



愿景："让编程不在难学，让技术与生活更加有趣"

更多课程请访问[xdclass.net](http://xdclass.net)

## 第一章 初识NodeJs

---

### 第1集 node.js课程介绍及案例演示

简介：介绍nodejs课程大纲及实战项目演示

vue+elementui+express

### 第2集 nodejs环境安装配置

简介：讲解windows和mac系统下nodejs环境安装配置

更换淘宝镜像：

```
export NVM_NODEJS_ORG_MIRROR=https://npm.taobao.org/mirrors/node
```

nvm 的bash\_profile文件里写入以下配置：

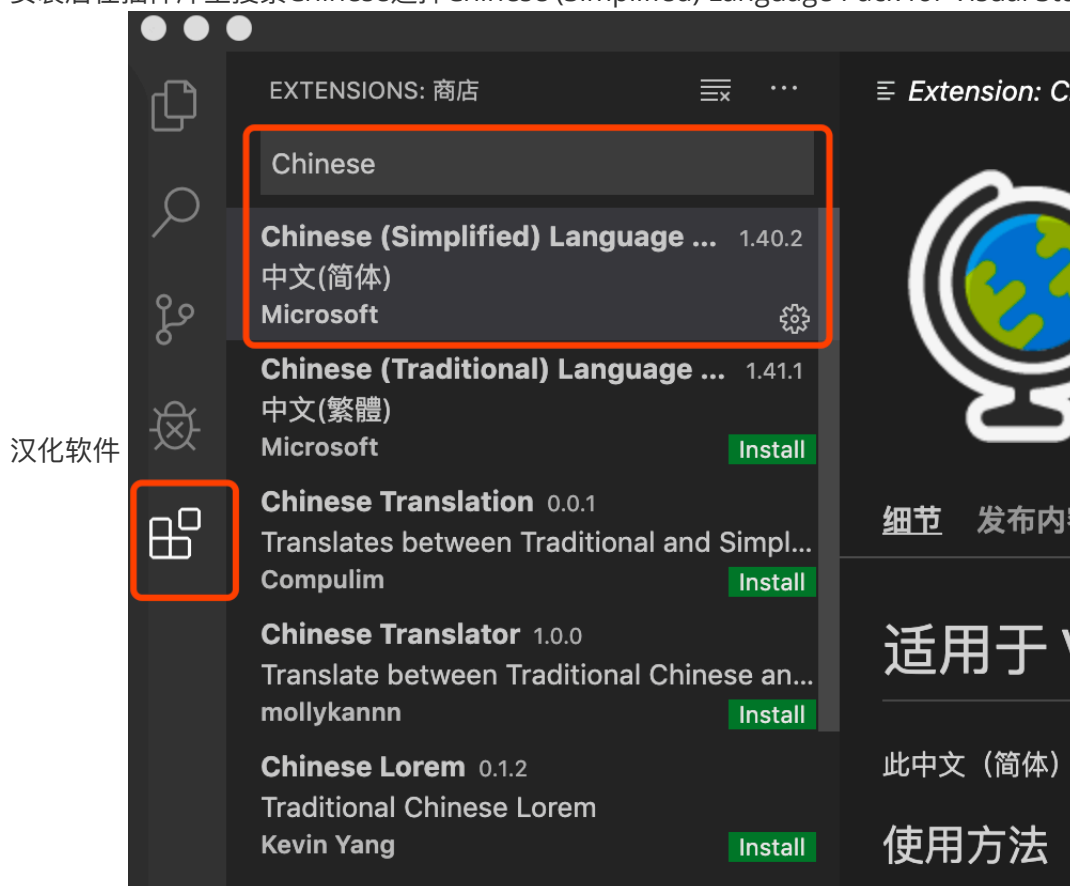
```
export NVM_DIR="$([ -z "${XDG_CONFIG_HOME-}" ] && printf %s "${HOME}/.nvm" ||  
printf %s "${XDG_CONFIG_HOME}/nvm")"  
[ -s "$NVM_DIR/nvm.sh" ] && \. "$NVM_DIR/nvm.sh" # This loads nvm
```

### 第3集 vscode编辑器和插件安装

简介：vscode安装步骤及常用插件安装

- vscode官方下载地址：<https://code.visualstudio.com/>

- 安装后在插件库里搜索Chinese选择Chinese (Simplified) Language Pack for Visual Studio Code



## 第4集 初建NodeJs应用及调试

简介：创建一个nodejs应用以及如何去调试nodejs应用

- nodejs和JavaScript有什么区别？  
nodejs是一个JavaScript运行环境（平台），JavaScript是编程语言。
- nodejs中无法运行alert方法，DOM和BOM这类方法也无法在node中运行。

## 第5集 深入理解commonjs模块规范

简介：讲解使用require和module引入、导出模块

- commonjs规范

每一个文件相当于一个模块，有自己的作用域，其模块里的变量、函数以及类都是私有的，对外不可见的。

- module.exports模块导出

```
function add(a,b){
  console.log(a+b)
}

function decrease(a,b){
  console.log(a-b)
}

module.exports = {
  add,
  decrease
}
```

- require模块引用

```
let cal = require('./calculate')

cal.add(10,5)
cal.decrease(100,50)
```

- loadsh

它是一个一致性、模块化、高性能的 JavaScript 实用工具库

初始化项目

```
npm init -y
```

安装命令

```
npm i loadsh --save/cnpm i loadsh --save
```

下载速度慢可替换成淘宝镜像

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

- nodejs中的全局对象是global，定义全局变量用global对象来定义

```
global.a = 2 //定义全局变量，可在其他模块中直接使用
```



愿景："让编程不在难学，让技术与生活更加有趣"

更多课程请访问[xdclass.net](http://xdclass.net)

## 第二章 NodeJs核心模块api-基础

### 第1集 Buffer缓冲器常用api （一）

简介：介绍buffer缓冲器类常用的api使用

- buffer用于处理二进制数据，在v8堆外分配物理内存，buffer实例类似0-255之间的整数数组，显示的数据为十六进制，大小是固定的，无法修改。
- 创建buffer

- **Buffer.alloc(size[, fill[, encoding]]):**

- `size` 新 Buffer 的所需长度。
- `fill` ||| 用于预填充新 Buffer 的值。默认值: 0。
- `encoding` 如果 `fill` 是一个字符串，则这是它的字符编码。默认值: 'utf8'。

```
console.log(Buffer.alloc(10)); //创建一个长度为10的Buffer，默认填充0
console.log(Buffer.alloc(10,2)); //创建一个长度为10的Buffer，填充2
console.log(Buffer.alloc(10,100)); //创建一个长度为10的Buffer，填充1000
console.log(Buffer.alloc(10,-1)); //创建一个长度为10的Buffer，-1强制转换成255
```

- **Buffer.allocUnsafe(size):**

- `size` 新 Buffer 的所需长度。

```
console.log(Buffer.allocUnsafe(10)); //创建一个长度为10未初始化的buffer
```

- **Buffer.from(array):**

- `array` 整数数组

```
console.log(Buffer.from([1,2,3])); //创建buffer，填充[1,2,3]
```

- **Buffer.from(string[, encoding]):**

- `string` 要编码的字符串。
- `encoding` `string` 的字符编码。默认值: 'utf8'。

```
console.log(Buffer.from('eric')); //创建buffer, 填充字符串

console.log(Buffer.from('eric','base64')); //创建buffer, 填充base64编码的字符串
```

- Buffer类上常用的属性、方法
  - **Buffer.byteLength**: 返回字符串的字节长度

```
console.log(Buffer.byteLength('eric')); //返回字符串的字节长度, 4

console.log(Buffer.byteLength('中文')); //6, 一个文字代表3个字节
```

- **Buffer.isBuffer(obj)**: 判断是否是buffer

```
console.log(Buffer.isBuffer({}));

console.log(Buffer.isBuffer(Buffer.from('eric')));
```

- **Buffer.concat(list[, totalLength])**: 合并buffer

```
const buf = Buffer.from('hello');
const buf2 = Buffer.from('eric');

console.log(buf);
console.log(buf2);
console.log(Buffer.concat([buf,buf2]));
console.log(Buffer.concat([buf,buf2],10));
```

官网文档地址: <http://nodejs.cn/api/buffer.html>

## 第2集 Buffer缓冲器常用api (二)

简介: 介绍buffer缓冲器实例常用的api使用

- buffer实例常用的属性、方法
  - **buf.write(string[, offset[, length]][, encoding])** 将字符写入buffer,返回已经写入的字节数
    - `string` 要写入 `buf` 的字符串。

- `offset` 从指定索引下写入。默认值: `0`。
- `length` 要写入的字节数。默认值: `buf.length - offset`。
- `encoding` 字符串的字符编码。默认值: `'utf8'`。

```
const buf = Buffer.allocUnsafe(20);
console.log(buf);

// console.log(buf.write('buffer'));
console.log(buf.write('buffer', 5, 3));
console.log(buf);
```

◦ **buf.fill(value[, offset[, end]][, encoding])** 填充buffer

- `value` | | | 用来填充 `buf` 的值。
- `offset` 开始填充 `buf` 的索引。默认值: `0`。
- `end` 结束填充 `buf` 的索引（不包含）。默认值: `buf.length`。
- `encoding` 如果 `value` 是字符串，则指定 `value` 的字符编码。默认值: `'utf8'`。

```
console.log(buf.fill('eric', 5, 10));
```

◦ **buf.length** buffer的长度

◦ **buf.toString([encoding[, start[, end]]])** 将buffer解码成字符串形式

- `encoding` 使用的字符编码。默认值: `'utf8'`。
- `start` 开始解码的字节索引。默认值: `0`。
- `end` 结束解码的字节索引（不包含）。默认值: `buf.length`。

```
const buf = Buffer.from('test');
console.log(buf.toString('utf8', 1, 3));
```

◦ **buf.toJSON** 返回 buffer 的 JSON 格式

```
const buf = Buffer.from('test');
console.log(buf.toJSON());
```

◦ **buf.equals (otherBuffer)** 对比其它buffer是否具有完全相同的字节

- `otherBuffer` 要对比的buffer

```
const buf1 = Buffer.from('ABC');
const buf2 = Buffer.from('414243', 'hex');
const buf3 = Buffer.from('ABCD');

console.log(buf1);
console.log(buf2);

console.log(buf1.equals(buf2));
// 打印: true
console.log(buf1.equals(buf3));
```

- **buf.indexOf/lastIndexOf** 查找指定的值对应的索引
- **buf.slice([start[, end]])** 切割buffer
  - `start` 新 `Buffer` 开始的位置。默认值: 0。
  - `end` 新 `Buffer` 结束的位置（不包含）。默认值: `buf.length`。

```
const buf = Buffer.from('abcdefghi');
console.log(buf.slice(2,7).toString()); //cdefg
```

- **buf.copy(target[, targetStart[, sourceStart[, sourceEnd]]])** 拷贝buffer
  - `target` | 要拷贝进的 `Buffer` 或 `Uint8Array`。
  - `targetStart` 目标 `buffer` 中开始写入之前要跳过的字节数。默认值: 0。
  - `sourceStart` 来源 `buffer` 中开始拷贝的索引。默认值: 0。
  - `sourceEnd` 来源 `buffer` 中结束拷贝的索引（不包含）。默认值: `buf.length`。

```
const buf = Buffer.from('abcdefghi');
const buf2 = Buffer.from('test');
// console.log(buf.slice(2,7).toString());

// console.log(buf.copy(buf2));
// console.log(buf2.toString());
console.log(buf2.copy(buf, 2, 1, 3));
console.log(buf.toString());
```

## 第3集 node.js文件系统模块常用api操作（一）

简介：讲解一些文件的常用api操作

- 引入文件系统模块fs

```
const fs = require('fs')
```

- **fs.readFile(path[, options], callback)** 读取文件

- `path` | 文件路径
- `callback` 回调函数
  - `err`
  - `data` | 读取的数据

```
fs.readFile('./hello.txt', 'utf8', (err, data) => {  
  if(err) throw err;  
  console.log(data);  
})
```

- **fs.writeFile(file, data[, options], callback)** 写入文件

- `file` | 文件名或文件描述符。
- `data` | 写入的数据
- `options` |
  - `encoding` | 写入字符串的编码格式 默认值: 'utf8'。
  - `mode` 文件模式(权限) 默认值: 0o666。
  - `flag` 参阅[支持的文件系统标志](#)。默认值: 'w'。
- `callback` 回调函数
  - `err`

```
fs.writeFile('./hello.txt', 'this is a test', err => {  
  if(err) throw err;  
  console.log('写入成功');  
})
```

- **fs.appendFile(path, data[, options], callback)** 追加数据到文件

- `path` | 文件名或文件描述符。
- `data` | 追加的数据
- `options` |
  - `encoding` | 写入字符串的编码格式 默认值: 'utf8'。
  - `mode` 文件模式(权限) 默认值: 0o666。
  - `flag` 参阅[支持的文件系统标志](#)。默认值: 'a'。
- `callback` 回调函数
  - `err`



```
const buf = Buffer.from('hello world!')
fs.appendFile('./hello.txt', buf, (err) => {
  if(err) throw err;
  console.log('追加成功');
})
```

- **fs.stat(path[, options], callback)** 获取文件信息，判断文件状态（是文件还是文件夹）

- path | |
- options
  - bigint 返回的 `fs.Stats` 对象中的数值是否应为 `bigint` 型。默认值: `false`。
- callback
  - err
  - stats <fs.Stats> 文件信息

```
fs.stat('./hello.txtt', (err, stats) => {
  if(err){
    console.log('文件不存在');
    return;
  }
  console.log(stats);
  console.log(stats.isFile());
  console.log(stats.isDirectory());
})
```

- **fs.rename(oldPath, newPath, callback)** 重命名文件

- oldPath | | 旧文件路径名字
- newPath | | 新文件路径名字
- callback 回调函数
  - err

```
fs.rename('./hello.txt', './test.txt', err => {
  if(err) throw err;
  console.log('重命名成功');
})
```

- **fs.unlink(path, callback)** 删除文件

- path | |
- callback
  - err

```
fs.unlink('./test.txt',err => {
  if(err) throw err;
  console.log('删除成功');
})
```

官方文档: [http://nodejs.cn/api/fs.html#fs\\_file\\_system\\_flags](http://nodejs.cn/api/fs.html#fs_file_system_flags)

## 第4集 node.js文件系统模块常用api操作（二）

简介: 讲解如何使用文件系统操作文件夹

- fs.mkdir(path[, options], callback) 创建文件夹
  - path | |
  - options |
    - recursive 是否递归创建 默认值: false。
    - mode 文件模式（权限）Windows 上不支持。默认值: 0o777。
  - callback
    - err

```
const fs =require('fs');

// fs.mkdir('./a',err => {
//   if(err) throw err;
//   console.log('创建文件夹成功');
// })

fs.mkdir('./b/c',{
  recursive:true
},err => {
  if(err) throw err;
  console.log('创建文件夹成功');
})
```

- fs.readdir(path[, options], callback) 读取文件夹
  - path | |
  - options |
    - encoding 默认值: 'utf8'。

- `withFileTypes` 默认值: `false`。
- `callback`
  - `err`
  - `files` `<string[]> | <buffer[]> | <fs.Dirent[]>`

```
fs.readdir('./',{
  encoding:'buffer', //设置buffer, files返回文件名为buffer对象
  withFileTypes:true //单上文件类型
},(err,files) => {
  if(err) throw err;
  console.log(files)
})
```

- `fs.rmdir(path[, options], callback)` 删除文件夹

- `path` | |
- `options`
  - `maxRetries` 重试次数。出现这类错误 `EBUSY`、`EMFILE`、`ENFILE`、`ENOTEMPTY` 或者 `EPERM`，每一个重试会根据设置的重试间隔重试操作。如果 `recursive` 不为 `true` 则忽略。默认值: `0`。
  - `retryDelay` 重试的间隔，如果 `recursive` 不为 `true` 则忽略。默认值: `100`。
  - `recursive` 如果为 `true`，则执行递归的目录删除。在递归模式中，如果 `path` 不存在则不报告错误，并且在失败时重试操作。默认值: `false`。
- `callback`
  - `err`

```
fs.rmdir('./b',{
  recursive:true
},err => {
  if(err) throw err;
  console.log('删除文件夹成功');
})
```

- 错误代码意义: <https://blog.csdn.net/a8039974/article/details/25830705>

- 监听文件变化 `chokidar`

- 安装 `chokidar`

```
npm install chokidar --save-dev
```

- `Chokidar.watch(path,[options])`

```
chokidar.watch('./',{
  ignored: './node_modules'
}).on('all',(event,path) => {
  console.log(event,path)
})
```

## 第5集 核心知识之文件流讲解

简介：讲解如何创建读取文件流和创建写入文件流

- Node.js 中有四种基本的流类型：
  - [Writable](#) - 可写入数据的流（例如 [fs.createWriteStream\(\)](#)）。
  - [Readable](#) - 可读取数据的流（例如 [fs.createReadStream\(\)](#)）。
  - [Duplex](#) - 可读又可写的流（例如 [net.Socket](#)）。
  - [Transform](#) - 在读写过程中可以修改或转换数据的 [Duplex](#) 流（例如 [zlib.createDeflate\(\)](#)）。
- 创建读取文件流 `fs.createReadStream(path[, options])`

```
const fs = require('fs');

let rs = fs.createReadStream('./streamTest.js',{
  highWaterMark:100 //每次on data的一个数据量
});

let count = 1;
rs.on('data',chunk => {
  console.log(chunk.toString())
  console.log(count++)
})

rs.on('end',()=>{
  console.log('读取完成')
});
```

- 创建写入文件流 `fs.createWriteStream(path[, options])`

```
let ws = fs.createWriteStream('./a.txt');
```

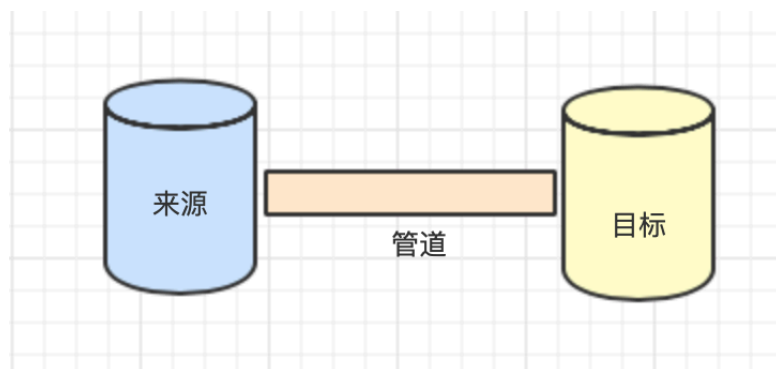
```

let num = 1;
let timer = setInterval(() => {
  if(num < 10){
    ws.write(num + '')
    num++
  }else{
    ws.end("写入完成");
    clearInterval(timer)
  }
}, 200);

ws.on('finish', ()=>{
  console.log('写入完成')
})

```

- 管道流



从数据流来源中一段一段通过管道流向目标。

- readable.pipe(destination[, options])

```

let rs = fs.createReadStream('./streamTest.js');
let ws = fs.createWriteStream('./a.txt');

rs.pipe(ws)

```

## 第6集 基础模块path常用api

简介：讲解path模块常用的一些api

- **path.basename(path[,ext])** 返回path的最后一部分
- **path.dirname(path)** 返回path的目录名
- **path.extname(path)** 返回path的扩展名
- **path.join(...paths)** 路径拼接
- **path.normalize(path)** 规范化路径
- **path.resolve(...paths)** 将路径解析为绝对路径
- **path.format(pathObject)** 从对象中返回路径字符串
- **path.parse(path)** 返回一个对象，包含path的属性
- **path.sep** 返回系统特定的路径片段分隔符
- **path.win32** 可以实现访问windows的path方法
- **\_\_filename** 表示当前正在执行的脚本的文件名
- **\_\_dirname** 表示当前执行脚本所在的目录

```
const {basename,dirname,extname,join,normalize,resolve,format,parse,sep,win32}
= require('path');

// console.log(basename('/nodejs/2-6/index.js','.js'))
// console.log(dirname('/nodejs/2-6/index.js'))
// console.log(extname('index.'))
// console.log(join('/nodejs/', '/index.js'))
// console.log(normalize('/nodejs/test/../../index.js'))
// console.log(resolve('./pathTest.js'))
// let pathObj = parse('/nodejs/test/index.js');
// console.log(pathObj)

// console.log(format(pathObj))
// console.log("当前系统下分隔符 " + sep)
// console.log("windows下分隔符 " + win32.sep)

console.log("filename " + __filename)
// console.log(__dirname)
console.log("resolve " + resolve('./pathTest.js'))
```

## 第7集 深度讲解node.js事件触发器

简介：讲解事件触发器events的使用方法

- **eventEmitter.on(eventName, listener)**注册监听器
  - `eventName` | 事件名称
  - `listener` 回调函数。
- **eventEmitter.emit(eventName[, ...args])** 触发事件

- `eventName` | 事件名称
- `...args` 参数
- **`eventEmitter.once(eventName, listener)`** 绑定的事件只能触发一次
- **`emitter.removeListener(eventName, listener)`** 从名为 `eventName` 的事件的监听器数组中移除指定的 `listener`。
- **`emitter.removeAllListener([eventName])`** 移除全部监听器或指定的 `eventName` 事件的监听器

```
const EventEmitter = require('events');

class MyEmitter extends EventEmitter{}

let myEmitter = new MyEmitter();

function fn1(a,b){
  console.log('触发了事件, 带参 ',a+b)
}

function fn2(){
  console.log("触发了事件, 不带参")
}

myEmitter.on('hi',fn1)

myEmitter.on('hi',fn2)

myEmitter.once('hello',()=>{
  console.log('触发了hello事件')
})

myEmitter.emit('hi',1,8);
// myEmitter.emit('hello')
// myEmitter.emit('hello')

// myEmitter.removeListener('hi',fn1)
myEmitter.removeAllListeners('hi');
myEmitter.emit('hi',1,8);
```

## 第8集 核心模块util常用工具

简介：讲解util模块里常用的工具

- **util.callbackify(original)** 将 `async` 异步函数（或者一个返回值为 `Promise` 的函数）转换成遵循异常优先的回调风格的函数

```
const util = require('util');

async function hello(){
  return 'hello world'
}

let helloCb = util.callbackify(hello);

helloCb((err,res) => {
  if(err) throw err;
  console.log(res)
})
```

- **util.promisify(original)** 转换成 promise 版本的函数

```
let stat = util.promisify(fs.stat)

// stat('./utilTest.js').then((data) => {
//   console.log(data)
// }).catch((err) => {
//   console.log(err)
// })

async function statFn () {
  try {
    let stats = await stat('./utilTest.js');
    console.log(stats)
  } catch (e) {
    console.log(e)
  }
}

statFn();
```

- **util.types.isDate(value)** 判断是否为date数据

```
console.log(util.types.isDate(new Date()))
```





愿景："让编程不在难学，让技术与生活更加有趣"

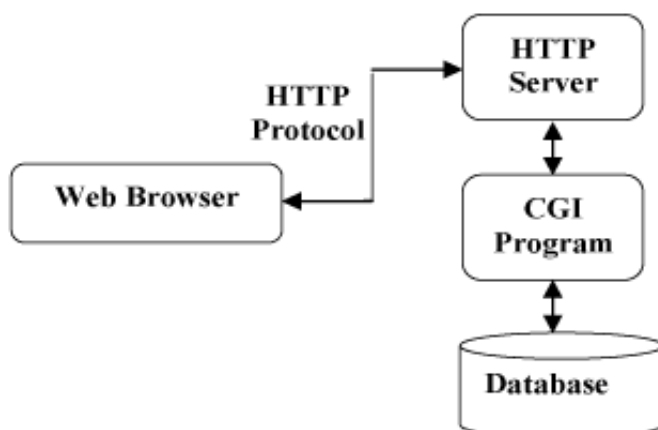
更多课程请访问[xdclass.net](http://xdclass.net)

## 第三章 http全面解析

### 第1集 http的发展历史

简介：讲解什么是http

- http是什么？ <http://www.xxx.com>
  - http协议（HyperText Transfer Protocol，超文本传输协议）是一种应用广泛的网络传输协议。
  - http是一个基于TCP/IP通讯协议来传递数据（HTML文件，图片文件，查询结果等）。
- http工作原理
  - http协议工作在客户端-服务端之间
  - 主流的三个web服务器：Apache、Nginx、IIS。
  - http默认端口为80
  - http协议通信流程



- 输入url发生了什么？
  - DNS解析
  - TCP连接
  - 发送http请求
  - 服务器处理请求

- 浏览器解析渲染页面
- 连接结束

## 第2集 走进http之请求方法和响应头信息

简介：讲解http的请求方法和响应头信息

- http请求方法
  - GET 请求指定的页面信息，并返回实体主体
  - HEAD 类似于get请求，只不过返回的响应中没有具体的内容，用于获取报头
  - POST 向指定资源提交数据进行处理请求。数据被包含在请求体中。
  - PUT 从客户端向服务器传送的数据取代指定的文档的内容
  - DELETE 请求服务器删除指定的页面
  - CONNECT HTTP/1.1协议中预留给能够将连接改为管道方式的代理服务器。
  - OPTIONS 允许客户端查看服务器的性能
  - TRACE 回显服务器收到的请求，主要用于测试或诊断
- HTTP响应头信息

应答头	说明
Allow	服务器支持哪些请求方法（如get、post等）
Content-Encoding	文档的编码方法。只有在解码之后才可以得到Content-Type头指定的内容类型。利用gzip压缩能减少HTML文档的下载时间。
Content-Length	表示内容长度。只有当浏览器使用持久http连接时才需要这个数据。
Content-Type	表示文档属于什么MIME类型。
Date	当前的GMT时间。
Expires	资源什么时候过期，不再缓存
Last-Modified	文档最后改动时间。
Location	重定向的地址
Server	服务器的名字
Set-Cookie	设置和页面关联的Cookie
WWW-Authenticate	定义了使用何种验证方式去获取对资源的链接

## 第3集 走进http之状态码和content-type

简介：讲解http的状态码和content-type

- 常见的http状态码
  - 200 请求成功
  - 301 资源被永久转移到其他URL
  - 404 请求的资源（网页等）不存在
  - 500 内部服务器错误
- http状态码分为5类：

分类	分类描述
1**	信息，服务器收到请求，需要请求者继续执行操作
2**	成功，操作被成功接收并处理
3**	重定向，需要进一步的操作以完成请求
4**	客户端错误，请求包含语法错误或无法完成请求
5**	服务器错误，服务器在处理请求的过程中发生了错误

- **Content-Type** 内容类型

- 常见的媒体格式类型如下
  - text/html:HTML格式
  - text/plain:纯文本格式
  - text/xml:XML格式
  - image/gif:gif图片格式
  - image/jpeg:jpg图片格式
  - image/png:png图片格式
  - multipart/form-data:需要在表单中进行文件上传时，就需要使用该格式
- 以application开头的媒体格式类型：
  - application/xhtml+xml:XHTML格式
  - application/xml:XML数据格式
  - application/atom+xml:Atom XML聚合格式
  - application/json:JSON数据格式
  - application/pdf:pdf格式
  - application/msword:Word文档格式
  - application/octet-stream:二进制流数据（常见的文件下载）
  - application/x-www-form-urlencoded:表单中默认的encType,表单数据被编码为key/value格式发送到服务器

## 第4集 搭建自己的第一个http服务器

简介：讲解如何使用nodejs中的http模块搭建服务器

- 引入http模块

```
const http = require('http')
```

- 创建http服务器

```
const http = require('http');

const server = http.createServer((req, res) => {
  res.writeHead(200, { 'content-type': 'text/html' });
  res.end('<h1>hello world</h1>');
})

server.listen(3000, () => {
  console.log('监听了3000端口')
})
```

## 第5集 实战案例之nodejs简易爬虫

简介：讲解如何使用http模块做一个简单的爬虫

- 简单爬虫实现

```
const https = require('https');
const fs = require('fs');

https.get('https://xdclass.net/#/index', (res) => {
  res.setEncoding('utf8');
  let html = '';
  res.on('data', chunk => {
    html += chunk;
  })
  res.on('end', () => {
    console.log(html)
    fs.writeFile('./index.txt', html, (err) => {
      if(err) throw err;
      console.log('写入成功')
    })
  })
})
```

- cheerio实现dom操作
  - 安装cheerio

```
npm install cheerio --save-dev
```

- 引入cheerio

```
const cheerio = require('cheerio')
```

- 通过title元素获取其内容文本

```
const $ = cheerio.load(html); //把html代码加载进去，就可以实现jq的dom操作
console.log($('title').text());
```



愿景："让编程不在难学，让技术与生活更加有趣"

更多课程请访问[xdclass.net](http://xdclass.net)

## 第四章 NodeJs核心模块api-路由与接口

### 第1集 如何处理客户端get/post请求

简介：讲解如何处理从页面中传来的参数

- `url.parse(urlString[, parseQueryString[, slashesDenoteHost]])`
  - `urlString` url字符串
  - `parseQueryString` 是否解析
  - `slashesDenoteHost`
    - -默认为false, `//foo/bar` 形式的字符串将被解释成 `{ pathname: '//foo/bar' }`
    - -如果设置成true, `//foo/bar` 形式的字符串将被解释成 `{ host: 'foo', pathname: '/bar' }`

```
console.log(url.parse('https://api.xdclass.net/pub/api/v1/web/product/find_list_by_type?type=2', true, true))
```

- get请求用于客户端向服务端获取数据，post请求是客户端传递数据给服务端
- 处理get请求

```

const url = require('url');
const http = require('http');

//
console.log(url.parse('https://api.xdclass.net/pub/api/v1/web/product/find
_list_by_type?type=2',true,true))

const server = http.createServer((req,res) => {
  let urlObj = url.parse(req.url,true);
  res.end(JSON.stringify(urlObj.query))
})

server.listen(3000,()=>{
  console.log('监听3000端口')
})

```

- 处理post请求

```

const url = require('url');
const http = require('http');

//
console.log(url.parse('https://api.xdclass.net/pub/api/v1/web/product/find
_list_by_type?type=2',true,true))

const server = http.createServer((req,res) => {
  let postData = '';
  req.on('data',chunk => {
    postData += chunk;
  })
  req.on('end',()=>{
    console.log(postData)
  })
  res.end(JSON.stringify({
    data:'请求成功',
    code:0
  })))
})

server.listen(3000,()=>{
  console.log('监听3000端口')
})

```

- 整合get/post请求

```

const url = require('url');
const http = require('http');

```

```
//
console.log(url.parse('https://api.xdclass.net/pub/api/v1/web/product/find
_list_by_type?type=2',true,true))

const server = http.createServer((req, res) => {
  if (req.method === 'GET') {
    let urlObj = url.parse(req.url, true);
    res.end(JSON.stringify(urlObj.query))
  } else if (req.method === 'POST') {
    let postData = '';
    req.on('data', chunk => {
      postData += chunk;
    })
    req.on('end', () => {
      console.log(postData)
    })
    res.end(JSON.stringify({
      data: '请求成功',
      code: 0
    })))
  }
})

server.listen(3000, () => {
  console.log('监听3000端口')
})
```

## 第2集 nodemon自动重启工具安装配置

简介：讲解nodemon安装以及配置

- nodemon安装

```
npm install -g nodemon
```

- 替换淘宝镜像

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



## 第3集 讲解初始化路由及接口开发

简介：讲解如何开发一个接口以及路由编写

- 通过pathname判断请求地址

```
//server.js    服务器文件
const http = require('http');
const routerModal = require('./router/index')

const server = http.createServer((req,res)=>{
  res.writeHead(200,{ 'content-type': 'application/json;charset=UTF-8' })
  let resultData = routerModal(req,res);
  if(resultData){
    res.end(JSON.stringify(resultData))
  }else{
    res.writeHead(404,{ 'content-type': 'text/html' })
    res.end('404 not found')
  }
})

server.listen(3000,() => {
  console.log('监听3000端口')
})
```

```
//router/index.js    路由文件
const url = require('url')
function handleRequest(req,res) {
  let urlObj = url.parse(req.url,true);
  console.log(urlObj)
  if(urlObj.pathname === '/api/getMsg' && req.method === 'GET'){
    return {
      msg: '获取成功'
    }
  }
  if(urlObj.pathname === '/api/updateData' && req.method === 'POST'){
    return {
      msg: '更新成功'
    }
  }
}

module.exports = handleRequest
```

## 第4集 案例实战用户列表增删改查

简介：讲解用户列表增删改查接口开发

- 使用promise处理post请求

```
const getPostData = (req) => {
  return new Promise((resolve, reject) => {
    if (req.method !== 'POST') {
      resolve({})
      return
    }
    let postData = '';
    req.on('data', chunk => {
      postData += chunk;
    })
    req.on('end', () => {
      console.log(postData)
      resolve(JSON.parse(postData))
    })
  })
}

const server = http.createServer((req, res) => {
  res.writeHead(200, { 'content-type': 'application/json;charset=UTF-8' })
  //获取post请求数据
  getPostData(req).then((data) => {
    req.body = data
    let resultData = routerModal(req, res);
    if (resultData) {
      res.end(JSON.stringify(resultData))
    } else {
      res.writeHead(404, { 'content-type': 'text/html' })
      res.end('404 not found')
    }
  })
})
```

- 处理数据的文件controller/user.js

```

module.exports = {
  getUserList(){
    return [
      {
        id:1,
        name:'eric',
        city:'北京'
      },
      {
        id:2,
        name:'xiaoming',
        city:'广州'
      },
      {
        id:3,
        name:'小红',
        city:'上海'
      }
    ]
  },
  addUser(userObj){
    console.log(userObj);
    return {
      code:0,
      msg:'新增成功',
      data:null
    }
  },
  delectUser(id){
    console.log(id)
    return {
      code:0,
      msg:'删除成功',
      data:null
    }
  },
  updateUser(id,userObj){
    console.log(id,userObj);
    return {
      code:0,
      msg:'更新成功',
      data:null
    }
  }
}
}

```

- 路由编写文件

```

const url = require('url')
const {getUserList,addUser,delectUser,updateUser} =
require('../controller/user')
function handleRequest(req,res) {
  let urlObj = url.parse(req.url,true);
  console.log(urlObj)
  if(urlObj.pathname === '/api/getUserList' && req.method === 'GET'){
    let resultData = getUserList()
    console.log(resultData)
    return resultData;
  }
  if(urlObj.pathname === '/api/addUser' && req.method === 'POST'){
    let resultData = addUser(req.body);
    return resultData;
  }
  if(urlObj.pathname === '/api/delectUser' && req.method === 'POST'){
    let resultData = delectUser(urlObj.query.id);
    return resultData;
  }
  if(urlObj.pathname === '/api/updateUser' && req.method === 'POST'){
    let resultData = updateUser(urlObj.query.id,req.body);
    return resultData;
  }
}

module.exports = handleRequest

```

## 第5集 教你轻松解决接口跨域问题

简介：讲解如何利用cors解决跨域问题

- 什么是跨域？

浏览器同源策略：协议+域名+端口三者相同就是同源。

http://www.baidu.com/a.js	http://www.baidu.com/b.js	
https://www.baidu.com/a.js	http://www.baidu.com/a.js	协议不同
https://www.baidu.com:8080/a.js	https://www.baidu.com/a.js	端口不同
https://www.baidu.com:8080/a.js	https://www.a.com:8080/a.js	域名不同

跨域：协议、域名、端口三者任意一个不同就是跨域。

- 前端请求跨域提示

```
✖ Access to XMLHttpRequest at 'http://127.0.0.1:3000/api/getUserList' from origin 'http://127.0.0.1:5500' has been blocked by CORS policy: No 'Access-Control-Allow-Origin' header is present on the requested resource. index.html:1
```

- 跨域的解决方法cors方法

```
//设置允许跨域的域名, *代表允许任意域名跨域
res.setHeader("Access-Control-Allow-Origin", "*");
```



愿景："让编程不在难学，让技术与生活更加有趣"

更多课程请访问[xdclass.net](http://xdclass.net)

## 第五章 NodeJS连接Mysql

### 第1集 mysql介绍

简介：介绍mysql及mysql软件推荐

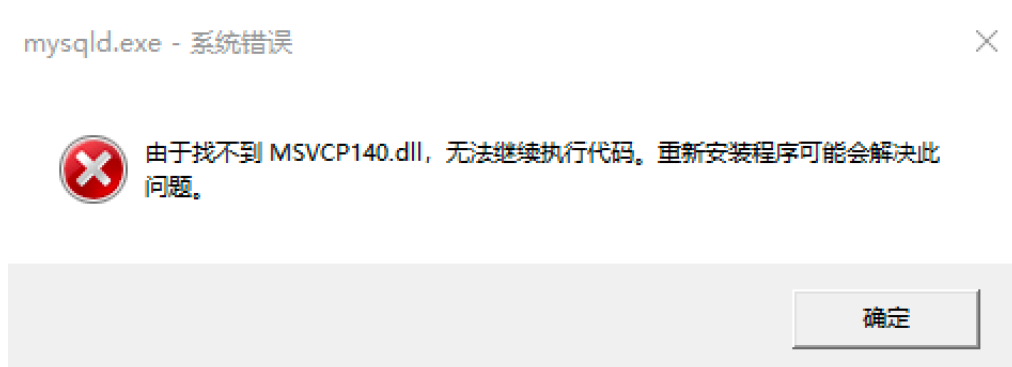
- 什么是mysql?  
mysql是一个数据库管理系统。数据库是存储、管理数据的仓库。
- mysql环境安装配置
  - windows安装及配置

windows安装配置教程

[https://blog.csdn.net/qq\\_37350706/article/details/81707862](https://blog.csdn.net/qq_37350706/article/details/81707862)

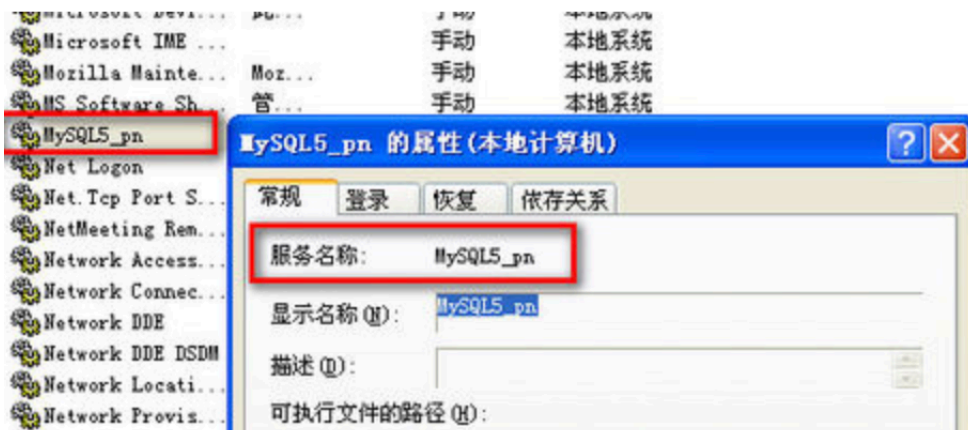
安装过程出现错误总结：

- 执行mysqld出现以下错误，可能是电脑缺少VC++ 2015运行库，安装一下就可以了



VC++2015下载地址：<https://www.microsoft.com/zh-CN/download/details.aspx?id=48145>

- net start mysql服务名无效
  1. win+R打开运行窗口，输入services.msc
  2. 在其中查看mysql的服务名，找到mysql开头的，如下图



3. 以管理员身份打开cmd，输入net start mysql服务名
4. 出现下图表示启动成功

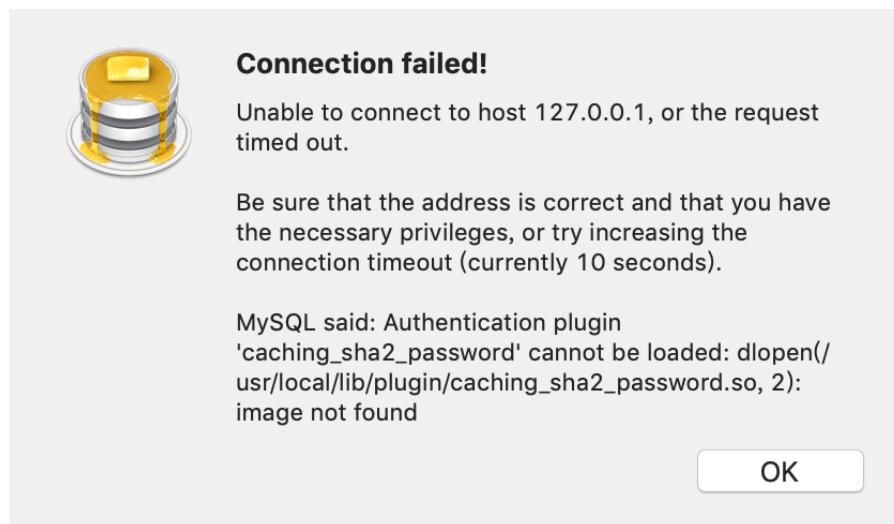
管理员: 命令提示符

```
Microsoft Windows [版本 10.0.18363.535]  
(c) 2019 Microsoft Corporation。保留所有权利。  
  
C:\Windows\system32>net stop mysql  
MySQL 服务正在停止。  
MySQL 服务已成功停止。  
  
C:\Windows\system32>net start mysql  
MySQL 服务正在启动。  
MySQL 服务已经启动成功。  
  
C:\Windows\system32>
```

- 出现拒绝访问，denied等字样，表示需使用管理员运行命令行
- mac安装及配置

mac下mysql8安装教程: <https://blog.csdn.net/luzhensmart/article/details/82948133>

- 客户端连接报错



客户端不支持mysql8的新密码格式，修改密码加密规则之后重新修改密码即可。

解决方案地址: <https://blog.csdn.net/u011182575/article/details/80821418>

- mysql数据库管理工具

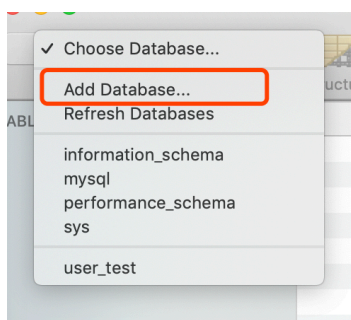
- MySQL Workbench: 下载地址: <https://dev.mysql.com/downloads/workbench/>

- navicat
- sequel pro: 下载地址<https://sequalpro.com/test-builds>

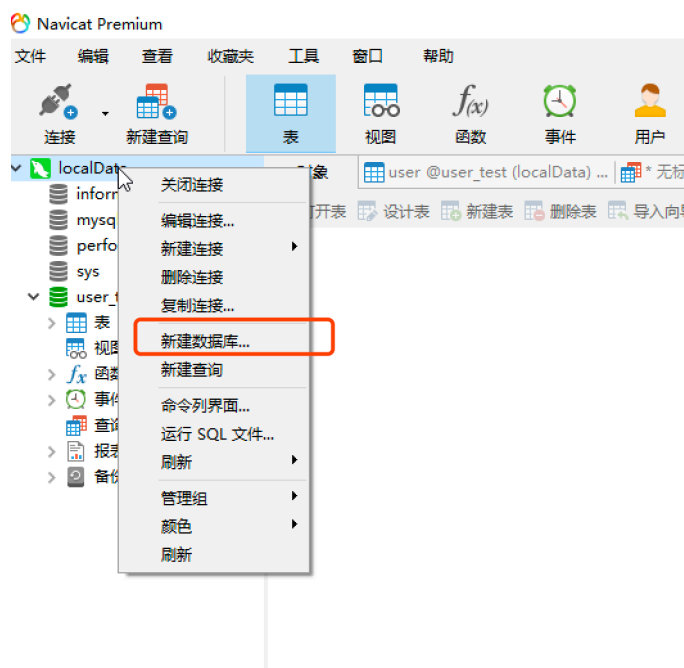
## 第2集 开发前准备之mysql数据库设计

简介: 讲解如何去创建一个数据库

- 创建数据库
  - Sequel pro

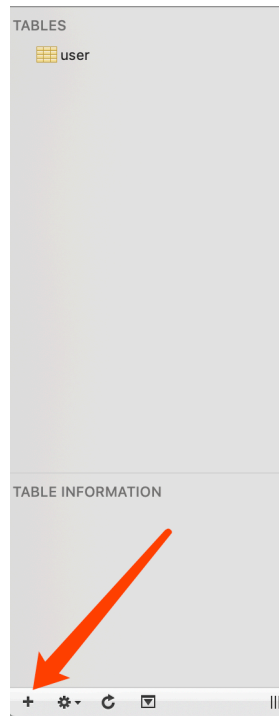


- Navicat

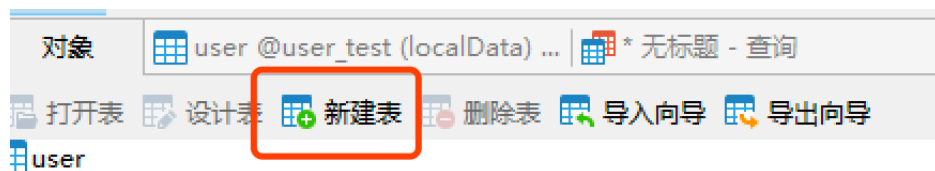


- 创建表
  - Sequel pro





# - Navicat



## - 设计表

### - navicat

字段	索引	外键	触发器	选项	注释	SQL 预览					
名			类型	长度	小数点	不是 null	虚拟	键	注释		
id			int	11	0	<input checked="" type="checkbox"/>	<input type="checkbox"/>	1	用户id		
name			varchar	128	0	<input type="checkbox"/>	<input type="checkbox"/>		用户名称		
city			varchar	255	0	<input type="checkbox"/>	<input type="checkbox"/>		城市		
sex			tinyint	2	0	<input type="checkbox"/>	<input type="checkbox"/>		1是女, 2是男		

### - sequel pro

Field	Type	Length	Unsign...	Zero...	Bina...	Allow Null	Key	Default	Extra	Encoding	Collation	Comment
id	INT	11	<input checked="" type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	PRI		auto_incre...			用户id
name	VARCHAR	128	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input checked="" type="checkbox"/>		NULL	None	UTF-8 Uni...	utf8_gener...	用户名称
city	VARCHAR	128	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input checked="" type="checkbox"/>		NULL	None	UTF-8 Uni...	utf8_gener...	城市
sex	TINYINT	2	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input checked="" type="checkbox"/>		NULL	None			1是女, 2...

- id设为主键（主键表示该字段为唯一标识，不能为空），自动递增
- varchar为字符类型，长度可根据业务需求设置
- int, tinyint为整数类型，tinyint占用1个字节，int占用4字节，tinyint允许从0到255的所有数字，int允许从-2 147 483 648~ 2 147 483 647

## 第3集 mysql常用数据库操作语句

简介：讲解数据库中常用的操作语句

- 增加表格数据

```
INSERT INTO table_name ( field1, field2,...fieldN ) VALUES (value1,
value2,...valueN)
```

//往user表插入一条数据

```
insert into user (name,city,sex) values ('小小','北京',1)
```

- 删除表格数据

```
DELETE FROM table_name [WHERE Clause]
```

//删除用户id为5的用户

```
delete from user where id = 5
```

- 修改表格数据

```
UPDATE table_name SET field1=new-value1, field2=new-value2 [WHERE Clause]
```

//修改用户id为5的信息

```
update user set name='小天',city='深圳' where id = 5
```

- 查询表格数据

```
SELECT column_name,column_name FROM table_name [WHERE Clause]
```

//查询所有用户信息，\*表示显示所有字段信息

```
select * from user
```

//查询所有用户信息，只显示name和city信息

```
select name,city from user
```

//查询id为4的用户

```
select name,city from user where id = 4
```

//同时满足两个条件用and

```
select name,city from user where city = '北京' and sex = 1
```

- 排序

```
SELECT field1, field2,...fieldN FROM table_name1, table_name2...  
ORDER BY field1 [ASC [DESC][默认 ASC]], [field2...] [ASC [DESC][默认 ASC]]
```

默认asc升序排序，desc降序排序

//根据id进行降序排序

```
select * from user order by id desc
```

- 模糊查询

```
SELECT field1, field2,...fieldN FROM table_name WHERE field1 LIKE  
condition1
```

//查询名字带有红的用户

```
select * from user where name like '%红%'
```

## 第4集 NodeJs连接mysql数据库讲解

简介：讲解nodejs安装mysql及连接mysql数据库方式

- mysql模块安装

```
npm install mysql --save
```

- 连接数据库

```
const mysql = require('mysql')

//创建连接
const conn = mysql.createConnection({
  host: 'localhost',
  user: 'root',
  password: '123456789',
  port: '3306',
  database: 'user_test'
})

//建立连接
conn.connect()

let sql = 'select * from user where id = ?'

//执行sql语句
conn.query(sql, [4], (err, result) => {
  if (err) throw err
  console.log(result)
})

//关闭连接
conn.end()
```

- 通过占位符实现传参,query方法第二参数就是会填充sql语句里的?

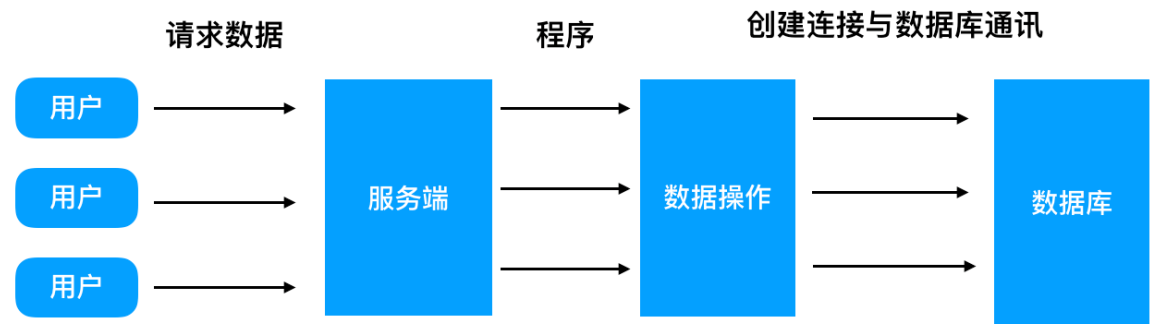
```
let sql = 'select * from user where id = ?'

//执行sql语句
conn.query(sql, [4], (err, result) => {
  if (err) throw err
  console.log(result)
})
```

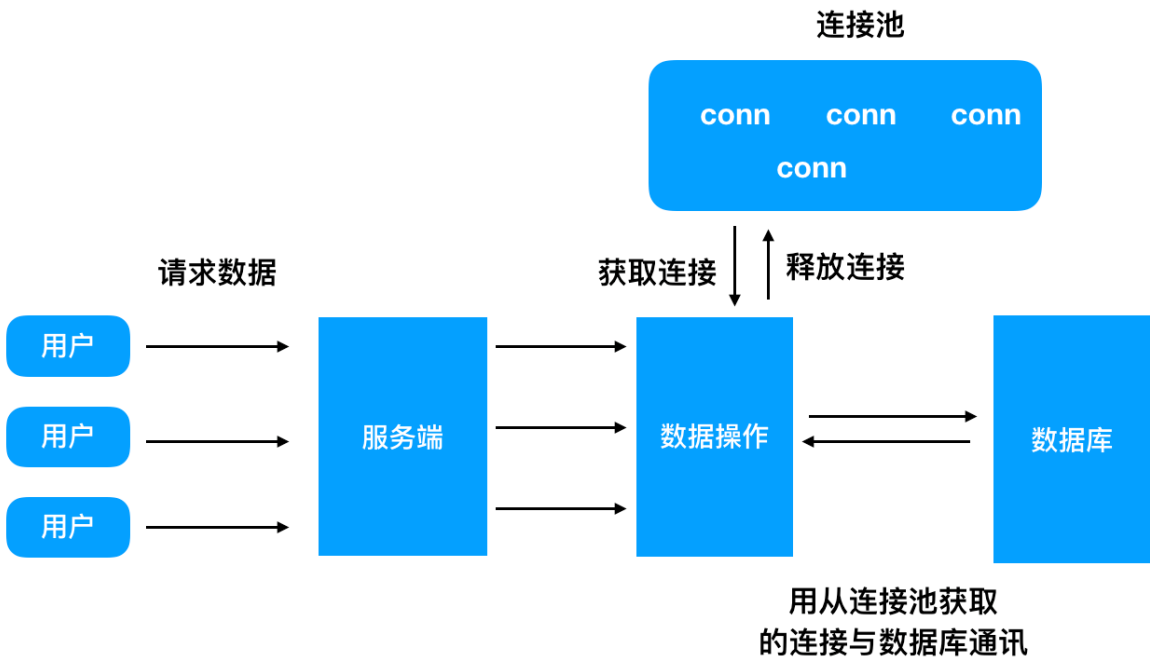
# 第5集 深度讲解mysql连接池

简介：mysql连接池与普通连接的区别以及它的使用方式

- 频繁的创作、关闭连接会减低系统的性能，提高系统的开销



- 连接池可以有效的管理连接，达到连接复用的效果



- 连接池的使用

```
const mysql = require('mysql')

//创建连接池
const pool = mysql.createPool({
  connectionLimit: 10,
  host: 'localhost',
  user: 'root',
  password: '123456789',
  port: '3306',
  database: 'user_test'
})

//获取连接
pool.getConnection((err, conn) => {
  if (err) throw err
  let sql = 'select * from user where city = ?'

  //执行sql语句
  conn.query(sql, ['广州'], (err, result) => {
    conn.release()
    if (err) throw err
    console.log(result)
  })
})
```

## 第6集 结合数据库改造用户列表接口（增）

简介：改造接口，通过sql语句操作数据库增加用户

- 数据库配置

```

let dbOption

dbOption = {
  connectionLimit: 10,
  host: 'localhost',
  user: 'root',
  password: '123456789',
  port: '3306',
  database: 'user_test'
}

module.exports = dbOption

```

- 数据库连接，以及query方法封装

```

const mysql = require('mysql')
const dbOption = require('../config/db_config')

//创建连接池
const pool = mysql.createPool(dbOption)

function query (sql,params) {
  return new Promise((resolve, reject) => {
    //获取连接
    pool.getConnection((err, conn) => {
      if (err){
        reject(err)
        return
      }
      //执行sql语句
      conn.query(sql, params, (err, result) => {
        conn.release()
        if (err) {
          reject(err)
          return
        }
        resolve(result)
      })
    })
  })
}

module.exports = query

```

- server通过获取的promise结果来获取数据，并且返回结果

```

let result = routerModal(req, res);
if (result) {
  result.then(resultData =>{
    res.end(JSON.stringify(resultData))
  })
} else {
  res.writeHead(404, { 'content-type': 'text/html' })
  res.end('404 not found')
}

```

## 第7集 结合数据库改造用户列表接口（删改）

简介：讲解用户列表删除更新接口的改造编写

- 更新用户接口

```

async updateUser(id,userObj){
  // console.log(id,userObj);
  let {name,city,sex} = jsonObj
  let sql = 'update user set name = ?,city = ?,sex = ? where id = ?'
  let resultData = await query(sql,[name,city,sex,id])
  if(resultData.affectedRows > 0){
    return {
      msg: '更新成功'
    }
  }else{
    return {
      msg: '更新失败'
    }
  }
}

```

- 删除用户接口



```

async delectUser(id){
    let sql = 'delete from user where id = ?'
    let resultData = await query(sql,[id])
    if(resultData.affectedRows > 0){
        return {
            msg: '删除成功'
        }
    }else{
        return {
            msg: '删除失败'
        }
    }
}

```

## 第8集 结合数据库改造用户列表接口（动态查询）

简介：讲解通过sql语句操作数据库实现动态查询

```

async getUserList(urlParams){
    let {name,city} = urlParams
    let sql = 'select * from user where 1=1 '
    if(name){
        sql += 'and name = ?'
    }
    if(city){
        sql += 'and city = ?'
    }
    let resultData = await query(sql,[name,city])
    return resultData
}

```



愿景："让编程不在难学，让技术与生活更加有趣"

更多课程请访问[xdclass.net](http://xdclass.net)

## 第六章 分布式文件储存数据库MongoDB

### 第1集 MongoDB的介绍及安装

简介：介绍MongoDB及安装方式

- MongoDB是用C++语言编写的非关系型数据库
- MongoDB与mysql的区别

	MongoDB	mysql
数据库模型	非关系型	关系型
表	collection集合	table二维表
表的行数据	document文档	row记录
数据结构	虚拟内存+持久化	不同引擎不同储存方式
查询语句	mongodb查询方式（类似js函数）	sql语句
数据处理	将热数据存储在物理内存中，从而达到快速读写	不同引擎有自己的特点
事务性	不支持	支持事务
占用空间	占用空间大	占用空间小
join操作	没有join	支持join

- mongodb数据库软件及可视化软件安装

数据库软件安装地址：<https://www.mongodb.com/download-center/community>

可视化软件安装地址：<https://www.mongodb.com/download-center/compass>

mac安装数据库教程

- 安装brew教程：<https://www.jianshu.com/p/dea776e7effb>
- 安装mongodb教程<https://www.cnblogs.com/georgeleoo/p/11479409.html>

- 启动mongodb命令

- mac

```
// 启动
brew services start mongodb-community@4.2
// 关闭
brew services stop mongodb-community@4.2
```

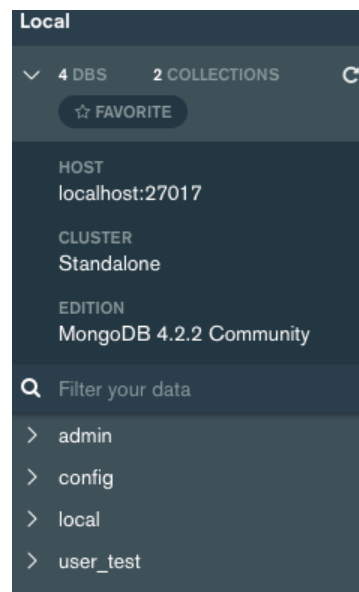
- windows

```
//启动
net start mongodb
//关闭
net stop mongodb
```


## 第2集 玩转MongoDB可视化工具

简介：讲解如何使用MongoDB可视化工具

- 数据库列表




- 集合列表,这里相当于mysql的表






Collections						
CREATE COLLECTION						
Collection Name	Documents	Avg. Document Size	Total Document Size	Num. Indexes	Total Index Size	Properties
users	4	77.8 B	311.0 B	1	36.9 KB	

- 集合里的文档,相当于mysql的一条数据

users |> insertMany

Documents Aggregations Schema Explain Plan


 FILTER

 ADD DATA  VIEW   

```

_id: ObjectId("5e0eec352c455987153a027a")
name: "eric"
city: "上海3"
sex: 1
__v: 0
age: 18

```



```

_id: ObjectId("5e0ef7b0e3220087de5cf05c")
sex: 1
name: "eric1"
city: "上海3"
__v: 0
age: 30

```

```

_id: ObjectId("5e0ef9fd69f3e187fca49fff")
sex: 1
name: "haha"
city: "上海3"
__v: 0
age: 40

```

```

_id: ObjectId("5e0f0123bfb762f60b77824d")
name: "xxx2"
city: "上海3"
age: 40

```

## 第3集 讲解第三方包mongoose的使用

简介：详细讲解如何使用mongoose连接数据库

- 安装mongoose包

```
npm install mongoose
```

- 数据库连接

```
const mongoose = require('mongoose')

//连接数据库,返回promise
mongoose.connect('mongodb://localhost/user_test',{
  useNewUrlParser: true,
  useUnifiedTopology: true
}).then(()=>{
  console.log('连接数据库成功')
}).catch(err => {
  console.log(err, '连接数据库失败')
})
```

官方文档地址: <https://mongoosejs.com/docs/connections.html>

## 第4集 MongoDB常用数据库操作之创建集合、文档

简介: 讲解MongoDB创建集合、创建文档

- mongodb不需要显示创建数据库, 如果数据库不存在, 它会自动创建。
- 创建集合

```
const Schema = new mongoose.Schema(options) //创建集合结构 (规则)  options集
合的结构 (规则)
const Model = mongoose.model(modelName, schema) //modelName集合名称
schema集合结构

//案例代码

//创建集合结构
const userSchema = new Schema({
  name:String,
  city:String,
  sex:Number
})

//创建集合
const Model = mongoose.model('user',userSchema)
```

- 创建文档
  - 第一种方式

```

const doc = new Model(options) //options符合集合规则的文档数据，以对象形式
doc.save() //将文档插入数据库中

//案例代码
//创建文档
const doc = new Model({
  name: 'eric',
  city: '深圳',
  sex: 2
})

//将文档插入数据库中
doc.save()

```

#### - 第二种方式

```

```js
//options符合集合规则的文档数据，以对象形式,err为报错信息，doc是插入的文档
Model.create(options, (err, doc))

//案例代码
//创建文档并把文档插入数据库中
Model.create({
  name: '嘻嘻',
  city: '广州',
  sex: 1
}, (err, doc) => {
  if(err) throw err
  console.log(doc)
})

```

## 第5集 讲解MongoDB如何导入文件数据

简介：讲解MongoDB里如何导入文件数据到数据库

- 数据库导入数据

```
mongoimport -d 数据库名称 -c 集合名称 --file 导入的数据文件路径
```

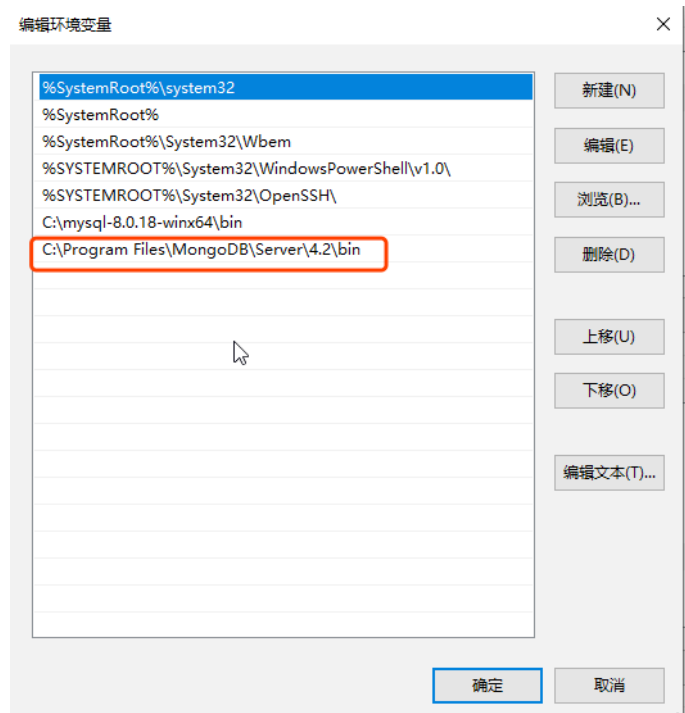
```
mongoimport -d test -c users --file ./user.json
```

- mongoimport环境变量配置

- mac

如果使用不了命令，访问该连接，用教程里的第二个方式配置环境变量<https://www.cnblogs.com/georgeleoo/p/11479409.html>

- window



配置系统环境变量，将mongodb安装路径添加进去（要进到bin文件夹）

## 第6集 MongoDB常用数据库操作之查询文档

简介：讲解MongoDB查询文档的多种用法

- 查询文档

```

Model.find(条件) //根据条件查询文档，条件为空则查询所有文档，      返回数组
Model.findOne(条件) //默认返回当前集合中的第一条文档      返回对象

Model.find({name:'6'}).then(res => {
  console.log(res)
})

Model.findOne({name:'2'}).then(res => {
  console.log(res)
})

```

- 区间查询

```

{key:{$gt:value,$lt:value}}  gt大于 lt小于  gte大于等于 lte小于等于

Model.find({age:{$gte:18,$lte:38}}).then(res => {
  console.log(res)
})

```

- 模糊查询

```

{key:正则表达式}

Model.find({city:/上/}).then(res => {
  console.log(res)
})

```

- 选择要查询的字段

```

Model.find().select(arg) //arg为要操作的字段  字段前加上-表示不查询该字段

Model.find().select('-name').then(res => {
  console.log(res)
})

```

- 排序

```

Model.find().sort(arg) //arg为要操作的字段  字段前加-表示降序排列

Model.find().sort('-age').then(res => {
  console.log(res)
})

```

- 跳过多少条数据、限制查询数量



```
Model.find().skip(num).limit(num)    //skip跳过多少条数据, limit限制查询数量

Model.find().skip(2).limit(3).then(res => {
  console.log(res)
})
```

## 第7集 MongoDB常用数据库操作之更新文档

简介：讲解MongoDB更新文档的用法

- 更新单个文档

```
//找到一个文档并更新,如果查询多个文档, 则更新第一个匹配文档    返回值为该文档
Model.findOneAndUpdate(条件,更新的值)
//更新指定条件文档,如果查询多个文档, 则更新第一个匹配文档
Model.updateOne(条件, 更新的值)

//全局配置
mongoose.set('useFindAndModify', false)

Model.findOneAndUpdate({name:'2'}, {city:'深圳'}).then(res => {
  console.log(res)
})

Model.updateOne({name:'2'}, {city:'上海'}).then(res => {
  console.log(res)
})
```

- 更新多个文档

```
Model.updateMany(条件, 更新的值)    //如果条件为空, 则会更新全部文档

Model.updateMany({name:'2'}, {city:"深圳"}).then(res => {
  console.log(res)
})
```

## 第8集 MongoDB常用数据库操作之删除文档

简介：讲解MongoDB删除文档的用法

- 删除单个文档

```
//找到一个文档并删除,如果查询多个文档, 则删除第一个匹配文档    返回值为该文档
Model.findOneAndDelete(条件)

//删除指定条件文档,如果查询多个文档, 则删除第一个匹配文档    返回值是一个成功对象
Model.deleteOne(条件)

Model.findOneAndDelete({name:'xx'}).then(res => {
  console.log(res)
})

Model.deleteOne({name:'2'}).then(res => {
  console.log(res)
})
```

- 删除多个文档

```
Model.deleteMany(条件)    //如果条件为空, 则会删除全部文档

Model.deleteMany().then(res => {
  console.log(res)
})
```

## 第9集 深度讲解MongoDB字段验证

简介：讲解集合中字段的验证

- required 验证字段是否为必须输入, 值为boolean

```
name:{
  type:String,
  required:[true, '该字段为必选字段'],
}
```

- minlength, maxlength 验证字符的值的的最小长度和最大长度

```
name:{
  type:String,
  required:[true, '该字段为必选字段'],
  minlength:[2, '输入值长度小于最小长度'],
  maxlength:[6, '输入值长度大于最大长度'],
}
```

- trim 去除字符串首尾空格, 值为boolean

```
name:{
  type:String,
  required:[true, '该字段为必选字段'],
  minlength:[2, '输入值长度小于最小长度'],
  maxlength:[6, '输入值长度大于最大长度'],
  trim:true
}
```

- min、max 验证最小最大数字

```
age:{
  type:Number,
  min:18,
  max:30
},
```

- default 默认值

```
createTime:{
  type>Date,
  default:Date.now
},
```

- enum 规定输入的值

```

hobbies:{
  type:String,
  enum:{
    values:['唱','跳','Rap'],
    message:'该值不在设定的值当中'
  }
},

```

- validate 根据自定义条件验证，通过validator函数处理输入值，message为自定义错误信息

```

validate:{
  validator:v => {
    //todo 返回布尔值验证输入值是否有效
  }
}

score:{
  type:Number,
  validate:{
    validator:v => {
      //返回布尔值
      return v&&v>0&&v<100
    },
    message:'不是有效的分数'
  }
}

```

- 通过catch获取errors对象，遍历对象从中获取对应字段自定义报错信息

```

Modal.create().then().catch(error => {
  //获取错误信息对象
  const errs = error.errors;
  //循环错误信息对象
  for(var i in errs ){
    console.log(err[i].message)
  }
})

```



愿景："让编程不在难学，让技术与生活更加有趣"

更多课程请访问[xdclass.net](http://xdclass.net)

## 第七章 NodeJs进阶知识线程与进程

### 第1集 进程与线程

简介：讲解进程与线程的概念

- 进程概念

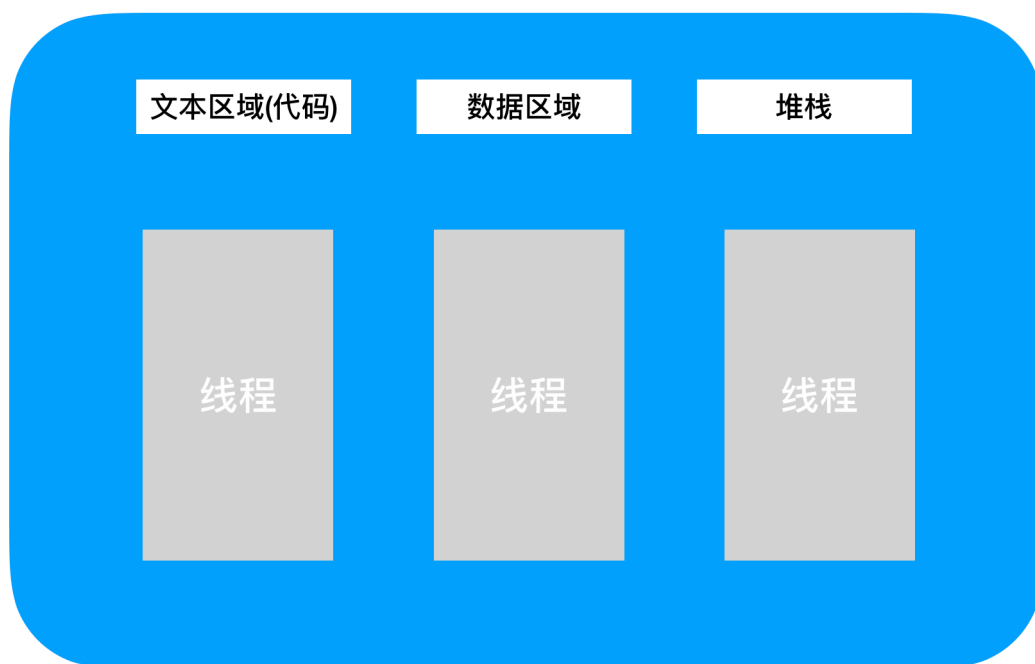
进程是正在运行的程序的实例，我们启动一个服务，运行一个实例，就是开一个服务进程。进程是线程的容器。进程包括文本区域（text region）、数据区域（data region）和堆栈（stack region）。

- 线程概念

线程是操作系统能够进行运算调度的最小单位。它被包含在进程之中，是进程中的实际运作单位。一个进程可以由一个或者多个线程组成，每条线程并行执行不同的任务。

- 进程与线程的关系

#### 进程



## 第2集 深度讲解process进程模块

简介：讲解process模块常用的功能点

- `process.env` 返回包含用户环境的对象，可设置环境变量，例如`Process.env.NODE_ENV`

```
windows上设置环境变量用set
set NODE_ENV=development && node index.js

mac上设置环境变量用export
export NODE_ENV=development && node index.js

安装cross-env, npm i cross-env --save-dev
cross-env NODE_ENV=development node index.js
```

- `process.pid` 返回进程的pid
- `process.platform` 返回当前进程的操作系统平台
- `process.title` 获取或指定进程名称
- `process.on('uncaughtException',cb)` 捕获异常信息
- `process.on('exit',cb)` 监听进程退出
- `process.cwd()` 返回当前进程的工作目录
- `process.uptime()` 返回当前进程运行时间秒长

## 第3集 进阶知识多进程之child\_process模块

简介：讲解如何使用child\_process创建子进程及常用操作

- 创建子进程的4个方式
  - `child_process.spawn()` 启动一个子进程执行命令

```
const work = spawn('node', ['work.js'])
work.stdout.on('data', (data) => {
  console.log(`stdout: ${data}`);
});
```

- `child_process.exec()` 启动一个子进程执行命令，与`spawn`不同地方的是有一个回调函数获知子进程状况

```
exec('node work.js', (err, stdout, stderr) => {
  if(err) throw err
  console.log(stdout)
})
```

- `child_process.execFile()` 启动一个子进程来执行可执行文件

```
execFile('node', ['work.js'], (err, stdout, stderr) => {
  if(err) throw err
  console.log(stdout)
  console.log(stderr)
})
```

- `child_process.fork()` 衍生一个新的 Node.js 进程，并调用一个指定的模块

```
fork('work.js')
```

## 第4集 深度讲解cluster模块

简介：讲解cluster模块的使用

- cluster创建子进程

```
const cluster = require('cluster');
const http = require('http');
const numCPUs = require('os').cpus().length;

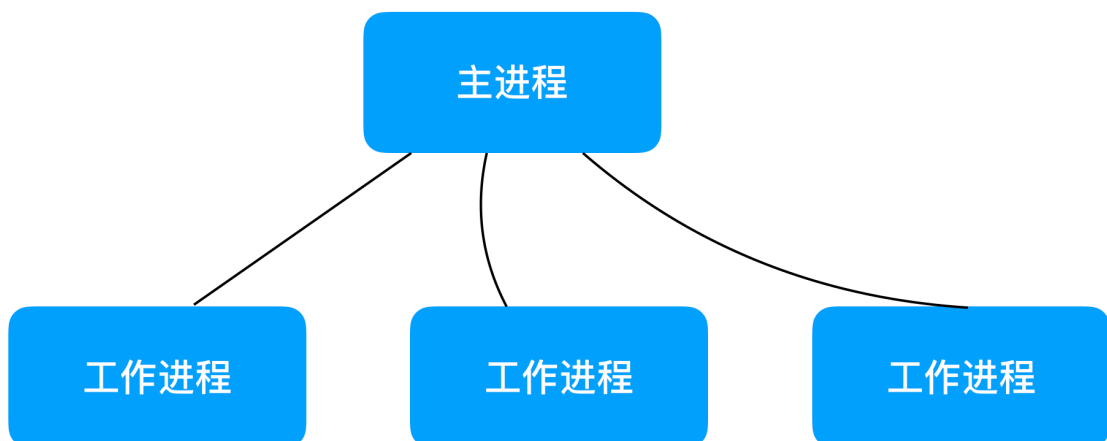
if (cluster.isMaster) {
  console.log(`主进程 ${process.pid} 正在运行`);

  // 衍生工作进程。
  for (let i = 0; i < numCPUs; i++) {
    cluster.fork();
  }

  cluster.on('exit', (worker, code, signal) => {
    console.log(`工作进程 ${worker.process.pid} 已退出`);
  });
} else {
  // 工作进程可以共享任何 TCP 连接。
  // 在本例子中，共享的是 HTTP 服务器。
  http.createServer((req, res) => {
    res.writeHead(200);
    res.end('你好世界\n');
  }).listen(8000);

  console.log(`工作进程 ${process.pid} 已启动`);
}
```

- cluster主从模型





## 第5集 深度讲解进程间的通信

简介：讲解进程间的通信方式

- 子进程通过send方法讲结果发送给主进程，主进程通过message监听到信息后处理并退出

```
//父进程index.js文件
const http = require('http')
const { spawn, exec, execFile, fork } = require('child_process')

const server = http.createServer((req, res) => {
  if (req.url === '/compute') {
    const work = fork('work.js')
    work.send('发送给子进程的信息')
    work.on('message', data => {
      console.log('父进程接收子进程的消息: ' + data)
    })
  } else {
    res.end('ok')
  }
})

server.listen(3000, () => {
  console.log('listen success')
})

//子进程work.js
process.title = '计算'

let sum = 0;
for(let i = 0;i<1e10;i++){
  sum += i
}

console.log(sum)

console.error('出错')

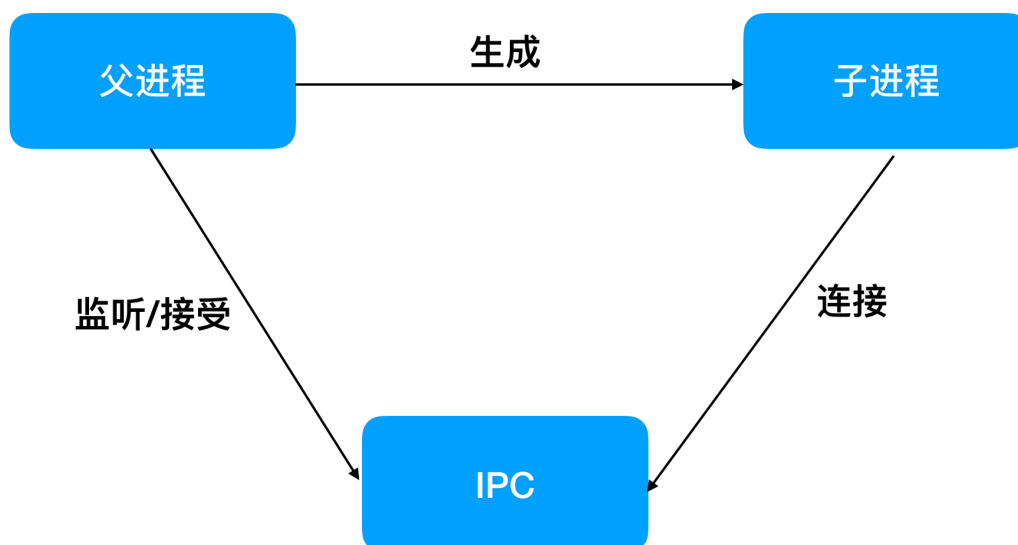
process.on('message', (data)=>{
  console.log('接收父进程的消息:'+data)
})

process.send(sum)
```

- 进程间通信原理

IPC的全称是Inter-Process Communication,即进程间通信。它的目的是为了让不同的进程能够互相访问资源并进行协调工作。

- 创建IPC管道的步骤图



父进程在实际创建子进程之前，会创建IPC通道并监听它，然后才真正的创建出子进程。此过程中会通过环境变量（NODE\_CHANNEL\_FD）告诉子进程这个IPC通道的文件描述符。子进程在启动的过程中，根据文件描述符连接这个IPC通道，从而完成父子进程之间的连接。

## 第八章 express框架知识讲解

### 第1集 express框架介绍

简介：介绍express框架知识

- express是什么？  
它是基于 [Node.js](https://nodejs.org/) 平台，快速、开放、极简的 Web 开发框架。
- express框架特性
  - 可以设置中间件来响应 HTTP 请求。
  - 提供方便的路由定义方式
  - 可以通过模板引擎动态渲染 HTML 页面
  - 简化获取http请求参数的方式
- express安装

```
npm install express --save
```

- express创建服务器  
//send方法会自动设置http状态码及相应的内容类型及编码

```
const express = require('express')

//创建服务器
const app = express()

app.get('/', (req, res) => {
  res.send('hello express')
})

//监听端口
app.listen(3000)
console.log('服务器已经启动')
```

## 第2集 深度讲解express处理get/post请求

简介：讲解express框架处理get/post请求，并获取请求参数

- express使用req.query获取get请求参数

```
app.get('/index', (req, res) => {  
  res.send(req.query)  
})
```

- express接收post请求参数需要使用第三方包body-parser
  - 安装body-parser

```
npm install body-parser
```

- post请求通过req.body获取参数

```
//拦截所有请求  
//extended为false表示用querystring解析字符串参数  
//extended为true表示用qs解析字符串参数  
app.use(bodyParser.urlencoded({ extended: false })))  
  
app.post('/add', (req, res) => {  
  res.send(req.body)  
})
```

## 第3集 介绍与剖析中间件

简介：介绍什么是中间件

- 中间件是Express 框架中的一个重要概念。中间件（Middleware）是一个函数，它可以访问请求对象 `req`，响应对象 `res`，和 web 应用中处于请求-响应循环流程中的中间件，一般被命名为 `next` 的变量。

```
//一个http请求可以有多个中间件,通过next可以交给下一个中间件进行处理
app.get('/index', (req, res, next) => {
  req.name = 'eric'
  next()
})

app.get('/index', (req, res) => {
  res.send(req.name)
})
```

## 第4集 深度讲解app.use中间件用法

简介：讲解app.use是做什么的以及用法

- `app.use` 加载用于处理http请求的middleware（中间件），当一个请求来的时候，会依次被这些middlewares处理。

```
/*表示匹配任何路由*/
app.use((req, res, next) => {
  console.log('这是第一个中间件')
  next()
})

//匹配所有/index的请求,无论是get或者post
app.use('/index', (req, res, next) => {
  console.log('这是第二个中间件')
  next()
})
```

## 第5集 讲解中间件常用的应用场景

简介：讲解中间件的一些应用场景

- 路由保护：客户端访问需要登录的请求时，可以使用中间件进行拦截，判断用户的登录状态，进而响应用户是否允许访问

```
//登录拦截
app.use((req,res,next)=>{
  let isLogin = true
  if(isLogin){
    next()
  }else{
    res.send('你需要登录后才可访问')
  }
})
```

- 网站维护：在所有中间件上定义一个接收所有请求的中间件，不使用next，直接给客户端响应表示网站维护中

```
app.use((req,res,next)=>{
  res.send('网站正在维护中')
})
```

- 自定义404页面

```
//自定义404
app.use((req,res)=>{
  res.status(404).send('404 NOT FOUND')
})
```

## 第6集 案例之处理错误的中间件

简介：讲解如何写一个处理错误的中间件

- 错误处理中间件

```
//错误处理中间件
app.use((err, req, res, next) => {
  res.status(500).send(err.message)
})
```

- 手动触发错误处理中间件，通过next方法，把错误信息通过参数传给next方法即可

```
app.get('/index', (req, res, next) => {
  fs.readFile('./index.js', 'utf8', (err, result) => {
    if (err) {
      next(err)
      return;
    } else {
      res.send(result)
    }
  })
})
```

## 第7集 讲解router构建模块化路由

简介：如何构建模块化路由

- 通过express的Router方法构建模块化路由

```
const express = require('express')
//创建服务器
```

```

const app = express()
//创建路由对象
const user = express.Router()

//将路由对象和请求地址进行匹配
app.use('/user',user)

//创建二级路由
user.get('/list',(req,res)=>{
  res.send('访问用户列表')
})

//监听端口
app.listen(3000)
console.log('服务器已经启动')

```

- 拆分路由模块

User.js

```

const express = require('express')
//创建用户路由对象
const user = express.Router()

//创建二级路由
user.get('/list',(req,res)=>{
  res.send('访问用户列表')
})
user.get('/index',(req,res)=>{
  res.send('访问用户首页')
})

module.exports = user

```

Blog.js

```

const express = require('express')
//创建博客路由对象
const blog = express.Router()

//创建二级路由
blog.get('/index',(req,res)=>{
  res.send('访问博客首页')
})

module.exports = blog

```

Index.js



```
const express = require('express')
//创建服务器
const app = express()
const user = require('./route/user')
const blog = require('./route/blog')

//将路由对象与请求地址匹配
app.use('/user',user)
app.use('/blog',blog)

//监听端口
app.listen(3000)
console.log('服务器已经启动')
```

## 第8集 讲解express框架访问静态资源

简介：如何访问静态资源

- 使用express.static方法可以管理静态文件，例如img、css、js文件等

```
const express = require('express')
const path = require('path')
const app = express()
// path.join(__dirname, 'public') 表示工程路径后面追加 public
app.use(express.static(path.join(__dirname, 'public')))

app.listen(8080, () => {
  console.log(`App listening at port 8080`)
})
```

## 第9集 讲解express模板引擎

简介：讲解express中template模板引擎

- 安装命令

```
npm install art-template express-art-template
```

- 模板引擎设置

```
//使用express-art-template渲染后缀为art的模板
app.engine('art', require('express-art-template'))
//设置模板的存放目录
app.set('views', path.join(__dirname, 'views'))
//渲染模板时默认拼接art后缀
app.set('view engine', 'art')

//渲染模板
res.render(模板名称, 模板数据)
```

- 案例代码

```
const express = require('express')
const path = require('path')

//创建服务器
```

```
const app = express()

//使用express-art-template渲染后缀为art的模板
app.engine('art',require('express-art-template'))
//设置模板的存放目录
app.set('views',path.join(__dirname,'views'))
//渲染模板时默认拼接art后缀
app.set('view engine','art')

app.get('/index',(req,res)=>{
  res.render('index',{
    msg:'访问首页'
  })
})

//监听端口
app.listen(3000)
console.log('服务器已经启动')
```



愿景："让编程不在难学，让技术与生活更加有趣"

更多课程请访问[xdclass.net](http://xdclass.net)

## 第九章 开发简易博客系统前端篇之项目搭建

---

# 第1集 博客系统核心业务需求分析

简介：分析博客系统的需求业务

- 登录注册
- 用户信息更新
- 博客列表
- 博客管理（查看、编辑、新增、删除）
- 博客详情
- 评论
- 用户头像上传

# 第2集 项目初始化及前端框架搭建

简介：讲解初始化项目及vue前端框架的搭建

- 安装vue

```
npm install -g @vue/cli
```

- 检查vue安装是否成功

```
vue --version
```

- 创建项目

```
vue create project(项目名称, 自定义)
```

- vue开发常用插件

```
eslint、Prettier Code formatter、vetur、vue 2 snippets、vue vscode snippets
```

## 第3集 路由设计及公共布局实现(一)

## 第4集 路由设计及公共布局实现(二)

简介：讲解公共顶部导航栏的菜单和底部编写

- 安装elementui

```
npm i element-ui -S
```

- 在main.js中引入elementui

```
import ElementUI from 'element-ui'
import 'element-ui/lib/theme-chalk/index.css'

Vue.use(ElementUI)
```

- 公共路由设计

```
const routes = [
  {
    path: "/",
    component: () => import('@components/CommonLayout.vue')
  }
];
```

- 公共顶部

```
<header>
  <div class="wrapper">
    <el-row>
      <el-col :span="4">
        <div class="logo">小滴博客</div>
      </el-col>
      <el-col :span="20">
```

```

        <el-menu mode="horizontal"
          class="nav"
          :default-active="activeIndex"
          @select="hanleSelect"
          background-color="#2d2d2d"
          text-color="#9d9d9d"
          active-text-color="#fff"
        >
          <el-menu-item index="1">
            <router-link to="/"><i class="iconfont icon-home"></i>首页</router-link>
          </el-menu-item>
          <el-menu-item index="2">
            <router-link to="/blog">我的博客</router-link>
          </el-menu-item>
          <el-menu-item index="3">
            <router-link class="signBtn" to="/">登
录</router-link>
          </el-menu-item>
        </el-menu>

      </el-col>
    </el-row>

  </div>
</header>

```

- 公共底部

```

<footer>
  <div class="wrapper">
    Copyright © 小滴
  </div>
</footer>

```

- 公共的布局组件中引入顶部和底部组件

```

<template>
  <div class="container">
    <common-header></common-header>
    <div>中间内容</div>
    <common-footer></common-footer>
  </div>
</template>

<script>
import CommonHeader from '@components/CommonHeader.vue'
import CommonFooter from '@components/CommonFooter.vue'

```

```
export default {
  components:{
    CommonHeader,
    CommonFooter
  }
};
</script>
```

## 第5集 封装列表组件实现首页博客列表布局

简介：讲解列表组件的布局及实现

- bloglist.vue

```
<div class="list">
  <div class="card">
    <router-link :to="'/detail/'+1">
      <p class="title">nodejs教程</p>
    </router-link>
    <p class="date">2020-03-20</p>
  </div>
  <div class="card">
    <router-link :to="'/detail/'+1">
      <p class="title">nodejs教程</p>
    </router-link>
    <p class="date">2020-03-20</p>
  </div>
  <div class="card">
    <router-link :to="'/detail/'+1">
      <p class="title">nodejs教程</p>
    </router-link>
    <p class="date">2020-03-20</p>
  </div>
</div>
```

- Home.vue 首页

```
<div class="home">
  <div class="wrapper">
    <div class="main">
      <blog-list></blog-list>
    </div>
  </div>
</div>
```

## 第6集 讲解博客系统详情页布局

简介：讲解博客详情页的布局

- 安装mavon-editor编辑器

```
npm install mavon-editor --save
```

- Main.js中引入编辑器

```
import mavonEditor from 'mavon-editor'
import 'mavon-editor/dist/css/index.css'

Vue.use(mavonEditor)
```

- 编辑器设置

```
<mavon-editor v-model="content"
              defaultOpen="preview"
              :toolbarsFlag="false"
              :subfield="false"></mavon-editor>
```



name 名称	type 类型	default 默认值	describe 描述
defaultOpen	String		edit: 默认展示编辑区域, preview: 默认展示预览区域, 其他 = edit
toolbarsFlag	Boolean	true	工具栏是否显示
subfield	Boolean	true	true: 双栏(编辑预览同屏), false: 单栏(编辑预览分屏)

## 第7集 讲解博客系统详情页评论模块设计（一）

## 第8集 讲解博客系统详情页评论模块设计（二）

简介：讲解登录状态以及未登录状态下评论模块的布局

- 根据登录状态判断是否显示评论框

```
<div v-if="isSignIn===0" class="signInText">登录留言吧</div>
<div v-else class="input-box">
  <div class="input-top">
    <div class="img">
      <img class="avatar" src="" alt="">
      <p class="username">hahaha</p>
    </div>
    <div class="text">
      <textarea class="comment-content" v-model="submitText">
    </div>
  </div>
  <div class="input-bottom">
    <a href="javascript:void(0)" class="submit">发表评论</a>
```

```
</div>  
</div>
```



愿景："让编程不在难学，让技术与生活更加有趣"

更多课程请访问[xdclass.net](http://xdclass.net)

## 第十章 开发简易博客系统前端篇之登录模块

### 第1集 博客实战之登录注册模块页面设计

简介：讲解博客系统的登录注册页面布局

- 输入框验证

```
loginRules:{  
    name:[  
        { required: true, message: '请输入账号', trigger:  
'blur' },  
    ],  
    password:[  
        { required: true, message: '请输入密码', trigger:  
'blur' },  
    ],  
}
```

- 自定义验证

```

var validatePass = (rule, value, callback) => {
    if (value === '') {
        callback(new Error('请输入密码'));
    } else {
        if (this.regForm.checkPass !== '') {
            this.$refs.regForm.validateField('checkPass');
        }
        callback();
    }
};

var validatePass2 = (rule, value, callback) => {
    if (value === '') {
        callback(new Error('请再次输入密码'));
    } else if (value !== this.regForm.password) {
        callback(new Error('两次输入密码不一致!'));
    } else {
        callback();
    }
};

regRules:{
    name:[
        { required: true, message: '请输入账号', trigger:
'blur' },
    ],
    password: [
        { validator: validatePass, trigger: 'blur' }
    ],
    checkPass: [
        { validator: validatePass2, trigger: 'blur' }
    ],
}

```

## 第2集 讲解模块登录态下的展示

简介：讲解不同登录状态下页面模块的展示

- 全屏加载动画

```
<div v-loading.fullscreen.lock="loading"></div>
```

- 利用vuex存储登录态

```
state: {  
  isSignIn: 0 // 0未登录, 1登录  
},  
mutations: {  
  changeIsSignIn (state, n) {  
    state.isSignIn = n  
  }  
}
```

## 第3集 博客实战之用户个人信息页面布局

简介：讲解登录后用户个人页面的布局

- 头像上传

```
<el-upload  
  class="avatar-uploader"  
  action=""  
  :show-file-list="false"  
  :on-success="handleAvatarSuccess"  
  :before-upload="beforeAvatarUpload">  
    
</el-upload>
```



愿景："让编程不在难学，让技术与生活更加有趣"

更多课程请访问[xdclass.net](http://xdclass.net)

## 第十一章 开发简易博客系统前端篇之博客管理

### 第1集 博客管理页面布局设计

简介：讲解博客管理页面的布局

- 表格数据

```
articleList:[{
  id:1,
  title:'hhhh1',
  create_time:'2020-02-16'
},
{
  id:2,
  title:'nodejs教程',
  create_time:'2020-02-17'
},
{
  id:3,
  title:'express教程',
  create_time:'2020-02-18'
}]
```

- 通过插槽自定义表格

```
<el-table-column
  label="日期"
  width="180">
  <template slot-scope="scope">
    <i class="el-icon-time"></i>
    <span>{{ scope.row.create_time }}</span>
  </template>
</el-table-column>
```

## 第2集 核心业务之博客管理功能（上）

简介：讲解博客新增及编辑功能实现及markdown编辑器引入

- Maven-editor文档使用

<https://www.npmjs.com/package/mavon-editor>

## 第3集 核心业务之博客管理功能（下）

简介：讲解博客删除功能弹窗提示

```
this.$confirm('此操作将永久删除该文章，是否继续?', '提示', {
  confirmButtonText: '确定',
  cancelButtonText: '取消',
  type: 'warning'
})
.then(() => {
  //请求删除api todo
})
.catch(() => {
  this.$message({
```

```
    type: 'info',  
    message: '已取消删除'  
  })  
})
```



愿景："让编程不在难学，让技术与生活更加有趣"

更多课程请访问[xdclass.net](http://xdclass.net)

## 第十二章 开发简易博客系统服务端篇之框架搭建

### 第1集 搭建express框架

简介：讲解安装express框架，搭建项目

- Node.js 8.2.0版本以上

```
npx express-generator
```

- req.app.get('env')相当于process.env.NODE\_ENV
- mogan文档地址

```
https://www.npmjs.com/package/morgan
```

## 第2集 博客系统后台mysql数据库设计

简介：讲解如何设计博客的数据库

- 用户表设计
  - 用户id
  - 用户账号
  - 用户密码
  - 用户昵称
  - 用户头像
- 文章表设计
  - 文章id
  - 文章标题
  - 创建时间
  - 文章内容
  - 作者用户id
  - 删除状态
- 评论表设计
  - 评论id
  - 评论内容
  - 用户id
  - 文章id
  - 创建时间
  - 用户头像



愿景："让编程不在难学，让技术与生活更加有趣"

更多课程请访问[xdclass.net](http://xdclass.net)

## 第十三章 开发简易博客系统服务端篇之登录注册

---

### 第1集 注册接口开发讲解



简介：讲解注册接口的编写和对接

- 安装跨域中间件cors

```
npm install cors --save
```

- md5密码加密
  - 安装crypto

```
npm install crypto --save
```

- 加密方法

```
function md5(s){  
    //注意参数需要为string类型，否则会报错  
    return crypto.createHash('md5').update(String(s)).digest('hex');  
}
```

## 第2集 登录实战jwt加密生成token讲解

简介：讲解jwt生成token开发登录接口

- 什么是token

Token是服务端生成的一串字符串，作为客户端进行请求的一个令牌。当用户登录后，服务器生成一个Token返回给客户端，之后客户端只需带上这个Token来请求数据即可，无需每次都输入用户名和密码来鉴权。

- token的组成
  - Header

```
{  
    type: "jwt",  
    alg: "HS256"  
}
```

- Payload

```
iss (issuer): 签发人
exp (expiration time): 过期时间
sub (subject): 主题
aud (audience): 受众
nbf (Not Before): 生效时间
iat (Issued At): 签发时间
jti (JWT ID): 编号
```

//除上面信息外,可自定义其他私有字段,比如一些用户信息,但要注意敏感的信息不要存进来

- Signature (**secret**)

```
HMACSHA256(
  base64UrlEncode(header) + "." +
  base64UrlEncode(payload),
  secret)
```

- 安装jsonwebtoken

```
npm install jsonwebtoken -S
```

## 第3集 登录拦截jwt解密校验

简介: 讲解token的解密及校验

- 安装express-jwt

```
npm install express-jwt
```

- 校验token, 获取headers里的Authorization的token

前端token值格式 Bearer token

```
app.use(expressJWT({
  secret: PRIVATE_KEY
}).unless({
  path: [ '/api/user/register', '/api/user/login' ] //白名单,除了这里写的地址,其他的URL都需要验证
}));
```

- 中间件错误处理

```
if (err.name === 'UnauthorizedError') {
  // 这个需要根据自己的业务逻辑来处理
  res.status(401).send({code:-1,msg:'token验证失败'});
}
```

- 前端请求拦截

```
import axios from 'axios'
import store from './store/index'
axios.defaults.baseURL = 'http://127.0.0.1:3000/';

export default function setAxios(){
  //请求拦截
  axios.interceptors.request.use(
    config=>{
      if(store.state.token){
        config.headers['Authorization']= `Bearer
${store.state.token}`
      }
      return config
    }
  )
}
```

## 第4集 讲解前端cookie缓存修改登录态

简介：缓存接口返回的token根据登录状态修改页面展示

- 安装js-cookie

```
npm install js-cookie --save
```

- 路由守卫

```
router.beforeEach((to, from, next) => {
  store.commit('setToken', Cookie.get('token'))
  if (store.state.token) {
    store.commit('changIsSignIn', 1) // 设置登录态
  }
  // 根据登录权限判断
  if (to.meta.requireAuth) {
    if (store.state.token) {
      next()
    } else {
      next({ path: '/login' })
    }
  } else {
    next()
  }
})
```

## 第5集 讲解获取用户信息接口

简介：讲解用户信息接口的编写

- 获取用户信息接口

```
//获取用户信息接口
router.get('/info', async(req, res, next) => {
  let {username} = req.user
  try {
    let userinfo = await querySql('select * from user where username = ?',
[username])
    res.send({code:0,msg:'成功',data:userinfo[0]})
  }catch(e){
    console.log(e)
    next(e)
  }
})
```

## 第6集 讲解用户头像上传接口开发

简介：讲解如何开发处理用户头像上传

- 安装multer

```
npm install --save multer
```

- 配置multer

```
let upload = multer({
  storage: multer.diskStorage({
    // 设置文件存储位置
    destination: function (req, file, cb) {
      let date = new Date()
      let year = date.getFullYear()
      let month = (date.getMonth() + 1).toString().padStart(2, '0')
      let day = date.getDate()
      let dir = path.join(__dirname, '../public/uploads/' + year + month
+ day)

      // 判断目录是否存在，没有则创建
      if (!fs.existsSync(dir)) {
        fs.mkdirSync(dir, {recursive: true})
      }

      // dir就是上传文件存放的目录
    }
  })
})
```

```

        cb(null, dir)
    },
    // 设置文件名称
    filename: function (req, file, cb) {
        let fileName = Date.now() + path.extname(file.originalname)
        // fileName就是上传文件的文件名
        cb(null, fileName)
    }
})
})

```

- 上传头像接口

```

//头像上传接口
router.post('/upload',upload.single('head_img'),async(req,res,next) => {
    console.log(req.file)
    let imgPath = req.file.path.split('public')[1]
    let imgUrl = 'http://127.0.0.1:3000'+imgPath
    res.send({code:0,msg:'上传成功',data:imgUrl})
})

```

## 第7集 讲解用户信息更新接口开发

简介：讲解更新用户信息接口开发

- 用户信息更新接口

```
router.post('/updateUser', async(req, res, next) => {
  let {nickname, head_img} = req.body
  let {username} = req.user
  try {
    let result = await querySql('update user set nickname = ?,head_img = ?
where username = ?', [nickname, head_img, username])
    console.log(result)
    res.send({code:0,msg:'更新成功',data:null})
  }catch(e){
    console.log(e)
    next(e)
  }
})
```



愿景："让编程不在难学，让技术与生活更加有趣"

更多课程请访问[xdclass.net](http://xdclass.net)

## 第十四章 开发简易博客系统服务端篇之博客管理

### 第1集 博客核心业务之博客接口开发（增）

简介：讲解添加博客接口的开发及对接

- 新增博客接口代码

```

/* 新增博客接口 */
router.post('/add', async(req, res, next) => {
  let {title,content} = req.body
  let {username} = req.user
  try {
    let result = await querySql('select id from user where username = ?',
[username])
    let user_id = result[0].id
    await querySql('insert into article(title,content,user_id,create_time)
values(?,?,?,NOW())',[title,content,user_id])
    res.send({code:0,msg:'新增成功',data:null})
  }catch(e){
    console.log(e)
    next(e)
  }
});

```

## 第2集 博客核心业务之博客接口开发（查）

简介：讲解博客查询接口的开发及对接

- 转换时间格式

```
DATE_FORMAT(create_time,"%Y-%m-%d %H:%i:%s") AS create_time
```

- 所有博客列表接口开发



```
// 获取全部博客列表接口
router.get('/allList', async(req, res, next) => {
  try {
    let sql = 'select id,title,content,DATE_FORMAT(create_time,"%Y-%m-%d
%H:%i:%s") AS create_time from article'
    let result = await querySql(sql)
    res.send({code:0,msg:'获取成功',data:result})
  }catch(e){
    console.log(e)
    next(e)
  }
});
```

- 我的博客列表开发

```
// 获取我的博客列表接口
router.get('/myList', async(req, res, next) => {
  let {username} = req.user
  try {
    let userSql = 'select id from user where username = ?'
    let user = await querySql(userSql,[username])
    let user_id = user[0].id
    let sql = 'select id,title,content,DATE_FORMAT(create_time,"%Y-%m-%d
%H:%i:%s") AS create_time from article where user_id = ?'
    let result = await querySql(sql,[user_id])
    res.send({code:0,msg:'获取成功',data:result})
  }catch(e){
    console.log(e)
    next(e)
  }
});
```

## 第3集 博客核心业务之博客详情接口开发（查）

简介：讲解博客详情接口的开发及对接

- 博客详情接口开发

```
// 获取博客详情接口
router.get('/detail', async(req, res, next) => {
  let article_id = req.query.article_id
  try {
    let sql = 'select id,title,content,DATE_FORMAT(create_time,"%Y-%m-%d
%H:%i:%s") AS create_time from article where id = ?'
    let result = await querySql(sql,[article_id])
    res.send({code:0,msg:'获取成功',data:result[0]})
  }catch(e){
    console.log(e)
    next(e)
  }
});
```

## 第4集 博客核心业务之博客接口开发（改）

简介：讲解博客修改更新接口的开发及对接

- 博客修改接口代码

```
// 获取博客详情接口
router.post('/update', async(req, res, next) => {
  let {article_id,title,content} = req.body
  try {
    let sql = 'update article set title = ?,content = ? where id = ?'
    let result = await querySql(sql,[title,content,article_id])
    res.send({code:0,msg:'更新成功',data:null})
  }catch(e){
    console.log(e)
    next(e)
  }
});
```

## 第5集 博客核心业务之博客接口开发（删）

简介：讲解博客删除接口的开发及对接

- 博客删除接口代码

```
// 删除博客接口
router.post('/delete', async(req, res, next) => {
  let {article_id} = req.body
  let {username} = req.user
  try {
    let userSql = 'select id from user where username = ?'
    let user = await querySql(userSql,[username])
    let user_id = user[0].id
    let sql = 'delete from article where id = ? and user_id = ?'
    let result = await querySql(sql,[article_id,user_id])
    res.send({code:0,msg:'删除成功',data:null})
  }catch(e){
    console.log(e)
    next(e)
  }
});
```

## 第6集 博客核心业务之评论接口开发

简介：讲解新增评论接口的开发及对接

```
//发表评论接口
router.post('/public', async(req, res, next) => {
  let {content, article_id} = req.body
  let {username} = req.user
  try {
    let userSql = 'select id, head_img, nickname from user where username = ?'
    let user = await querySql(userSql, [username])
    let {id: user_id, head_img, nickname} = user[0]
    let sql = 'insert into
comment(user_id, article_id, cm_content, nickname, head_img, create_time)
values(?, ?, ?, ?, ?, NOW())'
    let result = await querySql(sql,
[user_id, article_id, content, nickname, head_img])
    res.send({code: 0, msg: '发表成功', data: null})
  } catch (e) {
    console.log(e)
    next(e)
  }
})
```

## 第7集 博客核心业务之评论列表数据

简介：讲解博客详情评论列表接口

```
//评论列表接口
router.get('/list', async(req, res, next) => {
  let {article_id} = req.query
  try {
    let sql = 'select
id,head_img,nickname,cm_content,DATE_FORMAT(create_time,"%Y-%m-%d %H:%i:%s")
AS create_time from comment where article_id = ?'
    let result = await querySql(sql,[article_id])
    res.send({code:0,msg:'成功',data:result})
  }catch(e){
    console.log(e)
    next(e)
  }
})
})
```



愿景："让编程不在难学，让技术与生活更加有趣"

更多课程请访问[xdclass.net](http://xdclass.net)

## 第十五章 项目上线与部署

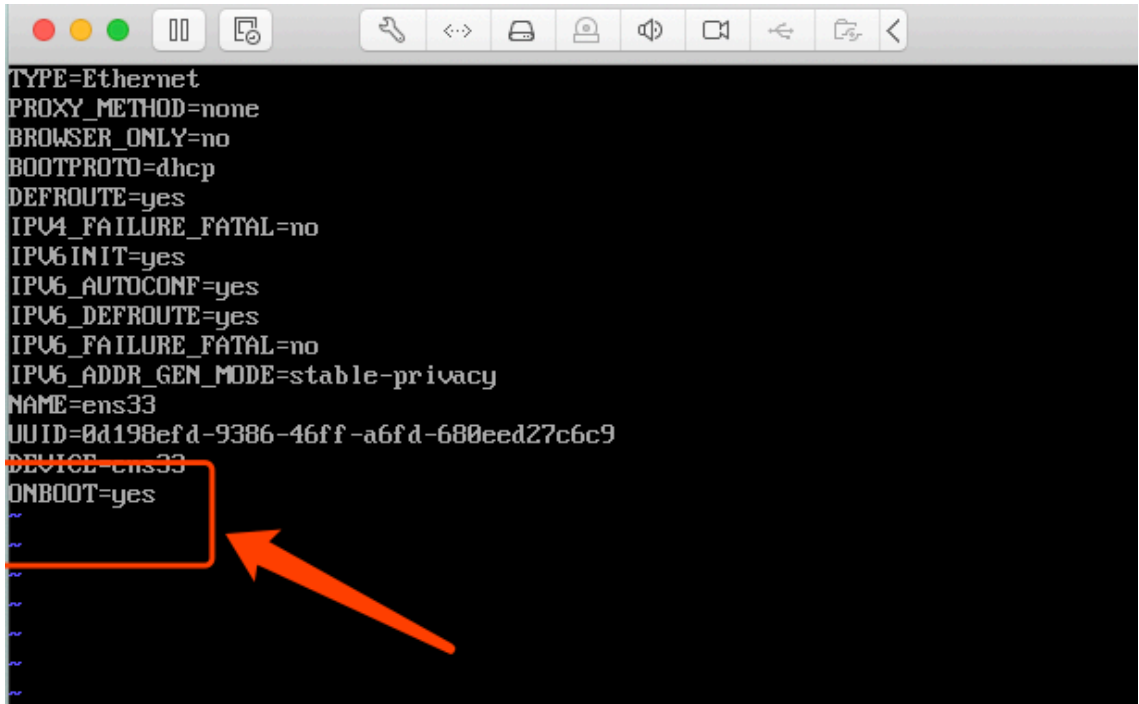
### 第1集 定制linux虚拟机服务器

简介：讲解如何搭建自己的linux的服务器

- 安装linux
- 查看linux服务器ip
  - 输入 ip addr 查看服务器ip，第一次查看ens33没有net属性

```
[root@localhost local]# ip addr
1: lo: <LOOPBACK,UP,LOWER_UP> mtu 65536 qdisc noqueue state UNKNOWN group default qlen 1000
    link/loopback 00:00:00:00:00:00 brd 00:00:00:00:00:00
    inet 127.0.0.1/8 scope host lo
        valid_lft forever preferred_lft forever
    inet6 ::1/128 scope host
        valid_lft forever preferred_lft forever
2: ens33: <BROADCAST,MULTICAST,UP,LOWER_UP> mtu 1500 qdisc pfifo_fast state UP group default qlen 1000
    link/ether 00:0c:29:47:ae:d2 brd ff:ff:ff:ff:ff:ff
    inet 172.16.11.129/24 scope global dynamic noprefixroute ens33
        valid_lft 1704sec preferred_lft 1704sec
    inet6 fe80::7243:516c:573b:c36f/64 scope link noprefixroute
        valid_lft forever preferred_lft forever
```

- 查看一下ens33配置，用vi /etc/sysconfig/network-scripts/ifcfg-ens33命令查看，把下图ONBOOT部分改成yes（输入a进去编辑模式，按esc退出编辑，再次输入:wq保存并退出）



```
TYPE=Ethernet
PROXY_METHOD=none
BROWSER_ONLY=no
BOOTPROTO=dhcp
DEFROUTE=yes
IPV4_FAILURE_FATAL=no
IPV6INIT=yes
IPV6_AUTOCONF=yes
IPV6_DEFROUTE=yes
IPV6_FAILURE_FATAL=no
IPV6_ADDR_GEN_MODE=stable-privacy
NAME=ens33
UUID=0d198efd-9386-46ff-a6fd-680eed27c6c9
DEVICE=ens33
ONBOOT=yes
```

- 修改完成后启动网卡,输入service network start

```
[root@localhost local]# service network start
Starting network (via systemctl): [ OK ]
[root@localhost local]#
```

- 再次输入ip addr可看到ens33有ip地址

```
[root@localhost local]# ip addr
1: lo: <LOOPBACK,UP,LOWER_UP> mtu 65536 qdisc noqueue state UNKNOWN group default qlen 1000
    link/loopback 00:00:00:00:00:00 brd 00:00:00:00:00:00
    inet 127.0.0.1/8 scope host lo
        valid_lft forever preferred_lft forever
    inet6 ::1/128 scope host
        valid_lft forever preferred_lft forever
2: ens33: <BROADCAST,MULTICAST,UP,LOWER_UP> mtu 1500 qdisc pfifo_fast state UP group default qlen 1000
    link/ether 00:0c:29:47:ae:d2 brd ff:ff:ff:ff:ff:ff
    inet 172.16.11.129/24 scope global dynamic noprefixroute ens33
        valid_lft 1704sec preferred_lft 1704sec
    inet6 fe80::7243:516c:573b:c36f/64 scope link noprefixroute
        valid_lft forever preferred_lft forever
```

- 远程连接服务器
  - mac系统使用终端输入ssh root@服务器IP地址，输入服务器密码连接服务器
  - windows用户可用xshell连接服务器
- 连接服务器出现以下错误

```
bash
Last login: Sun Mar  8 17:23:17 on ttys000
ericdeMacBook-Pro:~ ericxie$ ssh root@172.16.11.130
@@
@    WARNING: REMOTE HOST IDENTIFICATION HAS CHANGED!    @
@@
IT IS POSSIBLE THAT SOMEONE IS DOING SOMETHING NASTY!
Someone could be eavesdropping on you right now (man-in-the-middle attack)!
It is also possible that a host key has just been changed.
The fingerprint for the ECDSA key sent by the remote host is
SHA256:1iZGE4UrS4Nr8p+fM7YmgPx5v3J9IX1Tp9WtHU6gQLo.
Please contact your system administrator.
Add correct host key in /Users/ericxie/.ssh/known_hosts to get rid of this messa
ge.
Offending ECDSA key in /Users/ericxie/.ssh/known_hosts:4
ECDSA host key for 172.16.11.130 has changed and you have requested strict check
ing.
Host key verification failed.
ericdeMacBook-Pro:~ ericxie$
```

```
ssh-keygen -R XX.XX.XX.XX 清除ip
```

## 第2集 预备知识之linux服务器node环境安装

简介：讲解如何在linux服务器上安装node环境

- 安装nodejs
  - 去淘宝镜像复制linux版本的nodejs下载链接

## Mirror index of <https://nodejs.org/dist/v13.0.0/>

../	10-Oct-2019 10:30	-
docs/	10-Oct-2019 10:19	-
win-x64/	10-Oct-2019 10:22	-
node-v13.0.0-aix-ppc64.tar.gz	22-Oct-2019 14:38	40731606 (38.84MB)
node-v13.0.0-darwin-x64.tar.gz	22-Oct-2019 15:47	28522538 (27.2MB)
node-v13.0.0-darwin-x64.tar.xz	22-Oct-2019 15:48	18451564 (17.6MB)
node-v13.0.0-headers.tar.gz	22-Oct-2019 14:31	552478 (539.53kB)
node-v13.0.0-headers.tar.xz	22-Oct-2019 14:31	363640 (355.12kB)
node-v13.0.0-linux-arm64.tar.gz	22-Oct-2019 13:49	31784870 (30.31MB)
node-v13.0.0-linux-arm64.tar.xz	22-Oct-2019 13:51	19519044 (18.61MB)
node-v13.0.0-linux-armv7l.tar.gz	22-Oct-2019 13:58	29839531 (28.46MB)
node-v13.0.0-linux-armv7l.tar.xz	22-Oct-2019 14:00	17967640 (17.14MB)
node-v13.0.0-linux-ppc64le.tar.gz	22-Oct-2019 13:54	31776891 (30.3MB)
node-v13.0.0-linux-ppc64le.tar.xz	22-Oct-2019 13:56	19790440 (18.87MB)
node-v13.0.0-linux-s390x.tar.gz	22-Oct-2019 13:54	32223472 (30.73MB)
node-v13.0.0-linux-s390x.tar.xz	22-Oct-2019 13:55	19690804 (18.78MB)
node-v13.0.0-linux-x64.tar.gz	22-Oct-2019 13:54	31734944 (30.26MB)
node-v13.0.0-linux-x64.tar.xz	22-Oct-2019 13:56	20111304 (19.18MB)
node-v13.0.0-sunos-x64.tar.gz	22-Oct-2019 14:22	35977414 (34.31MB)
node-v13.0.0-sunos-x64.tar.xz	22-Oct-2019 14:23	23290528 (22.21MB)
node-v13.0.0-win-x64.7z	22-Oct-2019 14:16	16301094 (15.55MB)
node-v13.0.0-win-x64.zip	22-Oct-2019 14:17	26634439 (25.4MB)
node-v13.0.0-win-x86.7z	22-Oct-2019 14:32	15249752 (14.54MB)
node-v13.0.0-win-x86.zip	22-Oct-2019 14:40	25164080 (24MB)
node-v13.0.0-x64.msi	22-Oct-2019 14:17	28336128 (27.02MB)
node-v13.0.0-x86.msi	22-Oct-2019 14:49	26771456 (25.53MB)
node-v13.0.0.pkg	22-Oct-2019 15:14	28784879 (27.45MB)
node-v13.0.0.tar.gz	22-Oct-2019 14:24	59466425 (56.71MB)
node-v13.0.0.tar.xz	22-Oct-2019 14:28	31977112 (30.5MB)
SHASUMS256.txt	22-Oct-2019 16:15	3121 (3.05kB)
SHASUMS256.txt.asc	22-Oct-2019 16:15	3658 (3.57kB)
SHASUMS256.txt.sig	22-Oct-2019 16:15	310 (310B)

### ○ 安装wget

```
yum -y install wget
```

```
ssh
Last login: Sat Mar 7 19:26:58 2020 from 172.16.11.1
[root@localhost ~]# wget https://dev.mysql.com/downloads/file/?id=484922
-bash: wget: 未找到命令
[root@localhost ~]# yum -y install wget
已加载插件: fastestmirror
Loading mirror speeds from cached hostfile
 * base: mirrors.aliyun.com
 * extras: mirrors.ustc.edu.cn
 * updates: mirrors.aliyun.com
base | 3.6 kB | 00:00
extras | 2.9 kB | 00:00
updates | 2.9 kB | 00:00
正在解决依赖关系
--> 正在检查事务
--> 软件包 wget.x86_64.0.1.14-18.el7_6.1 将被安装
--> 解决依赖关系完成

依赖关系解决

=====
Package          架构      版本                源                大小
-----
正在安装:
wget             x86_64    1.14-18.el7_6.1    base              547 k
```

### ○ 输入wget 复制的链接地址下载node

```
wget https://npm.taobao.org/mirrors/node/v12.16.1/node-v12.16.1-linux-x64.tar.xz
```

```
[root@localhost local]# wget https://npm.taobao.org/mirrors/node/v12.16.1/node-v12.16.1-linux-x64.tar.xz
```



- 解压下载node包

```
tar xvf node-v12.16.1-linux-x64.tar.xz
```

```
[root@localhost local]# tar xvf node-v12.16.1-linux-x64.tar.xz
```

- 修改文件夹名称

```
mv 文件夹名 修改后的文件夹名
```

```
[root@localhost local]# mv node-v12.16.1-linux-x64 node
```

- 创建软连接，就可以在任意目录下直接使用node和npm命令

```
ln -s /usr/local/node/bin/node /usr/local/bin/node  
ln -s /usr/local/node/bin/npm /usr/local/bin/npm
```

如果不小心输错了路径，重新创建会提示：‘ln: 无法创建符号链接"/usr/local/bin/npm": 文件已存在’，输入`rm /usr/local/bin/npm`命令清除后可以重新创建

```
[root@localhost local]# ln -s /usr/local/node/bin/node /usr/local/bin/node  
[root@localhost local]# ln -s /usr/local/node/bin/npm /usr/local/bin/npm
```

- 测试安装是否成功

```
node -v 查看node版本  
npm -v 查看npm版本
```

```
[root@localhost local]# node -v  
v12.16.1  
[root@localhost local]# npm -v  
6.13.4
```

## 第3集 预备知识之linux服务器mysql环境安装

简介：讲解如何在linux服务器上安装mysql环境

- 网易云上复制mysql压缩包地址，使用wget下载到服务器上

```
wget http://mirrors.163.com/mysql/Downloads/MySQL-8.0/mysql-8.0.17-linux-glibc2.12-x86_64.tar.xz
```

- 解压缩文件夹，重命名文件夹

```
tar xvf mysql-8.0.18-linux-glibc2.12-x86_64.tar.xz  
mv mysql-8.0.18-linux-glibc2.12-x86_64 mysql
```

- 进入到mysql根目录下创建data文件夹

```
mkdir data
```

- 创建mysql用户组和mysql用户

```
groupadd mysql  
useradd -g mysql mysql
```

- 改变mysql目录权限

```
chown -R mysql:mysql /usr/local/mysql/
```

- 修改配置

```
vi /etc/my.cnf  
  
[mysqld]  
basedir=/usr/local/mysql  
datadir=/usr/local/mysql/data  
socket=/tmp/mysql.sock  
  
[mysqld_safe]  
log-error=/usr/local/mysql/log/mysql.errlog  
pid-file=/usr/local/mysql/data/$hostname.pid
```

- 初始化数据库，进入到mysql的bin目录下执行以下语句
- 6jq+>k&Jkqit

```
./mysqld --initialize --user=mysql --basedir=/usr/local/mysql --  
datadir=/usr/local/mysql/data
```

- 配置mysql自启动服务

```
cp -a ./support-files/mysql.server /etc/init.d/mysql
```

```
chkconfig --add mysql 添加mysql服务
```

```
chkconfig --list mysql 检查mysql服务是否设置成功
```

- mysql启停

```
service mysql start
```

```
service mysql stop
```

```
service mysql status
```

- 创建软连接

```
ln -s /usr/local/mysql/bin/* /usr/bin/
```

- 登录mysql

```
mysql -uroot -p
```

- 修改密码

```
alter user 'root'@'localhost' identified by 'xdclass123';
```

```
flush privileges;刷新权限
```

```
quit;退出登录
```

- 开放防火墙端口

```
firewall-cmd --list-ports 查看已开放端口
```

```
firewall-cmd --zone=public --add-port=3306/tcp --permanent 开放3306端口
```

```
firewall-cmd --reload 重启
```

- 开启远程登录

```
mysql> update user set host='%' where user='root';
```

```
mysql> flush privileges;
```

- 出现连接密码格式不对



### Connection failed!

Unable to connect to host 172.16.11.130, or the request timed out.

Be sure that the address is correct and that you have the necessary privileges, or try increasing the connection timeout (currently 10 seconds).

MySQL said: Authentication plugin 'caching\_sha2\_password' cannot be loaded: dlopen(/usr/local/mysql/lib/plugin/caching\_sha2\_password.so, 2): image not found

OK

```
ALTER USER 'root'@'%' IDENTIFIED BY 'password' PASSWORD EXPIRE NEVER;  
#修改加密规则
```

```
ALTER USER 'root'@'%' IDENTIFIED WITH mysql_native_password BY  
'Xdclass123';  
#更新一下用户的密码
```

## 第4集 上线必备知识之pm2介绍和安装

简介：介绍和安装进程管理工具pm2

- pm2介绍

PM2 (process manager 2) 是具有内置负载均衡器的node.js应用程序的生产进程管理器。它能使你的程序永久保持活跃状态，无需停机即可重新加载它们，简化常见的系统管理任务。

- pm2特性

- 应用程序的日志管理

- 集群模式：Node.js负载均衡和零宕机时间重新加载

- 性能监控：可以在终端中监控您的应用程序并检查应用程序运行状况（CPU使用率，使用的内存等）

- 多平台支持：适用于Linux（稳定）和macOS（稳定）和Windows（稳定）。
- pm2安装

```
npm install pm2 -g
```

- pm2启动项目

```
pm2 start app.js
```

## 第5集 pm2的常用命令

简介：讲解pm2的常用命令

- 启动服务

```
pm2 start app.js  
pm2 start app.js --name app_name命名进程名称  
pm2 start app.js --watch --ignore-watch="node_modules"监听当文件变化时重启项目
```

- 停止服务

```
pm2 stop app_name|id|'all'
```

- 删除服务

```
pm2 delete app_name|id|'all'
```

- 重启服务

```
pm2 restart app_name|id|'all'
```

- 0秒停机重加载服务

```
pm2 reload app_name|id|'all'
```

- 查看进程信息

```
pm2 [list|ls|status] 查看进程状态  
pm2 describe app_name|id 查看进程所有信息
```

- 集群模式

```
pm2 start app.js -i 0根据cpu数目启动最大进程数目  
pm2 start app.js -i 3启动3个进程
```

- 日志

```
pm2 logs [app-name]查看日志  
pm2 flush 清空所有日志文件
```

## 第6集 深度讲解pm2的配置文件

简介：讲解pm2的配置文件

- 生成配置文件

```
pm2 ecosystem
```

- 配置文件

```
module.exports = {  
  apps : [{  
    name: 'app',  
    script: './bin/www',  
    // Options reference:  
    https://pm2.keymetrics.io/docs/usage/application-declaration/  
    instances: 0,  
  }],  
}
```

```
    autorestart: true,
    watch: true,
    ignore_watch: [                                // 不用监听的文件
        "node_modules",
        "logs"
    ],
    max_memory_restart: '1G',
    "error_file": "./logs/app-err.log",             // 错误日志文件
    "out_file": "./logs/app-out.log",
    "log_date_format": "YYYY-MM-DD HH:mm:ss", // 给每行日志标记一个时间
    env: {
        NODE_ENV: 'development'
    },
    env_production: {
        NODE_ENV: 'production'
    }
  }],
};
```

## 第7集 后端项目自动化部署上线

简介：讲解怎么把后端项目部署到服务器上

- 安装git

```
yum -y install git
```

- 配置用户信息

```
git config --global user.name "eric"
git config --global user.email "1873556804@qq.com"
```

- 生成ssh key

```
ssh-keygen -t rsa -C "1873556804@qq.com"
```

- 查看密钥

```
cat ~/.ssh/id_rsa.pub
```

- 安装pm2

```
npm i pm2 -g  
ln -s /usr/local/node/bin/pm2 /usr/local/bin/pm2
```

- 自动化部署设置

```
deploy : {  
  production : {  
    user : 'root',  
    host : ['172.16.11.130'],  
    ref: "origin/master",  
    // 远程仓库地址  
    repo: "git@github.com:xd-eric/myblog.git",  
    // 指定代码拉取到服务器的目录  
    path: "/usr/local/myProject",  
    ssh_options: "StrictHostKeyChecking=no",  
    'post-deploy' : 'npm install && pm2 reload ecosystem.config.js --env  
production',  
    "env": {  
      "NODE_ENV": "production"  
    }  
  }  
}
```

- 初始化服务器远程文件夹

```
pm2 deploy ecosystem.config.js production setup
```

- 部署项目

```
pm2 deploy ecosystem.config.js production
```

- 免密连接服务器

```
scp ~/.ssh/id_rsa.pub root@172.16.11.130:/root/.ssh/authorized_keys
```

- 开放防火墙端口



```
firewall-cmd --list-ports  查看已开放端口
firewall-cmd --zone=public --add-port=3000/tcp --permanent  开放3000端口
firewall-cmd --reload  重启
```

## 第8集 前端项目实战部署nginx网站服务器

简介：讲解如何使用nginx在服务器上搭建前端项目

- nginx简介

Nginx是一款高性能的 HTTP 和反向代理服务器

- 编译安装：

安装gcc编译环境：

```
yum install -y gcc-c++
```

安装zlib-devel库：

```
yum install -y zlib-devel
```

安装OpenSSL密码库：

```
yum install -y openssl openssl-devel
```

安装pcre正则表达式库

```
yum install -y pcre pcre-devel
```

安装nginx

```
nginx下载官网: http://nginx.org/en/download.html  
wget http://nginx.org/download/nginx-1.16.0.tar.gz  
tar -xf nginx-1.16.0.tar.gz  
mkdir nginx  
cd nginx-1.16.0  
./configure --prefix=/usr/local/nginx --with-http_ssl_module --with-  
http_stub_status_module --with-pcre  
make && make install
```

- 启停nginx服务

```
启动:  
/usr/local/nginx/sbin/nginx -c /usr/local/nginx/conf/nginx.conf  
查看是否启动成功:  
ps -ef | grep nginx  
关闭:  
/usr/local/nginx/sbin/nginx -s stop
```

- 开放80端口防火墙

```
firewall-cmd --zone=public --add-port=80/tcp --permanent  开放80端口  
firewall-cmd --reload  重启
```

- 把vue打包后的文件上传到nginx/html下

filezilla下载: <https://filezilla-project.org/download.php?type=client>



愿景："让编程不在难学，让技术与生活更加有趣"

更多课程请访问[xdclass.net](http://xdclass.net)

## 第十六章 课程总结

---

### 第1集 nodejs课程总结与学习体系

简介：回顾课程内容与学习的知识体系

- mysql (mongodb)
- linux服务器操作语法
- nginx