* Table of Content
* [1. 模块化](file:///G:\01.First\node-01\1-%E6%95%99%E5%AD%A6%E8%B5%84%E6%96%99\01-%E7%AC%94%E8%AE%B0\01-%E6%A8%A1%E5%9D%97%E5%8C%96.html#1)
  + [1.1. 为什么要学习模块化开发](file:///G:\01.First\node-01\1-%E6%95%99%E5%AD%A6%E8%B5%84%E6%96%99\01-%E7%AC%94%E8%AE%B0\01-%E6%A8%A1%E5%9D%97%E5%8C%96.html#101)
    - [1.1.1. 为什么要学习模块化](file:///G:\01.First\node-01\1-%E6%95%99%E5%AD%A6%E8%B5%84%E6%96%99\01-%E7%AC%94%E8%AE%B0\01-%E6%A8%A1%E5%9D%97%E5%8C%96.html#10101)
    - [1.1.2. 什么是模块化](file:///G:\01.First\node-01\1-%E6%95%99%E5%AD%A6%E8%B5%84%E6%96%99\01-%E7%AC%94%E8%AE%B0\01-%E6%A8%A1%E5%9D%97%E5%8C%96.html#10102)
  + [1.2. 模块化的演变过程](file:///G:\01.First\node-01\1-%E6%95%99%E5%AD%A6%E8%B5%84%E6%96%99\01-%E7%AC%94%E8%AE%B0\01-%E6%A8%A1%E5%9D%97%E5%8C%96.html#102)
    - [1.2.1. 非模块化带来的问题](file:///G:\01.First\node-01\1-%E6%95%99%E5%AD%A6%E8%B5%84%E6%96%99\01-%E7%AC%94%E8%AE%B0\01-%E6%A8%A1%E5%9D%97%E5%8C%96.html#10201)
    - [1.2.2. 全局函数](file:///G:\01.First\node-01\1-%E6%95%99%E5%AD%A6%E8%B5%84%E6%96%99\01-%E7%AC%94%E8%AE%B0\01-%E6%A8%A1%E5%9D%97%E5%8C%96.html#10202)
    - [1.2.3. 对象封装](file:///G:\01.First\node-01\1-%E6%95%99%E5%AD%A6%E8%B5%84%E6%96%99\01-%E7%AC%94%E8%AE%B0\01-%E6%A8%A1%E5%9D%97%E5%8C%96.html#10203)
    - [1.2.4. 隔离私有空间](file:///G:\01.First\node-01\1-%E6%95%99%E5%AD%A6%E8%B5%84%E6%96%99\01-%E7%AC%94%E8%AE%B0\01-%E6%A8%A1%E5%9D%97%E5%8C%96.html#10204)
    - [1.2.5. 模块的扩展](file:///G:\01.First\node-01\1-%E6%95%99%E5%AD%A6%E8%B5%84%E6%96%99\01-%E7%AC%94%E8%AE%B0\01-%E6%A8%A1%E5%9D%97%E5%8C%96.html#10205)
    - [1.2.6. 依赖第三方模块](file:///G:\01.First\node-01\1-%E6%95%99%E5%AD%A6%E8%B5%84%E6%96%99\01-%E7%AC%94%E8%AE%B0\01-%E6%A8%A1%E5%9D%97%E5%8C%96.html#10206)
    - [1.2.7. 使用模块化开发的价值](file:///G:\01.First\node-01\1-%E6%95%99%E5%AD%A6%E8%B5%84%E6%96%99\01-%E7%AC%94%E8%AE%B0\01-%E6%A8%A1%E5%9D%97%E5%8C%96.html#10207)
  + [1.3. 模块化开发框架seajs](file:///G:\01.First\node-01\1-%E6%95%99%E5%AD%A6%E8%B5%84%E6%96%99\01-%E7%AC%94%E8%AE%B0\01-%E6%A8%A1%E5%9D%97%E5%8C%96.html#103)
    - [1.3.1. 浏览seajs官网（文档）](file:///G:\01.First\node-01\1-%E6%95%99%E5%AD%A6%E8%B5%84%E6%96%99\01-%E7%AC%94%E8%AE%B0\01-%E6%A8%A1%E5%9D%97%E5%8C%96.html#10301)
    - [1.3.2. 使用seajs的步骤](file:///G:\01.First\node-01\1-%E6%95%99%E5%AD%A6%E8%B5%84%E6%96%99\01-%E7%AC%94%E8%AE%B0\01-%E6%A8%A1%E5%9D%97%E5%8C%96.html#10302)
    - [1.3.3. seajs.use使用模块](file:///G:\01.First\node-01\1-%E6%95%99%E5%AD%A6%E8%B5%84%E6%96%99\01-%E7%AC%94%E8%AE%B0\01-%E6%A8%A1%E5%9D%97%E5%8C%96.html#10303)
    - [1.3.4. 导出模块中的成员](file:///G:\01.First\node-01\1-%E6%95%99%E5%AD%A6%E8%B5%84%E6%96%99\01-%E7%AC%94%E8%AE%B0\01-%E6%A8%A1%E5%9D%97%E5%8C%96.html#10304)
    - [1.3.5. 模块之间的依赖](file:///G:\01.First\node-01\1-%E6%95%99%E5%AD%A6%E8%B5%84%E6%96%99\01-%E7%AC%94%E8%AE%B0\01-%E6%A8%A1%E5%9D%97%E5%8C%96.html#10305)
    - [1.3.6. config](file:///G:\01.First\node-01\1-%E6%95%99%E5%AD%A6%E8%B5%84%E6%96%99\01-%E7%AC%94%E8%AE%B0\01-%E6%A8%A1%E5%9D%97%E5%8C%96.html#10306)
    - [1.3.7. seajs原理](file:///G:\01.First\node-01\1-%E6%95%99%E5%AD%A6%E8%B5%84%E6%96%99\01-%E7%AC%94%E8%AE%B0\01-%E6%A8%A1%E5%9D%97%E5%8C%96.html#10307)
  + [1.4. 模块化框架requireJS](file:///G:\01.First\node-01\1-%E6%95%99%E5%AD%A6%E8%B5%84%E6%96%99\01-%E7%AC%94%E8%AE%B0\01-%E6%A8%A1%E5%9D%97%E5%8C%96.html#104)
  + [1.5. 模块化规范](file:///G:\01.First\node-01\1-%E6%95%99%E5%AD%A6%E8%B5%84%E6%96%99\01-%E7%AC%94%E8%AE%B0\01-%E6%A8%A1%E5%9D%97%E5%8C%96.html#105)
    - [1.5.1. CMD规范](file:///G:\01.First\node-01\1-%E6%95%99%E5%AD%A6%E8%B5%84%E6%96%99\01-%E7%AC%94%E8%AE%B0\01-%E6%A8%A1%E5%9D%97%E5%8C%96.html#10501)
    - [1.5.2. AMD规范](file:///G:\01.First\node-01\1-%E6%95%99%E5%AD%A6%E8%B5%84%E6%96%99\01-%E7%AC%94%E8%AE%B0\01-%E6%A8%A1%E5%9D%97%E5%8C%96.html#10502)
    - [1.5.3. CMD规范和AMD规范区别](file:///G:\01.First\node-01\1-%E6%95%99%E5%AD%A6%E8%B5%84%E6%96%99\01-%E7%AC%94%E8%AE%B0\01-%E6%A8%A1%E5%9D%97%E5%8C%96.html#10503)
  + [1.6. 路径问题](file:///G:\01.First\node-01\1-%E6%95%99%E5%AD%A6%E8%B5%84%E6%96%99\01-%E7%AC%94%E8%AE%B0\01-%E6%A8%A1%E5%9D%97%E5%8C%96.html#106)

**1. 模块化**

**1.1. 为什么要学习模块化开发**

**1.1.1. 为什么要学习模块化**

了解模块化思想

为后面的node学习打基础

**1.1.2. 什么是模块化**

* 生活中的模块化
  + 组装电脑
  + google的模块化手机（流产了，moto继承了）
  + 乐高积木
* 软件开发中的模块化
  + 日期模块
  + 数学计 算模块
  + 日志模块
  + 登录认证模块
  + 报表展示模块
* 模块化的好处
  + 生产效率高
  + 方便维护

**1.2. 模块化的演变过程**

**1.2.1. 非模块化带来的问题**

命名冲突

文件依赖

**1.2.2. 全局函数**

* 问题：命名可能会冲突（变量污染）
* [代码](file:///G:\01.First\node-01\1-%E6%95%99%E5%AD%A6%E8%B5%84%E6%96%99\01-%E7%AC%94%E8%AE%B0\01-模块化\01-计算器-全局函数.html)

**1.2.3. 对象封装**

* 解决：解决命名冲突 （类似命名空间的方式）
* 新问题：没有自己的私人空间(不能设置私有成员)
* [代码](file:///G:\01.First\node-01\1-%E6%95%99%E5%AD%A6%E8%B5%84%E6%96%99\01-%E7%AC%94%E8%AE%B0\01-模块化\02-计算器-对象封装.html)

**1.2.4. 隔离私有空间**

* 解决：不能设置私有成员问题
* 新问题：如何扩展一个功能？
* [代码](file:///G:\01.First\node-01\1-%E6%95%99%E5%AD%A6%E8%B5%84%E6%96%99\01-%E7%AC%94%E8%AE%B0\01-模块化\03-计算器-隔离私有成员.html)

**1.2.5. 模块的扩展**

* 对扩展开放，对修改封闭
* 解决：扩展一个求平方的功能
* [代码](file:///G:\01.First\node-01\1-%E6%95%99%E5%AD%A6%E8%B5%84%E6%96%99\01-%E7%AC%94%E8%AE%B0\01-模块化\04-计算器-模块的扩展.html)

**1.2.6. 依赖第三方模块**

* 封装到js文件中
* [代码](file:///G:\01.First\node-01\1-%E6%95%99%E5%AD%A6%E8%B5%84%E6%96%99\01-%E7%AC%94%E8%AE%B0\01-模块化\05-计算器-依赖第三方模块.html)

**1.2.7. 使用模块化开发的价值**

<https://github.com/seajs/seajs/issues/547>

**1.3. 模块化开发框架seajs**

**1.3.1. 浏览seajs官网（文档）**

<http://seajs.org>

**1.3.2. 使用seajs的步骤**

* 导入seajs
* define() 定义一个模块，define的回调的三个参数名字不可更改

define(function (require, exports, module) {});

* exports或者module.exports将模块中的成员暴露出来
* seajs.use 使用一个模块
* config() 配置一些基本信息（别名、路径）

**1.3.3. seajs.use使用模块**

//使用单个模块，回调函数中的obj就是模块中的exports对象

seajs.use("modules/demo1/calc.js", function (obj) {

console.log(obj.add(5,6));

console.log(obj.mul(5,6));

});

//使用多个模块

seajs.use(["modules/demo1/calc.js","modules/demo1/power.js"] , function (o1,o2)

{

//o1 对应calc模块中的exports对象

console.log(o1.add(1,2));

//o2 对应power模块中的exports对象

console.log(o2.power(5));

});

**1.3.4. 导出模块中的成员**

* 使用exports导出成员

define(function (require, exports, module) {

exports.add = function (a, b) {

return a + b;

}

exports.sub = function (a, b) {

return a - b;

}

});

* 使用module.exports导出

//使用module.exports导出成员和exports用法一样

define(function (require, exports, module) {

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

return a + b;

}

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

return a - b;

}

});

//使用module.exports导出对象，exports不可以

define(function (require, exports, module) {

//可以使用module.exports导出对象

module.exports = {

add: function (a, b) {

console.log("module");

return a + b;

},

sub: function (a, b) {

return a - b;

}

};

});

* exports和module.exports的关系
  + exports = module.exports = {};
  + 画图解释
  + 不能为exports直接赋值，但是可以动态添加成员

**1.3.5. 模块之间的依赖**

* require() 引用另一个模块

define(function (require, exports, module) {

//依赖另一个模块，js后缀可以省略

var o = require("./power");

module.exports = {

add: function (a, b) {

console.log("module");

return a + b;

},

sub: function (a, b) {

return a - b;

},

three: function (a) {

return o.power(a) \* parseInt(a) ;

}

};

});

**1.3.6. config**

* config的作用就是为了简化调用模块
  + base 设置路径
  + alias 设置模块的别名，简化调用

seajs.config({

//设置路径

base: "modules/demo4",

//设置别名

alias: {

c: "calc",

p: "power"

}

});

**1.3.7. seajs原理**

function loadJS(path, callback) {

var head = document.getElementsByTagName("head")[0];

var node = document.createElement("script");

node.src = path;

head.appendChild(node);

//浏览器兼容处理

var supportOnload = "onload" in node;

if(supportOnload) {

node.onload = function () {

callback();

}

}else{

node.onreadystatechange = function () {

if(node.readyState == "loaded" || node.readyState == "complete") {

callback();

}

}

}

}

//调用

loadJS("js/test.js", function () {

test();

})

**1.4. 模块化框架requireJS**

**1.5. 模块化规范**

**1.5.1. CMD规范**

* seajs遵守CMD规范
* CMD规范的地址 <https://github.com/cmdjs/specification/blob/master/draft/module.md>
* CMD规范的特点 所有的require()都是懒加载模式，用到的时候才去加载，提升初始化时加载的性能

**1.5.2. AMD规范**

* requirejs遵守AMD规范 requirejs官网 <http://www.requirejs.org/>
* AMD规范的地址[https://github.com/amdjs/amdjs-api/wiki/AMD/](https://github.com/amdjs/amdjs-api/wiki/AMD)
* AMD规范的特点 所有的require()都是一个预加载模式

**1.5.3. CMD规范和AMD规范区别**

CMD是懒加载

AMD是预加载

**1.6. 路径问题**

* seajs中使用相对路径
  + demo 和 ./demo在seajs中稍有不同
  + demo 一般表示相对于当前引入的sea.js文件的位置
  + ./demo 一般表示相对当前编写的文件的位置
* Table of Content
* [1. Node介绍](file:///G:\01.First\node-01\1-%E6%95%99%E5%AD%A6%E8%B5%84%E6%96%99\01-%E7%AC%94%E8%AE%B0\02-Node%E4%BB%8B%E7%BB%8D.html#1)
  + [1.1. 复习相关概念](file:///G:\01.First\node-01\1-%E6%95%99%E5%AD%A6%E8%B5%84%E6%96%99\01-%E7%AC%94%E8%AE%B0\02-Node%E4%BB%8B%E7%BB%8D.html#101)
  + [1.2. 为什么要学习Nodejs](file:///G:\01.First\node-01\1-%E6%95%99%E5%AD%A6%E8%B5%84%E6%96%99\01-%E7%AC%94%E8%AE%B0\02-Node%E4%BB%8B%E7%BB%8D.html#102)
  + [1.3. 安装Node.js](file:///G:\01.First\node-01\1-%E6%95%99%E5%AD%A6%E8%B5%84%E6%96%99\01-%E7%AC%94%E8%AE%B0\02-Node%E4%BB%8B%E7%BB%8D.html#103)
  + [1.4. 学习Node.js的网站](file:///G:\01.First\node-01\1-%E6%95%99%E5%AD%A6%E8%B5%84%E6%96%99\01-%E7%AC%94%E8%AE%B0\02-Node%E4%BB%8B%E7%BB%8D.html#104)

**1. Node介绍**

**1.1. 复习相关概念**

* 什么是JavaScript
  + 一种遵守ECMAScript标准的脚本语言
  + 最初只能运行在浏览器端
* 浏览器中的 JavaScript 可以做什么？
  + 操作DOM：表单验证、动画
  + 数据交互：ajax
  + 通常所指的 JavaScript ：BOM + DOM （W3C）+ ECMAScript
* JavaScript 只可以运行在浏览器中吗？
  + JavaScript 不仅仅能运行在浏览器中
  + JavaScript 还可以运行在服务器环境中，例如：node
  + JavaScript引擎

| **浏览器** | **JavaScript引擎** |
| --- | --- |
| Firefox | SpiderMonkey |
| IE | Chakra |
| Safari | JavaScriptCore |
| Chrome | V8 |

* 客户端和服务器

面向用户的一端就是客户端

从数据角度来说：客户端只负责界面交互，主动的发起 HTTP 请求，请求后台服务器，把数据渲染到界面。

服务器提供的 HTTP 服务，接收 客户端的 HTTP 请求，然后处理 HTTP 请求， 最后把要发送给客户端的响应数据通过网络发送给客户端。

**1.2. 为什么要学习Nodejs**

* 什么是 Node.js
* Node.js 是一种建立在Google Chrome's V8 引擎上的non-blocking(非阻塞),event-driven(基于事件的)I/O平台
* Node.js平台使用的开发语言是JavaScript，平台提供了操作系统底层的API，方便做服务器端编程，
* 具体包括：文件操作、进程操作、通信操作等系统模块，支持模块化的开发
  + Node.js官网：<https://nodejs.org>
  + Node.js 是一个构建与 Chrome 的 V8 JavaScript 引擎之上的一个 JavaScript 运行时环境
  + Node 可以用来解析和执行 JavaScript 代码(因为Node基于V8引擎)
  + Node 无法解析 DOM 和 BOM 等对象
  + Node 只关心数据部分，例如 读写文件，提供 Web 服务
  + Node 是 JavaScript 吗？
  + Node 不是一种新的语言
  + Node 是一个可以用来解析和执行 JavaScript 语言的一个环境或者说是一个平台
  + Node 环境或者说平台 给 JavaScript 语言，提高了 一些 例如文件操作、网络操作等接口API
  + Node.js 的包管理系统 npm 是世界上最大的开源库生态系统
* Node 的诞生历程
  + Node 的作者：Ryan Dahl
  + 2004年还在纽约读数学系博士
  + 2006年退学（世界那么大，我想去看看），生活所迫，选择了码农
  + 接项目、工作、旅行，两年之后，成为了高性能Web服务器的专家
  + 2009年5月，Ryan Dahl 在 github 上发布了最初的 Node 版本
  + 2010年底，Ryan Dahl 加入 Joyent 公司全职负责 Node 的发展
  + 2011年7月，Node 在微软的支持下发布了 Windows 版
  + 2012年1月底，Ryan Dahl 将掌门人身份转交给了 Isaac Z.Schlueter，自己转向一些研究项目
  + 2014年12月，多位重量级Node开发者不满 Joyent 对 Node 的管理，自立门户创建了 io.js
  + 2015年9月，Node 与 io.js 合并，Node 的版本从 0.12.7 直接升级到了 4.0.0
  + 合并后的 io.js 和 Node 在 Joyent 公司的维护下并行了两个版本：
    - 一个是4.x.x 还是原来的 Node，这个版本是稳定版
    - 一个是5.x.x，目前已经更新到了 6.6.0，其实就是 io，最新特性版，不建议生产环境使用
* Node的特点
  + 事件驱动
  + 异步IO模型
  + 跨平台
* 为什么要学习 Node
  + Node 打破了过去 JavaScript 只能在浏览器中运行的局面
  + 前后端编程环境统一，大大降低了前后端语言切换的代价
  + 全栈开发工程师
    - 后端开发工程师
    - 前端开发工程师
    - 移动端开发工程师
* 使用 Node 可以做什么
  + 高性能的网站服务器
  + 简单易用的命名行应用程序
    - i5ting、gulp、less
  + 实时多人游戏后台服务器
  + 高大上的桌面应用程序
    - 使用 Web 技术 作为解决方案
  + 底层的物联网开发
  + 移动开发
    - Ionic --> 安卓 IOS
    - 使用 Web 技术作为解决方案

**1.3. 安装Node.js**

* 两种安装方式
  + 直接下载Nodejs的安装包
  + 使用NVM安装，可以切换Nodejs的版本
    - NVM安装不同版本的Node  
      nvm install 6.5.0
    - 查看现在可用的Node版本 nvm list
    - 切换Node版本 nvm use 6.6.0 32/64
* 查看Node是否安装成功/查看Node的当前版本 node -v
* 切换源 国内 <https://registry.npm.taobao.org> 国外 <http://registry.npmjs.org/>

切换 npm config set registry <https://registry.npm.taobao.org> 配置后可通过下面方式来验证是否成功 npm config get registry

**1.4. 学习Node.js的网站**

* 官网 <https://nodejs.org>
* Node.js开源包管理网站 <https://www.npmjs.com> npm.taobao.org
* 开源代码托管 <https://github.com>
* 问答社区 <http://stackoverflow.com>
* Table of Content
* [1. 命令行](file:///G:\01.First\node-01\1-%E6%95%99%E5%AD%A6%E8%B5%84%E6%96%99\01-%E7%AC%94%E8%AE%B0\03-%E7%BB%88%E7%AB%AF%E5%91%BD%E4%BB%A4%E5%92%8C%E7%8E%AF%E5%A2%83%E5%8F%98%E9%87%8F.html#1)
  + [1.1. path 环境变量](file:///G:\01.First\node-01\1-%E6%95%99%E5%AD%A6%E8%B5%84%E6%96%99\01-%E7%AC%94%E8%AE%B0\03-%E7%BB%88%E7%AB%AF%E5%91%BD%E4%BB%A4%E5%92%8C%E7%8E%AF%E5%A2%83%E5%8F%98%E9%87%8F.html#101)

## 1. 命令行

* 图形用户界面（GUI） VS 命令行界面（CLI）

GUI: graphical user interface，简单的任务更容易完成

CLI: command line interface，完成复杂的任务成为可能

* 为什么使用命令行

在电影中一个“超级黑客”坐在电脑前，从不摸一下鼠标， 就能够在30秒内侵入到超安全的军事计算机中。 作为人类，本能的知道让计算机圆满完成工作的唯一途径，就是用键盘来操作计算机。

* 介绍

学习终端最正确的姿势是学习 Linus 中的 shell， 在 Linus 中，shell 就是一个程序，它接受用户从键盘输入的命令， 然后把命令传递给操作系统去执行。

在Windows中也有一个程序类似于 shell，就是我们常见的控制台或者说是 cmd

终端、shell、bash、terminal

一般在 Windows 中，终端就是指的 cmd，也叫控制台

在 Linus 或者 mac 中，一般叫做 shell、bash、terminal

* 进入与退出终端

在 Windows 中，有两个终端可以让我们使用：

cmd：输入 win + r 打开运行，输入 cmd 敲回车就可以打开cmd控制台了。

powershell：输入 win + r 打开运行，输入 powershell 敲回车就可以打开cmd控制台了。

powershell 是 cmd 的一个加强版的终端。powershell 对 Linus / mac 上的终端的，命令做了一个兼容

在终端中输入 exit 敲回车就可以退出终端了

* 命令提示符
* 常用命令
* 快速打开系统应用 notepad、mspaint、calc、sysdm.cpl、write
* 终端命令 cd、dir、md、rd、del、copy、type
  + cd（change directory）：切换目录
    - 盘符: 然后敲回车就可以切换盘符
    - 相对路径
      * cd ./目录名 切换到相对于当前目录下的某个目录
      * cd ../ 返回上一级目录
    - 绝对路径
      * 切换绝对路径的时候，路径中不要包含空格
      * 如果有目录名称有空格，两边加上单引号即可
  + mkdir（md）（make directory）：创建目录
  + rd（remove）：删除文件或目录
    - -r 递归删除：不会提示，直接将整个目录包括里面的内容都删掉
  + cls：清屏
  + 方向键的左右可以切换 历史敲过的命令

以后多使用，就会越用越熟。

## 1.1. path 环境变量

当要求系统运行一个程序而没有告诉它程序所在的完整路径时， 系统首先在当前目录下面寻找该程序， 如果找不到，则系统会跑到path中指定的路径去找，如果找到，直接运行， 如果path环境变量中也没有找到，则直接提示不是内部或外部命令，也不是可运行的程序

当设置完环境变量之后要重启cmd

添加 path 环境变量的两种方式：

一：直接在path环境变量中加入程序所属目录的绝对路径， 两边以 **英文分号** 进行分隔。

例如 feiq.exe 程序的绝对路径是 C:\Program Files\feiQ， 则在 path 中添加 ;C:\Program Files\feiQ;

二：也可以在外部先定义一个变量名，值就是程序所属目录的绝对路径， 然后在 path 中以 %变量名% 的方式引入，两边以分号分隔

例如 qq.exe 程序的绝对路径是 C:\Program Files (x86)\Tencent\QQ\Bin， 则先定义一个变量名 QQ\_HOME ，变量值就是C:\Program Files (x86)\Tencent\QQ\Bin， 然后在 path 中填入变量名：;%QQ\_HOME%;

* Table of Content
* [1. Node的REPL环境](file:///G:\01.First\node-01\1-%E6%95%99%E5%AD%A6%E8%B5%84%E6%96%99\01-%E7%AC%94%E8%AE%B0\04-Node%E7%9A%84%E5%85%A8%E5%B1%80%E5%AF%B9%E8%B1%A1.html#1)
  + [1.1. Node 执行js文件](file:///G:\01.First\node-01\1-%E6%95%99%E5%AD%A6%E8%B5%84%E6%96%99\01-%E7%AC%94%E8%AE%B0\04-Node%E7%9A%84%E5%85%A8%E5%B1%80%E5%AF%B9%E8%B1%A1.html#101)
  + [1.2. Node.js的快速体验](file:///G:\01.First\node-01\1-%E6%95%99%E5%AD%A6%E8%B5%84%E6%96%99\01-%E7%AC%94%E8%AE%B0\04-Node%E7%9A%84%E5%85%A8%E5%B1%80%E5%AF%B9%E8%B1%A1.html#102)
  + [1.3. Node中的全局对象](file:///G:\01.First\node-01\1-%E6%95%99%E5%AD%A6%E8%B5%84%E6%96%99\01-%E7%AC%94%E8%AE%B0\04-Node%E7%9A%84%E5%85%A8%E5%B1%80%E5%AF%B9%E8%B1%A1.html#103)

## 1. Node的REPL环境

* 什么是REPL

REPL全称 Read-eval-print-loop，交互式解析器

REPL可以提供给程序员对Node.js的一些api快速测试

* REPL的基本操作
  + 定义和运行变量、函数、对象
  + 下划线\_ 可以获取表达式的运算结果
* REPL的常用命令
  + 退出REPL，连按两次ctrl+c，输入.exit
  + tab 打印Node.js中的所有对象
  + 向上/向下 查看历史命令
  + .save filename 保存输入的命令
  + .load filename 加载文件

## 1.1. Node 执行js文件

* node filename

## 1.2. Node.js的快速体验

* 读取文件

//加载文件读写模块

var fs = require("fs");

fs.readFile("data.json", function(error, data) {

if(error){

throw error;

}

console.log(data.toString());

});

* 创建一个简易web服务器

var http = require("http");

var server = http.createServer();

var count = 0;

server.on("request", function (request,response) {

count ++;

console.log("有请求进来");

response.write("hello world" + count);

response.end();

});

server.listen(3333);

console.log("服务开启");

## 1.3. Node中的全局对象

* global
  + global顶级对象，类似于浏览器中的window
  + 在任何地方可以直接使用，不需要导包
  + setTimeout/setImmediate/process.nextTick，异步的时候setTimeout的优先级高于setImmediate,process.nextTick优先级最高
  + setInterval
  + 在REPL环境中定义的变量直接挂载到global中
  + 在js文件中定义的变量不会挂载到global中
* 获取路径
  + \_\_dirname 当前执行的js所在的文件夹
  + \_\_filename 当前执行的js的路径
  + process.cwd() 当前执行node.exe的路径
* process
  + process.abort() 退出node.exe
  + process.arch 获取执行当前node的平台是32还是64位
  + process.argv 获取当前执行node时候的参数数组

默认数组由两个值 1、node的路径 2、当前js的路径

* + process.env 获取环境变量

结合命令行 命令 set xxx="" 可以指定执行的环境 例如：set xxx=dev && node 07-env判断执行环境.js

* 标准输入输出
  + process.stdout.write(); console.log内部调用的write()并且加了一个换行
  + 案例：输出控制台动画
  + var index = 0;
  + setInterval(function () {
  + //输出之前清空之前的内容
  + process.stdout.write("\u001b[2J\u001b[0;0H");
  + var content = fcontent[index];
  + process.stdout.write(content);
  + index++;
  + if(index >= fcontent.length) {
  + index =0;
  + }
  + }, 100);
  + process.stdin.read();
  + process.stdin.on("readable", function () {
  + //监控键盘的输入
  + //只要有输入，就会执行function
  + var data = process.stdin.read();
  + if(data == null) {
  + console.log("请输入一些内容");
  + return;
  + }
  + process.stdout.write("你输入的内容：" + data);

});

* Table of Content
* [1. Node.js的模块化](file:///G:\01.First\node-01\1-%E6%95%99%E5%AD%A6%E8%B5%84%E6%96%99\01-%E7%AC%94%E8%AE%B0\05-Node%E7%9A%84%E6%A8%A1%E5%9D%97%E5%8C%96.html#1)
  + [1.1. Node.js中的模块化简介](file:///G:\01.First\node-01\1-%E6%95%99%E5%AD%A6%E8%B5%84%E6%96%99\01-%E7%AC%94%E8%AE%B0\05-Node%E7%9A%84%E6%A8%A1%E5%9D%97%E5%8C%96.html#101)
  + [1.2. Node.js的核心模块](file:///G:\01.First\node-01\1-%E6%95%99%E5%AD%A6%E8%B5%84%E6%96%99\01-%E7%AC%94%E8%AE%B0\05-Node%E7%9A%84%E6%A8%A1%E5%9D%97%E5%8C%96.html#102)
  + [1.3. 核心模块存在哪里？](file:///G:\01.First\node-01\1-%E6%95%99%E5%AD%A6%E8%B5%84%E6%96%99\01-%E7%AC%94%E8%AE%B0\05-Node%E7%9A%84%E6%A8%A1%E5%9D%97%E5%8C%96.html#103)
  + [1.4. 文件模块（自定义模块）](file:///G:\01.First\node-01\1-%E6%95%99%E5%AD%A6%E8%B5%84%E6%96%99\01-%E7%AC%94%E8%AE%B0\05-Node%E7%9A%84%E6%A8%A1%E5%9D%97%E5%8C%96.html#104)
  + [1.5. 包](file:///G:\01.First\node-01\1-%E6%95%99%E5%AD%A6%E8%B5%84%E6%96%99\01-%E7%AC%94%E8%AE%B0\05-Node%E7%9A%84%E6%A8%A1%E5%9D%97%E5%8C%96.html#105)
  + [1.6. 发布包](file:///G:\01.First\node-01\1-%E6%95%99%E5%AD%A6%E8%B5%84%E6%96%99\01-%E7%AC%94%E8%AE%B0\05-Node%E7%9A%84%E6%A8%A1%E5%9D%97%E5%8C%96.html#106)
  + [1.7. Node.js的调试](file:///G:\01.First\node-01\1-%E6%95%99%E5%AD%A6%E8%B5%84%E6%96%99\01-%E7%AC%94%E8%AE%B0\05-Node%E7%9A%84%E6%A8%A1%E5%9D%97%E5%8C%96.html#107)

# 1. Node.js的模块化

## 1.1. Node.js中的模块化简介

* 为什么Node.js中需要模块化
* 在后台开发语言中，比如Java、C#。他们都是隐含模块化的，Node.js默认帮我们提供了模块化这种机制。
* 在服务器端，我们想要使用底层的一些功能需要导入一些“包”来对其操作，比如操作文件、网络需要导入对应的包。其它语言中都是基于类来实现的模块化的思想，使用类来组织文件和文件之间的关联。
* 而Node.js中使用的是JavaScript语言，ECMAScript仅仅规定了基本的语法的书写，并没有规定文件之间
* 关联，也就是说每个js文件之间是独立的，Node.js已经帮我们实现了js文件之间的关联（模块化）
* Node.js中的模块化是基于CommonJS规范的
* JavaScript的局限性
  + 没有模块系统
  + 系统提供的接口较少，比如：缺少操作文件、I/O流等常用的接口
  + 没有标准接口，缺少如web服务器、数据库等统一接口
  + 缺乏管理系统导师JavaScript应用中基本没有自动加载和安装依赖的能力
* CommonJS规范
  + Node.js开发之初遵守了CommonJS规范
  + 使JavaScript达到像Java、Python、PHP等语言一样有开发大型应用的基本能力
  + CommonJS规范规定每一个模块都有一个单独的作用域
  + CommonJS规范规定每个模块对外公布的成员使用module.exports或者exports
  + 有了模块化系统之后，Node.js提供了许多系统模块：文件、Buffer、I/O流、Socket等

## 1.2. Node.js的核心模块

* 使用核心模块之前首先要导入模块
* path模块
  + 导入模块 var path = require("path");
  + basename() 获取文件名+后缀
  + path.basename("/foo/hello/world/123.html")
  + //第二个参数，去掉获取的文件名中的相同部分
  + path.basename("c:/foo/hello/world/123.html",".html")
  + dirname() 获取目录
  + path.dirname("/foo/hello/world/123.html")
  + extname() 获取文件的扩展名
  + path.extname("/foo/hello/world/123.html")
  + join() 合并路径
  + var p1 = "c://abc/xyz";
  + var p2 = "/123/456";
  + console.log(path.join(p1,p2));
  + parse() 把路径转换为一个对象
  + path.parse("c:\\home\\hello\\world\\123.html")
  + { root: 'c:/',
  + dir: 'c://home/hello/world',
  + base: '123.html',
  + ext: '.html',
  + name: '123' }
  + format() 把一个路径对象转换成一个路径字符串
  + var obj = { root: 'c:\\',
  + dir: 'c:\\home\\hello\\world',
  + base: '123.html',
  + ext: '.html',
  + name: '123' }
  + console.log(path.format(obj));
  + delimiter 环境变量的分隔符，可以跨平台 windows下是; 其它平台 :
  + path.sep 路径的分隔符 windows下是\ 其它下是/
  + isAbsolute() 是否是绝对路径
* url模块
  + 导入模块 var url = require("url");
  + parse() 把字符串的路径转换成对象
  + var uri = "http://www.baidu.com:8080/images/1.jpg?version=1.0&time=1123#abcd";
  + console.log(url.parse(uri));
  + format() 把路径对象转换成字符串
  + var obj = {
  + protocol: 'http:',
  + slashes: true,
  + auth: null,
  + host: 'www.baidu.com:8080',
  + port: '8080',
  + hostname: 'www.baidu.com',
  + hash: '#abcd',
  + search: '?version=1.0&time=1123',
  + query: 'version=1.0&time=1123',
  + pathname: '/images/1.jpg',
  + path: '/images/1.jpg?version=1.0&time=1123',
  + href: 'http://www.baidu.com:8080/images/1.jpg?version=1.0&time=1123#abcd' };
  + var str = url.format(obj);
  + console.log(str);
* querystring模块
  + 导入模块 var querystring = require("querystring");
  + parse() 把参数字符串解析成对象
  + var obj = querystring.parse("version=1.0&time=123");
  + console.log(obj);
  + stringify() 把一个对象转换成一个字符串
  + escape() url进行编码
  + unescape() url进行解码

## 1.3. 核心模块存在哪里？

* 核心模块存储在node.exe中，当node.exe运行的时候，核心模块会被加载，require的时候会加载到内存
* 在github上可以找到源代码，lib文件夹下
* 核心模块的执行速度比较快

## 1.4. 文件模块（自定义模块）

* 定义文件模块 add.js
* function add(a,b) {
* return a + b;
* }
* //导出成员
* exports.add = add;
* //module.exports.add = add;
* 使用文件模块 main.js
* var obj = require("./add.js");
* console.log(obj.add(5,6));
  + 注意引用js的方式和核心模块不同
  + //使用相对于main.js 的方式查找add.js
  + var obj = require("./add.js");
  + var obj = require("./add");
  + //下面这种方式是引用核心模块或者包
  + //var obj = require("add");

## 1.5. 包

* CommonJS的包规范给程序员提供了组织模块的标准，减少沟通成本
* 包的使用：
  + 所有模块放在一个文件夹(包名)
  + 包放在当前项目中的node\_modules文件夹下
  + 包中定义一个index.js（文件名不可以更改）导出所有模块
  + 引用包（约定大于配置）
* 导入包的执行过程 require("calc")
  + 将calc当做核心模块加载，加载不成功
  + 自动去当前目录中的node\_modules中找文件名为calc的包
  + 自动去calc找index.js的出口模块（导出的模块）
  + 如果找不到index.js报错，如果想要改出口模块，需要package.json配置文件
* package.js

| **名称** | **功能** |
| --- | --- |
| name | 包名称 |
| description | 包介绍，介绍包的功能 |
| version | 版本号，用于版本控制 |
| keywords | 关键词数组，用于在npm中搜索 |
| main | require引入包时优先检查此字段 |
| dependencies | 标记当前包所依赖的包列表，npm会自动加载依赖的包 |
| Author | 包作者 |
| License | 开源许可 |

{

"name": "calcpack",

"version": "1.0.0",

"description": "",

"main": "app.js",

"scripts": { //可以通过npm run来执行

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

},

"keywords": [],

"author": "",

"license": "ISC"

}

* 创建包的标准方式
  + npm init -y 自动创建package.json
* 一个标准包的结构

名称 |功能 |---|---| package.js |包描述文件 bin |存放可执行文件 lib |存放JavaScript代码 doc |存放文档 test |存放单元测试用例代码 README.md |说明文档，描述包的作用和用法

* 标准包执行过程
  + 将calcpack当做核心模块加载，加载不成功
  + 自动去当前目录中的node\_modules中找文件名为calcpack的包
  + 如果在calcpack中有package.json的话，并且指定了main属性的值，优先加载main指定的.js模块（出口模块）
  + 如果没有package.json，或没有指定main属性，自动去calcpack找index.js的出口模块（导出的模块）
  + 如果找不到index.js报错

## 1.6. 发布包

* 把包发布到NPM官网 <https://www.npmjs.com/>
  + 建立一个包，设置package.json
  + 在npmjs中注册账号
  + 在包的根目录下执行
    - npm adduser 添加发布包的用户信息,登录网站
    - npm publish 发布或者更新包 package.json中一定要指定 maintainers:[{
    - "name":"nllcode",
    - "email":"xxxxx@qq.com"

}]

* + - npm cache clear 清除npm本地缓存，用于对使用相同版本号发布新版本
    - npm unpublish @ 删除发布过的版本代码 npm unpublish haha@1.0.0
* 错误
  + only admin can publish this module
  + 解决：修改源 npm config set registry <http://registry.npmjs.org>
* 安装包
  + 从网络安装
    - 当前目录安装 npm install 包名
    - 全局安装 npm install 包名 -g
  + 本地安装 npm install 包的路径
  + 卸载包 npm uninstall 包名
* require()加载规则
  + 优先从缓存加载模块或者包
  + 加载文件模块要使用相对路径 ./ ../
  + 文件模块的加载可以不写后缀名，如果不写后缀名按照 .js > .node > .json的顺序加载
  + 加载json文件，推荐写上后缀.json
  + 加载核心模块或包，不写路径和后缀
  + module.paths 加载node\_modules的时候，按此数组的顺序加载

## 1.7. Node.js的调试

* console.log()
* Node.js自带的调试器
  + node debug a.js
  + 常用命令
    - help 查看可用命令
    - n 进入下一步
    - s 单步进入
    - o 步出
* WebStorm中调试
* Table of Content
* [1. ECMAScript6语法](file:///G:\01.First\node-01\1-%E6%95%99%E5%AD%A6%E8%B5%84%E6%96%99\01-%E7%AC%94%E8%AE%B0\06-ECMASript6%E8%AF%AD%E6%B3%95.html#1)

# 1. ECMAScript6语法

* 为什么要学习ES6
  + ES6也叫ECMAScript2015，2015-6月正式发布，向下兼容ES5.1
  + ES6来编写Node.js程序
  + ES6让JavaScript语言能够开发复杂应用程序
* 严格模式
  + ES5中出现的模式
  + 严格模式的目的：
    - 消除Javascript语法的一些不合理、不严谨之处，减少一些怪异行为;
    - 消除代码运行的一些不安全之处，保证代码运行的安全；
    - 提高编译器效率，增加运行速度；
    - 为未来新版本的Javascript做好铺垫
    - 参考

[严格模式的限制](https://jsmean.com/blog/post/55a9db88a0367a75336e8884) [严格模式的限制](http://www.cnblogs.com/jiqing9006/p/5091491.html)

* + 开启严格模式
    - 在作用域的最上面 "use strict"
      * js文件的最开头
      * function的开头
* 变量和常量
  + 变量 let
    - 没有变量提升，必须先定义再使用
    - 有块级作用域
    - 解决问题
    - window.onload = function () {
    - var ul = document.getElementById("list");
    - var lis = ul.getElementsByTagName("li");
    - for(let i = 0, length=lis.length; i < length; i++) {
    - // (function (i) {
    - // lis[i].onclick = function () {
    - // console.log(i);
    - // }
    - // })(i);
    - lis[i].onclick = function () {
    - console.log(i);
    - }
    - }
    - }
  + 常量 const
    - 常量的一旦赋值，不能改变
    - 有块级作用域
    - 没有变量提升，先定义再使用
    - 不可以重复定义
    - 必须有初始值，否则报错
    - 在Node.js中，所有接收require()获得的对象都使用const修饰
* string的扩展方法
  + includes() 返回布尔值，是否包含，第二个参数从第几个位置开始查找
  + startsWith() 返回布尔值，是否以...开头，第二个参数从第几个位置开始查找
  + endsWith() 返回布尔值，是否以...结尾，第二个参数, 前n个字符以...结尾
  + repeat() 重复次数
* 模板
  + 示例1：
  + let name = "steve jobs";
  + let str = `hello ${name}`;
  + 示例2：
  + let obj = {name:"jobs", age: 18, salary: 1};
  + let template = `
  + 姓名：${obj.name}
  + 年龄：${obj.age}
  + 工资：${obj.salary}
  + `.trim();
  + 示例3：生成页面
* 箭头函数
  + 格式： => (读作：goesto) 左边是参数，右边是方法体
  + 演变：function f(x,y) {} -----> 演变成箭头函数
  + (x, y) => {}
  + 语法格式简单
  + 箭头函数的几种形式：
    - 没有参数 () => console.log("hello");
    - 有一个参数 a => ++a;
    - 有多个参数 (a,b) => a + b;
    - 方法体有多条语句 (a,b) => { a=1; b=2; console.log(a+b)};
  + 箭头函数的注意事项
    - 箭头函数不会影响this的指向
    - 箭头函数根本没有自己的this，箭头函数内部的this就是外层代码块的this
    - 箭头函数中没有arguments对象，箭头函数内部的arguments指向外层函数的arguments
    - 不可以当做构造函数
* ES6的学习站点
  + 各个浏览器对ES6的支持 <http://kangax.github.io/compat-table/es6/>
  + 学习站点 <http://es6.ruanyifeng.com/#README>
* Table of Content
* [1. 文件操作和网络通信](file:///G:\01.First\node-01\1-%E6%95%99%E5%AD%A6%E8%B5%84%E6%96%99\01-%E7%AC%94%E8%AE%B0\07-%E6%96%87%E4%BB%B6%E6%93%8D%E4%BD%9C%E5%92%8C%E7%BD%91%E7%BB%9C%E9%80%9A%E4%BF%A1.html#1)
  + [1.1. 文件操作](file:///G:\01.First\node-01\1-%E6%95%99%E5%AD%A6%E8%B5%84%E6%96%99\01-%E7%AC%94%E8%AE%B0\07-%E6%96%87%E4%BB%B6%E6%93%8D%E4%BD%9C%E5%92%8C%E7%BD%91%E7%BB%9C%E9%80%9A%E4%BF%A1.html#101)
    - [1.1.1. 为什么要进行文件操作](file:///G:\01.First\node-01\1-%E6%95%99%E5%AD%A6%E8%B5%84%E6%96%99\01-%E7%AC%94%E8%AE%B0\07-%E6%96%87%E4%BB%B6%E6%93%8D%E4%BD%9C%E5%92%8C%E7%BD%91%E7%BB%9C%E9%80%9A%E4%BF%A1.html#10101)
    - [1.1.2. Buffer对象](file:///G:\01.First\node-01\1-%E6%95%99%E5%AD%A6%E8%B5%84%E6%96%99\01-%E7%AC%94%E8%AE%B0\07-%E6%96%87%E4%BB%B6%E6%93%8D%E4%BD%9C%E5%92%8C%E7%BD%91%E7%BB%9C%E9%80%9A%E4%BF%A1.html#10102)
  + [1.2. Node.js的异步编程](file:///G:\01.First\node-01\1-%E6%95%99%E5%AD%A6%E8%B5%84%E6%96%99\01-%E7%AC%94%E8%AE%B0\07-%E6%96%87%E4%BB%B6%E6%93%8D%E4%BD%9C%E5%92%8C%E7%BD%91%E7%BB%9C%E9%80%9A%E4%BF%A1.html#102)
  + [1.3. 网络通信](file:///G:\01.First\node-01\1-%E6%95%99%E5%AD%A6%E8%B5%84%E6%96%99\01-%E7%AC%94%E8%AE%B0\07-%E6%96%87%E4%BB%B6%E6%93%8D%E4%BD%9C%E5%92%8C%E7%BD%91%E7%BB%9C%E9%80%9A%E4%BF%A1.html#103)

# 1. 文件操作和网络通信

## 1.1. 文件操作

### 1.1.1. 为什么要进行文件操作

* 浏览器端JavaScript不能操作文件（安全性考虑）
* Node.js需要文件操作，因为
  + 可以开发命令行的工具,例如：i5ting
  + 开发高效率的web服务器，需要操作文件
* 文件种类繁多，计算机最终存储的时候都是以2进制形式存储
* 存储单位
  + 1byte = 8bit
  + 1k =
  + 1m = 1024k

### 1.1.2. Buffer对象

* Buffer
  + Buffer对象是一个像Array的对象，主要存储字节
  + Node.js作为后端开发，需要处理图片、js、css等文件，图片等资源是以二进制形式存在的，所以在Node.js中增加了一个buffer对象，专门用来存放二进制数据
  + Buffer对象其实就是在内存中申请一块空间，用于存放数据
* Buffer对象的使用(了解)
  + 第一种创建方式
  + //开辟4个字节的空间存储数据
  + let buffer = new Buffer(4);
  + //创建完成后，buffer分配的内存可能有数据
  + //清空buffer对象
  + buffer.fill(0);
  + 第二种创建方式
  + //直接存储数据在buffer中
  + let buffer = new Buffer("hello itcast");
  + 读取文件中遇到的Buffer
  + var fs = require("fs");
  + fs.readFile("01-buffer.js", function (err,content) {
  + //此处的content就是Buffer
  + console.log(content);
  + });
  + fs.readFile("01-buffer.js","utf8", function (err,content) {
  + //指定了编码后，获取到的content是字符串
  + console.log(content);
  + });
  + Buffer支持的编码
    - ascii
    - utf8 默认
    - utf16
    - ucs2
    - base64
    - hex
    - Binary
  + 支持gb2312-第三方包 iconv-lite
  + Buffer的常见方法
    - buf.write() 往buffer写入内容
    - buf.toString([编码]) 把buffer中的内容转换成字符串
    - buf.length 获取buffer的长度
    - Buffer.isEncoding("utf8") 检查支持的编码（不支持gb2312）
* File System
  + 操作文件夹
    - mkdirSync() 同步创建文件夹
    - mkdir() 异步创建文件夹
    - rmdirSync() 同步删除文件夹
    - rmdir() 异步删除文件夹
    - 判断文件是否存在
  + 操作文件
    - writeFile() 写入文件，会覆盖以前的内容
    - appendFile() 追加文件
    - readFile() 读取文件
    - rename() 更改文件名
    - unlink() 删除文件
    - stat() 获取文件的相关属性
    - watchFile() 监控文件或文件夹内文件的改变
  + 练习：监视文件abc/abc.txt，如果文件发生变化把内容拷贝到target/123.txt
  + 拷贝文件：
    - 之前的方式只能拷贝小文件
    - 拷贝大文件
  + 文件流操作
    - createReadStream()
    - createWriteStream()
    - 拷贝大文件带进度

```javascript "use strict"

const fs = require("fs");

let sourceFile ="xdy.avi"; let targetFile = "123.itcast";

//文件流读写 进行大文件复制 //创建读流对象 let rs = fs.createReadStream(sourceFile); //创建写流对象 let ws = fs.createWriteStream(targetFile);

//文件大小

let fileSize = fs.statSync(sourceFile).size;

//总共写入的大小

let writeTotal = 0;

//监听data事件

rs.on("data", chunk => {

//每次读到的数据 buffer对象

writeTotal += chunk.length;

//把chunk写入目标文件

ws.write(chunk);

console.log(writeTotal/fileSize \* 100);

});

//读取完毕

rs.on("end", ()=>{

console.log("文件读取完毕");

//写入结束

ws.end(()=>{

console.log("文件写入完毕");

});

});

```

- 管道拷贝大文件

```javascript

"use strict"

const fs = require("fs");

let sourceFile ="xdy.avi";

let targetFile = "123.itcast";

let rs = fs.createReadStream(sourceFile);

let ws = fs.createWriteStream(targetFile);

rs.pipe(ws);

```

## 1.2. Node.js的异步编程

* 同步和异步的概念
* 非阻塞式I/O
* 事件驱动
* 回调函数
* 监视js的变化重新运行的插件 npm install nodemon -g
  + 使用 nodemon js文件路径

## 1.3. 网络通信

* 网络的概念
* IP和端口的概念
* Socket的概念
* 打开telnet

程序和功能-->

* 自己开发一个小服务器

"use strict";

const net = require("net");

//创建server对象

let server = net.createServer();

//监听连接到服务器上的客户端

server.on("connection", (socket) => {

console.log("haha");

//服务器的socket

//向客户端发送消息

socket.write("welcome to my server");

socket.on("data", (content) => {

//服务器发送的内容。buffer

console.log(content.toString());

});

});

//监听端口

server.listen(8888,()=>{

console.log("正在监听...");

});

* 自己开发的客户端

"use strict";

var net = require("net");

let port = 8888;

let ip = "127.0.0.1";

//连接服务器

let client = net.createConnection(port,ip,()=>{

console.log("连接服务器成功");

//向服务器发送消息

client.write("你好服务器");

});

//接收服务器的消息

client.on("data", (content) => {

//服务器发送的内容。buffer

console.log(content.toString());

});

* 处理错误

//注册error事件

socket.on("error",()=>{

console.log("客户端下线");

})

* 练习：聊天机器人
* 练习：聊天室
* Table of Content
* [1. HTTP核心模块](file:///G:\01.First\node-01\1-%E6%95%99%E5%AD%A6%E8%B5%84%E6%96%99\01-%E7%AC%94%E8%AE%B0\08-http.html#1)
  + [1.1. HTTP协议](file:///G:\01.First\node-01\1-%E6%95%99%E5%AD%A6%E8%B5%84%E6%96%99\01-%E7%AC%94%E8%AE%B0\08-http.html#101)
  + [1.2. HTTP模块](file:///G:\01.First\node-01\1-%E6%95%99%E5%AD%A6%E8%B5%84%E6%96%99\01-%E7%AC%94%E8%AE%B0\08-http.html#102)
  + [1.3. xtpl模板的使用](file:///G:\01.First\node-01\1-%E6%95%99%E5%AD%A6%E8%B5%84%E6%96%99\01-%E7%AC%94%E8%AE%B0\08-http.html#103)
  + [1.4. Express第三方框架](file:///G:\01.First\node-01\1-%E6%95%99%E5%AD%A6%E8%B5%84%E6%96%99\01-%E7%AC%94%E8%AE%B0\08-http.html#104)

# 1. HTTP核心模块

## 1.1. HTTP协议

* 请求报文
* 响应报文

## 1.2. HTTP模块

* 使用之前的方式拼接响应报文

let msg = "<h1>hello world</h1>";

//构造响应报文

let responseText = "HTTP/1.1 200 OK\r\n";

responseText += "Server: mynodeserver 1.0\r\n";

responseText += "Content-Length:"+ msg.length +"\r\n";

responseText += "Content-Type: text/html;charset=utf-8\r\n";

responseText += "\r\n";

responseText += msg;

socket.write(responseText);

socket.end();

* 使用http模块简化web服务器的开发

"use strict";

const http = require("http");

let server = http.createServer((req,res)=>{

//获取请求头 req.headers

//获取http的版本 req.httpVersion

//请求的方法 req.method

//请求的路径 req.url

console.log(req.headers);

console.log(req.httpVersion);

console.log(req.method);

console.log(req.url);

res.write("hhhhhh中文");

res.end();

}).listen(8888,()=>{

console.log("开始监听...");

});

* 请求对象的常用属性
  + req.headers 获取请求头
  + req.method 请求的方法
  + req.httpVersion 获取http的版本
  + req.url 请求的路径
* 响应对象的常用方法
  + res.end() 结束请求
  + res.write() 发送内容
  + res.setHeader("Content-Type","text/html;charset=utf-8");
* 异常处理

请求会有很多很多，如果一个请求出错会导致服务器挂掉，别的请求无法访问服务器.

所以再做服务器开发的时候要保证如果一个请求出错，不影响其他请求

如何处理异常：

try{

}catch(err){

console.log("出错了:" + err);

//此处要写日志，让程序员发现错误，进行修改

}

* 实现自己的模板，改变页面数据

const http = require("http");

const fs= require("fs");

const path = require("path");

let server = http.createServer((req,res)=>{

if(req.url.trim() === "/" || req.url.includes("index.html")) {

//读取模板html文件

fs.readFile("./template/tmp.html", (err, content)=>{

let tmp = content.toString();

let dataArr = require("./data.json");

//变量数据，替换模板

let array = [];

for (var i = 0; i < dataArr.length; i++) {

let item = dataArr[i];

array.push("<li>"+item+"</li>");

}

tmp = tmp.replace("${list}",array.join(""));

tmp = tmp.replace("${title}","列表");

res.write(tmp);

//结束请求

res.end();

});

}else if(req.url.includes(".css")){

//加载css

fs.readFile(path.join("template",req.url) ,(err,content)=>{

res.setHeader("Content-Type","text/css;charset=utf8");

res.end(content);

});

}else if(req.url.includes(".jpg")) {

//加载jpg的图片

fs.readFile(path.join("image",req.url) ,(err,content)=>{

res.setHeader("Content-Type","image/jpeg");

res.end(content);

});

}else{

res.end("你闹啥呢");

}

}).listen(8888,()=>{

console.log("开启监听...");

});

## 1.3. xtpl模板的使用

* 官网
  + <https://github.com/xtemplate/xtpl>
* 安装
  + npm install xtpl xtemplate --save
* xtemplate模板语法
  + <https://github.com/xtemplate/xtemplate/blob/master/docs/syntax-cn.md>
* 使用

var xtpl = require('xtpl');

xtpl.renderFile('./x.xtpl',{

x:1

},function(error,content){

});

* 音乐播放器

## 1.4. Express第三方框架

* 中文文档
  + <http://www.expressjs.com.cn/4x/api.html>
* express的安装
  + npm install express
* 基本使用
* 代码结构
* "use strict";
* const express = require("express");
* let app = express();
* app.get("/", (req,res)=>{
* res.setHeader("Content-Type","text/html;charset=utf8");
* res.write("我的天哪");
* res.end();
* });
* app.listen(8888,()=>{
* console.log("开始监听...");
* });
* express的get

app.get("/", (req, res) => {

res.setHeader("Content-Type", "text/html;charset=utf-8");

res.write("欢迎");

res.end();

});

app.get("/photo", (req, res) => {

res.write("<h1>没有照片</h1>");

res.end();

});

* express的get获取参数 方式1

//请求地址的形式 http://127.0.0.1:8888/post?id=1&token=abcd

app.get("/post", (req, res) => {

//解析url中的参数 方式1

//let query = url.parse(req.url, true).query;

//console.log(query.id);

//console.log(query.token);

//解析url中的参数 方式2

//console.log(req.query);

//console.log(req.query.id);

////过期的方法，不推荐使用

//console.log(req.param("id"));

res.write("ceshi");

res.end();

});

* express的get获取参数 方式2

//请求地址的形式 http://127.0.0.1:8888/zs/1002.html

app.get("/:username/:id.html", (req, res) => {

//获取参数

console.log(req.params);

console.log(req.params.username);

res.send("haha");

});

* express的通配符,next跳到下一个路由

//优先跟注册地址注册的顺序有关

app.get("/\*", (req, res, next) => {

res.setHeader("Content-Type", "text/html;charset=utf-8");

res.write("通配符哈<br>");

next();

});

* express的post和all
* send方法
* 简化读取静态文件

//app.use(express.static("template"));

app.use("/public", express.static("template"));

* 路由router

"use strict";

const express = require("express");

let app = express();

let mainRouter = express.Router();

mainRouter.get("/main/list", (req, res) => {

res.end("/main/list");

});

mainRouter.get("/main/haha", (req, res) => {

res.end("/main/haha");

});

app.use("/", mainRouter);

app.listen(8888, ()=>{

console.log("正在加载...");

});

* 把不同功能的router放到不同的位置
* 重做音乐播放器
* Table of Content
* [1. 数据库MySQL](file:///G:\01.First\node-01\1-%E6%95%99%E5%AD%A6%E8%B5%84%E6%96%99\01-%E7%AC%94%E8%AE%B0\09-mysql.html#1)
  + [1.1. 数据库的相关概念](file:///G:\01.First\node-01\1-%E6%95%99%E5%AD%A6%E8%B5%84%E6%96%99\01-%E7%AC%94%E8%AE%B0\09-mysql.html#101)
  + [1.2. Node.js中操作MySQL](file:///G:\01.First\node-01\1-%E6%95%99%E5%AD%A6%E8%B5%84%E6%96%99\01-%E7%AC%94%E8%AE%B0\09-mysql.html#102)

# 1. 数据库MySQL

## 1.1. 数据库的相关概念

* 什么是数据库
  + 存储数据的仓库
  + 使用数据库管理数据的好处
    - 方便存储、查询、修改、删除
  + 关系型数据库
  + - MySQL
  + - Sql Server
  + - Orcale
  + - SQLite
* 数据表
  + 二维的表，类似于Excel表
  + 由行和列组成，列：字段，行：记录
  + 字段的类型
    - int 整数、double 浮点数、varchar(255) 字符串、text 文本、datetime 日期、float 浮点数
* MySQL的服务器
* MySQL的客户端
  + 使用navicat创建数据库
  + 使用navicat创建表，添加数据
  + 使用SQL语句操作数据
    - 插入数据 INSERT INTO users (uname,upwd,uqq) values('zs','123',"12345")
    - 修改数据 UPDATE users SET uname='zsxxx' WHERE uid=1
    - 删除数据 DELETE FROM users WHERE uid = 1
    - 查询数据 SELECT \* FROM users

SELECT \* FROM users WHERE uname='zs' AND uname='000000'

SELECT \* FROM users WHERE uname='zs' OR uname='ls'

SELECT \* FROM users WHERE uname in ('zs','ls')

SELECT \* FROM users WHERE uname like '%s%'

SELECT \* FROM users WHERE uage > 18 ORDER by desc

SELECT \* FROM users ORDER BY DESC

SELECT \* FROM users LIMIT 3 //取前3条数据

SELECT \* FROM users ORDER BY DESC LIMIT 3

* + - 分页 //跳过3条，取2条 SELECT \* FROM users ORDER BY DESC LIMIT 3,2

SELECT COUNT(\*) FROM users

## 1.2. Node.js中操作MySQL

* 使用mysql操作
  + 安装 npm install mysql
  + 使用
  + //设置数据库参数
  + var connection = mysql.createConnection({
  + host : 'localhost',
  + user : 'root',
  + password : '',
  + database : 'musicsys'
  + });
  + //创建连接
  + connection.connect();
  + //增加一条数据
  + let data = {uname: "hahaha", upwd: "xxxx", uqq: "11111", uage: 18, create: Date.now()};
  + connection.query("insert into users set ?", data, (err, result) => {
  + console.log(result);
  + });
  + //查询数据
  + connection.query("select \* from users", (err, rows, fields) => {
  + console.log(rows);
  + });
  + //关闭连接
  + connection.end();
* 安装node-orm2，安装 orm的依赖 mysql
  + <https://github.com/dresende/node-orm2>
  + npm install orm
  + npm install mysql
* 查询数据表users的所有数据

//步骤一： 在服务器启动的时候连接数据库

app.use(orm.express("mysql://root:@localhost:3306/musicSys",{

define: (db, models, next) => {

models.users = db.define("users",{

uid:{type:"serial", key:true},

uname: String,

upwd: String,

uqq: String,

uage: Number

});

next();

}

}));

//步骤二：当请求到达后查询数据库

app.get("/", (req, res)=>{

let where = {};

req.models.users.find(where, (err, data)=>{

//data是对象

res.send(data);

});

});

* 条件查询
  1. 查询所有数据
  2. 查询id=3的用户 方式1
  3. 查询id=3的用户 方式2
  4. 条件查询 uname=zs && uage=100
  5. 条件查询 uid=4 || uage=5
  6. 模糊查询 uanme like '%f%'
  7. 大于 uage > 18
  8. 排序 A -- asc升序 Z -- desc 降序
  9. 取最前面的两条数据 limit
  10. 分页的基础 limit 4, 2
  11. 分页
  12. 获取总共有多少条数据，第一个参数是条件
  13. 获取对应的数据表 的数据

//1 查询所有数据

//req.models.user.find({}, (err, users) => {

// res.send(users);

//});

//获取url参数

//let id = req.query.id;

//2 查询id=3的用户 方式1

//req.models.user.get(id, (err, user) => {

// //console.log(err);

// res.send(user);

//});

//3 查询id=3的用户 方式2 find 返回的是数组

//req.models.user.find({uid: id}, (err, users) => {

// console.log(id);

// console.log(err);

// res.send(users);

//});

//4 条件查询 uname=zs && uage=100

//let where = { uname : "zs", uage : 100 };

//req.models.user.find(where, (err, users) => {

// res.send(users);

//});

//5 条件查询 uid=4 || uage=5

//let where = {uid: [4, 5]};

//req.models.user.find(where, (err, users) => {

// res.send(users);

//});

//6 模糊查询 uanme like '%f%'

//let where = {uname: orm.like('%f%') };

//req.models.user.find(where, (err, users) => {

// res.send(users);

//});

//7 大于 uage > 18

//let where = {uage: orm.gt(18)};

//req.models.user.find(where, (err, users) => {

// res.send(users);

//});

//8 排序 A -- asc升序 Z -- desc 降序

//req.models.user.find({}, ["uid", "Z"], (err, users) => {

// res.send(users);

//});

//9 取最前面的两条数据 limit

//req.models.user.find({}, {limit: 2}, ["uid", "Z"], (err, users) => {

// res.send(users);

//});

//10 分页的基础 limit 4, 2 offset: 2 跳过2条 limit: 2 取几条

//req.models.user.find({}, {offset: 4, limit: 2}, ["uid", "Z"], (err, users) => {

// res.send(users);

//});

//11 分页 pageIndex = 0; 页码 pageCount = 2 页容量

//

//let pageIndex = 1; //页码

//let pageCount = 2; //页容量

//req.models.user.find({}, {offset: (pageIndex - 1) \* pageCount, limit: pageCount}, ["uid", "Z"], (err, users) => {

// res.send(users);

//});

//12 获取总共有多少条数据，第一个参数是条件

//req.models.user.count({}, (err, count) => {

// res.send({count: count});

//});

//13 获取对应的数据表 的数据

//req.models.user.exists({}, (err, exists) => {

// res.send({exists: exists});

//});

* 增删改

//增加一条数据

//req.models.users.create({

// uname:"xxx",

// upwd:"0000",

// uqq:"9999",

// uage:30

//},(err,user)=>{

// console.log("创建成功");

// console.log("user");

//

// res.end(JSON.stringify(user));

//});

//修改一条数据

//根据主键获取一条数据

//req.models.users.get(3,(err,user)=>{

// user.upwd = "woxxxxx";

// //保存

// user.save();

//})

//修改数据2

//req.models.users.find({uname:"zs"},(err,users)=>{

// //users是数组

// users[0].uage = 100;

// users[0].save();

//})

//删除

//req.models.users.get(5,(err,user)=>{

// user.remove((err)=>{

// //item当前删除的数据

// res.end("删除成功");

// })

//})

* 执行sql语句

app.get("/", (req, res)=>{

req.db.driver.execQuery("select \* from users",(err, data)=>{

res.end(JSON.stringify(data));

});

});

* Table of Content
* [1. Node项目](file:///G:\01.First\node-01\1-%E6%95%99%E5%AD%A6%E8%B5%84%E6%96%99\01-%E7%AC%94%E8%AE%B0\10-node%E9%A1%B9%E7%9B%AE.html#1)
  + [1.1. 项目开发流程](file:///G:\01.First\node-01\1-%E6%95%99%E5%AD%A6%E8%B5%84%E6%96%99\01-%E7%AC%94%E8%AE%B0\10-node%E9%A1%B9%E7%9B%AE.html#101)
  + [1.2. 搭建项目结构](file:///G:\01.First\node-01\1-%E6%95%99%E5%AD%A6%E8%B5%84%E6%96%99\01-%E7%AC%94%E8%AE%B0\10-node%E9%A1%B9%E7%9B%AE.html#102)
  + [1.3. glob 获取目录下的文件路径](file:///G:\01.First\node-01\1-%E6%95%99%E5%AD%A6%E8%B5%84%E6%96%99\01-%E7%AC%94%E8%AE%B0\10-node%E9%A1%B9%E7%9B%AE.html#103)
  + [1.4. 加载所有路由模块](file:///G:\01.First\node-01\1-%E6%95%99%E5%AD%A6%E8%B5%84%E6%96%99\01-%E7%AC%94%E8%AE%B0\10-node%E9%A1%B9%E7%9B%AE.html#104)
  + [1.5. 错误处理](file:///G:\01.First\node-01\1-%E6%95%99%E5%AD%A6%E8%B5%84%E6%96%99\01-%E7%AC%94%E8%AE%B0\10-node%E9%A1%B9%E7%9B%AE.html#105)
  + [1.6. 设置模板，加载首页](file:///G:\01.First\node-01\1-%E6%95%99%E5%AD%A6%E8%B5%84%E6%96%99\01-%E7%AC%94%E8%AE%B0\10-node%E9%A1%B9%E7%9B%AE.html#106)
  + [1.7. 从数据库中加载数据](file:///G:\01.First\node-01\1-%E6%95%99%E5%AD%A6%E8%B5%84%E6%96%99\01-%E7%AC%94%E8%AE%B0\10-node%E9%A1%B9%E7%9B%AE.html#107)
  + [1.8. 首页渲染数据](file:///G:\01.First\node-01\1-%E6%95%99%E5%AD%A6%E8%B5%84%E6%96%99\01-%E7%AC%94%E8%AE%B0\10-node%E9%A1%B9%E7%9B%AE.html#108)
  + [1.9. 获取帖子的详细内容](file:///G:\01.First\node-01\1-%E6%95%99%E5%AD%A6%E8%B5%84%E6%96%99\01-%E7%AC%94%E8%AE%B0\10-node%E9%A1%B9%E7%9B%AE.html#109)
  + [1.10. 获取评论列表](file:///G:\01.First\node-01\1-%E6%95%99%E5%AD%A6%E8%B5%84%E6%96%99\01-%E7%AC%94%E8%AE%B0\10-node%E9%A1%B9%E7%9B%AE.html#110)
  + [1.11. 支持、反对](file:///G:\01.First\node-01\1-%E6%95%99%E5%AD%A6%E8%B5%84%E6%96%99\01-%E7%AC%94%E8%AE%B0\10-node%E9%A1%B9%E7%9B%AE.html#111)
  + [1.12. 发表评论](file:///G:\01.First\node-01\1-%E6%95%99%E5%AD%A6%E8%B5%84%E6%96%99\01-%E7%AC%94%E8%AE%B0\10-node%E9%A1%B9%E7%9B%AE.html#112)

# 1. Node项目

## 1.1. 项目开发流程

* 项目需求

自有产品 --> 产品部门 外包项目 --> 需求部门

-->

需求文档 -->

产品经理：交互稿：原型设计 -->

UI设计设：UI图：PSD图 -->

前端：根据交互稿和PSD图 实现html和里面的特效 -->

后端：实现数据库和后端功能和接口的实现 -->

测试人员：测试功能是否正常，记录到bug反馈系统 -->

项目发布到生成环境

## 1.2. 搭建项目结构

* dist
* src
  + controller
  + views
  + model
  + route
  + www
  + app.js
* package.json
* README.md

## 1.3. glob 获取目录下的文件路径

* 安装
  + npm install glob --save
* 使用

var glob = require("glob")

glob("\*\*/\*.js", function (er, files) {

// files is an array of filenames.

})

## 1.4. 加载所有路由模块

* 方式一：

//读取静态资源 访问路径 www/index.html

app.use("/www", express.static(path.join(\_\_dirname,"www")));

//导入路由模块

const home = require("./route/home.js");

const sign = require("./route/sign.js");

app.use("/home",home);

app.use("/sign",sign);

* 方式二：

//加载所有路由模块

//const glob = require("glob");

let files = glob.sync("./route/\*.js");

//files文件的路径数组

//files.forEach((item)=>{

// const route = require(item);

// //获取文件的名字

// let name = path.basename(item, ".js");

// app.use("/"+name,route);

//});

let files = glob.sync("./route/\*.js");

//files文件的路径数组

files.forEach((item)=>{

const route = require(item);

app.use(route.prefix, route);

});

## 1.5. 错误处理

//处理错误--请求的文件不存在-404

app.use((req,res,next)=>{

//res.status(404).send("4041111");

res.status(404);

res.redirect("/404/index.html");

});

//处理错误--服务器错误-500

app.use((err,req,res,next)=>{

res.status(500);

res.redirect("/www/500/index.html");

});

## 1.6. 设置模板，加载首页

const xtpl = require("xtpl");

app.set('views','./views');

app.set('view engine', 'html');

app.engine('html',xtpl.renderFile);

//读取静态资源 访问路径 http://127.0.0.1:8888/404/index.html

app.use(express.static(path.join(\_\_dirname,"www")));

//控制器中

const controller = module.exports;

controller.getindex = (req,res)=>{

res.render('blog/index',{array: data},(err,html)=>{

res.send(html);

});

};

//路由中

const home = module.exports = express.Router();

home.prefix = "/home";

const controller = require("../controller/home");

home.get("/index",controller.getindex);

## 1.7. 从数据库中加载数据

* 导入包 mysql和orm
  + npm install orm --save
  + npm install mysql --save
* model中创建对象和表的映射

module.exports = (db, models)=>{

models.post = db.define("posts", {

id: { type: 'serial', key: true },

slug: String,

title: String,

excerpt: String,

content: String,

status: String,

comment\_status: String,

comment\_count: Number,

view\_count: Number,

created: Date,

user\_id: Number,

parent\_id: Number,

type: String

});

models.user = db.define("users",{

id: { type: 'serial', key: true },

slug: String,

username: String,

password: String,

nickname: String,

email: String,

status: String,

created: Date,

role: String,

token: String

});

models.comment = db.define("comments",{

id: { type: 'serial', key: true },

author: String,

author\_email: String,

author\_ip: String,

content: String,

status: String,

support: Number,

oppose: Number,

created: Date,

post\_id:Number,

parent\_id: Number,

user\_id: Number

});

}

* app.js中创建连接，导入映射

//设置orm

const orm = require("orm");

app.use(orm.express("mysql://root:@localhost:3306/myblog",{

define: function(db, models, next){

//

let m = require("./model/model.js");

m(db,models);

next();

}

}));

* 控制器中获取数据

controller.getindex = (req,res)=>{

let where = {};

req.models.post.find(where,(err,data)=>{

if(err)throw err;

//获取到所有post

console.log(data);

res.render('blog/index',{array: data},(err,html)=>{

res.end(html);

});

});

};

## 1.8. 首页渲染数据

controller.getindex = (req, res)=>{

let where = {};

req.models.post.find(where,{limit:5},["id","A"],(err,data)=>{

if(err)throw err;

//获取到所有post

//把日期转换一下

data.forEach((item)=>{

item.created = item.created.toDateString();

});

res.render('blog/index',{array: data},(err,html)=>{

res.send(html);

});

});

};

## 1.9. 获取帖子的详细内容

* 设置路由

post.get("/:userid/:id", controller.getpost);

* 设置控制器

//获取帖子

controller.getpost = (req, res)=>{

//获取参数

let userid = req.params.userid;

//检查用户是否存在

req.models.user.find({id:userid}, (err, user)=>{

if(err)throw err;

if(!user || user.length == 0) {

res.status(404);

res.redirect("/404/index.html");

}

});

//获取帖子

let id = req.params.id;

req.models.post.find({user\_id:userid,id:id},(err,post)=>{

if(err)throw err;

if(!post || post.length == 0) {

res.status(404);

res.redirect("/404/index.html");

return;

}

post[0].created = post[0].created.toDateString();

//输出页面

res.render('blog/post',{post:post[0]},(err,html)=>{

res.send(html);

});

});

}

* 设置模板
  + 模板中使用 {{{ }}} 不转移内容的html

## 1.10. 获取评论列表

* 设置路由，传递帖子的id，新建comment.js路由

post.prefix = "/comment";

const controller = require("../controller/comment");

//id是帖子的id

post.get("/:id", controller.getcomment);

* 设置控制器

controller.getcomment = (req, res)=>{

//根据帖子的id 查询该帖子下的所有评论

let id = req.params.id;

//查询某个帖子下的所有评论

req.models.comment.find({post\_id: id}, (err, datas) => {

if(err)throw err;

res.send(datas);

});

}

* 页面，发送ajax请求，设置内容

## 1.11. 支持、反对

* 设置路由
* 设置控制器
* 页面发送ajax请求
* 根据ip判断是否已经发过评论

controller.support = (req, res) => {

//获取客户端的ip

//console.log(req.socket.remoteAddress);

req.models.comment.find({author\_ip:req.socket.remoteAddress}, (err, data)=>{

if(data && data.length>0) {

let data = {d:-1};

res.send(data);

}else{

//支持 id 是评论的id

req.models.comment.get(req.params.id,(err,comment)=>{

comment.support++;

//保存

comment.save();

let data = {d:comment.support};

res.send(data);

})

}

});

}

## 1.12. 发表评论

* 获取post过来的参数
  + 安装body-parser npm install body-parser --save

const bodyparser = require("body-parser");

//处理application/x-www-form-urlencoded的 parser

app.use(bodyparser.urlencoded({ extended: false }));

//获取post来的数据

req.body

* 路由
* 控制器

//发布评论

controller.create = (req, res) => {

req.models.comment.create({

author: req.body.name,

author\_email: req.body.email,

content: req.body.comment,

author\_ip: req.socket.remoteAddress,

post\_id: req.body.id,

support: 0,

oppose: 0

},(err, comment)=> {

if(err) throw err;

//查询某个帖子下的所有评论

req.models.comment.find({post\_id: req.body.id}, (err, datas) => {

if(err)throw err;

//修改时间的格式

datas.forEach((item)=>{

item.created = item.created.toDateString();

});

res.send(datas);

});

})

}

* 页面

//发布评论

$(".btn").click(function () {

var json = $("#f1").serialize();

$.post("/comment/create",json, function (datas) {

if(datas) {

//使用layer插件弹出层

//layer.alert("发表成功");

layer.msg("发布成功");

var html = template("tpl",{datas:datas});

$("#comment").html(html);

}else{

layer.msg("发布失败");

}

});

return false;

});

* Table of Content
* [1. Node项目](file:///G:\01.First\node-01\1-%E6%95%99%E5%AD%A6%E8%B5%84%E6%96%99\01-%E7%AC%94%E8%AE%B0\11-node%E9%A1%B9%E7%9B%AE2.html#1)
  + [1.1. 后台管理](file:///G:\01.First\node-01\1-%E6%95%99%E5%AD%A6%E8%B5%84%E6%96%99\01-%E7%AC%94%E8%AE%B0\11-node%E9%A1%B9%E7%9B%AE2.html#101)
  + [1.2. Cookie](file:///G:\01.First\node-01\1-%E6%95%99%E5%AD%A6%E8%B5%84%E6%96%99\01-%E7%AC%94%E8%AE%B0\11-node%E9%A1%B9%E7%9B%AE2.html#102)
  + [1.3. Session](file:///G:\01.First\node-01\1-%E6%95%99%E5%AD%A6%E8%B5%84%E6%96%99\01-%E7%AC%94%E8%AE%B0\11-node%E9%A1%B9%E7%9B%AE2.html#103)
  + [1.4. 登录](file:///G:\01.First\node-01\1-%E6%95%99%E5%AD%A6%E8%B5%84%E6%96%99\01-%E7%AC%94%E8%AE%B0\11-node%E9%A1%B9%E7%9B%AE2.html#104)
  + [1.5. 管理首页](file:///G:\01.First\node-01\1-%E6%95%99%E5%AD%A6%E8%B5%84%E6%96%99\01-%E7%AC%94%E8%AE%B0\11-node%E9%A1%B9%E7%9B%AE2.html#105)
  + [1.6. 模板重构](file:///G:\01.First\node-01\1-%E6%95%99%E5%AD%A6%E8%B5%84%E6%96%99\01-%E7%AC%94%E8%AE%B0\11-node%E9%A1%B9%E7%9B%AE2.html#106)
  + [1.7. 用户管理](file:///G:\01.First\node-01\1-%E6%95%99%E5%AD%A6%E8%B5%84%E6%96%99\01-%E7%AC%94%E8%AE%B0\11-node%E9%A1%B9%E7%9B%AE2.html#107)
  + [1.8. UEditor](file:///G:\01.First\node-01\1-%E6%95%99%E5%AD%A6%E8%B5%84%E6%96%99\01-%E7%AC%94%E8%AE%B0\11-node%E9%A1%B9%E7%9B%AE2.html#108)
  + [1.9. 分页](file:///G:\01.First\node-01\1-%E6%95%99%E5%AD%A6%E8%B5%84%E6%96%99\01-%E7%AC%94%E8%AE%B0\11-node%E9%A1%B9%E7%9B%AE2.html#109)

# 1. Node项目

## 1.1. 后台管理

## 1.2. Cookie

* 在客户端保存数据的一种方式
  + 保存少量数据4k左右，客户端和服务器都可以读写cookie
  + 如果只是存储数据，可以使用localStorage替换，可以存储5M
  + 不可以跨域
* 服务端写Cookie
  + 存储在内存
  + res.setHeader("Set-Cookie","name=zhangsan1");
  + 存储在文件
  + res.setHeader("Set-Cookie","name=zhangsan1; path=/; domain=.tt.com; max-age=180");
  + 参数
    - path 设置路径，哪个路径可以访问cookie
    - expires 设置过期时间，是日期，服务器的日期可能和客户端的日期不一样
    - max-age 设置过期时间，秒，第一次访问的时间开始
    - domain 设置可以跨子域访问cookie
* 封装读取cookie

let cookie = module.exports;

cookie.getCookie = (req,key)=>{

var Cookies = {};

req.headers.cookie && req.headers.cookie.split(';').forEach(function( Cookie ) {

var parts = Cookie.split('=');

Cookies[ parts[ 0 ].trim() ] = ( parts[ 1 ] || '' ).trim();

});

return Cookies[key];

}

## 1.3. Session

* 保持会话状态
* 服务端+客户端实现
* 模拟Session的实现

const express = require("express");

const cookie = require("../tools/cookie");

let demo = module.exports = express.Router();

demo.prefix = "/session";

let session = {sids:[]};

//{sids:["1111","222"], "1111": {name:zhangsan},}

demo.get("/get", (req, res) => {

res.send(req.headers.cookie);

});

demo.get("/login/:name",(req, res) => {

let sid =cookie.getCookie(req, "sid");

//第一次登录，判断是否有sid，如果没有sid生成一个，并跳转会当前页面

if(!sid) {

//唯一标示一个客户端

sid = Math.random();

//模拟登录

res.setHeader("Set-Cookie","sid="+ sid + "; path=/session");

session.sids.push(sid);

res.redirect(req.originalUrl);

//console.log(req.originalUrl);

}else{

//模拟登录

res.setHeader("Set-Cookie","name="+req.params.name+"; path=/session");

session[sid] = {name: req.params.name};

res.send("<a href='/session'>到首页</a>");

}

});

demo.get("/",(req, res) => {

//req.headers.cookie

//判断是否登录成功，先获取sid，根据sid获取当前的登录信息

let sid =cookie.getCookie(req, "sid");

//根据sid获取 登录信息

let obj = session[sid];

//模拟权限的判断

if(obj && obj.name) {

res.send("后台首页")

}else{

res.send("请先登录");

}

});

* 使用express-session简化session开发
  + 安装 npm install express-session --save
  + 使用，在设置路由之前
  + app.use(session({
  + secret: '12345',
  + resave: true,
  + saveUninitialized: true,
  + //maxage 单位毫秒
  + cookie: { maxAge: 3600000 }
  + }))
  + //然后就可以使用session
  + req.session.userinfo = user;
  + 参数
    - secret：用来对session数据进行加密的字符串.这个属性值为必须指定的属性。
    - name：表示cookie的name，默认cookie的name是：connect.sid。
    - maxAge：cookie过期时间，毫秒。
    - resave：是指每次请求都重新设置session cookie，假设你的cookie是6000毫秒过期，每次请求都会再设置6000毫秒。
    - saveUninitialized： 是指无论有没有session cookie，每次请求都设置个session cookie ，默认给个标示为 connect.sid。

## 1.4. 登录

* 路由
* 控制器
* 登录成功转到首页
* 添加session
* md5加密
  + 安装md5的包
    - npm install blueimp-md5 --save
    - 服务端使用代码
    - //4297f44b13955235245b2497399d7a93
    - let md5 = require("blueimp-md5");
    - console.log(md5("123123"));
    - 客户端使用
    - <script src="/assets/vendor/md5.js"></script>
    - console.log(md5("123123"));

## 1.5. 管理首页

* 管理首页获取数据

account.getindex = (req, res) => {

if(req.session.userinfo) {

//获取帖子

req.models.post.find({},{limit:5},["id","A"],(err,blogs)=>{

if(err) throw err;

res.render("admin/index",{user:req.session.userinfo,blogs:blogs}, (err,html)=>{

if(err) throw err;

res.send(html);

});

})

}else{

res.send("<script>alert('请先登录!'); location.href='/admin/login'</script>");

}

}

* 修改首页模板
* 帖子的列表
  + 删除帖子
  + 修改帖子

## 1.6. 模板重构

* 父亲模板 parent.html

<!doctype html>

<html>

<head>

<meta name="charset" content="utf-8" />

<title>{{title}}</title>

{{{block ("head")}}}

</head>

<body>

{{{block ("body")}}}

</body>

</html>

* 子模板 index.html

{{extend ("./parent")}}

{{#block ("head")}}

<link type="text/css" href="test.css" rev="stylesheet" rel="stylesheet"/>

{{/block}}

{{#block ("body")}}

<h2>{{title}}</h2>

{{/block}}

* 渲染和以前一样

## 1.7. 用户管理

## 1.8. UEditor

* 获取ueditor
* 引入js

<script type="text/javascript" charset="utf-8" src="ueditor.config.js"></script>

<script type="text/javascript" charset="utf-8" src="ueditor.all.min.js"> </script>

<!--建议手动加在语言，避免在ie下有时因为加载语言失败导致编辑器加载失败-->

<!--这里加载的语言文件会覆盖你在配置项目里添加的语言类型，比如你在配置项目里配置的是英文，这里加载的中文，那最后就是中文-->

<script type="text/javascript" charset="utf-8" src="lang/zh-cn/zh-cn.js"></script>

* 页面上添加

<script type="text/plain" id="content" style="height:300px"></script>

* 配置ueditor

var ue = UE.getEditor('content',{

//这里可以选择自己需要的工具按钮名称,此处仅选择如下五个

toolbars:[['Source', 'Undo', 'Redo','bold']],

//focus时自动清空初始化时的内容

autoClearinitialContent:true,

//关闭字数统计

wordCount:false,

//关闭elementPath

elementPathEnabled:false,

//更多其他参数，请参考ueditor.config.js中的配置项

});

* 获取内容

var content = ue.getContent();

* 设置内容

ue.setContent('欢迎使用ueditor', isAppendTo);

## 1.9. 分页

* Table of Content
* [1. 项目搭建步骤](file:///G:\01.First\node-01\1-%E6%95%99%E5%AD%A6%E8%B5%84%E6%96%99\01-%E7%AC%94%E8%AE%B0\12-%E9%A1%B9%E7%9B%AE%E6%90%AD%E5%BB%BA%E6%AD%A5%E9%AA%A4.html#1)
  + [1.1. 搭建项目目录结构](file:///G:\01.First\node-01\1-%E6%95%99%E5%AD%A6%E8%B5%84%E6%96%99\01-%E7%AC%94%E8%AE%B0\12-%E9%A1%B9%E7%9B%AE%E6%90%AD%E5%BB%BA%E6%AD%A5%E9%AA%A4.html#101)
  + [1.2. 安装express](file:///G:\01.First\node-01\1-%E6%95%99%E5%AD%A6%E8%B5%84%E6%96%99\01-%E7%AC%94%E8%AE%B0\12-%E9%A1%B9%E7%9B%AE%E6%90%AD%E5%BB%BA%E6%AD%A5%E9%AA%A4.html#102)
  + [1.3. 新建app.js 实现监听](file:///G:\01.First\node-01\1-%E6%95%99%E5%AD%A6%E8%B5%84%E6%96%99\01-%E7%AC%94%E8%AE%B0\12-%E9%A1%B9%E7%9B%AE%E6%90%AD%E5%BB%BA%E6%AD%A5%E9%AA%A4.html#103)
  + [1.4. 添加路由，统一的方式添加路由](file:///G:\01.First\node-01\1-%E6%95%99%E5%AD%A6%E8%B5%84%E6%96%99\01-%E7%AC%94%E8%AE%B0\12-%E9%A1%B9%E7%9B%AE%E6%90%AD%E5%BB%BA%E6%AD%A5%E9%AA%A4.html#104)
* 添加glob模块
  + [1.5. 统一处理错误404 500](file:///G:\01.First\node-01\1-%E6%95%99%E5%AD%A6%E8%B5%84%E6%96%99\01-%E7%AC%94%E8%AE%B0\12-%E9%A1%B9%E7%9B%AE%E6%90%AD%E5%BB%BA%E6%AD%A5%E9%AA%A4.html#105)
  + [1.6. 输出404页面 500页面](file:///G:\01.First\node-01\1-%E6%95%99%E5%AD%A6%E8%B5%84%E6%96%99\01-%E7%AC%94%E8%AE%B0\12-%E9%A1%B9%E7%9B%AE%E6%90%AD%E5%BB%BA%E6%AD%A5%E9%AA%A4.html#106)
* 设置加载静态资源
  + [1.7. 模板输出首页](file:///G:\01.First\node-01\1-%E6%95%99%E5%AD%A6%E8%B5%84%E6%96%99\01-%E7%AC%94%E8%AE%B0\12-%E9%A1%B9%E7%9B%AE%E6%90%AD%E5%BB%BA%E6%AD%A5%E9%AA%A4.html#107)
* 由控制器管理view和model
  + [1.8. 创建数据库，导入数据](file:///G:\01.First\node-01\1-%E6%95%99%E5%AD%A6%E8%B5%84%E6%96%99\01-%E7%AC%94%E8%AE%B0\12-%E9%A1%B9%E7%9B%AE%E6%90%AD%E5%BB%BA%E6%AD%A5%E9%AA%A4.html#108)
  + [1.9. 安装orm，添加models](file:///G:\01.First\node-01\1-%E6%95%99%E5%AD%A6%E8%B5%84%E6%96%99\01-%E7%AC%94%E8%AE%B0\12-%E9%A1%B9%E7%9B%AE%E6%90%AD%E5%BB%BA%E6%AD%A5%E9%AA%A4.html#109)
  + [1.10. app.js中设置orm的连接](file:///G:\01.First\node-01\1-%E6%95%99%E5%AD%A6%E8%B5%84%E6%96%99\01-%E7%AC%94%E8%AE%B0\12-%E9%A1%B9%E7%9B%AE%E6%90%AD%E5%BB%BA%E6%AD%A5%E9%AA%A4.html#110)
  + [1.11. 把获取的数据渲染到首页](file:///G:\01.First\node-01\1-%E6%95%99%E5%AD%A6%E8%B5%84%E6%96%99\01-%E7%AC%94%E8%AE%B0\12-%E9%A1%B9%E7%9B%AE%E6%90%AD%E5%BB%BA%E6%AD%A5%E9%AA%A4.html#111)
  + [1.12. 获取帖子的详细内容](file:///G:\01.First\node-01\1-%E6%95%99%E5%AD%A6%E8%B5%84%E6%96%99\01-%E7%AC%94%E8%AE%B0\12-%E9%A1%B9%E7%9B%AE%E6%90%AD%E5%BB%BA%E6%AD%A5%E9%AA%A4.html#112)
  + [1.13. 获取评论列表](file:///G:\01.First\node-01\1-%E6%95%99%E5%AD%A6%E8%B5%84%E6%96%99\01-%E7%AC%94%E8%AE%B0\12-%E9%A1%B9%E7%9B%AE%E6%90%AD%E5%BB%BA%E6%AD%A5%E9%AA%A4.html#113)
  + [1.14. 评论的支持 反对](file:///G:\01.First\node-01\1-%E6%95%99%E5%AD%A6%E8%B5%84%E6%96%99\01-%E7%AC%94%E8%AE%B0\12-%E9%A1%B9%E7%9B%AE%E6%90%AD%E5%BB%BA%E6%AD%A5%E9%AA%A4.html#114)
  + [1.15. 根据ip判断是否允许 支持 反对](file:///G:\01.First\node-01\1-%E6%95%99%E5%AD%A6%E8%B5%84%E6%96%99\01-%E7%AC%94%E8%AE%B0\12-%E9%A1%B9%E7%9B%AE%E6%90%AD%E5%BB%BA%E6%AD%A5%E9%AA%A4.html#115)
  + [1.16. 发表评论](file:///G:\01.First\node-01\1-%E6%95%99%E5%AD%A6%E8%B5%84%E6%96%99\01-%E7%AC%94%E8%AE%B0\12-%E9%A1%B9%E7%9B%AE%E6%90%AD%E5%BB%BA%E6%AD%A5%E9%AA%A4.html#116)
  + [1.17. cookie和session](file:///G:\01.First\node-01\1-%E6%95%99%E5%AD%A6%E8%B5%84%E6%96%99\01-%E7%AC%94%E8%AE%B0\12-%E9%A1%B9%E7%9B%AE%E6%90%AD%E5%BB%BA%E6%AD%A5%E9%AA%A4.html#117)
  + [1.18. 登录，md5](file:///G:\01.First\node-01\1-%E6%95%99%E5%AD%A6%E8%B5%84%E6%96%99\01-%E7%AC%94%E8%AE%B0\12-%E9%A1%B9%E7%9B%AE%E6%90%AD%E5%BB%BA%E6%AD%A5%E9%AA%A4.html#118)
  + [1.19. 管理首页](file:///G:\01.First\node-01\1-%E6%95%99%E5%AD%A6%E8%B5%84%E6%96%99\01-%E7%AC%94%E8%AE%B0\12-%E9%A1%B9%E7%9B%AE%E6%90%AD%E5%BB%BA%E6%AD%A5%E9%AA%A4.html#119)
  + [1.20. logout](file:///G:\01.First\node-01\1-%E6%95%99%E5%AD%A6%E8%B5%84%E6%96%99\01-%E7%AC%94%E8%AE%B0\12-%E9%A1%B9%E7%9B%AE%E6%90%AD%E5%BB%BA%E6%AD%A5%E9%AA%A4.html#120)
  + [1.21. 帖子管理](file:///G:\01.First\node-01\1-%E6%95%99%E5%AD%A6%E8%B5%84%E6%96%99\01-%E7%AC%94%E8%AE%B0\12-%E9%A1%B9%E7%9B%AE%E6%90%AD%E5%BB%BA%E6%AD%A5%E9%AA%A4.html#121)
  + [1.22. 模板重构](file:///G:\01.First\node-01\1-%E6%95%99%E5%AD%A6%E8%B5%84%E6%96%99\01-%E7%AC%94%E8%AE%B0\12-%E9%A1%B9%E7%9B%AE%E6%90%AD%E5%BB%BA%E6%AD%A5%E9%AA%A4.html#122)
  + [1.23. 用户管理](file:///G:\01.First\node-01\1-%E6%95%99%E5%AD%A6%E8%B5%84%E6%96%99\01-%E7%AC%94%E8%AE%B0\12-%E9%A1%B9%E7%9B%AE%E6%90%AD%E5%BB%BA%E6%AD%A5%E9%AA%A4.html#123)
  + [1.24. UEditor](file:///G:\01.First\node-01\1-%E6%95%99%E5%AD%A6%E8%B5%84%E6%96%99\01-%E7%AC%94%E8%AE%B0\12-%E9%A1%B9%E7%9B%AE%E6%90%AD%E5%BB%BA%E6%AD%A5%E9%AA%A4.html#124)
  + [1.25. 分页](file:///G:\01.First\node-01\1-%E6%95%99%E5%AD%A6%E8%B5%84%E6%96%99\01-%E7%AC%94%E8%AE%B0\12-%E9%A1%B9%E7%9B%AE%E6%90%AD%E5%BB%BA%E6%AD%A5%E9%AA%A4.html#125)
  + [1.26. 搜索](file:///G:\01.First\node-01\1-%E6%95%99%E5%AD%A6%E8%B5%84%E6%96%99\01-%E7%AC%94%E8%AE%B0\12-%E9%A1%B9%E7%9B%AE%E6%90%AD%E5%BB%BA%E6%AD%A5%E9%AA%A4.html#126)