### 什么是NodeJs

NodeJs是基于V8引擎的JavaScript运行环境

### NodeJs特点

\* 单线程

\* 非阻塞式I/O

\* 事件驱动

### NodeJs 简介

\* 2009年5月 Ryan Dahl发布了最初版本的部分Node.js包

\* 目前的LTS版本是8.11.3

### NodeJs 安装

https://nodejs.org/en/

### 模块

\* 在NodeJs中,一个文件就是一个模块,这个模块的本质其实就是一个函数,每个模块都有自己的属性

### 模块化

\* CommonJS exports 和 require

\* ES6 export 和 import

\* AMD define 和 require 语句

### NodeJs模块分类

\* 核心模块

> \* 系统核心模块在Node源代码编译过程中被编译成了二进制文件,在node启动时被直接加载到了内存中

\* 自定义模块

> \* 在目录node\_modules下的文件或者包,例如用npm安装的模块

> \* 自定义模块的加载路径可以通过module.paths查看

\* 文件模块

> \* 文件模块是用户自己编写的模块,以相对或者绝对路径形式引入

\* 注意点

> \* Node对所有加载过的模块对象都会缓存

> \* Node对二次加载的模块一律采用缓存优先

> \* 核心模块的优先级高于自定义模块

### NodeJs文件模块

\* 模块定义

> \* module.exports, 在module.exports对象上添加属性导出值

> \* exports,在exports对象上添加属性导出值

> \* exports对象和module.exports对象是同一个对象

> \* exports对象添加属性来导出值

> \* module.exports对象可以赋值一个对象来导出值

> \* 导出的始终是module.exports指向的对象

\* 模块引用

> \* require('文件名'),执行对应的文件并且返回该文件对应的modeule.exports对象

\* 模块标识

> \* 模块标识其实就是传递给require()方法的参数

> \* 模块标识可以是绝对路径也可以是相对路径

> \* 用绝对路径的话系统会直接加载路径指定的文件

> \* 用相对路径时注意,当前目录一定要写“./”,否则表明要加载系统的核心模块

> \* 模块加载规则

>> \* 首先按照模块的文件名进行查找

>> \* 如果没有找到,则会在模块名称后面加上.js后缀进行查找

>> \* 如果还没有找到,则会在模块名称后面加上.json后缀进行查找

>> \* 如果还没有找到,则会在模块名称后面加上.node 后缀进行查找

>> \* 如果还没有找到,则会报错

### NPM(Node Package Manager)

\* NPM是一个工具,安装node时会自动安装

\* NPM主要用来完成第三方模块的发布,安装和依赖

### NPM 常用命令

\* 查看版本 npm -v

\* 搜索包 npm search 包名关键字

\* 初始化 npm init [--yes]

> \* 参数--yes表明不出现提问设置,所有参数采用默认值

\* 安装包 npm install [-g] 包名 [--save] [--save-dev]

> \* 如果带参数-g,表明全局安装,这样安装的包在/usr/local/lib/node\_modules文件夹中，可以在命令行中使用,但不可以在项目中require,使用-g参数时没有--save

> \* 如果不带参数-g表明本地安装,包文件安装在当前文件夹的node\_modules文件夹中,只能在当前项目中使用require引入模块

> \* 参数--save:把包的版本信息添加到package.json中dependencies 中,一般是需要最终需要打包到业务代码中的依赖,如果执行npm install就会安装 package.json中dependencies的依赖

> \* 参数--save-dev: 依赖会加在package.json的devDependencies中,一般是辅助开发的依赖,不会打包上线的

\* 卸载包 npm uninstall [-g] 包名 [--save] [--save-dev]

> 安装用什么参数卸载就用什么参数

yarn

### 什么是全局变量

在所有的模块中都可以使用,不需要引入

### 作用域在模块内的变量

\* \_\_dirname 当前模块的文件夹名次

\* \_\_filename 当前模块文件的绝对路径

\* module 当前的模块信息

\* exports module.exports对象,用来导出模块

\* require() 引入模块

### global全局的命名空间对象

### process进程对象

\* process 和 global.process是同一个对象

\* process.argv

\* process.env

\* process.pid

\* process.nextTick()

### 定时器

\* setInterval()/clearInterval()

\* setTimeout()/clearTimeout()

\* setImmediate()/clearImmediate()

### 控制台对象

### Buffer类

\* Buffer是用来存放二进制数据的容器

\* Buffer.from

\* Buffer.alloc

### event

\* 概念

> \* Node中的event没有默认行为和事件冒泡

> \* const EventEmitter = require('events');//返回的是一个类

> \* 通常我们需要继承EventEmitter类来实现事件

> \* 一个EventEmitter对象默认最大可以有10个监听,可以通过emitter.setMaxListeners(n)来设置最大监听数

> \* 大多数 Node.js 核心对象都继承事件类

\* 添加监听

> \* emitter.addListener(eventName, listener)

> \* emitter.on(eventName, listener)

> \* emitter.once(eventName, listener)

> \* 注意点:

>> \* 和浏览器端的事件不同,监听函数的第一个参数就是触发时传入的参数而不是event对象

>> \* emitter.addListener和emitter.on(eventName, listener)是同一个方法

\* 触发事件

> \* emitter.emit(eventName[, ...args])

> \* 注意点:

>> \* 和浏览器端的事件不同,传入参数不用数组而是参数列表

\* 移除事件

> \* emitter.removeListener(eventName, listener)

> \* emitter.off(eventName, listener)

> \* 注意点:

>> \* emitter.removeListener和emitter.off是同一个方法

>> \* emitter.off 新增于: v10.0.0

\* 事件

> 'newListener'事件,当有新的监听被添加是触发,回调函数接受两个参数分别是添加的事件名称和函数的

### stream

\* 概念

> \* 流stream是一种中处理数据的抽象接口,是将数据分割成一段一段的来处理,提高效率

> \* 流可以是可读的、可写的、或是可读写的,所有的流都是 EventEmitter 的实例

\* readable.pipe(writable) 将可读流的数据传递给可写流

\* writable 流指的是只能流进不能流出的流

\* readable 流是产出数据的流

\* process.stdin 代表控制台的可读流

\* process.stdout 代表控制台的可写流

### fs(File System)

\* 同步读写文件

\* 异步读写文件

\* 读写文件流

### http

\* http.createServer((req,res)=>{}) 创建一个服务器实例

\* req 它实现了 可读流 接口

\* res 它实现了 可写流 接口

### url

### path

### querystring

### crypto

MongoDB

### 数据库介绍

\* 什么是数据库

> 数据库(DataBase,简称DB),是一种专门用来管理数据的软件

\* 为什么要数据库

> 因为应用程序中的数据是保存在内存中的,随着程序的执行完毕或者退出,数据就会销毁,而对于大多数情况下我们是需要把用户的数据保存下来的,一个最简单的方法是把用户数据写入文件,但是随着程序的功能越来越复杂,数据量越来越大,用文件管理数据就显得力不从心了,所以需要一个专门来管理数据的软件来帮忙,他就是数据库

\* 数据的分类和选择

> \* SQL数据库

>> \* Oracle ,SQL Server,DB2,Sybase

>> \* MySQL,PostgreSQL,sqlite

> \* NoSQL(not only sql)数据库

>> \* MongoDB

>> \* CouchDB

>> \* HBase

>> \* Redis

> \* SQL数据库结构化,适合中大型系统

> \* NoSQL数据库数据模型比较简单,但操作比较灵活,适合微架构

> \* SQL数据库和NoSQL数据库不是二选一的关系,而是相互辅助和补充的关系

### mongodb 的安装

https://www.mongodb.com/

https://docs.mongodb.com/manual/reference/

http://www.mongodb.org.cn/tutorial/

\* 注意环境变量的配置

### mongodb 存储基本组成

\* database 数据仓库,数据仓库中存放集合,一个mongodb中可以有多个数据仓库

\* collection 集合,集合中存放文档,一个集合可以有多个文档

\* document 文档,文档是有字段(field)和值组成,数据结构和JSON基本一样,是数据库中的最小单位,是存储和操作的内容

### mongodb 常用终端命令

\* 启动Mongodb: mongod

> 注意点:

>> \* 启动前需要创建默认数据库存储目录

>> \* 如果不用默认的存储目录,可以用 mongod --dbpath 自定义路径 来指定数据库存储目录

\* 连接MongoDB: mongo

### 数据库操作

\* 创建数据库:use DATABASE\_NAME,如果数据库不存在，则创建数据库，否则切换到指定数据库。

\* 查看数据库:show dbs

> 注意:

>> \* 只有数据库中插入了数据后用show dbs才能查看到

>> \* 默认的数据库为test,如果你没有创建新的数据库,集合将存放在test数据库中

\* 查看当前数据库名:db

\* 删除数据库:db.dropDatabase()

> 注意:

>> \* 删除前要切换到需要删除的数据库

### 集合操作

\* 显示当前数据中的集合: show collections

\* 删除当前数据库中指定的集合:db.COLLECTION\_NAME.drop()

### 文档操作

\* 插入文档:db.COLLECTION\_NAME.insert(document)

> 注意:

>> \* 新插入的数据系统会自动添加一个\_id的主键,也可以自己指定

>> \* db.COLLECTION\_NAME.save(document),不指定 \_id 和insert一样,指定 \_id 则是更新该 \_id 的数据

\* 查看文档:db.COLLECTION\_NAME.find(query,projection)

> 注意:

>> \* query是查询的条件,是一个对象,不传的话是查询所有

>> \* query查询可以用查询操作符,常见的查询操作符有

>>> \* 比较查询操作符

>>>> \* $lt 小于

>>>> \* $lte 小于等于

>>>> \* $gt 大于

>>>> \* $gte 大于等于

>>>> \* $in 包含

>>> \* 逻辑查询操作符

>>>> \* $and 与

>>>> \* $or 或

>>>> \* $not 非

>> \* 该方法返回一个返回指向符合条件的所有文档的游标对象

>> \* 游标对象的常用方法有

>>> \* skip(num)

>>> \* limit(num)

>>> \* toArray()

>>> \* length()

>>> \* count()

>>> \* sort({ field: value }),1升序,-1降序

>> \* projection是指定那些字段显示或者隐藏,格式为{ field1: <value>, field2: <value> ... },1显示,0隐藏

\* 更新文档:db.COLLECTION\_NAME.update(query,update)

> 注意:

>> \* query是更新的条件,是一个对象

>> \* update是有更新操作符和更新后的值组成的对象

>>> \* 常见更新操作符有

>>>> \* $set,设置某一个字段的值,如果匹配到的文档有该字段则修改没有则新增

>>>> \* $unset,删除字读

>> \* update默认更新复合条件的第一条文档,如果需要更新所有

>>> \* 可以给update方法增加一个参数{multi:true}

>>> \* 也可以用updateMany方法

\* 删除文档:db.COLLECTION\_NAME.remove(query,justOne)

> 注意:

>> \* query是删除的条件,是一个对象,如果是一个空对象则删除所有

>> \* justOne如果设置为true表示只删除第一条

### 集合间关系

\* 一对一 1:1, 例如老公和老婆

> 可以通过文档关联或者内嵌文档的形式来实现

\* 一对多 1:n, 例如老爸和孩子

> 可以通过文档关联或者内嵌文档的形式来实现

\* 多对多 n:n, 例如老师和学生

> 可以通过文档关联的形式来实现

### 索引

\* 为了提高查询效率,通常会为经常需要查询的字段创建索引,索引是对文档中的字段排序的一种数据结构,可以理解为书的目录

\* 创建索引

db.COLLECTION\_NAME.createIndex( {field1: 1|-1, field2: 1|-1,….}<,{name:”indexName”, unique:true|fase}> )

> \* 1是升序,-1是降序

> \* 第二个参数是可选的，可以指定索引的名字,设置索引是否唯一

> \* 如果设置了unique为true,则该字段不能重复插入

> \* 如果新建索引时字段中有重复值，新建会失败

\* 查看索引 db.COLLECTION\_NAME.getIndexes()

\* 删除索引

> \* db.COLLECTION\_NAME.dropIndex( {field1: 1|-1} 删除指定的索引

> \* db.COLLECTION\_NAME.dropIndexes() 删除所有索引(除了\_id)

### NodeJs Driver

http://mongodb.github.io/node-mongodb-native/3.1/quick-start/quick-start/

\* mongodb NodeJs Driver 安装

\* CRUD 体验

### mongoose 基本使用

\* mongoose 安装

\* CRUD 体验

> 连接DB

> 定义Schema

> 用Schema生成Model

\* CRUD 进阶

> \* 新增

>> \* Model.insertMany()

>> \* Model.prototype.save()

>> \* Model.create()

> \* 查找

>> \* Model.find()

>> \* Model.findById()

>> \* Model.findOne()

> \* 更新

>> \* Model.update()

>> \* Model.updateMany()

>> \* Model.updateOne()

> \* 删除

>> \* Model.remove()

>> \* Model.deleteOne()

>> \* Model.deleteMany()

> \* 获取去重后的值

>> \* Model.distinct()

\* Schema 定义文档的模型

> \* 常见定义类型

>> \* String

>> \* Number

>> \* Date

>> \* Boolean

>> \* ObjectId(mongoose.Schema.Types.ObjectId)

>> \* Array

> \* 注意点

>> \* 在存储和更新数据时,如果数据的类型和定义字段的类型不一致,mongoose内部会尝试将数据转换为定义的字段类型,如果转换失败则操作失败

>> \* 定义类型的方法有两种,一种是直接用类型,另一种是用一个对象,类型是对象type属性的值

>> \* MongoDB存储的是格林尼治标准时间(GMT时间),和们的时区错8个小时,存储时会减去8小时

>> \* 通常我从数据库中取出的时间需要做格式化,可以借助 moment 包来格式化时间

>> \* MongoDB会自动添加ObjectId的id,如果字段类型是ObjectId,插入时会把字符串转化为ObjectId

> \* 验证

>> \* 内置验证

>>> \* 所有的类型都有required(必须)验证

>>> \* Number类型有min(最小)和max(最大)值验证

>>> \* String有enum(枚举),maxlength(最大长度)和minlength(最小长度)验证.

>> \* 每一个验证都可以写为: 规则:[值,错误消息]的格式,也可以写为:规则:值

>> \* 自定义验证举例

```javascript

validate:{

validator:function(v){

return /1[358]\d{9}/.test(v)

},

message:'{VALUE} 不是合法电话号码'

}

```

> \* 方法

>> \* 自定义实例方法举例

```javascript

//注意不要用箭头函数

UserSchema.methods.findMyBlogs = function(callback){

// this是 UserModel的一个实例

// 在Model的原型上有Model.prototype.model()方法,该方法返回一个指定的Model

this.model('Blog').find({author:this.\_id},(err,docs)=>{

callback(null,docs)

})

}

```

>> \* 自定义模型静态方法举例

```javascript

//注意不要用箭头函数

UserSchema.statics.findByPhone = function(phone,callback){

//this 是 UserModel

//Model.model()方法返回一个指定的Model,因此this和this.model('User'))相等

this.find({phone:phone},(err,docs)=>{

callback(null,docs)

})

}

```

\* 关联查询 populate