

        module.exports = 'hello'

        module.exports = function add(a, b) {

            return a + b

        }

        通过以下方法导出多个成员：

        module.exports = {

            foo = 'foo',

            add: function(a, b) {

                return a + b

            }

        }

## 3.3 模块原理

exports和module.exports的一个引用：

console.log(exports === module.exports);    // true

exports.foo = 'bar'

// 等价于

module.exports.foo = 'bar'

// 但是

exports = 'bar'

// 不等价于

module.exports = 'bar'

// 当给exports重新赋值后，exports！= module.exports.

// 最终return的是module.exports，而exports中的成员是何无用

// 在使用的时候：

    -导出单个成员：exports.xxx = xxx

    -导出多个成员：module.exports 或 module.exports = {}

总结：

// 引用服务

var http = require('http');

var fs = require('fs');

// 引用模板

var template = require('art-template');

// 创建服务

var server = http.createServer();

// 公共路径

var wwwDir = 'D:/app/www';

server.on('request', function (req, res) {

    var url = req.url;

    // 读取文件

    fs.readFile('./template-apche.html', function (err, data) {

        if (err) {

            return res.end('404 Not Found');

        }

        fs.readdir(wwwDir, function (err, files) {

            if (err) {

                return res.end('Can not find www Dir.')

            }

            // 使用模板引擎解析替换data中的模板字符串

            // 去xmpTempleteList.html中编写模板语法

            var htmlStr = template.render(data.toString(), {

                title: 'D:/app/www/ 的索引',

                files:files

            });

            // 发送响应数据

            res.end(htmlStr);

        })

    })

});

server.listen(3000, function () {

    console.log('running....');

})

1.jQuery中的each 和 原生JavaScript方法forEach的区别：

  提供源头：

      原生js是es5提供的（不兼容IE8）,

        jQuery的each是jQuery第三方库提供的（如果要使用需要用2以下的版本也就是1.版本）,它的each方法主要用来遍历jQuery实例对象（伪数组）,同时也可以做低版本forEach的替代品,jQuery的实例对象不能使用forEach方法，如果想要使用必须转为数组（[].slice.call(jQuery实例对象)）才能使用

2.模块中导出多个成员和导出单个成员

3. 301和302的区别：

    -301永久重定向，浏览器会记住

    -302临时重定向

4. exports 和 module.exports 的区别：

    每一个模块中都有一个 module 对象

        module 对象中有一个exports对象

        我们可以把需要导出的成员都挂载到 module.exports 接口对象中：module.exports.xxx = xxx

        但是每次写太多会很麻烦，所以node为了简化代码，在每一个模块中都提供了一个成员：exports

        exports === module.exports 为true，即可用 exports.xxx = xxx 代替

        当一个模块需要导出单个成员的时候，需使用 module.exports = xxx 的方式

        而使用 exports = xxx 不管用，因为每个模块最终return 的是 module.exports, exports 只是module.exports的一个引用，所以，即使 exports 重新赋值，也不会影响 module.exports 【因为他们都是对象】

        除非：exports = module.exports 来新建立引用关系。

# 四、require的加载规则

- 核心模块

  - 模块名

- 第三方模块

  - 模块名

- 用户自己写的

  - 路径

## 4.1 require的加载规则

-优先从缓存加载

-判断模块标识符

    -核心模块

    -自己写的模块（路劲形式的模块）

    -第三方模块（node\_modules)

        - 第三方模块的标识就是第三方模块的名称（不可能有第三方模块和核心模块的名字一致）

        - npm

            - 开发人员可以把写好的框架库发布到npm上

            - 使用者通过npm命令来下载

        - 使用方式：`var 名称 = require('npm install【下载包】 的包名')`

            - node\_modules/express/package.json main

            -如果package.json或main不成立，则查找被选择项：index.js

            - 如果以上条件都不满足，则继续进入上一级目录中的node\_modules按照上面的规则依次查找，直到当前文件所属此盘根目录都找不到最后报错

路径形式的标识：

    ./ 当前目录 不可省略

    ../ 上一级目录 不可省略

    /xxx 即 D:/xxx

    带有绝对路径几乎不用（D:/a/foo.js）

    首位表示的是当前文件模块所属磁盘根目录

    require('./a')

核心模块

核心模块本质也是文件，核心模块文件已经被编译到了二级制文件中，只需要按照名字来加载即可

require('fs')

// 第三方模块

// 凡是第三方模块都必须通过npm下载（npm i node\_modules），使用的时候就可以通过require('包名')来加载才可以使用

// 第三方包的名字不可能和核心模块的名字是一样的

// 既不是核心模块，也不是路径形式的模块

//      先找到当前文所述目录的node\_modules

//      然后找node\_modules/art-template目录

//      node\_modules/art-template/package.json

//      node\_modules/art-template/package.json中的main属性

//      main属性记录了art-template的入口模块

//      然后加载使用这个第三方包

//      实际上最终加载的还是文件

//      如果package.json不存在或者mian指定的入口模块不存在

//      则node会自动找该目录下的index.js

//      也就是说index.js是一个备选项，如果main没有指定，则加载index.js文件

//

        // 如果条件都不满足则会进入上一级目录进行查找

// 注意：一个项目只有一个node\_modules，放在项目根目录中，子目录可以直接调用根目录的文件

var template = require('art-template');

模块标识符中的‘/’和文件操作路径中的‘/’

    -文件操作路径：

    // 咋们所使用的所有文件操作的api都是异步的。就像ajax一样

    // 读取文件

    // 文件操作中的 ./ 相当于当前模块所处磁盘根目录

    // ./index.txt 相对于当前目录

    // /index.txt 相对于当前目录

    // /index.txt 绝对路径，当前文件模块所处根目录

    // d:express/index.txt 绝对路径

    fs.readFile('./index.txt', (err,data) => {

        if(err) {

            return console.log('read error');

        }

        console.log(data.toString());

    })

    -模块操作路径：

    // 在模块加载中，相对路径中的./ 不能省略

    // 省略了.  也是磁盘根目录

    require('./index') ('hello')

# 五、npm

- node package manage(node包管理器)

- 通过npm命令安装jQuery包（npm install --save jquery），在安装时加上--save会主动生成说明书文件信息（将安装文件的信息添加到package.json里面）

## 5.1 npm网站

> ​ npmjs.com 网站   是用来搜索npm包的

## 5.2 npm命令行工具

npm是一个命令行工具，只要安装了node就已经安装了npm。

npm也有版本概念，可以通过`npm --version`来查看npm的版本

升级npm(自己升级自己)：

```javascript

npm install --global npm

```

## 5.3 常用指令

- npm init(生成package.json说明书文件)

  - npm init -y(可以跳过向导，快速生成)

- npm install

  - 一次性把dependencies选项中的依赖项全部安装

  - 简写（npm i）

- npm install 包名

  - 只下载

  - 简写（npm i 包名）

- npm install --save 包名

  - 下载并且保存依赖项（package.json文件中的dependencies选项）

  - 简写（npm i  包名）

- npm uninstall 包名

  - 只删除，如果有依赖项会依然保存

  - 简写（npm un 包名）

- npm uninstall --save 包名

  - 删除的同时也会把依赖信息全部删除

  - 简写（npm un 包名）

- npm help

  - 查看使用帮助

- npm 命令 --help

  - 查看具体命令的使用帮助（npm uninstall --help）

## 5.4 解决npm被墙问题

npm存储包文件的服务器在国外，有时候会被墙，速度很慢，所以需要解决这个问题。

> https://developer.aliyun.com/mirror/NPM?from=tnpm淘宝的开发团队把npm在国内做了一个镜像（也就是一个备份）。

>

安装淘宝的cnpm：

```javascript

npm install -g cnpm --registry=https://registry.npm.taobao.org;

```

```shell

#在任意目录执行都可以

#--global表示安装到全局，而非当前目录

#--global不能省略，否则不管用

npm install --global cnpm

```

安装包的时候把以前的`npm`替换成`cnpm`。

```shell

#走国外的npm服务器下载jQuery包，速度比较慢

npm install jQuery;

#使用cnpm就会通过淘宝的服务器来下载jQuery

cnpm install jQuery;

```

如果不想安装`cnpm`又想使用淘宝的服务器来下载：

```shell

npm install jquery --registry=https://npm.taobao.org;

```

但是每次手动加参数就很麻烦，所以我们可以把这个选项加入到配置文件中：

```shell

npm config set registry https://npm.taobao.org;

#查看npm配置信息

npm config list;

```

只要经过上面的配置命令，则以后所有的`npm install`都会通过淘宝的服务器来下载

# 六、package.json

每一个项目都要有一个`package.json`文件（包描述文件，就像产品的说明书一样）

这个文件可以通过`npm init`自动初始化出来

D:\code\node中的模块系统>npm init

This utility will walk you through creating a package.json file.

It only covers the most common items, and tries to guess sensible defaults.

See `npm help json` for definitive documentation on these fields

and exactly what they do.

Use `npm install <pkg>` afterwards to install a package and

save it as a dependency in the package.json file.

Press ^C at any time to quit.

package name: (node中的模块系统)

Sorry, name can only contain URL-friendly characters.

package name: (node中的模块系统) cls

version: (1.0.0)

description: 这是一个测试项目

entry point: (main.js)

test command:

git repository:

keywords:

author: xiaochen

license: (ISC)

About to write to D:\code\node中的模块系统\package.json:

{

  "name": "cls",

  "version": "1.0.0",

  "description": "这是一个测试项目",

  "main": "main.js",

  "scripts": {

    "test": "echo \"Error: no test specified\" && exit 1"

  },

  "author": "xiaochen",

  "license": "ISC"

}

Is this OK? (yes) yes

对于目前来讲，最有用的是`dependencies`选项，可以用来帮助我们保存第三方包的依赖信息。

如果`node\_modules`删除了也不用担心，只需要在控制面板中`npm install`就会自动把`package.json`中的`dependencies`中所有的依赖项全部都下载回来。

- 建议每个项目的根目录下都有一个`package.json`文件

- 建议执行`npm install 包名`的时候都加上`--save`选项，目的是用来保存依赖信息

## ## package.json和package-lock.json

npm 5以前是不会有`package-lock.json`这个文件

npm5以后才加入这个文件

当你安装包的时候，npm都会生成或者更新`package-lock.json`这个文件

- npm5以后的版本安装都不要加`--save`参数，它会自动保存依赖信息

- 当你安装包的时候，会自动创建或者更新`package-lock.json`文件

- `package-lock.json`这个文件会包含`node\_modules`中所有包的信息（版本，下载地址。。。）

  - 这样的话重新`npm install`的时候速度就可以提升

- 从文件来看，有一个`lock`称之为锁

  - 这个`lock`使用来锁版本的

  - 如果项目依赖了`1.1.1`版本

  - 如果你重新install其实会下载最细版本，而不是`1.1.1`

  - ``package-lock.json``的另外一个作用就是锁定版本号，防止自动升级

## 6.1 path路径操作模块

> 参考文档：https://nodejs.org/docs/latest-v13.x/api/path.html

- path.basename：获取路径的文件名，默认包含扩展名

- path.dirname：获取路径中的目录部分

- path.extname：获取一个路径中的扩展名部分

- path.parse：把路径转换为对象

  - root：根路径

  - dir：目录

  - base：包含后缀名的文件名

  - ext：后缀名

  - name：不包含后缀名的文件名

- path.join：拼接路径

- path.isAbsolute：判断一个路径是否为绝对路径![image-20200315150610001](C:\Users\A\AppData\Roaming\Typora\typora-user-images\image-20200315150610001.png)

# 七、node中的其他成员(\_\_dirname, \_\_filename)

在每个模块中，除了require和exports等模块相关API之外，还有两个特殊的成员：

\_\_dirname: 动态的 获取当前文件模块所属目录的绝对路径

\_\_filename: 动态的 获取当前文件的绝对路径

在文件操作中，使用相对路径不可靠。因为在node中，文件操作的路径被设计为相对于执行node命令所处的路径。（不是bug，这样是有使用场景的）,而不是文件的相对路径

所以，为了解决这个问题，只需要将相对路径，变为绝对路径。

这里就可以使用：\_\_dirname和\_\_filename

例如：fs.readFile()

'./a.txt' ===> \_\_dirname + '/a.txt'

在拼接路径的过程中，为了避免手动拼接带来的一些错误，推荐使用path.join() 来辅助拼接，例如：

path.join(\_\_dirname, './a.txt')

在文件操作中，使用的相对路径统一转换为动态的绝对路径。

-注意，模块中的路径表示和这里的路径没有关系，不受影响（相对于文件模块）。例如：

// 模块中的路径标识和文件操作中的相对路径表示不一样

// 模块中的路径标识,就是相对于当前文件模块,不受执行node命令所处路径影响

require('./b.js')

# 八、Express

作者：Tj

原生的http在某些方面表现不足以应对我们的开发需求，所以就需要使用框架来加快我们的开发效率，框架的目的就是提高效率，让我们的代码高度统一。

在node中有很多web开发框架。主要学习express

- `http://expressjs.com/`,其中主要封装的是http。

 // 1 安装

  // 2 引包

  var express = require('express');

  // 3 创建服务器应用程序

  //      也就是原来的http.createServer();

  var app = express();

  // 公开指定目录

  // 只要通过这样做了，就可以通过/public/xx的方式来访问public目录中的所有资源

  // 在Express中开放资源就是一个API的事

  app.use('/public/',express.static('./public/'));

  //模板引擎在Express中开放模板也是一个API的事

  // 当服务器收到get请求 / 的时候，执行回调处理函数

  app.get('/',function(req,res){

      res.send('hello express');

  })

  // 相当于server.listen

  app.listen(3000,function(){

      console.log('app is runing at port 3000');

  })

## 8.1 学习Express，起步，安装

cnpm install express

// 引入express

var express = require('express');

// 1. 创建app

var app = express();

//  2.

app.get('/',function(req,res){

    // 1

    // res.write('Hello');

    // res.write('World');

    // res.end()

    // 2

    // res.end('hello world');

    // 3

    res.send('hello world');

})

app.listen(3000,function(){

    console.log('express app is runing...');

})

## 8.2 第一个程序Hello World

## 8.3 基本路由

路由：

-请求方法

-请求路径

-请求处理函数

get:

```javascript

//当你以get方法请求/的时候，执行对应的处理函数

app.get('/',function(req,res){

    res.send('hello world');

})

```

post:

```javascript

//当你以post方法请求/的时候，执行对应的处理函数

app.post('/',function(req,res){

    res.send('hello world');

})

```

## 8.4 Express静态服务API

// app.use不仅仅是用来处理静态资源的，还可以做很多工作(body-parser的配置)

app.use(express.static('public'));

app.use(express.static('files'));

app.use('/stataic',express.static('public'));

```javascript

// 引入express

var express = require('express');

// 创建app

var app = express();

// 开放静态资源

// 1.当以/public/开头的时候，去./public/目录中找对应资源

// 访问：http://127.0.0.1:3000/public/login.html

app.use('/public/',express.static('./public/'));

// 2.当省略第一个参数的时候，可以通过省略/public的方式来访问

// 访问：http://127.0.0.1:3000/login.html

// app.use(express.static('./public/'));

// 3.访问：http://127.0.0.1:3000/a/login.html

// a相当于public的别名

// app.use('/a/',express.static('./public/'));

//

app.get('/',function(req,res){

    res.end('hello world');

});

app.listen(3000,function(){

    console.log('express app is runing...');

});

```

## 8.5 在Express中配置使用art-template模板引擎

- [art-template官方文档](https://aui.github.io/art-template/)

- 在node中，有很多第三方模板引擎都可以使用，不是只有`art-template`

  - 还有ejs，jade（pug），handlebars，nunjucks

安装：

```shell

npm install --save art-template

npm install --save express-art-template

//两个一起安装

npm i --save art-template express-art-template

```

配置：

```javascript

app.engine('html', require('express-art-template'));

```

使用：

```javascript

app.get('/',function(req,res){

    // express默认会去views目录找index.html

    res.render('index.html',{

           title:'hello world'

    });

})

```

如果希望修改默认的‘views’视图渲染存储目录，可以：

```javascript

// 第一个参数views千万不要写错

app.set('views',目录路径);

```

## 8.6 在Express中获取表单请求数据

### 8.6.1 获取get请求数据

Express内置了一个api，可以直接通过`req.query`来获取数据

```javascript

// 通过requery方法获取用户输入的数据

// req.query只能拿到get请求的数据

 var comment = req.query;

```

### 8.6.2 获取post请求数据

在Express中没有内置获取表单post请求体的api，这里我们需要使用一个第三方包`body-parser`来获取数据。

安装：

```javascript

npm install --save body-parser;

```

配置：

// 配置解析表单 POST 请求体插件（注意：一定要在 app.use(router) 之前 ）

```javascript

var express = require('express')

// 引包

var bodyParser = require('body-parser')

var app = express()

// 配置body-parser

// 只要加入这个配置，则在req请求对象上会多出来一个属性：body

// 也就是说可以直接通过req.body来获取表单post请求数据

// parse application/x-www-form-urlencoded

app.use(bodyParser.urlencoded({ extended: false }))

// parse application/json

app.use(bodyParser.json())

使用：

```javascript

app.use(function (req, res) {

  res.setHeader('Content-Type', 'text/plain')

  res.write('you posted:\n')

  // 可以通过req.body来获取表单请求数据

  res.end(JSON.stringify(req.body, null, 2))

})

```

## 8.7 在Express中配置使用express-session插件操作

参考文档： <https://www.npmjs.com/package/express-session>

安装：

配置：

使用：

var session = require('express-session')

// 再路由之前

app.use(session({

    // // 配置加密字符串,它会在原有加密基础之上和这个字符串拼接起来,加密

    // 安全性,防止用户端恶意伪造

    secret: 'itcat',

    resave: false,

    // 无论你是否使用session, 我都默认直接给你分配一把钥匙,设为true,改为false是在只有数据操作的时候

    saveUninitialized: false

  }))

// 注册成功,使用session记录用户的登录状态

            // 服务器重启,session数据丢失

            req.session.user = user

-默认session数据是内存存储的，服务器一旦重启就会丢失，真正的生产环境会把session进行持久化存储。

-插件也是工具，只需要明确目标就可以了

-我们的目标就是使用session帮我们管理一些敏感信息数据状态。例如保存登录状态

-写session、读session、删session（req.session.xxx= null 更严谨的做法是： delete req.session.xxx）

## 8.8 利用Express实现CRUD项目

### 8.8.1 模块化思想

模块如何划分：

-模块职责单一

-前端三大框架

-JavaScript模块化：

-node中的commonjs

-浏览器中的

-AMD require.js

-CMD sea.js

-es6中的

-编译工具

### 8.8.2 起步

### 8.8.3 路由设计

### 8.8.4 提取路由模块

### 8.8.5 设计操作数据的API文件模块

### 8.8.6 步骤

-处理模板

-配置开放静态资源

-配置模板引擎

-简单路由，/students渲染静态页出来

-路由设计

-提取路由模块

-由于接下来的一系列的操作需要处理文件数据，所以我们需要封装students.js

-先写好students.js文件结构

-查询所有学生列表API find

-findById

-save

-updateById

-deleteById

-实现具体功能

-通过路由收到请求

-就收请求中的数据（get: req.query, post: req.body）

-调用数据API 处理数据

-根据操作结果给客户端发送响应

-业务功能顺序

-列表

-添加

-编辑

-删除

### 8.8.7 子模版和模板的继承（模板引擎高级语法）【include、extend、block】

# 九、MongoDB

-灵活、不用设计数据表

-业务的改动不需要关心数据表结构

+DBA 架构师 级别的工程师都需要掌握这项技能：设计、维护、分布式计算

## 9.1 关系型和非关系型数据库

### 9.1.1 关系型数据库

表就是关系

或者说表与表之间存在关系

-所有的关系型数据库都需要通过sql语言来操作

-所有的关系型数据库再操作之前都要设计表结构

-而且数据表还支持约束

-唯一的

-主键

-默认值

-非空

### 9.1.2 非关系型数据库

-非常灵活

-有的非关系型数据库就是key-value对儿

-但是MongoDB就是长的最像关系型数据库的非关系型数据库

-数据库 -》数据库

-数据表 -》集合（数组）

-表记录 -》（文档对象）

-MongoDB不需要设计表结构

-即你可以任意的网里面存放数据，没有结构性这么一说

## 9.2 安装

## 9.3 启动和关闭数据库

启动：

Mongod默认执行 mongod 命令所处盘符根目录下的 /data/db 作为自己的数据存储目录。所以在启动之前，手动在磁盘根目录下新建一个 /data/db

-控制台：mongod

如果想修改默认的数据存储目录，可以：

-mongod –dbpath=数据存储目录路径

停止：

Ctrl+c 或关闭控制台

## 9.4 连接和退出数据库

连接：

连接之前，确保已开启

默认连接本机服务

-直接mongo

退出：

在连接状态输入ecxit退出连接

-直接exit

## 9.5 基本命令

show dbs : 查看所有数据库

db：查看当前操作的数据库

use 数据库名称：切换到指定的数据库（如果没有会新建）

插入数据：



指定连接的数据库不一定存在，当你插入第一条数据之后会被自动创建出来

## 9.6 在node中如何操作MongoDB数据库

### 9.6.1 使用官方的MongoDB包来操作

<https://github.com/mongodb/node-mongodb-native>

### 9.6.2 使用第三方包Mongoose来操作MongoDB数据库

第三方包：mongoose 基于mongodb官方的mongdb 包再次做了封装，在实际开发中真正使用的，使操作mongodb数据库更方便。WordPress项目团队开发

官网：<https://mongoosejs.com/>

官方指南：<https://mongoosejs.com/docs/guides.html>

官方API文档：

-设计Schema

-发布Model（得到模型构造函数）

-数据操作

### 9.6.3 MongoDB数据库的基本概念

数据库（多个）

集合（多个）（表）

文档（多个）（表记录）

-文档结果灵活，不限制

-无需像MySQL一样，先创建数据库、表、设计表结构

-一切由mongodb自动完成建库建表

{

    // 数据库

    qq: {

        // 集合(表): 数组

        users: [

            // 文档: 对象

            {name: 'name', age: 1},

            {},

        ],

        product: [

        ]

    },

    taobao: {

    },

    db: {

    }

}

## 9.7 学习指南（步骤）

### 9.7.1 设计scheme发布Model（创建表）

var mongoose = require('mongoose')

var schema = mongoose.Schema

// 1, 连接数据库

mongoose.connect('mongodb://localhost/itcast')

// 2, 设计集合结构(表结构)

// 字段名称就是表结构中的属性名称

// 值

// 约束: 保证数据的完整性,不要由脏数据

var userSchema = new Schema({

    title: {

        type: String,

        required: true  // 必须有

    },

    password: {

        type: String,

        required: true  // 必须有

    },

    email: String,

    // comments: [{body: String, data: Date}],

    // date: { type: Date, default: Date.now}

})

// 3, 将文档结构发布为模型

//  mongoose.model

//      第一个参数

//      第二个参数: 架构Schema

//      返回值: 模型构造函数

var User = mongoose.model('User', userSchema)

// 4, 使用模型构造函数,对users集合中的数据操作

### 9.7.2 添加数据（增）

var admin = new User({

    username: 'user1',

    password: '111111',

    email: 'admin@ad.com'

})

// 持久化

admin.save((err, ret) => {

    if(err){

        console.log('save error');

    } else {

        console.log('save success');

        console.log(ret);

    }

})

### 9.7.3 删除（删）

// deleteOne deleteMany

User.deleteMany({

    username: 'user1'

    }, (err, ret) => {

        if(err){

            console.log('delete error');

        } else {

            console.log('delete success');

            console.log(ret);

        }

    })

### 9.7.4 更新（改）

// update

// User.update({

//         username: 'user2'

//     }, { password: '121212' }, (err, ret) => {

//         if(err){

//             console.log('update error');

//         } else {

//             console.log('update success');

//             console.log(ret);

//         }

// })

### 9.7.5 查询（查）

// search

// User.find((err, ret) => {

//     if(err){

//         console.log('save error');

//     } else {

//         console.log('save success');

//         console.log(ret);

//     }

// })

User.findOne({

    username: 'user1'

}, (err, ret) => {

    if(err){

        console.log('save error');

    } else {

        console.log('save success');

        console.log(ret);

    }

})

# 十、使用node操作MySql数据库

# 十一、异步编程

异步代码无法保证：执行的顺序就按照代码的顺序执行

为了解决这个问题：可以使用嵌套

嵌套的太多：产生回调地狱问题。Callback hell 回调地狱

## 11.1 回调函数

-异步编程

-如果需要得到一个函数内部异步操作的结果，需要使用回调函数

-在调用的地方，传递一个函数进来

-在封装的函数内部调用传递进来的函数

-如何熟练达到像定义一个变量一样，来封装一个带有回调函数的方法；

-Js编程的一大特色：异步编程

function add(x, y, callback) {

    setTimeout(() => {

        var res = x + y

        callback(res)

    }, 1000)

}

add(1,4, (res) => {

    // res 才是我们得到的结果

})

基于原生XMLHTTPRequest封装get方法：

function get (url, callback) {

            var oq = new XMLHttpRequest()

            // 请求加载成功后,调用指定的函数

            oq.onload = function() {

                console.log(oq.responseText);

                callback(oq.responseText)

            }

            oq.open('get', url, true)

            oq.send()

        }

        get('data.json', (data) => {

            console.log(data);

        })

## 11.2 Promise

封装promise版本的readFile

var fs = require('fs')

// var p1 = new Promise(function(resolved, rejected) {

//     fs.readFile('./a.txt', 'utf8' , (err, data) => {

//         if(err) {

//             // console.log('err');

//             rejected(err)

//         }

//         // console.log(data);

//         resolved(data)

//     })

// })

// var p2 = new Promise((resolved, rejected) => {

//     fs.readFile('./b.txt', 'utf8' , (err, data) => {

//         if(err) {

//             console.log('err');

//             rejected(err)

//         }

//         // console.log(data);

//         resolved(data)

//     })

// })

// var p3 = new Promise((resolved, rejected) => {

//     fs.readFile('./c.txt', 'utf8' , (err, data) => {

//         if(err) {

//             console.log('err');

//             rejected(err)

//         }

//         // console.log(data);

//         resolved(data)

//     })

// })

// 封装

function addPromise(url) {

    return new Promise(function(resolved, rejected) {

        fs.readFile(url, 'utf8' , (err, data) => {

            if(err) {

                // console.log('err');

                rejected(err)

            }

            // console.log(data);

            resolved(data)

        })

    })

}

// then的链式调用

addPromise('./a.txt')

    .then(function(data) {

        console.log(data);

        // 这里return的作为下一个then的resolved和rejected函数

        return addPromise('./b.txt')

    }, function(error) {

        console.log(error);

    })

    .then((data) => {

        console.log(data);

        return addPromise('./c.txt')

    })

    .then((data) => {

        console.log(data);

    },)

Promise的数据库操作：

// Mongoose所有的API 都支持promise

User.find()

    .then((data) => {

        console.log(data);

    })

    // 注册模拟

// User.findOne({username: 'ww'})

//     .then((user) => {

//         // console.log(user);

//         if(user) {

//             console.log('has');

//         } else {

//             console.log('no');

//             return new User({

//                 // username: user,

//                 username: 'ww',

//                 password: '111',

//                 email: 'ww'

//             }).save()

//         }

//     })

//     .then((data) => {

//         console.log(data);

//     })

## 11.3 Generator

# 十二、其他

## 12.1 修改完代码自动重启

## 12.2 封装异步API

## 12.3 数组的遍历方法，都是对函数作为一种参数

## 12.4 es6

# 十三、项目案例

## art-template的模板继承，模板填充

<https://aui.github.io/art-template/zh-cn/docs/syntax.html#%E5%AD%90%E6%A8%A1%E6%9D%BF>

-只需要写一个母模板，在母模板中，写公共的部分，剩下的不同的内容部分使用block 加名字，留坑；需要引入其他的模板，使用include加地址，引入

<!DOCTYPE html>

<html lang="en">

<head>

    <meta charset="UTF-8">

    <meta name="viewport" content="width=device-width, initial-scale=1.0">

    <title>Document</title>

    <link rel="stylesheet" href="../node\_modules/bootstrap/dist/css/bootstrap.min.css">

    {{ block 'head'}} {{ /block }}

</head>

<body>

    {{ include './header.html' }}

    <!-- 让子去补 -->

    {{ block 'content' }}

        <h1>layout content</h1>

    {{ /block }}

    {{ include './footer.html' }}

</body>

<script src="/node\_modules/jquery/dist/jquery.js"></script>

<script src="/node\_modules/bootstrap/dist/js/bootstrap.js"></script>

{{ block 'script' }} {{ /block}}

</html>

子模版，使用extend继承母模板

{{ extend './layout.html' }}

{{ block 'content' }}

<div>

    <h2>index content</h2>

</div>

{{ /block }}

## -补充：

**表单同步提交和异步提交**：早期的表单是同步的，即没有ajax的时候，form表单的提交是同步的，它会将服务器返回的内容显示到页面上，替换当前的页面； ——》 后来有人用render进行页面的重新渲染，此时页面会刷新，将表单中，用户所填的数据，进行template模块引擎重新渲染到页面上，【现在也有人这么用，例如GitHub的登录页面】——》 后来才提出ajax异步操作。

http就是web中的沟通语言。字符串交互：请求（报文，一定格式的字符串）、响应（字符串）。

**服务器重定向问题**：服务器可以进行重定向，但是当浏览器中的相对应的操作是异步的，在服务器中重定向无效，只能在浏览器端重定向； ——》 只有当浏览器是同步操作，服务器的重定向才有效。

## 13.1 目录结构

App.js 项目的入口文件

Controllers

Models 存储使用mongoose设计的数据模型

Node\_modules 第三方包

Package.json 包描述文件

Package-lock.json 第三方包版本锁定文件（npm5之后才有的）

Public 公共静态资源

Routes 统一存储路由文件

Views 存储视图页面目录

## 13.2 模板页

子模版

模板继承

## 13.3 路由设计

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| 路径 | 方法 | get参数 | post参数 | 备注 | 是否需要权限 |
| / | Get |  |  | 首页 |  |
| /register | Get |  |  | 注册 |  |
| /register | Post |  | Email, nickname, password |  | 注册请求 |
| /login | Get |  |  | 登录 |  |
| /login | Post |  | Email，password | 登录请求 |  |
| /logout | Get |  |  | 退出请求 |  |

使用md5对密码加密：<https://github.com/pvorb/node-md5#readme>

## 13.4 模型设计

## 13.5 功能实现

## 13.6 步骤

创建目录结构

整和静态页-模板页

Include、block、extend

设计用户登录、退出、注册的路由

用户注册

先处理好客户端页面的内容（表单控件的name、发送请求、收集表单数据）

服务端

-获取客户端表单请求的数据

-操作数据库：有错，发送500，告诉客户端服务器错了；

其他的根据业务发送不同的响应数据

用户登录

用户退出

# 十四、Express中间件

## 14.1 中间件的概念

处理请求的函数

-当请求进来，会从第一个中间件开始匹配。

-如果匹配，则会进来

-如果请求进入中间件之后，没有调用next，则不会继续匹配下一个中间件

-如果调用了next，则会继续向后找到第一个满足匹配的中间件

-如果不匹配，则继续匹配下一个中间件

如果没有匹配的中间件，则express会默认输出cannot GET/

## 14.2 中间件的分类

不关心请求路径和请求方法的中间件，即任何请求都会进入这个中间件：

app.use((req, res) => {

    console.log('任何请求都会进来');

})

中间件本身是一个方法，接收三个参数：request、response、next（下一个中间件）。当一个请求进入也给中间件之后，如果不调用next，则不会调用下一个中间件，所以next是一个调用下一个中间件的方法：

app.use((req, res, next) => {

    console.log('任何请求都会进来1');

    // 通过next进入下一个中间件

    next()

})

app.use((req, res, next) => {

    console.log('任何请求都会进来2');

})

关系请求路径的中间件：以/xxx 开头的路径中间件：

// 以 /a 开头的

app.use('/a', (req, res, next) => {

    console.log('任何请求都会进来1');

    // 通过next进入下一个中间件，next也是要匹配的，不一定调用紧挨着的那一个

    next()

})

除了以上中间件之外，还有一种最常用的，严格匹配请求方法和请求路径的中间件：

app.get app.post

### 14.2.1 应用程序级别的中间件

### 14.2.2 路由级别的中间件

## 14.3 总

### 14.3.1 错误处理中间件

### 14.3.2 内置中间件

### 14.3.3 第三方中间件