

# 信息与软件工程学院

## 综合设计报告

题目全称： 多人博客系统 Nblog 的设计与实现

指导老师： 惠宇 职称： 副教授

序号	学生姓名	学号	班号	成绩
1	孙韬	2014220402020	0402	
2				
3				
4				
5				
6				
7				
8				
9				
10				

(注：学生姓名填写按学生对该综合设计的贡献及工作量由高到底排列，分数按排名依次递减。序号排位为“1”的学生成绩最高，排位为“10”的学生成绩最低。)

指导老师评语：

指导签字： \_\_\_\_\_

### 本小组成员任务分工情况

序号	姓名	学号	任务分工	完成情况
1	孙韬	2014220402020	设计, 实现, 测试	完成
2				
3				
4				
5				
6				
7				
8				
9				
10				

## 摘 要

博客, 是 Web Logger 的混成词, 正式名称为网络日志. 在网络上发表 Blog 的构想始于 1998 年, 但到了 2000 年才开始真正流行. 自从微型博客流行起来之后, 传统的博客才日益衰落, 但是直到今天 (2016 年), 博客依旧在媒体中有着一定的影响力.

多人博客系统 Nblog, 主要基于 NodeJS 和 HTML 开发, 考虑到较小的数据量, 以 sqlite 作为默认数据库.

全文主要分为五大部分:

1. 需求分析文档, 提出系统的主要功能/非功能需求.
2. 设计文档, 根据需求分析, 进行概要设计, 对系统层面进行
3. 根据概要设计, 实现系统各个子模块, 并给出单元测试结果.
4. 测试文档, 测试用例及测试结果.
5. 部署及维护, 详细描述服务器部署过程.

作为一个 Java 后端工程师, 在行文中我将或多或少的比对 NodeJS 和 Java 实现的异同, 并给出自己的看法.

**关键词:** 博客, NodeJS, HTML

## ABSTRACT

Blog is a portmanteau word of ‘Web Logger’ , the official name is web log. The idea of presenting blog in the Internet began in 1998, but it became really popular since 2000. Cause of popular micro blog, traditional blog, is on the wane, but until today (2016), it still has a certain influence in social media.

Multi-user blog system--Nblog, mainly based on NodeJS and HTML, considered the limited data size, it use the SQLite as the default database.

This paper is mainly divided into five parts:

The first part, the demand analysis of the document, the main function of the system / non functional requirements.

The second part, the design document, according to the demand analysis, carries on the outline design, carries on the

The third part, according to the outline design, realize the system each sub module, and give the unit test results.

The fourth part, test documents, test cases and test results.

The fifth part, deployment and maintenance, detailed description of the server deployment process.

As a Java backend engineer, I will write the similarities and differences between NodeJS and Java in the text more and less, and give out my own views.

**Keywords:**    **Blog,NodeJS,HTML**

## 目录

<b>需求分析 .....</b>	<b>4</b>
1.1    引言 .....	4
1.2    任务概述 .....	5
1.3    需求规定 .....	6
1.4    功能和非功能需求 .....	7
1.5    环境需求 .....	10
<b>设计 .....</b>	<b>12</b>
2.1    引言 .....	12
2.2    总体设计 .....	12
2.3    用户 UI 设计 .....	16
2.4    系统数据结构设计 .....	17
2.5    系统出错处理设计 .....	20
<b>实现 .....</b>	<b>21</b>
3.1    开发环境 .....	21
3.2    开发技术 .....	22
3.3    调试技术 .....	23
3.4    版本管理 .....	23
3.5    模块实现 .....	25
3.6    实现效果 .....	39
<b>测试 .....</b>	<b>47</b>
4.1    引言 .....	47
4.2    测试环境 .....	47
4.3    测试用例及测试结果 .....	47
4.4    负载和压力测试 .....	48
<b>部署及维护 .....</b>	<b>55</b>
5.1    部署 .....	55
5.2    维护 .....	59
<b>总结与展望 .....</b>	<b>60</b>
<b>参考文献 .....</b>	<b>61</b>

---

## 需求分析

### 1.1 引言

市面上的博客系统有很多,其中最负盛名的就是基于 PHP 的 WordPress,之所以要使用 NodeJS 再实现一次,其主要有以下几个目的:

学习异步编程的思想. 与 Java 或 C# 这些工业级语言不同, JS 是一个以异步和回调为核心的语言, 方法是 JS 的一等公民, 可以当做参数传递, 轻松实现了回调, 这在 Java 里是做不到的(通常情况下你需要编写实现 Callable 接口的类, 才可以实现异步编程, 这是因为 Java 中 Class 是一等公民, 当然匿名类和 lambda 表达式的引入已经简化了这个过程)

掌握一门可以快速开发的语言. 基于原型的开发模型, 以及各种快速开发模型, 或多或少的都需要一个快速原型, 而在制作快速原型方面, NodeJS+Express+Sqlite 在这方面做得尤其优秀, 搭建一个快速的 API 服务器甚至不需要 1 分钟, 并且不会有各种异常(作为比较, 使用 Spring Boot + JPA(Hibernate)+Mysql+druid+Tomcat 搭建一个 maven 项目也只需要 5 分钟, 但是由于各种各样的原因, 可能在各种步骤出现问题, 最终结果可能是 5 分钟+N 小时), 或许 Python 或者 Ruby 也可以制作快速原型, 但是它们实质上是包含了各种各样库函数的通用语言, 而 NodeJS 其本质上只是一个 Web 服务器, 其限制也是其优点, 大大降低了学习曲线, 加速了原型的开发.

学习动态语言的开发与调试. 与 Java 不同, JS 这种弱类型的语言对于开发人员的素质有很高的要求, 由于其高度动态的特性, 优点是开发速度快, 动态程度高, 并且编程十分灵活. 而缺点也很明显, 就是没有静态检测, 错误经常需要运行时才能发现(哪怕是拼写错误), 因此, 弱类型语言对于开发人员的素质有很高的要求, 如果开发人员对于语言了解不深刻, 没有自己的一套编程规范, 将会是一件非常痛苦的事情.

权限管理的实现. 权限管理自古以来就是一件令人头疼的事情, Java 有 Shiro 和 Spring Security 等安全框架, 而 NodeJS 并没有这样的框架(有一个 passport 认证框架, 并没有权限管理功能), 这样的话, 就可以自己实现一个简单的权限管理子系统, 当然, 由于时间有限, 该子系统的性能将不予考虑.

API 设计. API 的设计是一个非常有哲学的问题, 我在设计另一个系统的时候, 就由于系统设计问题, 导致 API 的 URL 非常的冗长(为了保证可读性, 但是长名称虽然保证了可读性, 冗长的 API 也让人不想多看一眼), 在做这个系统的时候, 我将尝试不同风格的 API, 以确定实际开发中最优 API 设计方式. (API 的设计也取决于权限管理的设计)

---

## 1.2 任务概述

### 1.2.1 目标

一个可用的具有博客和评论功能的多用户网页博客系统.

### 1.2.2 用户的特点

普通用户：具有基本计算机使用经验的人员.

管理员：具有系统管理经验的管理人员

### 1.2.3 假定和约束

假定管理员会仔细阅读项目文档，明白权限管理系统的实现，不会误操作取消掉自己的管理权限，而使整个系统失效.

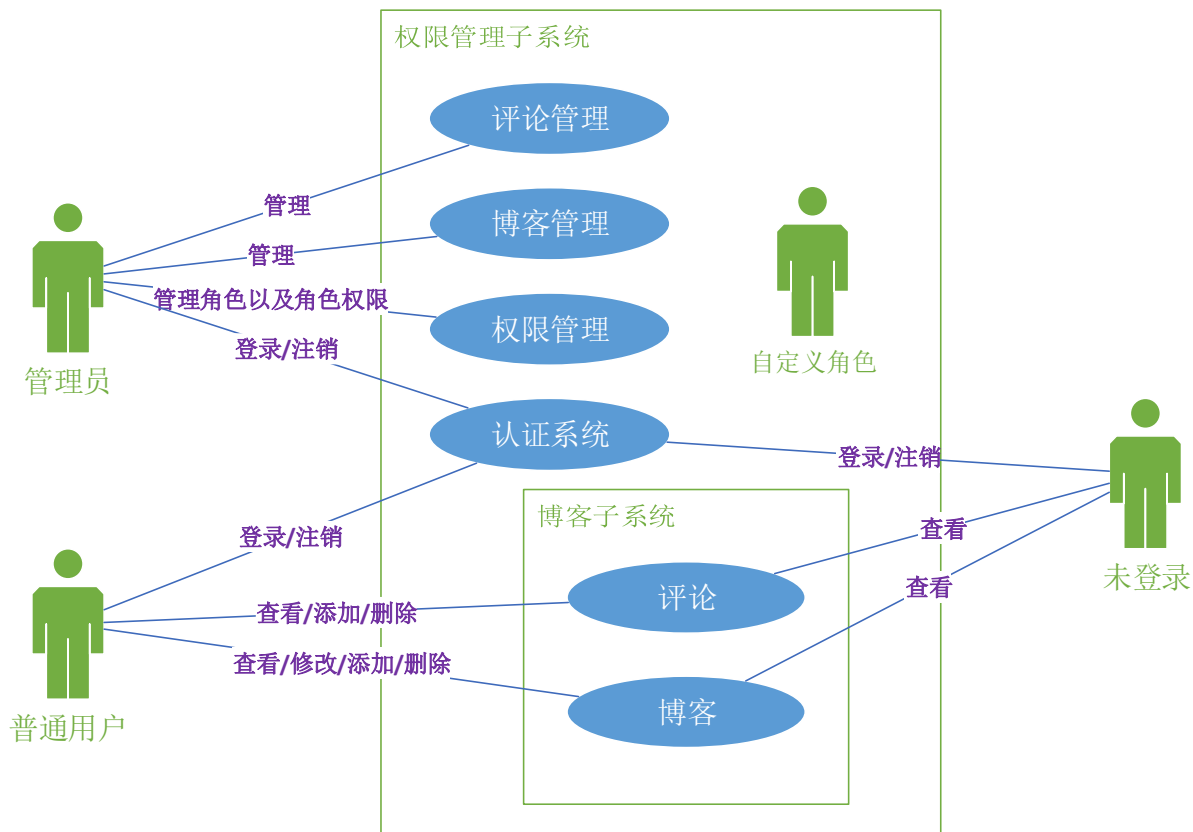
假定用户对于系统没有恶意，不会使用 XSS 等 Web 攻击手段以达到非法的目的.

假定在开发周期内，所涉及到的开发技术都处于稳定状态，主要 API 不会发生变更.

假定在部署时，所部署的实体机，不会受到其他软件的限制. 例如端口，数据库，项目路径都不会因为被占用而导致部署失败.

## 1.3 需求规定

### 1.3.1 用例图(模块图)



高层次的简化用例图

### 1.3.2 用例简介

未登录用户,默认有着查看博客和评论的权限,并且可以登录为其他类型的用户.

普通用户,默认有着自己博客及评论 **CRUD** 的功能,并且可以登录和注销.

管理员,默认情况下,除了拥有普通用户的权限之外,拥有管理博客/评论的权限,此外管理员可以 **CRUD** 系统的角色,增加自定义的用户,也拥有管理角色权限的功能,也就是说上面的用例图只是默认情况下的用例图..



## 1.4 功能和非功能需求

### 1.4.1 功能需求

#### 1.4.1.1 权限管理模块

此模块不是核心模块,但是后述的模块将使用此模块,因此在此首先提出.

定义: API 权限,POST 方法提交 URL,如果没有权限将会返回未授权.

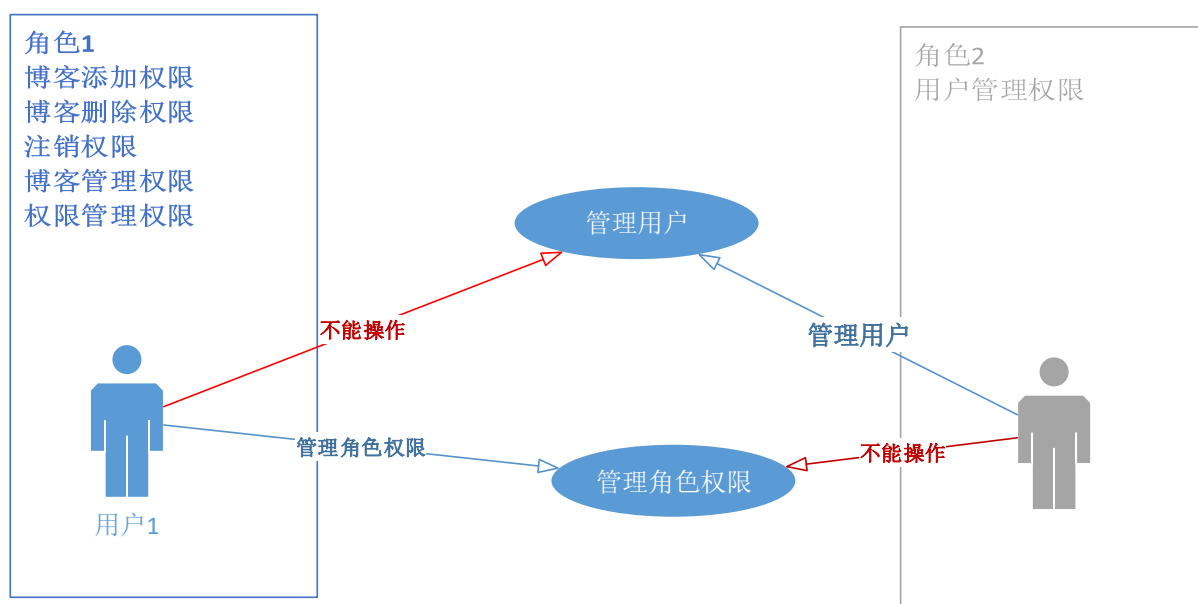
定义: View 权限,GET 方法请求 URL,如果没有权限将会返回未授权.

定义:角色,具有 API 和 View 权限属性的数据库记录.

定义:用户,具有角色属性的数据库记录.

创建用户的时候必须指定一个角色(默认为普通用户,具有管理自己博文评论的权限).

角色的权限 可以被具有 角色权限管理权限的角色下的用户 管理.



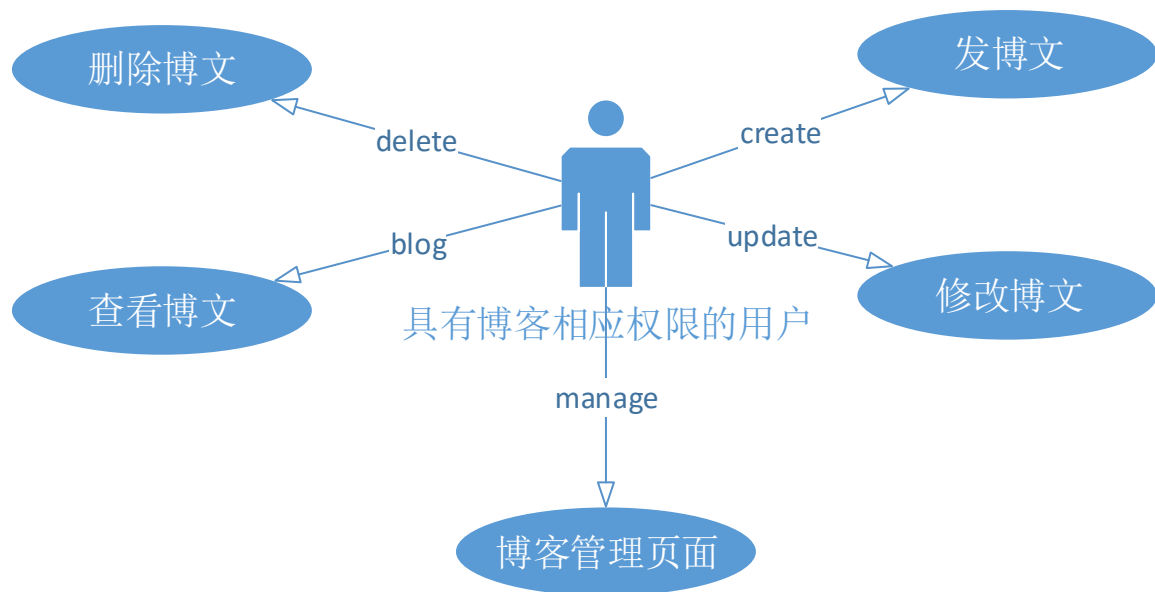
具有权限管理权限但没有用户管理权限的角色的用户

#### 1.4.1.2 博客模块

博客模块是本项目的核心功能.

此模块需要实现的功能是,实现博文的添加编辑查看页面,实现博客删除操作.尤其需要注意的是博文需要能够存储一定的格式而不是 plain text,并且能够上传图片插入博文中.还要实现一个管理所有博客的界面.

默认情况下用户只能管理自己的博客,只有管理界面才能管理所有博文.

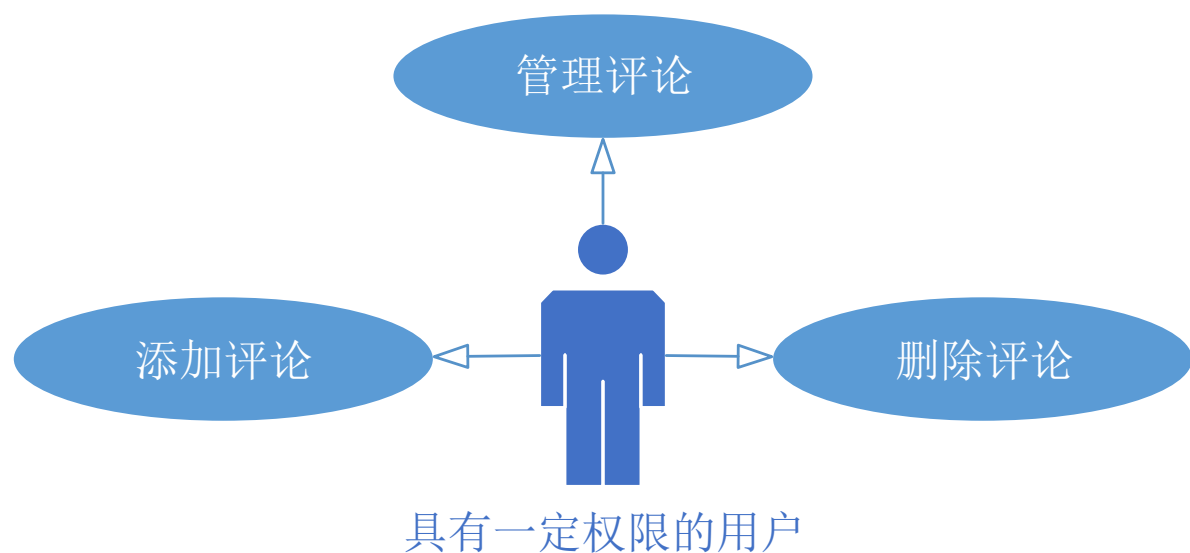


#### 1.4.1.3 评论模块

互联网上任意一个博客，基本上都有评论的功能，博客的核心也是分享和交流，所以博客次要的功能就是评论。

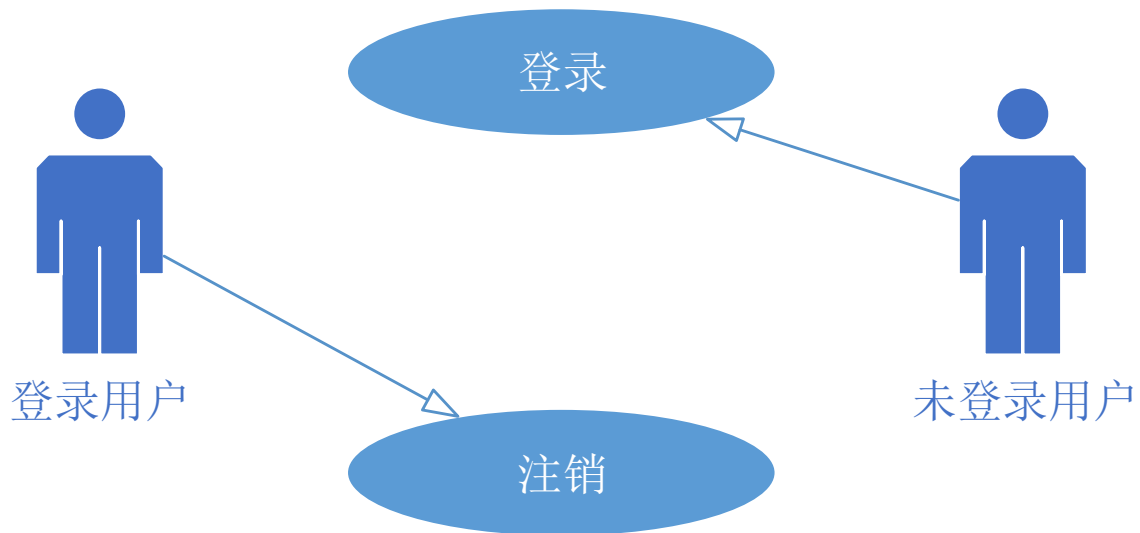
系统需要实现. 对于博客的评论,查看,删除功能. 同时需要实现一个管理所有评论的页面.

默认情况下用户只能操作自己的评论，只有管理页面才能管理所有评论。



#### 1.4.1.4 认证模块

认证模块主要是实现用户的登录注销,连接信息维持功能。

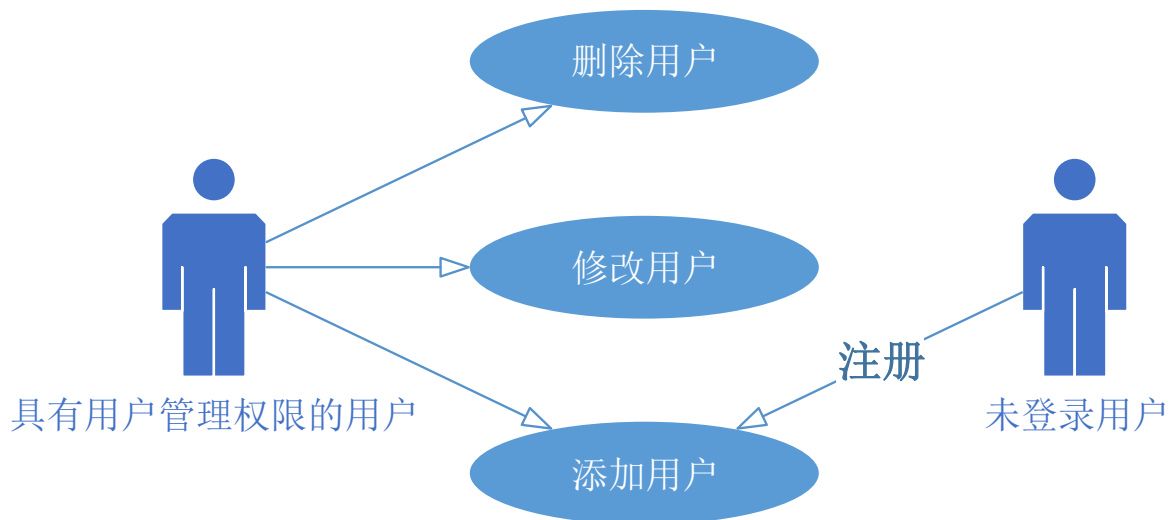


#### 1.4.1.5 用户管理模块

用户管理模块, 实现用户的 **CRUD**, 具体来说就是注册, 删除用户, 信息修改, 用户检索等功能.

默认情况下只有管理界面可以管理所有用户, 也只有管理页面可以删除用户.

默认情况下, 注册用户的角色是普通用户, 具有较低的权限..



#### 1.4.1.6 性能需求

本系统对于性能的要求并不高.

忽略网络传输, 也就是开发模式下, 要求整个系统响应时间在 1000ms 以下.

要求至少支持 10 人以上并发访问.

#### 1.4.1.7 其他

还有一些功能需求并未并入一些模块中, 或者被多个模块包含.

---

例如前端界面的配置,数据持久化模块等.

## 1.4.2 非功能需求

### 1.4.2.1 安全性需求

防御一般的互联网攻击.例如:

- XSS 跨站攻击,
- SQL 注入,
- CSRF 请求伪造
- 重放攻击
- 中间人攻击
- DoS 拒绝攻击.

### 1.4.2.2 稳定性需求

满足一般的稳定性即可:

- 正常运行时间高于 1D 以上.
- 在主要流程上正常负载下不发生 bug(博客模块)
- 不发生内存泄漏.
- 项目发生彻底崩溃之后,可以自动重新部署.

### 1.4.2.3 其他

保证项目的独立性,不会由于本项目的原因,导致服务器上的其他项目崩溃.

保证项目的完整性,尽量交付完整的项目.

## 1.5 环境需求

### 1.5.1 开发环境

正常 PC 机即可.

---

2010 年及以后的处理器

1G 以上内存

15G 以上存储空间

显示器

鼠标 键盘

XP 以上操作系统

Pow designer, Navicat 套装, Office, Putty, NodeJS v4, WebKit 内核浏览器, TortoiseSVN, Vbox.

任何可用的文本编辑器.

## 1.5.2 运行环境

50M 内存, 开启交换区情况下可以更低.

任何 CPU, 但至少需要有对应 GCC 编译器或者二进制 Python 安装程序.

15 MB 以上存储空间.

NodeJS v4, pm2, apache2, ssh, vim.

---

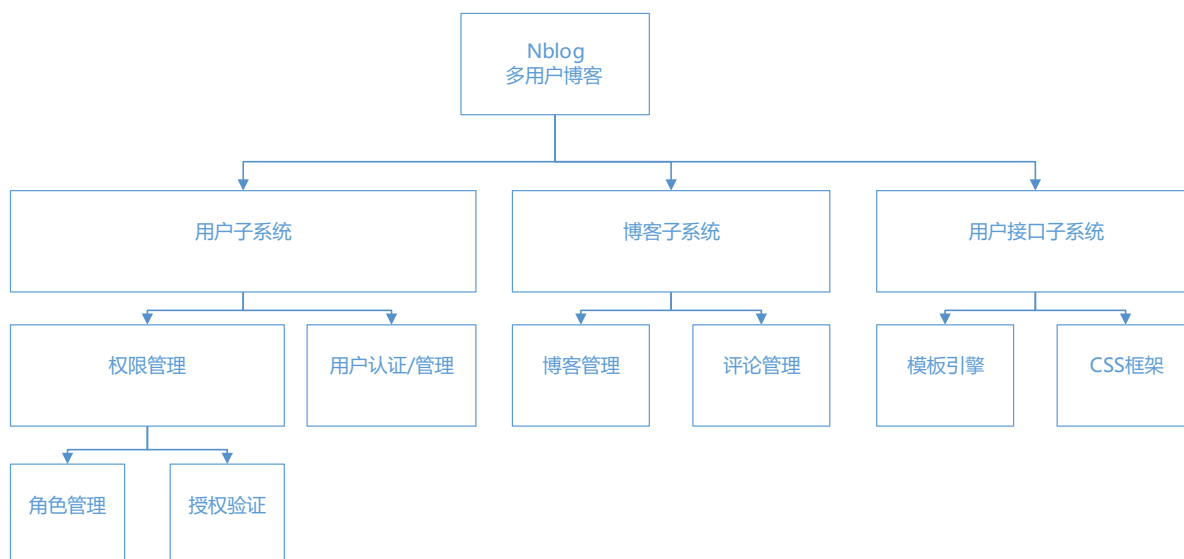
# 设计

## 2.1 引言

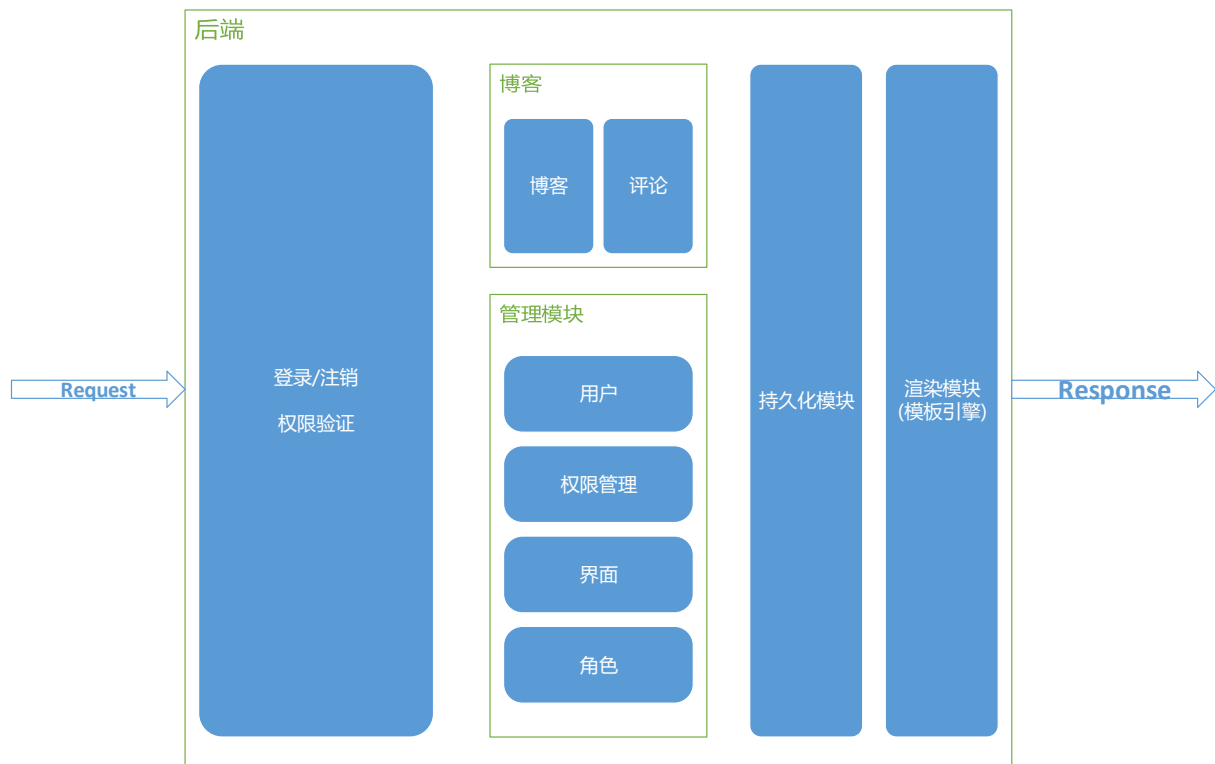
设计是一个系统的基石，也是集成/系统测试需要的文档.设计的好坏直接决定了项目的质量.

## 2.2 总体设计

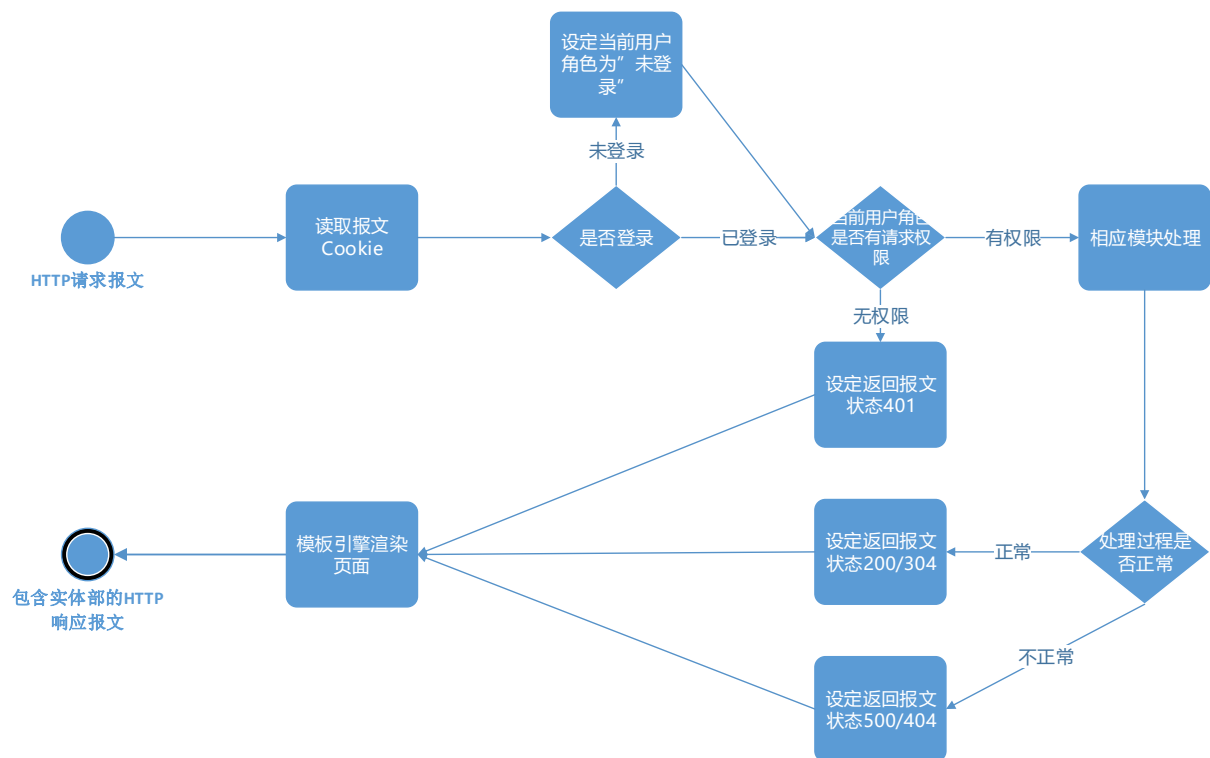
Nblog 博客系统主要功能结构如下图所示.



## 2.2.1 基本设计概念和处理流程



后端模块图



处理流程

### 2.2.2 用户子系统概要设计

此子系统提供用户管理和权限管理两类功能.

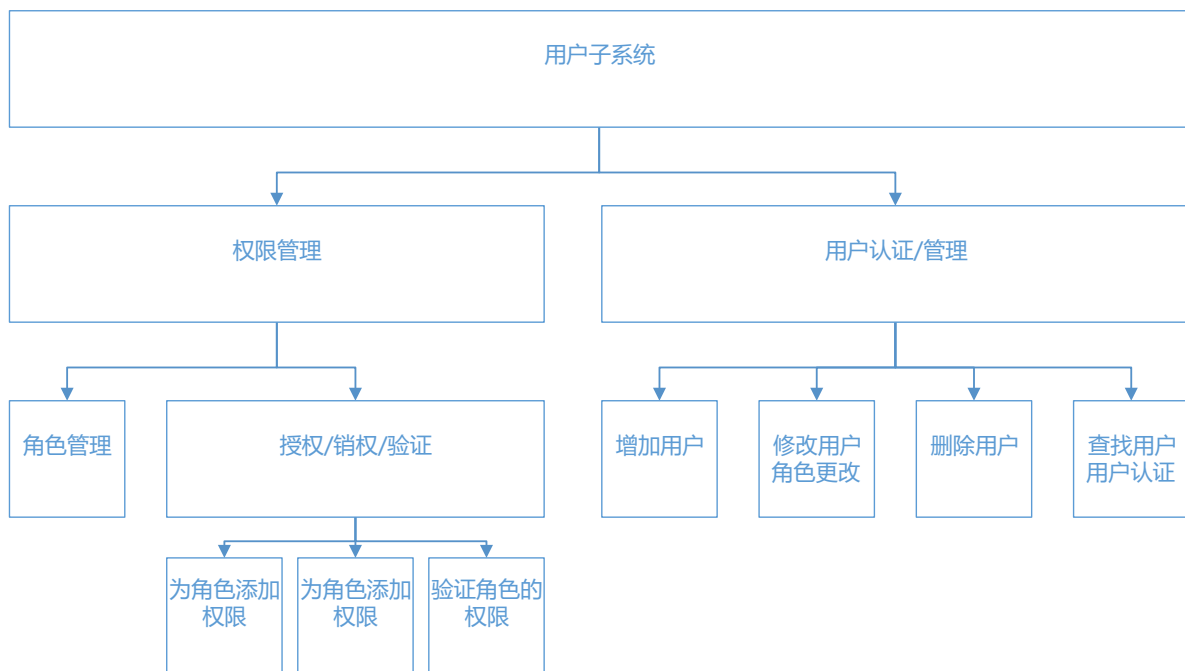
权限管理又分为角色管理和角色权限管理两部分功能

角色管理包含角色的 CRUD 四类功能

角色权限管理包含授权,销权,检验三类功能.

用户管理包含用户的 CRUD 四类功能..

如下图



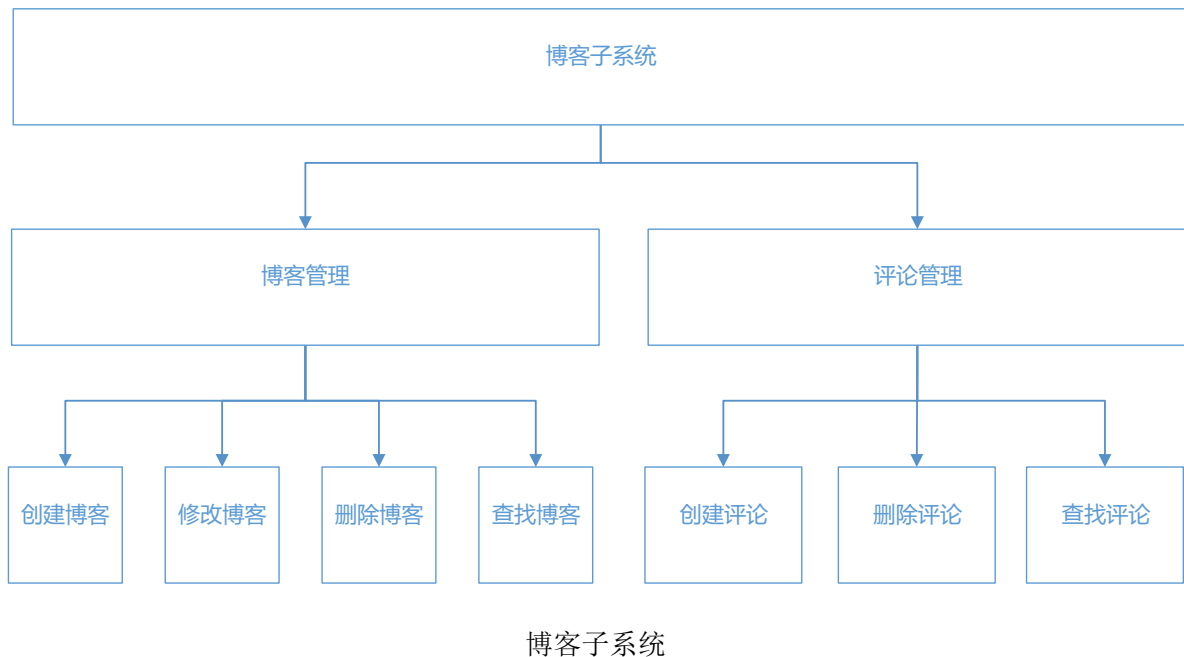
用户子系统

### 2.2.3 博客子系统概要设计

博客子系统是本系统的核心功能.

包含博客的发布,修改,删除,查阅及评论的发布,删除,查阅七项功能.





#### 2.2.4 用户接口子系统概要设计

用户接口,即 User Interface,是人机接口,是人与机器交互的方式.

由于采用了模板引擎的方式渲染前端(HTML)页面,所以其实用户接收到的是静态页面,只是接受的静态页面是动态渲染的,处理压力主要是后端服务器,前端的 JS 将会比较少,浏览器压力比较小,用户体验比较好.

为了减少前端页面的重复编写,例如 head 标签,header,footer 以及一些常用前端模块,使用模板引擎,将可以把分布在多个模板文件中的 HTML 片段组合起来,大大减少了 HTML 的重复编写,提供一致性和完整性,大大加速了前端开发过程. 有 J2EE 开发经验的开发者,可以将模板引擎视作 JSP,但它们有着更加灵活的使用方式和更加强大的功能(Java 也有很多强大的模板引擎,但由于各种原因,Java 其实并不适合渲染页面这样的工作,在大型系统中,Java 系统主要提供统一 API,页面渲染往往交给 PHP/NodeJS 来做).

本项目将采用 EJS(Embedded JavaScript templates)模板引擎,并不强大但是足够使用了.

用户接口子系统将提供用户接口,也就是 HTML 页面,具体的 HTML 页面将如下图. 接口将会调用其它子系统以完成任务.

系统包含的页面如下



## 2.3 用户 UI 设计

系统将使用 MIT 开源协议下的 Materialize 现代化响应式前端框架，该框架主要是一个 CSS 样式框架,提供 Google Material Design 的设计理念.并且提供一定程度上的移动页面适配.



## 2.4 系统数据结构设计

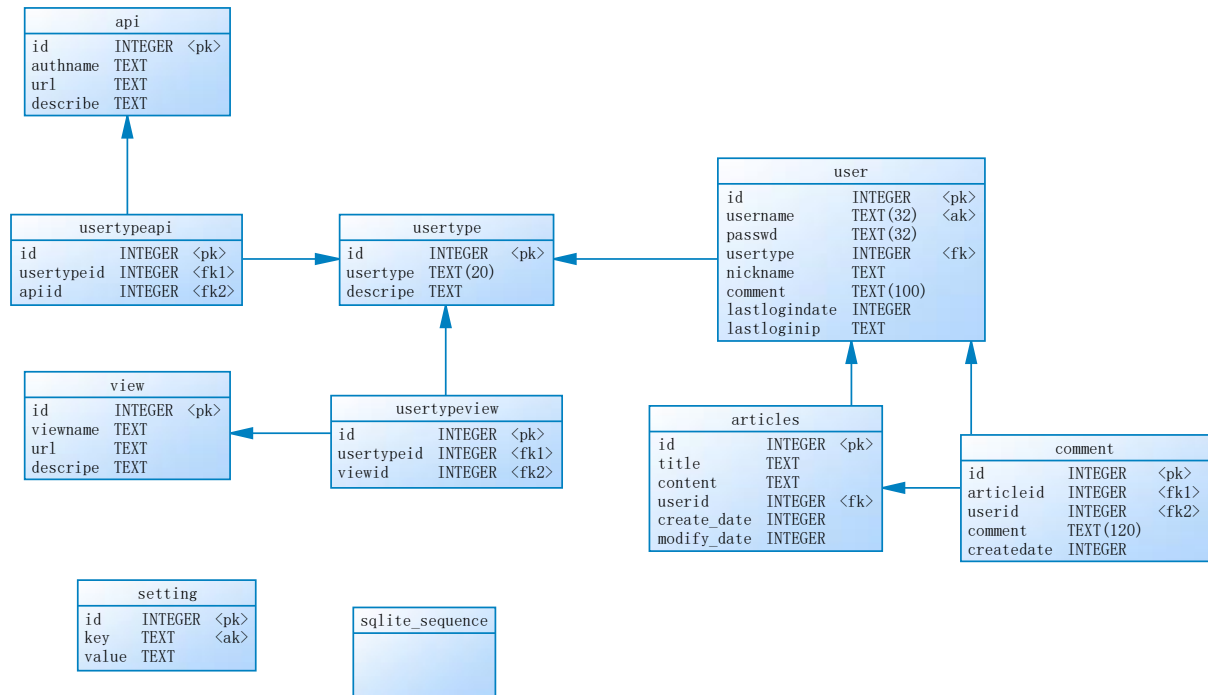
### 2.4.1 物理模型设计

(由于系统简单，直接跳过概要和逻辑模型)

Nblog 系统设计了如下 9 个表

<i>Name</i>	<i>Comment</i>
api	存放 API 信息
articles	存放文章
comment	存放评论
setting	存放 KV 数据
user	存放用户信息
usertype	存放用户角色
usertypeapi	存放用户角色的 api 权限
usertypeview	存放用户角色的视图权限
view	存放视图信息

### 2.4.1.1 Diagram



### 2.4.1.2 API

<i>Name</i>	<i>Data Type</i>	<i>Primary</i>	<i>Foreign Key</i>
id	INTEGER	X	
authname	TEXT		
url	TEXT		
describe	TEXT		

### 2.4.1.3 Articles

<i>Name</i>	<i>Data Type</i>	<i>Primary</i>	<i>Foreign Key</i>
id	INTEGER	X	
title	TEXT		
content	TEXT		
userid	INTEGER		X
create_date	INTEGER		
modify_date	INTEGER		

---

#### 2.4.1.4 Comment

<i>Name</i>	<i>Data Type</i>	<i>Primary</i>	<i>Foreign Key</i>
id	INTEGER	X	
articleid	INTEGER		X
userid	INTEGER		X
comment	TEXT(120)		
createdate	INTEGER		

#### 2.4.1.5 User

<i>Name</i>	<i>Data Type</i>	<i>Primary</i>	<i>Foreign Key</i>
id	INTEGER	X	
username	TEXT(32)		
passwd	TEXT(32)		
usertype	INTEGER		X
nickname	TEXT		
comment	TEXT(100)		
lastlogindate	INTEGER		
lastloginip	TEXT		

#### 2.4.1.6 Usertype

<i>Name</i>	<i>Data Type</i>	<i>Primary</i>	<i>Foreign Key</i>
id	INTEGER	X	
usertype	TEXT(20)		
descripe	TEXT		

#### 2.4.1.7 Usertypeapi

<i>Name</i>	<i>Data Type</i>	<i>Primary</i>	<i>Foreign Key</i>
id	INTEGER	X	
usertypeid	INTEGER		X
apiid	INTEGER		X

---

#### 2.4.1.8 Usertypeview

<i>Name</i>	<i>Data Type</i>	<i>Primary</i>	<i>Foreign Key</i>
id	INTEGER	X	
usertypeid	INTEGER		X
viewid	INTEGER		X

#### 2.4.1.9 View

<i>Name</i>	<i>Data Type</i>	<i>Primary</i>	<i>Foreign Key</i>
id	INTEGER	X	
viewname	TEXT		
url	TEXT		
describe	TEXT		

#### 2.4.1.10 Setting

<i>Name</i>	<i>Data Type</i>	<i>Primary</i>	<i>Foreign Key</i>
id	INTEGER	X	
key	TEXT		
value	TEXT		

### 2.4.2 物理结构设计

Nblog 使用 SQLite 数据库，在目标数据库中，创建 9 个表，Sqlite 将自动生成一个 sequence 表。

## 2.5 系统出错处理设计

本系统将错误分为两类，一种是业务错误，一种是系统错误。

业务错误指用户在使用过程中，不正常的使用系统的各项功能所导致的错误。

系统错误是指用户正常按照文档使用系统，但是系统内部由于设计/编码上的缺陷，发生了错误，不能完成用户的请求。

对于前者，系统依旧会返回页面，页面中将描述用户的请求结果。

对于后者，视状况重启系统或者返回异常页面。

---

## 实现

### 3.1 开发环境

实际开发环境

Software	Version
OS	Windows 10 Pro 10586 64bit
NodeJS	V4.4.2 LTS 64bit
Sublime Text	3114
npm	2.15.0
SAP PowerDesigner	16.5.5.2(4734)
Office	2013 Plus 64bit
Visio	2013 64bit
Firefox	46.0.1
Chrome	51.0
Oracle VM VirtualBox	5.0.20
TortoiseSVN	1.9.4 64bit
Subversion	1.9.4
OpenSSL	1.0.2g
SQLite	3.12.1
Navicat Premium	11.2.7 64bit

Hardware	Detail
CPU	AMD Athlon(tm) II X4 635 Processor, 2900 Mhz
Memory	12.0 GB
Graphic	NVIDIA GeForce GTX 560 SE
Storage	SanDisk SSD i100 24GB ATA Device

---

## 3.2 开发技术

以下对一些开发技术做一些简介。

### 3.2.1 NodeJS

---

*Node.js 是一个 Javascript 运行环境(runtime)。实际上它是对 Google V8 引擎进行了封装。V8 引擎执行 Javascript 的速度非常快，性能非常好。Node.js 对一些特殊用例进行了优化，提供了替代的 API，使得 V8 在非浏览器环境下运行得更好。*

*Node.js 是一个基于 Chrome JavaScript 运行时建立的平台，用于方便地搭建响应速度快、易于扩展的网络应用。Node.js 使用事件驱动，非阻塞 I/O 模型而得以轻量 and 高效，非常适合在分布式设备上运行的数据密集型的实时应用。*

---

NodeJS 这种语言(或者叫做 JS 运行时)，其实和 PHP 差不多，它们都是服务器脚本，但是 PHP 的并发是靠多进程，而 NodeJS 并发是靠多事件，纵使 linux 开一个进程消耗再小，也不如就一个进程消耗的资源小(对于 NodeJS 来说，并没有“多线程”这个概念，是的，NodeJS 是纯单线程的语言，因此保证了绝对的线程安全，如何充分利用多核 CPU 反而成了一个问题)。

值得注意的是，虽然 NodeJS 现在很火(阿里内部前端网页渲染基本用 NodeJS 替代了 PHP)，但是 Java 和 C# 依旧是工业级语言，依旧是多人协作的首选(或者说是唯一选择，任何人都可以理解弱类型的语言对多人协作是多么不友好)。

### 3.2.2 Express

---

*基于 Node.js 平台，快速、开放、极简的 web 开发框架。*

---

Express 是对于 NodeJS 的简单封装，使用 Express 主要是它提供一些实用的中间件(或者说中间件支持)。例如：

- Multer 文件上传
- Helmet 安全防御
- ORM2 持久化框架
- Session 简单 Session 封装
- EJS ejs 模板引擎



---

Express 有一些术语,例如路由,中间件,模板引擎都非常好理解.整体框架设计下来,基本还是 MVC 的架构,只是 Model 放在了 ORM 框架中,View 放在了模板引擎中,Controller 则是路由.

相比于 Java, 其实 NodeJS 的请求处理方式更像是 Filter Chain 而不是 Servlet, 因为异步非阻塞的编程方式决定了它必然经过多个回调函数, 如果读者对于 Java NIO API 有一定了解的话, 会发现它们其实有很多类似的地方, 其根源是因为它们的非阻塞编程模型.

### 3.2.3 SQLite

---

*SQLite, 是一款轻型的数据库, 是遵守ACID的关系型数据库管理系统, 它包含在一个相对小的C库中。它是D.RichardHipp建立的公有领域项目。它的设计目标是嵌入式的, 而且目前已经在很多嵌入式产品中使用了它, 它占用资源非常的低, 在嵌入式设备中, 可能只需要几百K的内存就够了。它能够支持Windows/Linux/Unix等等主流的操作系统, 同时能够跟很多程序语言相结合, 比如 Tcl、C#、PHP、Java 等, 还有ODBC接口, 同样比起Mysql、PostgreSQL这两款开源的世界著名数据库管理系统来讲, 它的处理速度比他们都快。SQLite 第一个Alpha版本诞生于2000年5月。至2015年已经有15个年头, SQLite也迎来了一个版本 SQLite 3 已经发布。*

---

Sqlite 的优点就是易于使用,易于部署,并且提供了足够的数据库功能,性能也还 OK.

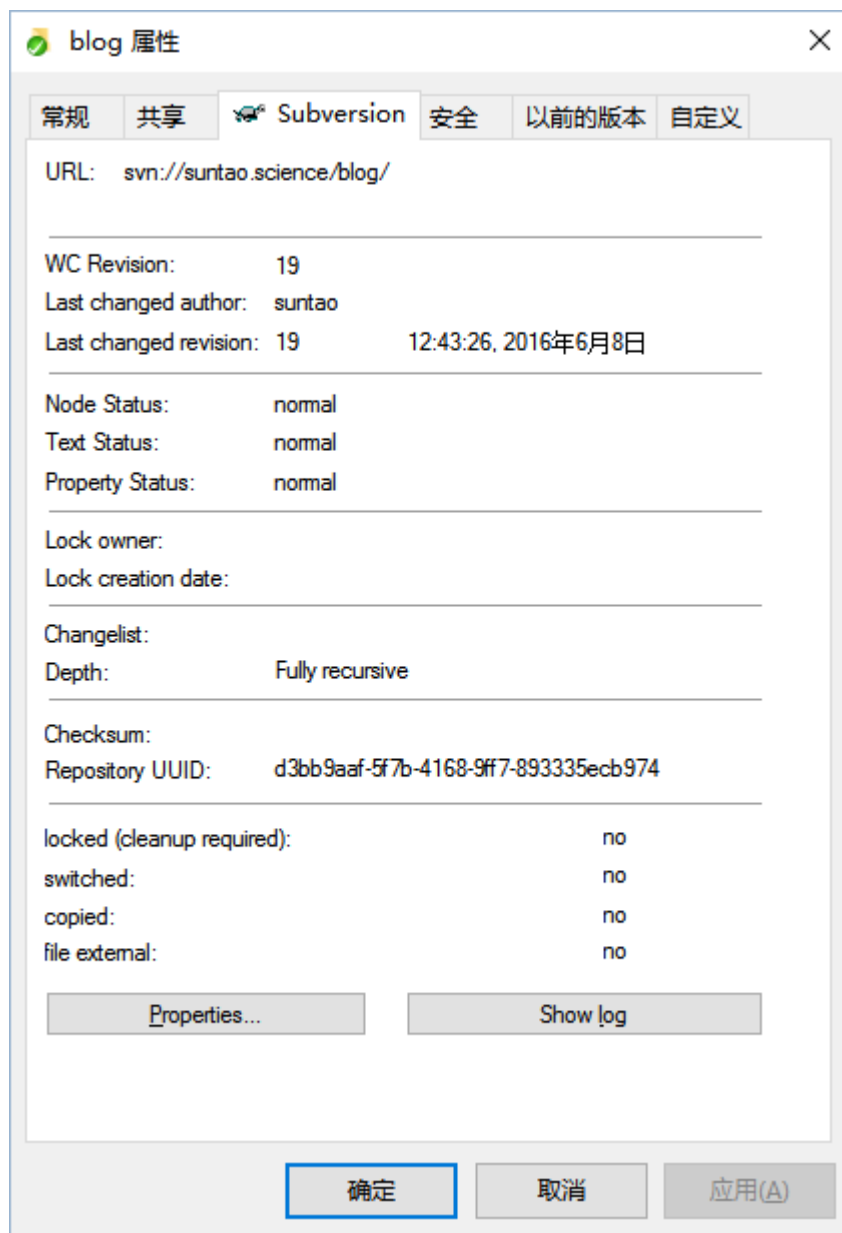
在小型系统中,Sqlite 的使用,大大降低了数据库的部署/维护成本,并且对于数据库的迁移提供了很好的支持(大多数数据库都是Sqlite的超集,在Sqlite能够正常使用的模型,其他数据库都能正常使用,而其他商业数据库往往有一些私有函数,虽然用起来舒服,但迁移起来往往十分费劲).

## 3.3 调试技术

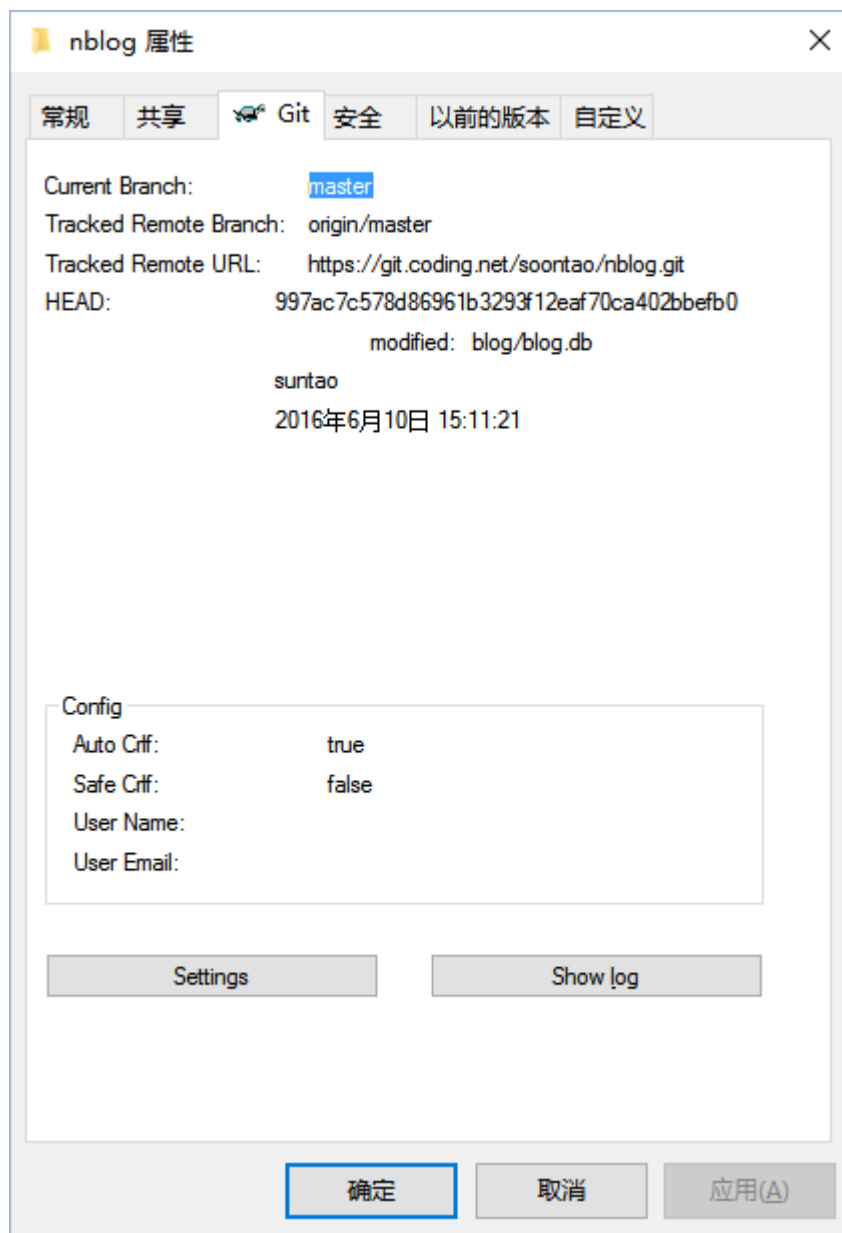
如果对于后端程序要进行调试, 需要在启动 node 程序的时候附上—debug 选项, 如 `node -debug app.js`, 然后需要启动 node-inspector(先使用 npm 安装), 然后就可以在(并且只能在)chrome 浏览器中调试 node 程序了, 比较好的一点是修改了之后可以直接热部署, 不需要重启项目, 类似于 JRebel.

## 3.4 版本管理

本项目先期基于 svn 管理项目,



后期为了发布到 github, 转移到 git 管理,并附上设计文档



(这是一个私有 git 仓库，截止文档撰写日期，暂时没有将项目 push 到 github)

### 3.5 模块实现

目录树 (tree -L 2)

```
.
├── app.js
├── bin
│   └── www
├── blog.db
├── design
│   └── blog.ndm
└── lib
```

---

```
|   |─── auth.js
|   |─── db.js
|   |─── renderp.js
|   └─── util.js
|─── node_modules
|   |─── body-parser
|   |─── cnvm
|   |─── cookie-parser
|   |─── debug
|   |─── ejs
|   |─── express
|   |─── express-session
|   |─── morgan
|   |─── multer
|   |─── orm
|   |─── serve-favicon
|   |─── sqlite
|   |─── sqlite3 -> .npminstall/sqlite3/3.1.4/sqlite3
|   └─── uploadify
|─── nohup.out
|─── package.json
|─── public
|   |─── admin
|   |─── css
|   |─── font
|   |─── fonts
|   |─── images
|   |─── index.html
|   |─── javascripts
|   |─── js
|   |─── LICENSE
|   |─── login.html
|   |─── README.md
|   |─── stylesheets
|   └─── umeditor
|─── routes
|   |─── admin.js
|   |─── articles.js
|   |─── auth.js
|   |─── index.js
|   └─── users.js
|─── test
|   └─── test.py
|─── views
```

---

- |—— allblogs.ejs
- |—— blogdetail.ejs
- |—— blogeditor.ejs
- |—— blogmanager.ejs
- |—— commentmanager.ejs
- |—— costume.ejs
- |—— editblog.ejs
- |—— error.ejs
- |—— footer.ejs
- |—— hasimageblogcard.ejs
- |—— head.ejs
- |—— header.ejs
- |—— index.ejs
- |—— login.ejs
- |—— newblog.ejs
- |—— noimageblogcard.ejs
- |—— overview.ejs
- |—— page.ejs
- |—— simplehead.ejs
- |—— user.ejs
- |—— usermanager.ejs
- |—— usertypeapimanager.ejs
- |—— usertype.ejs
- |—— usertypemanager.ejs
- |—— usertypeviewmanager.ejs

### 3.4.1 核心集成模块

本模块主要用于将各个子模块集成起来。

```
var express = require('express');
var path = require('path');
var favicon = require('serve-favicon');
var logger = require('morgan');
var cookieParser = require('cookie-parser');
var bodyParser = require('body-parser');
var session = require('express-session');
var orm = require('orm');
var mount_uploadify = require('uploadify');
var fs = require('fs');
var multer = require('multer');
var helmet = require('helmet');
```

---

```
var routes = require('./routes/index');
var users = require('./routes/users');
var admin = require("./routes/admin");
var articles = require("./routes/articles");
var auth = require("./routes/auth");

var ormsetup = require("./lib/db");
var authsetup = require("./lib/auth");
var renderp = require("./lib/renderp");
var uploadsetup = require("./lib/upload");

var app = express();

// view engine setup
app.set('views', path.join(__dirname, 'views'));
app.set('view engine', 'ejs');
app.set('env', 'development');

app.use(logger('dev'));
app.use(bodyParser.json());
app.use(bodyParser.urlencoded({
  extended: false
}));
// helmet 安全框架
app.use(helmet());
app.use(cookieParser());
// session配置
app.use(session({
  secret: 'user auth',
  resave: true,
  saveUninitialized: false
}));

// 数据库ORM实体配置
ormsetup(app);
// 权限配置
authsetup(app);
// 重新封装render 实现全局渲染
renderp(app);
// 配置文件上传
uploadsetup(app);
```

---

```
// 配置各个路由
app.use('/', routes);
app.use('/admin/users', users);
app.use('/admin', admin);
app.use('/articles', articles);
app.use('/admin/auth', auth);

// 经过所有路由之后,才查找静态文件
app.use(express.static(path.join(__dirname, 'public')));

// 如果没有找到一个文件或者路由来处理请求
// 就会到这里,抛出404 Error
app.use(function(req, res, next) {
    var err = new Error('Not Found');
    err.status = 404;
    next(err);
});

// error handlers

// development error handler
// will print stacktrace
if (app.get('env') === 'development') {
    app.use(function(err, req, res, next) {
        res.status(err.status || 500);
        res.render('error', {
            message: err.message,
            error: err
        });
    });
}

// production error handler
// no stacktraces leaked to user
app.use(function(err, req, res, next) {
    res.status(err.status || 500);
    res.render('error', {
        message: err.message,
        error: {}
    });
});

module.exports = app;
```

---

### 3.4.2 启动模块

本模块主要用于，将核心模块装载入 NodeJS 服务器中。  
主要是 Express 自动生成的代码。

```
#!/usr/bin/env node

/**
 * Module dependencies.
 */

var app = require('../app');
var debug = require('debug')('blog:server');
var http = require('http');
var https = require('https');
var fs = require('fs');

/**
 * Get port from environment and store in Express.
 */

var port = normalizePort(process.env.PORT || '937');
app.set('port', port);

/**
 * Create HTTP server.
 */
var server = http.createServer(app);

/**
 * Listen on provided port, on all network interfaces.
 */

server.listen(port);

server.on('error', onError);
server.on('listening', onListening);

/**
 * Normalize a port into a number, string, or false.
 */
```



---

```
*/

function normalizePort(val) {
  var port = parseInt(val, 10);

  if (isNaN(port)) {
    // named pipe
    return val;
  }

  if (port >= 0) {
    // port number
    return port;
  }

  return false;
}

/**
 * Event listener for HTTP server "error" event.
 */

function onError(error) {
  if (error.syscall !== 'listen') {
    throw error;
  }

  var bind = typeof port === 'string'
    ? 'Pipe ' + port
    : 'Port ' + port;

  // handle specific listen errors with friendly messages
  switch (error.code) {
    case 'EACCES':
      console.error(bind + ' requires elevated privileges');
      process.exit(1);
      break;
    case 'EADDRINUSE':
      console.error(bind + ' is already in use');
      process.exit(1);
      break;
    default:
      throw error;
  }
}
```

---

```
/**
 * Event listener for HTTP server "listening" event.
 */

function onListening() {
  var addr = server.address();
  var bind = typeof addr === 'string'
    ? 'pipe ' + addr
    : 'port ' + addr.port;
  debug('Listening on ' + bind);
}
```

### 3.4.3 授权管理模块

#### auth.js

```
module.exports = function auth(app) {
  // 通过Session记录认证信息
  app.use(function(req, res, next) {
    if (!req.session.isinit) {
      req.session.isinit = true;
      req.session.islogin = false;
      req.session.user = "";
    };
    next();
  });

  // 配置
  app.use(function(req, res, next) {
    var url = req.originalUrl;
    var method = req.method;
    var isStatic = false;
    //session user或者未注册用户
    var user = req.session.user || {
      usertype: 0
    }

    if (url.startsWith("/js") || url.startsWith("/css") ||
    url.startsWith("/images") || url.startsWith("/umeditor") || url == "/" ) {
      isStatic = true;
    }
  })
}
```

---

```

        if (!isStatic) {
            // 非静态文件
            if (method == 'GET') {
                // view
                req.db.driver.executeQuery("select * from usertypeview where viewid
in (select id from view where substr(?,1,length(url)) = url) and usertypeid = ?",
[url, user.usertype], function(err, views) {
                    if (err) throw err;
                    if (views.length > 0) {
                        next();
                    } else {
                        var err = new Error();
                        err.status = 401;
                        next(err);
                    }
                })
            } else if (method == 'POST') {
                // api
                req.db.driver.executeQuery("select * from usertypeapi where apiid in
(select id from api where substr(?,1,length(url)) = url) and usertypeid = ?",
[url, user.usertype], function(err, apis) {
                    if (err) throw err;
                    if (apis.length > 0) {
                        next();
                    } else {
                        var err = new Error();
                        err.status = 401;
                        next(err);
                    }
                })
            } else {
                var err = new Error("非标准方法");
                err.status = 500;
                next(err);
            }
        } else {
            next();
        }
    }

});
}

```

---

### 3.4.4 配置模块

这是几个配置模块

db.js

orm模块配置

```
var orm = require('orm');

module.exports = function(app) {

    // 数据库ORM实体配置
    app.use(orm.express("sqlite:blog.db", {
        define: function(db, models, next) {
            models.user = db.define("user", {
                id: {
                    type: 'serial',
                    key: true
                },
                username: String,
                passwd: String,
                usertype: Number,
                nickname: String,
                comment: String,
                lastlogindate: Number,
                lastloginip: String
            });
            models.articles = db.define("articles", {
                id: {
                    type: 'serial',
                    key: true
                },
                title: String,
                content: String,
                userid: {
                    type: 'integer'
                },
                create_date: Number,
                modify_date: Number
            });
        }
    });
```

---

```
models.setting = db.define("setting", {
  id: {
    type: 'serial',
    key: true
  },
  key: String,
  value: String
});

models.comment = db.define("comment", {
  id: {
    type: 'serial',
    key: 'true'
  },
  articleid: Number,
  userid: Number,
  comment: String,
  createdate: Number
});

models.usertype = db.define("usertype", {
  id: {
    type: 'serial',
    key: 'true'
  },
  usertype: String,
  describe: String,
});

models.usertypeapi = db.define("usertypeapi", {
  id: {
    type: 'serial',
    key: 'true'
  },
  usertypeid: Number,
  apiid: Number,
});

models.usertypeview = db.define("usertypeview", {
  id: {
    type: 'serial',
    key: 'true'
  },
  usertypeid: Number,
  viewid: Number,
});

next();
```

---

```
    }  
  });  
}
```

```
}
```

renderp.js

封装 render 实现全局渲染

```
module.exports = function renderp(app) {  
  // Express app  
  app.use(function(req, res, next) {  
    req.models.setting.find({}, function(err, keys) {  
      if (err) throw err;  
      res.renderp = function(template, data) {  
        data = data || {};  
        keys.forEach(function(record) {  
          data[record.key] = JSON.parse(record.value);  
        });  
        data['sessionuser'] = req.session.user;  
        return res.render(template, data);  
      }  
      next();  
    });  
  })  
}
```

upload.js

文件上传配置

```
var mount_uploadify = require('uploadify');
```

---

```
var fs = require('fs');
var multer = require('multer');

module.exports = function upload(app) {
  // Express app
  // 上传配置
  mount_uploadify(app, {
    path: '/images/upload',
    fileKey: 'upfile',
    multer: { dest: 'public/images' },
    callback: function(req) {
      req.files.forEach(function(file) {
        var suffix = file.originalname.split('.').pop()
        var npath = file.path + "." + suffix
        var nname = file.filename + "." + suffix
        file.filename = nname
        fs.rename(file.path, npath, (err) => {
          if (err) throw err;
        });
        file.path = npath;
        file.url = "/images/" + nname;
        file.state = "SUCCESS";
        file.name = nname;
        file.type = suffix;
      });
      return req.files[0]
    }
  });
}
```

## util.js

```
var crypto = require("crypto");
var util = new Object();

// 密码加密 如果需要加强密码 改这一个方法就可以了
util.cryptpass = function cryptpass(str) {
  return crypto.createHash('sha256').update(str).digest('hex');
}

module.exports = util;
```

---

### 3.4.5 路由

路由文件存放在 `routes` 路径下，主要实现 `Controller` 层功能，其一提供 `CRUD API`，其二提供页面渲染。

路由文件约 800 行，具体请参看项目文件。

### 3.4.6 视图

视图文件存放在 `views` 路径下，后缀为 `.ejs`，这些文件需要经过渲染才会被传输到客户端浏览器。视图文件太多太杂，请直接参看项目文件。



### 3.6 实现效果

#### 3.5.1 首页

见善-1

主页 所有文章 登录

致郭襄

我走过山时，山不说话，我路过海时，海不说话，小毛驴滴滴答答，箭天剑伴我走天涯。大家都说我因为爱看杨过大伙，才在峨眉山上出了家，其实我只是爱上了峨眉山上的云和霞，像极了十六岁那年的烟花。

更多

新的文章

我这一生最美好的场景 就是遇见你

更多



你好, 孙叔

滴滴答 福建大厦南方 发多少就爱你 发动机上来看 发的啥好看 你好哈哈 修改

更多

新的文章

分页测试

更多

如果说

减肥的拉萨减肥了肯定撒 放到了空间撒范德萨 附近的萨克雷锋

更多

你好呀

Hello World 与任何电子设备一样，当产品开启时请尤其小心。在极少数情况下，您可能会注意到产品中散发出 异味或者冒出烟雾或火花。或者会听到类似爆裂、裂音或嘶嘶声的声音。这些情况可能仅表示某个 内部电子元件发生故障，但仍处于安全和受控的状态。也可能表示存在安全隐患。但是请勿贸然采 取措施或尝试自行诊断这些情况。请联系客户支持中心寻求进一步的指导。要获取服务与支持电话 号码列表，请访问以下Web 站

更多

测试5

/t /n /t/n 换行 事实证明 这些换行都是无效的

更多

发挥是开发

灰化肥会发挥

更多

TEST1

涛山阻隔秦帝船 汉宫彻夜捧金盘 玉肌枉然生白骨 不知剑啸易水寒

更多

红豆

红豆生南国 春来发几枝 愿君多采撷 此物最相思

更多

少年游-1

林花谢了春红 太匆匆 -1

© 2016 Suntao Copyright -1

- 39 -

3.5.2 所有博客

见善-1

主页 所有文章 登录

无题

所爱隔山海 山海不可平 这是一篇没有标题的文章,所以它的标题应该显示为无题

更多

红豆

红豆生南国 春来发几枝 愿君多采撷 此物最相思

更多

1 2

共2页,共17条记录,每页15条

少年游-1

林花谢了春红 太匆匆 -1

© 2016 Suntao Copyright -1

3.5.3 登录

见善-1

主页 所有文章 登录

Username

Password

LOGIN REGISTER

少年游-1

林花谢了春红 太匆匆 -1

© 2016 Suntao Copyright -1

3.5.4 概览

见善-1

[主页](#) [所有文章](#) [我的文章](#) [新的文章](#) [管理](#)

管理概览

用户信息

用户名 suntao  
用户昵称 孙韬  
用户类型 1  
上次登录 2016-06-08 11:54:28  
上次登录IP ::1  
备注 用户管理员  
[点此修改](#)

管理消息

服务器平台 win32  
服务器时间 Wed Jun 08 2016 11:54:31 GMT+0800 (中国标准时间)  
进程号 6180  
内存占用 {"rss":38612992,"heapTotal":31100240,"heapUsed":24265992}  
运行时间 3623.119  
NodeJS 版本 v4.4.4  
项目位置 C:\Users\mrlls\Documents\repo\blog

操作

[博文管理](#)

[评论管理](#)

[用户管理](#)

[个性化](#)

[权限管理](#)

[退出](#)

少年游-1

林花谢了春红 太匆匆 -1

© 2016 Suntao Copyright -1

3.5.5 博文管理

见善-1

[主页](#) [所有文章](#) [我的文章](#) [新的文章](#) [管理](#)

博客管理

博客ID	标题	内容长度	用户昵称	创建日期	修改日期	操作	
22	致郭襄	124	孙韬	2016-05-30	2016-06-03	<a href="#">修改</a>	<a href="#">删除</a>
21	新的文章	16	孙韬	2016-05-30	2016-05-30	<a href="#">修改</a>	<a href="#">删除</a>
20	如果说	122	赵强	2016-05-30	2016-05-30	<a href="#">修改</a>	<a href="#">删除</a>
2	第二篇文章	47	王伟国	2016-05-30	2016-05-30	<a href="#">修改</a>	<a href="#">删除</a>
3	你好呀	207	赵强	2016-05-30	2016-05-30	<a href="#">修改</a>	<a href="#">删除</a>
6	测试5	65	孙韬	2016-05-30	2016-05-30	<a href="#">修改</a>	<a href="#">删除</a>
8	发挥是开发	135	孙韬	2016-05-30	2016-05-30	<a href="#">修改</a>	<a href="#">删除</a>
9	新的文章	53	孙韬	2016-05-30	2016-06-02	<a href="#">修改</a>	<a href="#">删除</a>
10	TEST1	56	孙韬	2016-05-30	2016-05-30	<a href="#">修改</a>	<a href="#">删除</a>
19	测试一下发送新文章	155	赵强	2016-05-30	2016-05-30	<a href="#">修改</a>	<a href="#">删除</a>

12

共2页,共15条记录,每页10条

少年游-1

林花谢了春红 太匆匆 -1

© 2016 Suntao Copyright -1

### 3.5.6 用户类型(角色)管理

## 见善-1

[主页](#) [所有文章](#) [我的文章](#) [新的文章](#) [管理](#)

### 用户类型

ID	用户类型	描述	操作			
0	未登录	尚未登录	<a href="#">修改</a>	<a href="#">删除</a>	<a href="#">API管理</a>	<a href="#">VIEW管理</a>
1	管理员	管理员账户	<a href="#">修改</a>	<a href="#">删除</a>	<a href="#">API管理</a>	<a href="#">VIEW管理</a>
2	普通用户	普通的博客用户	<a href="#">修改</a>	<a href="#">删除</a>	<a href="#">API管理</a>	<a href="#">VIEW管理</a>

[添加用户类型](#)

### 少年游-1

林花谢了春红 太匆匆 -1

© 2016 Suntao Copyright -1

普通用户角色的API权限

权限ID	用户类型	API名	API路径	描述	状态	操作
2	普通用户	overview	/admin/overview	overview访问权下	启用	禁用
3	普通用户	custume	/admin/custume	个性化访问权限	禁用	启用
4	普通用户	addblog	/articles/addblog	添加一个博客	启用	禁用
5	普通用户	editblog	/articles/editblog	修改一个博客	启用	禁用
6	普通用户	deleteblog	/articles/deleteblog	删除一个博客	启用	禁用
7	普通用户	addcmt	/articles/comment/add	添加一个评论	启用	禁用
8	普通用户	updatecmt	/articles/comment/update	修改一个评论	启用	禁用
9	普通用户	deletecmt	/articles/comment/delete	删除一条评论	启用	禁用
10	普通用户	adduser	/admin/users/add		禁用	启用
11	普通用户	deleteuser	/admin/users/delete		禁用	启用
12	普通用户	getuser	/admin/users/get		禁用	启用
13	普通用户	updateuser	/admin/users/update		禁用	启用
14	普通用户	login	/login		禁用	启用
15	普通用户	register	/register		禁用	启用
16	普通用户	logout	/logout		启用	禁用
17	普通用户	usertypeadd	/admin/auth/usertype/add		禁用	启用
18	普通用户	usertypeupdate	/admin/auth/usertype/update		禁用	启用
19	普通用户	usertypeapienable	/admin/auth/usertypeapi/enable		禁用	启用
20	普通用户	usertypeapidisable	/admin/auth/usertypeapi/disable		禁用	启用
21	普通用户	usertypereviewenable	/admin/auth/usertypereview/enable		禁用	启用
22	普通用户	usertypereviewdisable	/admin/auth/usertypereview/disable		禁用	启用
23	普通用户	costome	/admin/costume		禁用	启用
24	普通用户	usertypedelete	/admin/auth/usertype/delete		禁用	启用

3.5.8 视图管理

普通用户角色的视图

权限ID	用户类型	View名	View路径	描述	状态	操作
1	普通用户	newblog	/articles/newblog	添加博客 页面	启用	禁用
2	普通用户	editblog	/articles/editblog	修改博客页面	启用	禁用
3	普通用户	blogmanager	/articles/blogmanager	博客管理器	禁用	启用
4	普通用户	commentmanager	/articles/comment/commentmanager		禁用	启用
5	普通用户	allblogs	/articles/allblogs	博客列表	启用	禁用
6	普通用户	blog	/articles/blog	单独博客	启用	禁用
7	普通用户	register	/register	注册页面	禁用	启用
8	普通用户	usermanager	/admin/users/usermanager		禁用	启用
9	普通用户	adduser	/admin/users/add		禁用	启用
10	普通用户	updateuser	/admin/users/update		启用	禁用
11	普通用户	overview	/admin/overview		启用	禁用
12	普通用户	costume	/admin/costume		禁用	启用
13	普通用户	index	/index		启用	禁用
14	普通用户	login	/login		启用	禁用
15	普通用户	usertypepmanager	/admin/auth/usertypepmanager	用户类型管理	禁用	启用
16	普通用户	usertypepeapmanager	/admin/auth/usertypepeapi/manager	用户权限管理	禁用	启用
17	普通用户	usertypepviewmanager	/admin/auth/usertypepview/manager		禁用	启用
18	普通用户	usertypepeadd	/admin/auth/usertype/add		禁用	启用
19	普通用户	usertypepeupdate	/admin/auth/usertype/update		禁用	启用

少年游-1

林花谢了春红 太匆匆 -1

© 2016 Suntao Copyright -1

3.5.9 博客(修改,新建)

见善-1

[主页](#) [所有文章](#) [我的文章](#) [新的文章](#) [管理](#)

Page Title

致郭襄

富文本编辑器

我走过山时，山不说话，  
我路过海时，海不说话，  
小毛驴滴滴答答，倚天剑伴我走天涯。  
大家都说我因为爱着杨过大侠，才在峨眉山上出了家，  
其实我只是爱上了峨眉山上的云和霞，  
像极了十六岁那年的烟花。

保存

少年游-1

林花谢了春红 太匆匆 -1

© 2016 Suntao Copyright -1



## 测试

### 4.1 引言

单元测试,集成测试,系统测试已经在开发过程中测试过了,这里需要进行测试的就是验收测试.

### 4.2 测试环境

同开发主机,环境参看开发环境.

### 4.3 测试用例及测试结果

(非标准测试用例说明)

ID	用例描述	用户角色	预期结果	测试结果
1	登录(正确用户密码)	未登录	跳转首页	✓
2	登录(错误输入)	未登录	弹出提示	✓
3	注册	未登录	跳转首页	✓
4	注销	管理员	跳转首页	✓
5	查看"我的文章"	登录的用户	只显示我的文章列表	✓
6	查看"所有文章"	任何角色	显示系统内所有的文章	✓
7	新建文章	普通用户	跳转博客	✓
8	新建一个带图的文章	普通用户	跳转博客	✓
9	查看新建的文章	普通用户	博客页面	✓
10	添加评论	普通用户	刷新博客	✓
11	修改自己的文章	普通用户	刷新博客	✓
12	删除自己评论	普通用户	页面刷新	✓
13	删除自己文章	普通用户	跳转博客列表	✓
14	修改他人的文章	管理员	修改成功	✓
15	删除他人的文章	管理员	删除博客	✓
16	删除他人的评论	管理员	删除成功	✓
17	删除一个用户	管理员	删除成功	✓
18	新建一个管理员账户	管理员	创建成功	✓
19	创建一个普通账户	管理员	创建成功	✓
20	将一个普通账户转换为管理员账户	管理员	转换成功	✓

ID	用例描述	用户角色	预期结果	测试结果
21	将一个管理员账户转换为普通账户	管理员	转换成功	✓
22	修改博客右上角 Title	管理员	修改成功	✓
23	修改页脚 Title	管理员	修改成功	✓
24	修改页脚内容	管理员	修改成功	✓
25	修改版权内容	管理员	修改成功	✓
26	修改页头和页脚的颜色	管理员	修改成功	✓
27	为普通用户提供 Custome 权限 (API)	管理员	授权成功	✓
28	为普通用户提供 Custome 权限 (View)	管理员	授权成功	✓
29	普通用户修改页头页脚颜色	普通用户	修改成功	✓
30	对于普通用户，关闭授予的 costume 权限	管理员	关闭成功	✓
31	添加一个新的用户类型	管理员	添加成功	✓
32	修改一个用户类型	管理员	修改成功	✓
33	删除一个用户类型	管理员	管理员	✓

## 4.4 负载和压力测试

利用 apache2 自带的 ab 对 nodejs 项目进行压力测试。

ab 一般直接安装 apache2 就会直接可用，但不同发行版还略有不同，例如 debian8.4 直接安装 apache2 就直接可以使用 ab，而在 Ubuntu16.04 LTS 版本中，需要安装 apache2-utils 才可使用 ab。

为做压力测试，使用 vbox 新建了两台虚拟机

Key	Value
Name	vm1
OS	ubuntu 16.04 LTS
CPU	AMD 2.9 GHz * 1 core
Memory	2650
IP	192.168.1.28

Key	Value
Name	vm2
OS	Debian 8.4

CPU	AMD 2.9 GHz * 1 core
Memory	768
IP	192.168.1.29

在 vm2 中使用 pm2 部署了 nblog 项目之后, 利用 vm1 对 vm2 进行测试.

先简单的测试一下, 一个 1 个请求 1 个并发.

```
suntao@server1: ~
Server Hostname:      192.168.1.29
Server Port:         937

Document Path:       /index
Document Length:     8213 bytes

Concurrency Level:    1
Time taken for tests: 0.027 seconds
Complete requests:    1
Failed requests:      0
Total transferred:    8665 bytes
HTML transferred:     8213 bytes
Requests per second:  36.47 [#/sec] (mean)
Time per request:     27.419 [ms] (mean)
Time per request:     27.419 [ms] (mean, across all concurrent requests)
Transfer rate:        308.61 [Kbytes/sec] received

Connection Times (ms)
      min      mean[+/-sd] median   max
Connect:    1       1   0.0      1      1
Processing: 26      26   0.0     26     26
Waiting:    26      26   0.0     26     26
Total:      27      27   0.0     27     27
suntao@server1:~$
```

## 连通测试

100 并发下 100 请求

```
suntao@server1: ~  
HTML transferred:      821300 bytes  
Requests per second:   93.21 [#/sec] (mean)  
Time per request:      1072.860 [ms] (mean)  
Time per request:      10.729 [ms] (mean, across all concurrent requests)  
Transfer rate:          788.96 [Kbytes/sec] received  
  
Connection Times (ms)  
      min  mean[+/-sd] median  max  
Connect:    6    10   2.2    10   14  
Processing: 436 1008  58.9   1010  1054  
Waiting:    428  756 152.5   768  1054  
Total:      450 1018  58.4   1020  1066  
  
Percentage of the requests served within a certain time (ms)  
 50%    1020  
 66%    1027  
 75%    1030  
 80%    1031  
 90%    1036  
 95%    1041  
 98%    1063  
 99%    1066  
100%    1066 (longest request)  
suntao@server1:~$
```

#### 单核 100 并发

可见请求都在 1s 左右被处理了,还勉强可用。

让我们测试一下高并发的情况,1000 并发,1000 请求,即 1000 用户,每个用户一个请求。

```
suntao@server1: ~  
HTML transferred:      8213000 bytes  
Requests per second:   165.09 [#/sec] (mean)  
Time per request:      6057.242 [ms] (mean)  
Time per request:      6.057 [ms] (mean, across all concurrent requests)  
Transfer rate:          1397.43 [Kbytes/sec] received  
  
Connection Times (ms)  
      min  mean[+/-sd] median  max  
Connect:    72    94  14.8    93  120  
Processing: 1135 4582 1339.6  5756  5897  
Waiting:    970 3492 1315.0  3637  5672  
Total:      1253 4676 1326.9  5849  5985  
  
Percentage of the requests served within a certain time (ms)  
 50%    5849  
 66%    5871  
 75%    5893  
 80%    5900  
 90%    5919  
 95%    5931  
 98%    5934  
 99%    5944  
100%    5985 (longest request)  
suntao@server1:~$
```

### 单核 1000 并发

可见在大并发情况下,服务器的情况并不是很好,所有的请求都需要 6s 左右返回.

接下来进行压力测试, 10000 请求, 100 并发.(Linux 负载在 1.3 左右)

```
suntao@server1: ~  
HTML transferred:      82130000 bytes  
Requests per second:   173.06 [#/sec] (mean)  
Time per request:      577.831 [ms] (mean)  
Time per request:      5.778 [ms] (mean, across all concurrent requests)  
Transfer rate:         1464.87 [Kbytes/sec] received  
  
Connection Times (ms)  
      min      mean[+/-sd] median   max  
Connect:    0        2    1.6      1    10  
Processing: 242      574   56.0     572   989  
Waiting:    118      493   66.1     495   866  
Total:      253      576   55.7     573   990  
  
Percentage of the requests served within a certain time (ms)  
 50%    573  
 66%    586  
 75%    597  
 80%    617  
 90%    646  
 95%    666  
 98%    694  
 99%    724  
100%    990 (longest request)  
suntao@server1:~$
```

### 单核 100 并发 高负载

3min 处理完 10000 个请求, 服务时间在 0.7s 左右.

经过以上测试, 项目没有崩溃, 内存占用 double, 到了 100mb 左右..

下面进行一些拓展性的测试.

关闭 vm2,更改虚拟机 vm2 的配置,增加一个核心,其他不变,开机后开启 pm2 的 cluster 模式(开启两个 nblog 进程), 做负载均衡, .再次进行 1000 并发的测试.

```
suntao@server1: ~  
HTML transferred:      8213000 bytes  
Requests per second:   203.85 [#/sec] (mean)  
Time per request:      4905.552 [ms] (mean)  
Time per request:      4.906 [ms] (mean, across all concurrent requests)  
Transfer rate:         1725.49 [Kbytes/sec] received  
  
Connection Times (ms)  
      min  mean[+/-sd] median  max  
Connect:    73   103   7.4    105   113  
Processing:  71  2369 1375.0   2360  4756  
Waiting:    71  2357 1385.0   2353  4755  
Total:     184  2472 1369.6   2464  4861  
  
Percentage of the requests served within a certain time (ms)  
 50%    2464  
 66%    3260  
 75%    3686  
 80%    3914  
 90%    4376  
 95%    4595  
 98%    4724  
 99%    4762  
100%    4861 (longest request)  
suntao@server1:~$
```

双核 1000 并发

与单核相比, 50%-60%的连接将能在 3s 内返回.

```
suntao@server1: ~  
HTML transferred:      8213000 bytes  
Requests per second:   92.58 [#/sec] (mean)  
Time per request:      1080.122 [ms] (mean)  
Time per request:      10.801 [ms] (mean, across all concurrent requests)  
Transfer rate:         783.66 [Kbytes/sec] received  
  
Connection Times (ms)  
      min  mean[+/-sd] median  max  
Connect:     0    3   5.7     1    26  
Processing:  89 1028 170.4   1057  1356  
Waiting:    79 1019 170.8   1048  1348  
Total:     110 1032 166.7   1060  1375  
  
Percentage of the requests served within a certain time (ms)  
 50%    1060  
 66%    1093  
 75%    1107  
 80%    1116  
 90%    1146  
 95%    1171  
 98%    1190  
 99%    1234  
100%    1375 (longest request)  
suntao@server1:~$
```

双核心 100 并发

```
suntao@server1: ~
HTML transferred:      8213000 bytes
Requests per second:   201.14 [#/sec] (mean)
Time per request:      1242.902 [ms] (mean)
Time per request:      4.972 [ms] (mean, across all concurrent requests)
Transfer rate:         1702.56 [Kbytes/sec] received

Connection Times (ms)
      min  mean[+/-sd] median   max
Connect:    0    7  10.6      1    28
Processing: 81 1092 325.4    1216   1349
Waiting:    66 1086 329.6    1210   1339
Total:      101 1099 317.6    1218   1350

Percentage of the requests served within a certain time (ms)
 50%    1218
 66%    1254
 75%    1289
 80%    1296
 90%    1319
 95%    1329
 98%    1335
 99%    1340
100%    1350 (longest request)
suntao@server1:~$
```

### 双核心 250 并发

由上面 100 并发和 250 并发的处理情况来看，测试用机的单核心服务能力在 100 用户左右。

另外如果在 nodejs 前增加一个反向代理服务器，由于缓存的原因,1000 并发(单核)情况也会好很多。

```
suntao@server1: ~  
HTML transferred:      216000 bytes  
Requests per second:   155.70 [#/sec] (mean)  
Time per request:      6422.709 [ms] (mean)  
Time per request:      6.423 [ms] (mean, across all concurrent requests)  
Transfer rate:         68.27 [Kbytes/sec] received  
  
Connection Times (ms)  
      min     mean[+/-sd] median   max  
Connect:    94      507 335.1    495   1899  
Processing:  428    1741 1125.8   1597   5385  
Waiting:    428    1741 1125.8   1596   5385  
Total:      923    2248 1060.9   1954   6335  
  
Percentage of the requests served within a certain time (ms)  
 50%    1954  
 66%    2365  
 75%    2964  
 80%    3214  
 90%    4042  
 95%    4050  
 98%    4469  
 99%    4693  
100%    6335 (longest request)  
suntao@server1:~$
```

#### Apache2 反向代理 单核 1000 并发

根据测试情况分析,单位服务器(测试用机)核心(core)可以(在用户体验良好的情况下)服务 50-100 个用户. 并且在高负载的情况下,系统没有崩溃.没有出现明显的内存泄漏情况.基本达到了设计的要求.



## 部署及维护

### 5.1 部署

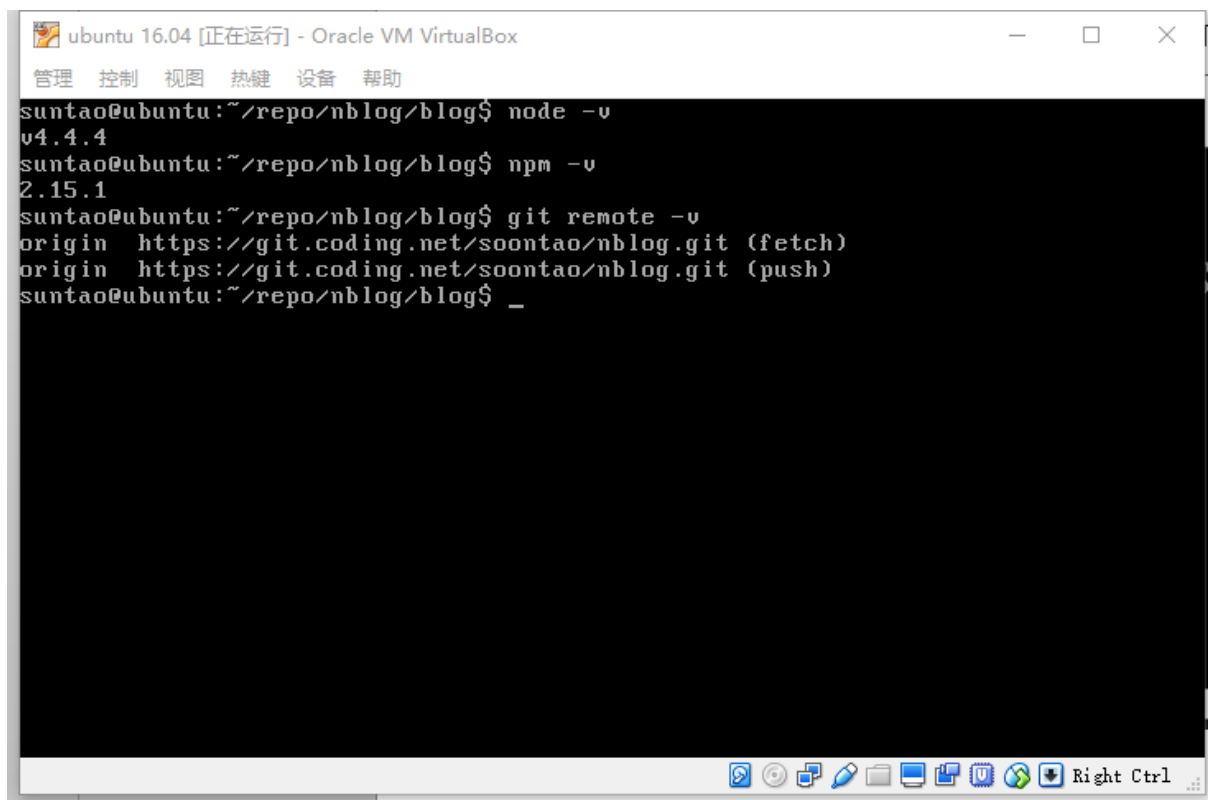
对于项目的实际部署,有以下几个方案.

#### 5.1.1 直接部署

直接部署很简单,直接在部署机器上安装 NodeJS 以及 NPM 就可以部署了.

(对于 windows 来说,nodejs 有相应的 msi 二进制发行包)

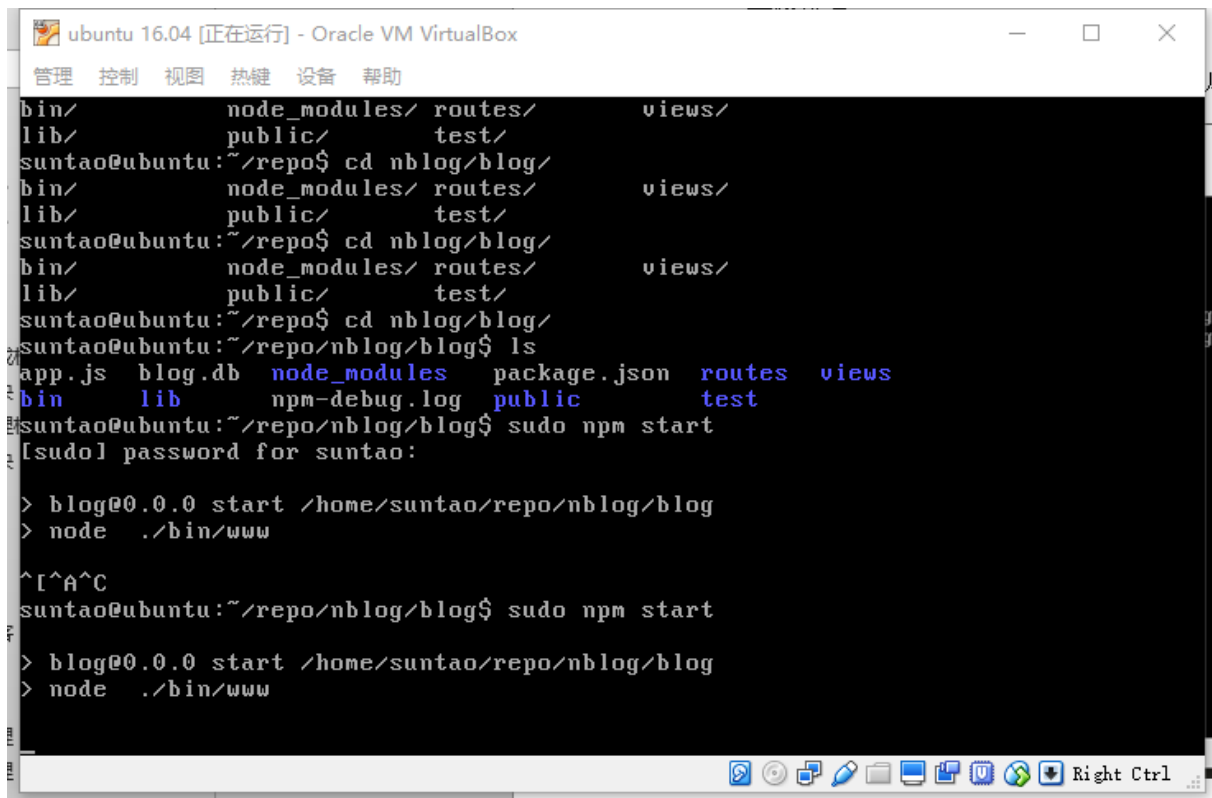
(对于 ubuntu 来说,就是 `sudo apt install nodejs-legacy npm`)



```
ubuntu 16.04 [正在运行] - Oracle VM VirtualBox
管理 控制 视图 热键 设备 帮助
suntao@ubuntu:~/repo/nblog/blog$ node -v
v4.4.4
suntao@ubuntu:~/repo/nblog/blog$ npm -v
2.15.1
suntao@ubuntu:~/repo/nblog/blog$ git remote -v
origin  https://git.coding.net/soontao/nblog.git (fetch)
origin  https://git.coding.net/soontao/nblog.git (push)
suntao@ubuntu:~/repo/nblog/blog$ _
```

在项目目录 blog 下,执行 `sudo npm install`, 安装依赖库.

执行 `sudo npm start`,开启应用



```
ubuntu 16.04 [正在运行] - Oracle VM VirtualBox
管理 控制 视图 热键 设备 帮助
bin/      node_modules/ routes/      views/
lib/      public/      test/
suntao@ubuntu:~/repo$ cd nblog/blog/
bin/      node_modules/ routes/      views/
lib/      public/      test/
suntao@ubuntu:~/repo$ cd nblog/blog/
bin/      node_modules/ routes/      views/
lib/      public/      test/
suntao@ubuntu:~/repo$ cd nblog/blog/
suntao@ubuntu:~/repo/nblog/blog$ ls
app.js  blog.db  node_modules  package.json  routes  views
bin     lib      npm-debug.log  public        test
suntao@ubuntu:~/repo/nblog/blog$ sudo npm start
[sudo] password for suntao:

> blog@0.0.0 start /home/suntao/repo/nblog/blog
> node ./bin/www

^C
suntao@ubuntu:~/repo/nblog/blog$ sudo npm start

> blog@0.0.0 start /home/suntao/repo/nblog/blog
> node ./bin/www
```

通过 <http://ip:937> 访问

### 5.1.2 通过 PM2 部署

pm2 是一个带有负载均衡功能的 Node 应用的进程管理器.

除了提供负载均衡之外, pm2 还提供一个守护进程,专门守护 nodejs 应用,当 nodejs 发生致命错误退出时,自动重启应用.

安装: `sudo npm install pm2 -g`

安装项目依赖[nblog/blog]: `sudo npm install`

启动[nblog/blog]: `sudo pm2 start bin/www -n blog`

这样就可以访问项目了

```
suntao@debian-vir: ~/repo/nblog/blog
- Real Time log display

Checkout

https://keymetrics.io/

-----

[PM2] Spawning PM2 daemon
[PM2] PM2 Successfully daemonized
[PM2] Starting bin/www in fork_mode (1 instance)
[PM2] Done.

| App name | id | mode | pid | status | restart | uptime | memory | watch |
|-----|-----|-----|-----|-----|-----|-----|-----|-----|
| blog | 0 | fork | 14091 | online | 0 | 0s | 20.676 MB | disabled |

Use `pm2 show <id|name>` to get more details about an app
suntao@debian-vir:~/repo/nblog/blog$ pm2 list
```

### 5.1.3 反向代理部署

80 端口只有一个,但是 Web 项目却不止一个,所以为了避免项目启动时端口已经被占用,web 项目的默认端口一般不是 80 端口,例如 J2EE 的默认端口是 8080 和 8443,NodeJS 的默认端口是 3000

以本人的服务器为例,服务器上面部署了两个项目,其中一个是 nblog 项目,此外还有一个 API 服务器,负责另一个项目的移动端和网页端的 API,很明显这两个项目都需要 80 端口(为了访问方便),但是一个服务器只有一个 80 端口,不能让两个完整的 http 服务器共用。

虽然开发人员也可以直接使用 8080 或者 3000 之类的周知端口访问,但是对于普通的非 IT 专业人员,他们只会使用简单的 URL 访问。

这种情况下,最简单的方式就是通过 apache 或者 ngx 反向代理。由于对于 apache 比较熟,所以这个示例使用 apache2 服务器做反向代理(实质上 ngx 性能更佳)。

首先在 DNS 域名服务商那里解析二级域名(我已经拥有该域名的顶级域名),将 blog.suntao.science 指向我的服务器。

然后编辑 apache 的 vhost 配置文件(由于我的 apache 是编译安装的,配置文件路径为 /usr/local/apache2/conf/extra/httpd-vhosts.conf)

```
suntao.science - PuTTY
<VirtualHost *:80>
  ServerAdmin service@domain.com
  ServerName blog.suntao.science
  Redirect / https://blog.suntao.science
</VirtualHost>

<VirtualHost *:443>
  ServerAdmin service@domain.com
  ServerName blog.suntao.science
  ProxyRequests Off

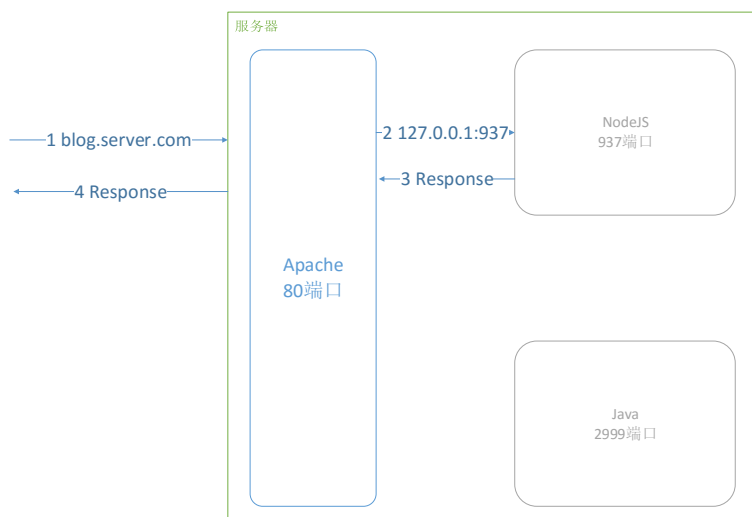
  #反向代理配置
  ProxyPass / http://suntao.science:937/
  ProxyPassReverse / http://suntao.science:937/
  SSLEngine on
  SSLProxyEngine on

  <FilesMatch "\.(cgi|shtml|phtml|php)$">
    SSLOptions +StdEnvVars
  </FilesMatch>
  <Directory /usr/lib/cgi-bin>
    SSLOptions +StdEnvVars
  </Directory>
  SSLCertificateFile /root/ssl/server.crt
  SSLCertificateKeyFile /root/ssl/server.key

  BrowserMatch "MSIE [2-6]" \
    nokeepalive ssl-unclean-shutdown \
    downgrade-1.0 force-response-1.0
  # MSIE 7 and newer should be able to use keepalive
  BrowserMatch "MSIE [17-9]" ssl-unclean-shutdown
</VirtualHost>

<VirtualHost *:443>
```

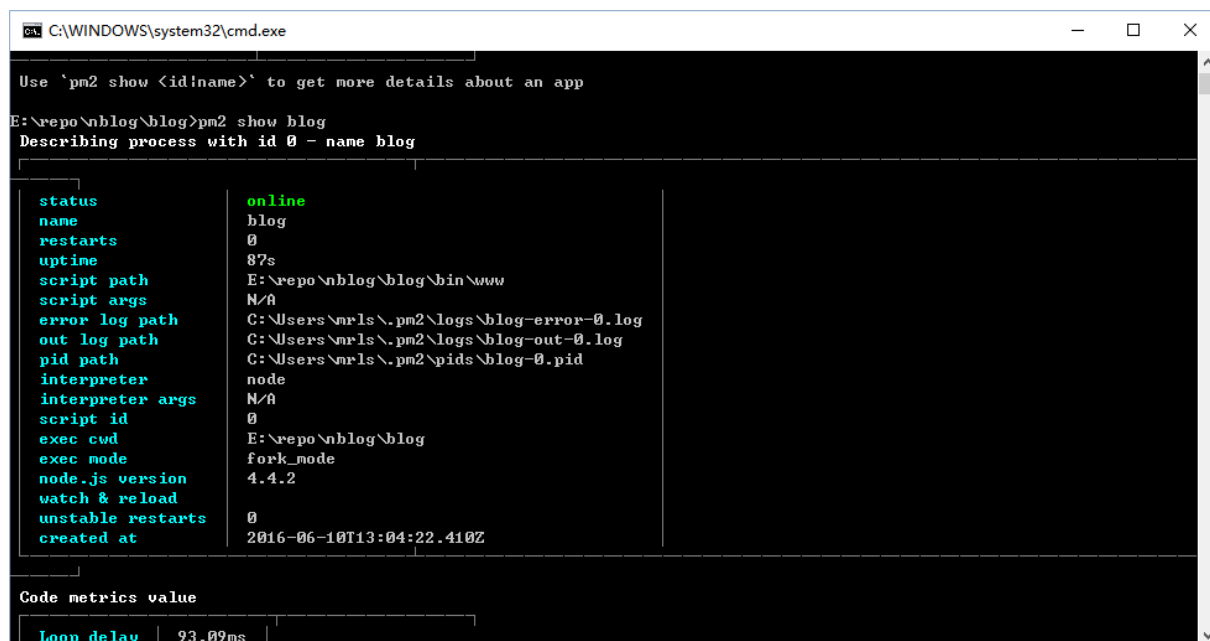
该配置文件指定了访问 <http://blog.suntao.science/> 会自动重定向至 https,同时指定了访问 <https://blog.suntao.science/> 会自动代理访问至主机的 937 端口,同时该配置也指定了 https 的证书.



像 5.1.1 或 5.1.2 那样部署 nblog 之后,然后重启 apache 即可访问.

## 5.2 维护

如果使用 pm2 进行部署的话,维护就会比较容易.pm2 会自动记录下输出和 error 日志,并提供自动重启的功能.



```
C:\WINDOWS\system32\cmd.exe

Use 'pm2 show <id|name>' to get more details about an app

E:\repo\nblog\blog>pm2 show blog
Describing process with id 0 - name blog



|                   |                                          |
|-------------------|------------------------------------------|
| status            | online                                   |
| name              | blog                                     |
| restarts          | 0                                        |
| uptime            | 87s                                      |
| script path       | E:\repo\nblog\blog\bin\www               |
| script args       | N/A                                      |
| error log path    | C:\Users\nrls\.pm2\logs\blog-error-0.log |
| out log path      | C:\Users\nrls\.pm2\logs\blog-out-0.log   |
| pid path          | C:\Users\nrls\.pm2\pids\blog-0.pid       |
| interpreter       | node                                     |
| interpreter args  | N/A                                      |
| script id         | 0                                        |
| exec cwd          | E:\repo\nblog\blog                       |
| exec node         | fork_mode                                |
| node.js version   | 4.4.2                                    |
| watch & reload    |                                          |
| unstable restarts | 0                                        |
| created at        | 2016-06-10T13:04:22.410Z                 |



Code metrics value



| Loop delay | 93.09ms |
|------------|---------|
|------------|---------|


```

如果直接使用命令行部署的话,可能需要使用 `nohup npm start > out.log &` 命令(Linux),后台运行并保证终端退出后不会终止系统,同时将输出重定向到 `out.log`.

虽然系统比较小,但还是需要使用 `top` 和 `ps` 查看进程信息,每周重启该项目,保证该系统的表现.

---

## 总结与展望

通过设计实现博客系统, 我对于 JavaScript 等弱类型动态语言有了更加深刻的了解, 对于权限管理系统的实现有了初步的想法, 掌握了前端 CSS 框架的基本使用方法.

就一个小型的博客系统而言, 本系统已经做得很好了, 在有限的访问量下, 响应时间足够短, 并且博客功能基本实现, 还兼有一些额外的个性化功能.

虽然系统完成了, 但是有以下不足:

1. 这个项目终究是一个偏后端项目, 所有的动态页面都是后端渲染的, 前端只有少量的 JS, 这样虽然简化了前端编程, 但事实上也使前端不够灵活(远远不及 C/S 架构灵活),
2. 对于移动端的扩展也很差.
3. 由于服务器承担了大量的渲染工作, 导致服务器的负载不够理想.
4. 由于模板引擎的存在, 前端的可移植性很差(必须要经过模板引擎渲染, 而其它 Web 平台不一定有 EJS 模板引擎)
5. 由于第一次接触 NodeJS, 项目的架构(实现)混乱, 一些文件放的很乱, 没有写出最佳实践.
6. 也是由于第一次接触 NodeJS, 陷入回调地狱中, 具体来说就是回调函数多层嵌套, 这样写没什么问题, 就是不太整洁.

以上不足可以有以下完善方案.

1. 前后端分离, 后端只提供 API, 前端调用 API, 通过浏览器进行动态渲染.
2. 使用 Promise 或者 Async, 使回调函数顺序调用, 避免回调函数嵌套.

希望后来者可以在这个项目的基础上, 保留优点, 改进缺点, 做出一个更加完善的 Web 项目.

---

## 参考文献

- [1] Node.js Foundation. Node.js v4.4.5 Documentation. <https://nodejs.org/dist/latest-v4.x/docs/api/>. 2016
- [2] 张丹(Conan). Nodejs 异步流程控制 Async. <http://blog.fens.me/nodejs-async/>. Sep 13, 2013
- [3] Materialize. Materialize. <https://github.com/Dogfalo/materialize> . 2016.
- [4] The Apache Software Foundation. Apache 虚拟主机文档. <http://httpd.apache.org/docs/2.2/vhosts/> . 2016.
- [5] iloveyin. Python Requests 快速入门. <http://blog.csdn.net/iloveyin/article/details/21444613> . 2014.
- [6] zhangxin09. EJS 模板快速入门. <http://blog.csdn.net/zhangxin09/article/details/18409119/> . 2014
- [7] dresende 等. node-orm2. <https://github.com/dresende/node-orm2> . 2016.
- [8] StrongLoop, Inc., and other expressjs.com contributors. Express 4.x API 中文手册.<http://www.expressjs.com.cn/4x/api.html> . 2016.