# 基于node.js的个人博客系统设计与实现

## 摘要

在社交网络飞速发展的今天，博客成为一种分享科技进步的方法。写博客的出发点是不尽相同，博客的内容也是多种多样的。从日志到对知识的总结，从自我的情绪的发泄，到团队合作的信息共享。

我设计并实现了一个个人博客系统，采用了浏览器/服务器模式构建。系统由前端页面展示内容和后端提供数据服务。最主要的是探索了由node.js提供后端实现。完成了一个能够实现博客主人发表管理自己的文章，上传并展示图片，访客能够浏览博主的文章与相册，同时可以给博主文章进行评论与对博主进行留言等诸多功能个人博客系统。最后对于这个系统采用了黑盒测试方法，覆盖了所有的功能点的测试。

关键词：浏览器/服务器模式，个人博客系统，异步I/O。

# 第一章 绪论

## 1.1 概述

由于人均网络带宽的提升，各种网络技术能实现的功能越来越强大，还有人们日益增长的对非物质消费的需求，web2.0被提了出来。Web2.0是资源平等的体现。Web应用可以让人了解到目前万为我网正在进行的一种改变-----从一系列网站到一个成熟的为最终用户提供网络应用的服务平台。平且随着网络技术的提升，web服务将在很多用途上最终取代桌面计算机应用。Web2.0并不是一个技术标准，不过他包含了技术架构及应用软件。他的特点是鼓励作为信息最终利用者通过分享，是的可供分享的资源变得更丰盛。网络成为了新的平台，内容由每一位用户产生，参与所产生的个人话内容，借由人与人的分享，形成了新的web的世界。

## 1.2 研究目的及现状

本课题的主要目的是通过运用新近发展的Node.js作为服务器端，Node.js上层的服务器框架Express.js ，前端使用JavaScript框架jQuery、还有层叠样式表CSS完成一个个人博客系统。通过完成一个博客系统，能够学习相关技术并且熟练使用。以期为今后的工作积累一定的开发经验与能力。

这里同时希望利用丰富的网络技术实现一个个人博客系统，完成对平时学习到的知识的总结归纳的记录，还有分享的目的。

个人博客来源于英文blogger，又被译作部落格等，通常是属于个人的，由个人管理，不定期的更新新的文章或者分享内容的网站。博客上的内容通常根据更新的时间，由新到旧进行展示。

许多的坐着专注于研究学习新的技术，然后对自己所学习到的知识进行整理总结然后分享出来，同时也可以写一些个人日记。一个博客，包含文字，图片，链接到其他文章或博客网站的超链接。也有作者会提供，浏览者评论的地方，可以让读者以互动的方式留下意见，或者可以提出疑问。目前网络上提供的许多个人博客系统有下列几种

网上几种流行的博客系统:

(1) Wordpress 是使用 PHP 同时前后端语言和 MySQL作为数据库支撑开发的开源、免费的 Blog引擎，用户可以安装后很方便的初始化属于自己的博客。

(2) PJ-Blog 是使用Asp+access技术的开发的博客系统。

(3) Z-Blog 与其他的相比较则较为小巧–同时著名的科技博客，月光博客 就是使用的z-blog的博客程序。

(4) EMLog 同样是一款PHP同时前后端寓言和MySQL作为数据库的博客系统。但是在支持个人编辑的同时还支持多人编辑。

网络上已经有了这么多成熟的博客系统，但是可以到，开发者为了满足越来越多的需求，代码量十分巨大，而且许多功能其实优先级并不高，但是实现他们花费了巨大的代价。已经脱离了，单纯的个人博客，发表文章，分享知识的目的。所以现在重新实现一个个人博客，让自己可以发表文章，分享知识或者记录自己的一点感受，同时学习技术，提升自己的能力，对这些新的技术能够熟练度运用，所以提出了这个课题。

## 1.3 相关技术介绍

### 1.3.1 Ajax

Ajax是异步的javascript与xml的简写（asynchronous javascript and xml)是一组web开发的技术。大部分的前端开发工作都是使用ajax来创建异步的web程序的。使用ajax，web程序可以传输数据给服务器，同样也可以在后台运行从服务器获取数据的工作而不会阻塞现有页面的工作与行为。传统的web应用允许用户填写表单，当提交表单时就想网页服务器发送一个请求。服务器接收并处理传来的表单，然后送会一个新的网页，但这个做法浪费了许多带宽，因为再前后两个网页终的大部分html码往往是相同的。由于每次应用的沟通都需要想服务器发送新的请求，引用的回应时间依赖于服务器的回应时间。这导致了和用户界面的回应比本机应用慢的多。而ajax应用可以仅向服务器发送并取回必须的数据，并再客户端采用javascript处理来自服务器的回应。因为在服务器和浏览器之间交换的数据大量减少。同时很多的处理工作可以在发出请求的和客户端机器上完成，因此web服务器的符合也减少了。Ajax不是指一种单一的技术，二十有机的利用了一系列相关的技术。虽然其名称包含xml，但实际上数据格式可以由json代替

### 1.3.2 git

版本控制是一种记录若干文件内容哦那个变化，以便将来查阅特定版本修订情况的系统。Git是分布式版本控制系统。客户端哥并不只提取最新的版本的文件快照，二十把原始的代码仓库完整的镜像下来。这么依赖，任何一处协同工作用服务器发生故障，时候都可以用任何一个镜像出来的本地仓库恢复因为每一次的提取操作，实际上都是一次对代码仓库的完整备份。

### 1.3.3 node.JS

Node.js是一个开放源代码、跨平台javascript语言运行环境。Node.js是由node.js基金会持有和维护的。采用google开发的V8运行代码，使用事件驱动、非阻塞和异步输入输出等技术来提高性能，可优化应用程序的传输量和规模。这些技术通常用于数据密集的事实应用程序。

Node大部分基本模块都是javascript语言编写。Node.js的出线使得javascript也能用于服务器端编程。并且node.js含有一些列的内置模块，使得程序可以脱离apache HTTP Server或IIS，作为独立服务器运行。目前node.js已经被IBM、Microsoft等企业采用。

Node.js已经有数十万玩模块，他们可以通过一个名为npm的管理器免费下载。

Node编写的web服务器可以同时处理 许多客户端链接，有新连接它就执行下回调函数， 所以每个连接只在堆里占用一小块内存。 这与当今大多数并发模型都不相同，它们都是使用操作系统的线程来实现的。基于 线程的网络交互相对而言不是那么高效，而且非常难以使用。Node在高负载的情况下 比那些要给每个链接分配2兆线程栈内存的系统要高效得多。Node里几乎就没有直接进行I／O的函数，所以线 程永远不会阻塞。正因为永不阻塞，非专家级的程序员也能快速开发系统了。 Node．JS的目标是成为快速搭建容易扩展的网络应用的工具。它的几个特点使得它 十分让人兴奋：

(1) javascript特別适合回调函数式的编程，即事件驱动。

(2) Javascript程序员早已理解如何构建事件驱动软件了：当用户点击、当ajax请求返回时、当页面加载完时，都会触发函数执行。

(3) node天生就很是非阻塞的,即使是文件系统或者DNS查询也是基于非阻塞的api。

### 1.3.4 express

Express.js是一个小而强大的Node.js的Web程序框架，它提供了一系列强大的功能来创建各种web应用程序。 在大量原生的HTTP工具方法和Connect中间件的帮助下，用户可以快速又方便的创建强健且用户友好的API。 Express仅仅是给应用程序包装了一层轻量级的功能函数，不会影响任何node.js特性。Express．JS是构建在Node．JS上的Web框架。Express．JS的多种功能使得架构Node 服务器变得非常快速。 它拥有健壮的路由，重定向帮助器，动态view帮助器，应用层view选项，内容写 上，应用挂载，高性能，view渲染和partials支持，环境配置，基于对话的flash提醒机 制等特点。另外，Express．JS是架构在Connect中间件上，基于Connect我们可以将很多 共性的东西抽象出来，这样就不用做重复的工作。

### 1.3.5 mysql

许多世界上最大和发展最快的组织包括facebook，google，adobe依靠mysql来节省事件和金钱，为大量的网站，关键业务系统和打包软件提供支持。Mysql再过去由于性能高，可靠性号已经成为最流行的开源数据库。并且mysql为多种编程语言提供api，这些语言包括C,C++,PHP。Mysql支持多线程，充分利用cpu资源，支持多用户。优化的sql查询算法，有效的提高查询速度。

### 1.3.6 jquery

Jquery是一个快速的，小巧的，功能丰富的javascript函数库。它使得对html元素的dom遍历与操作，相应事件，动画，还有ajax请求由于更加友好易于使用的api而工作再大多数的主流浏览器上。通过多功能和可扩展性的结合，jquery改变了数以百万人计的编写javascript方式。

# 第2章 个人博客应用的需求分析

### 2.1 需求概述

个人博客需要具有登录首页，进入个人博客详情页面，查看个人简介，还有博主发布的各种技术文章。并且博主发布的所有文章都必须是可以检索的，必须是可以很方便的被查找的。

所以可以给文章添加侧边栏链接，并且对文章进行按照时间进行排序。这样方便来访者对特定文章进行访问。

开放了访客功能，可以对博主进行留言，可以是和博主打个招呼，也可以是对博主发表的文章的点评，如果是文章中有些错误或者观点不能认同也可以指出来，欢迎大家进行交流。

### 2.2 系统功能性需求

在上面的可行性分析阶段已经大致的分析和讲述了本系统的基本需求，也论证了软件系统开发的可行性，但为了明确设计此软件系统的目标，了解用户的需求，现在对用户的需求做了明确清晰的调研，从而进一步明确所需要完成的每一个子模块的功能。

系统功能结构如图其中包括：

（1）博客首页界面，首页具有跳转留言页面，管理员登录页面，查看博客文章的页面，进入博客主人的个人相册的功能。

（2）查看博客文章界面：按照博客文章的分类进行检索文件的功能，查看所有文件总览的功能。

（3）查看详细文章的界面：查看每一篇详细文章，对当前查看的文章进行添加评论的功能，通过底部的链接快速跳转到下一篇文章，或者上一篇文章的功能。

（4）留言界面：对博客主人进行留言，需要对留言内容或者访客昵称填写为空是进行校验，不得提交。以及当留言成功时需要将最新添加的内容刷新显示到页面中给予用户成功添加的反馈。

（5)个人相册：以轮播图的形式展示博客主人的发表的个人图片。

(6)行使管理员权限：添加或者删除文章，添加删除管理文章的类别，添加或者删除发表的图片。

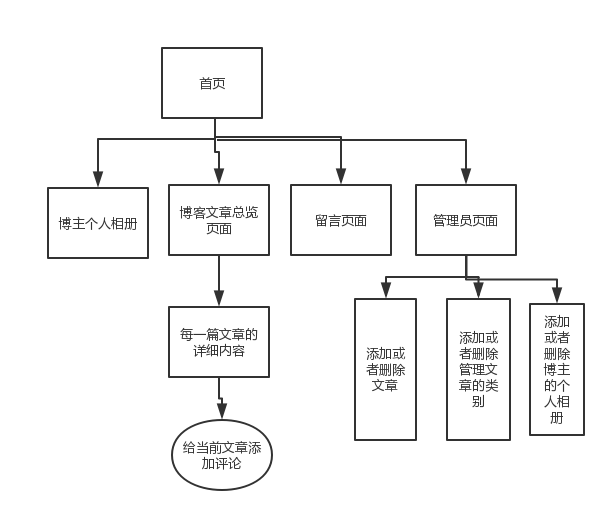


图 2-1 系统功能结构图

### 2.3 系统服务端功能模块划分

根据系统功能的分析可以画出服务器端功能模块图：

1. 管理员管理文章与图片等内容包括，添加或者删除文章，添加或者删除个人你相册的图片。
2. 文章模块包括按照文章类别进行对文章的检索，按照具体文章id进行检索，对数据库中文章的书进行添加和删除。
3. 留言模块包括向数据库中添加留言数据，向数据库中进行查询留言数据。
4. 返回数据库中对统一管理的图片的数据返回前端。

### 未命名文件 (2)

图 2-1 系统功能模块图

## 2.4 用例文档

用例用于描述系统中所有用户的功能。 用例文档详细的描述了系统用例参与的业务，该系统主要的参与者有：

访客用户：阅读博主发表的文章，查看博主的贴出来的个人相册，对博主进行留言。

系统管理员：管理自己的文章，管理自己的个人相册，查看自己的图片，查看自己的文章。

### 2.4.1 普通用户用例图

普通用户用例图清晰列出了，普通用户访问博客系统是主要的几项功能，有浏览文章，对博主进行留言，查看博主的相册。

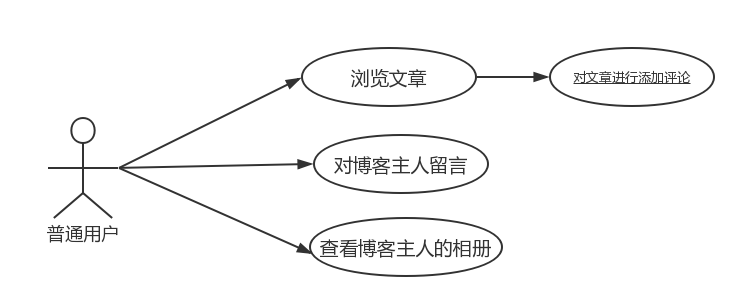


图 2.4.1普通用户用例图

### 2.4.2 管理员用户用例图

管理员用户用例图也显示管理员能够行使的主要管理员权限，有管理自己的文章和管理自己的相册。

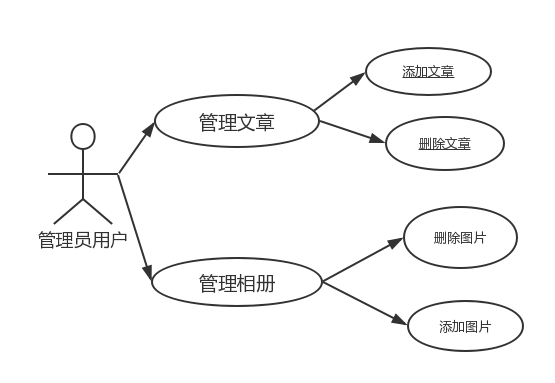


图 2.4.2管理员用户用例图

## 2.5 系统非功能性需求

### 2.5.1 系统可靠性

系统能否正常运行取决于系统的可靠性、我们采用的html,css,javascript等前端开发技术都是经过大量的工程检验的技术。而且node.js的后端开发也已经被例如百度、阿里、腾讯等大型公司加入具体的项目中实践。Mysql作为全世界使用最为广泛的关系型数据库也是非常可靠的。所以该系统从理论上来说是可靠的。

2.5.2 系统可扩展性

在要添加新功能或者新的模板时，扩展应该是方便、即插即用的。在设计本系统时， 应该充分考虑模块化设计，分割数据处理、页面展示、逻辑控制等层面。这样以后加入 新的功能的时候，无需全盘改动，只需嵌入相应的代码即可。

2.5.3 系统安全性

因为javascript代码是单线程的，当代码运行出错的时候会阻塞掉，前端页面会卡死，后端node.js程序也会报错。所以我们在对json数据的解析或者是各种事件的控制的回调我们都采用了try...catch来捕获错误。还有对ajax的请求的部分也对error出现的时候采用专门的函数来捕获。

2.5.4 系统可交互性

Web应用的优势在于它的可操作性。无论是在什么平台，比如Linux，Windows， OS X桌面端还是在手机移动端，只要有浏览器就可以浏览，评论。浏览器帮助我们屏蔽了操作系统之间的差异，是的不论实在什么操作系统，什么终端上都能获得统一的体验。

### 系统的开发方法

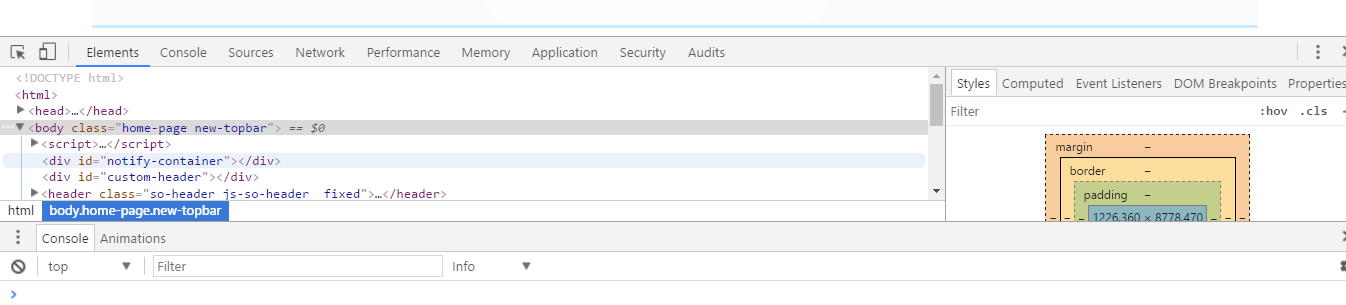
前端页面的开发使用了较为原始的开发方式，jQuery作为javascript库来进行各种动画，计算，事件处理。通过css（层叠样式表）来控制页面的样式与色彩，增加网页的美感。合理的配色总是能给人更好的体验。

同时页面是可交互的，交互的操作使用了ajax（异步请求），在不阻塞页面的情况下向后端请求需要更新的数据然后执行局部的刷新页面增强体验，或者想后端传递数据进行存储。

所有的交互操作都是需要路径（url)的才能到达后端的服务器的，所以我们需要定义好所有的请求的url链接。定义链接的过程中，所有的url都希望是语义化的，这样在真正的项目开发过程中很多时候是前后端分离的，方便大家开发的时候进行协作。能够从众多的url路径中较好的分辨哪一个路径是需要完成一个什么样的操作。

后端是采用Node．JS开发完成，所有相关技术均是围绕Node．JS配套的工具。 Node．JS的基础上，使用Express．JS Web框架，构建应用。将路由配置尽量统一放在某一个目录下面这样方便后续添加，在文件中定义好每一条url请求的controller（即对应的回调函数），当访问相应页面的时候，路由系统解析到路由，然后根据配置好的路由数据，找到对应的模板，和对应的页面文件将页面返回给前端浏览器，浏览器做出跳转的动作，然后进行渲染html文件。或者调用对应的controller函数完成操作，操作可能只是对数据库进行增加一条数据，或者删除一条数据，但是我们还是希望能够返回对应的操作结果给前端。前端能够知道操作是否完成的具体信息，然后将结果以合适的信息进行展示出来这样对用户更加友好。

还有连接mysql部分的代码，放在一个文件中，然后其他的文件需要操作mysql的时候进行导入就好了，避免过多的重复代码。

用叻

## 第3章基于node.js的个人博客应用概要设计

**3.1系统架构设计**

Node．JS博客系统使用三层B／S(浏览器/服务器)架构模式，B／S模式是C／S模式的发展产物，它与C／S模式的区别在于，用户在客户端无需安装任何专业软件只需要 通过浏览器并且确保可以连接互联网就可以实现对服务器的访问。 B／S模式的优点在于：

1. 与C／S模式比较，B／S模式维护与升级简单。如果使用C／S模式，必须在所 有的客户端安装专业软件，无疑增加系统维护难度以及系统维护人员的工作量。而使用 B／S模式对于客户端而言只需要安装浏览器，系统维护人员可以通过浏览器实现对系统 的远程维护与升级，减少了系统维护的工作量与时间，提高了工作效率。

B／S模式可以实现使用户跨平台操作系统，因为各种平台用户能够通过浏览 器访问服务器。

降低系统开发成本，提高系统运行稳定性。目前主流服务器包括Windows 服务器与Linux服务器，与Windows服务器相比，Linux服务器基本安装开源软件无需 支付软件费用，Linux内核安装保护功能模块、减少碎片文件，确保系统的稳定性。

提高系统的安全性，B／S模式把数据集中于数据库服务器，客户端没有存储 任何业务数据以及与数据库连接的信息，保证了系统的安全性。

数据实时性，B／S模式通过浏览器的刷新可以实时掌握业务数据变化，方便 企业快速决策。

1. 降低服务器运行数据负荷，如果同时把事务逻辑程序与数据库放在一台服务 器(应用服务器)中，那么必然导致应用服务器的负荷增加，延长服务响应时间，甚至 可能导致服务器崩溃，造成数据丢失。 B／S模式的缺点在于：

无法满足个性化功能需求。

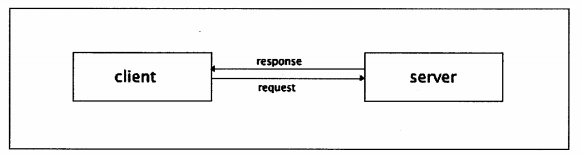
以鼠标为基本操纵方式，无法满足快速操纵要求。

页面动态刷新，响应时间明显下降。

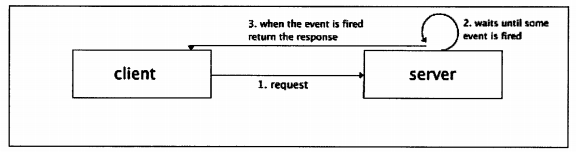
无法实现分页显示，数据库访问压力增加。

(5)功能弱化，难以实现传统模式的特殊功能要求。

我们可以通过两张图来看看Node工作的方式与普通的web服务器工作方式有何不 同。这时普通的web服务器的工作方式。当请求到来的时候，服务器尽快回复，让应用越快越好，这是每一个web开发小组的目标。



这里服务器并不立马回复，而是当“事件”发生时才回复。可以把Node看做一个 经典的聊天服务器。没有人发信息，其他人也收不到信息。Node开启服务候便一直等待事件发生，当事件发生时才激活然后服务。



### 3.1可行性分析

所谓可行性分析，是关于项目的内容以及条件：例如供应需求，设备规模，环境利 益等方面进行调查和分析，预测项目完成后可能获得的经济和环境效益进行预测，提出项目是否应该投资以及运行方式的意见的一种系统分析方法。可行性研究是为保证投 资效果而在项目实施之前从经济、技术、管理等角度对项目进行的全面综合分析。它的 基本任务包括从经济视角出发进行全面的分析研究、对项目运营后所产生的经济效益进 行预测、对已经确定的几种方案论证进行选择。它的目标是合理利用资源，实现预定的 社会效益与经济效益。

### 3.1.1技术可行性

主要从项目实施的技术角度，合理设计技术方案， 并进行比选和评价。 Node．JS上的Express．JS框架可以方便实现众多功能：新建Server，配置，设置路 由，中间件，错误处理，参数预处理，渲染，Partials，View查找，模板引擎合作，Session 支持都非常全面。 同时开发过程中所使用的各种基本技术html，css，javascript都是经过很长时间的发展，都是非常可靠，资料十分丰富的技术。

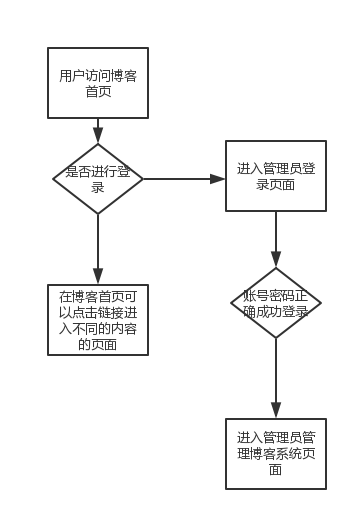
**3.1.2经济可行性**

本系统主要使用的是开源软件，atom编辑器，和开源框架如jquery，express等，同时服务器可以后期转换成阿里云的个人云服务器。因为需要满足的同时访问请求并不高，所以并不需要投入特别多的费用，本项目成本控制基本可控。

**3.1.3操作可行性**

本系统使用的是用户前端使用鼠标点击然后完成交互，目前绝大多数的web系统都是采用的这种交互方式，通过前端点击表明需要完成的操作，然后后端完成对数据等的修改完成整个交互。因此本系统是具有操作可行性的。

这是用户访问博客的流程图



3.2.1博客首页：

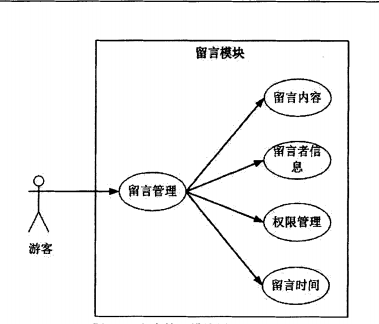


博客首页页面，包含进入留言/个人随笔文章/我的github等的链接选项。

3.2.2留言模块：

包含访问者的昵称（个人ID），留言内容，留言时间等各种基本信息，通过输入或者系统自动生成，然后点击提交按钮通过ajax请求发送到后端，node后端接收到前端传来的留言数据插入到数据库完成，同时给予前端成功或者失败的信息，前端接收到如果成功则展示，如果失败则给予用户留言失败的提示。

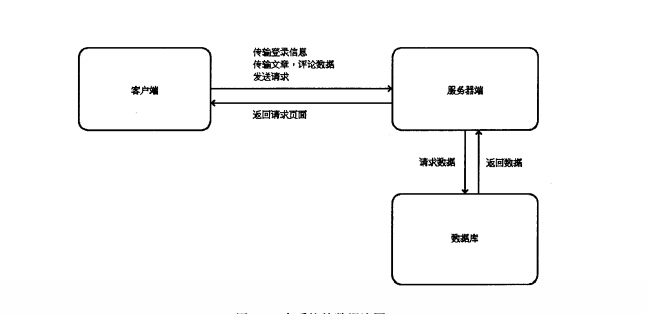
留言模块用例图



留言管理的具体页面：



数据流图：本系统是属于web应用，web应用分客户端，服务端的。因此本系统的数据遍在三个模块之间流动：客户端，即浏览器；服务端，即node服务器；数据库，此处为mysql数据库。



我们可以看到，所有的操作由前端触发，发出相应的请求，可能是登录请求，也可能是写入评论的请求，可能是发表留言的请求，或者请求新的页面。这些请求都会出给node服务器端。然后node服务器向数据库请求相应的数据或者执行相应的修改，写入操作，最后组织

成相关页面或者返回执行结果给前端。

3.3.3文章管理

本系统的主要内容就是文章。所以我们要为博主提供文章的增删改查功能(CRUD)

发布新文章：新建文章，并且在数据库中存储，使之持久； 基于Node．IS博客系统的设计与实

删除文章：已有文章可以删除； 另外，每篇文章有它自己的评论内容。

3.3.4文章的分类的类别管理

文章可以被添加标签，标示属于哪一个分类的。这个类别可以被添加或者删除。

3.3.5个人相册管理

(1)博客主人可以在博客的个人相册进行晒以下自己的日常，可以是出去旅行的也可以是和朋友的聚会的照片，这些照片需要可以添加也可以被删除。

## 3.3数据库设计

数据库是存储，管理，组织系统的数据的部分。根据数据之间的相关性，来组织他们之间的关系，构建数据表格，来构建数据库的架构，这里使用的是mysql，是一种关系型数据库，通过数据之间的关系来连接，存储他们。使用数据库有很多的优点：节省存储数据的空间，降低数据冗余度，节省存储空间。现在的数据库系统相比之前有很多优点，例如操作直观，使用灵活，数据处理能力强等优点。

同时系统的性能与可扩展性也很依赖与数据库的设计。数据库的设计需要满足系统的几项基本需求：

数据库信息要能满足整个系统对数据的所有处理需求

能够容易被数据库管理员维护

数据的一致性，完整性要好

数据库的选择

此处是以博客的主人和游客之类的以用户为单位进行存储数据的。因此，这里每一篇文章都属于博客的主人。同时mysql是世界上应用最广泛的关系型数据库，经过 工程的实践检验。著名的月光博客就是采用的mysql来实现的，所以这里采用了mysql作为本博客系统的数据库软件。

数据图

数据库存储着数据，支撑整个系统，对数据库的设计影响整个系统的性能与可扩展性。整个系统包含的信息有：博客主人的文章信息，访客对主人的留言信息，访客对主人的文章的评论性信息，最后为了系统的图片的统一调度设置了图片信息存储的表单。

文章：文章id（primary key）， 文章名， 年份， 详细时间， 存储地址， 类别id（foreign key）

文章所属类别： 类别id（primary key）, 类别名称

访客对主人的留言： 留言id（primary key）， 用户名， 图片， 内容， 时间

访客对主人文章的评论： 评论id（primary key），用户名， 图片， 内容， 时间， 文章id（foreign key）

图片： 图片id（primary key）, 图片地址

### 数据库的物理设计

1、所有数据库表格汇总：

|  |  |
| --- | --- |
| 表名 | 功能说明 |
| Article | 储存文章的相关信息 |
| Category | 储存文章所属类别的名称与对应id的信息 |
| Comment | 储存不同的文章的评论 |
| Message | 储存所有的对主人的留言 |
| Image | 储存相册的所有图片 |

1. 文章表格

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| 字段名称 | 字段含义 | 类型（精度） | 是否可以为空 | 约束条件 |
| articldId | 文章对应的id | Int(11) | 否 | 主键（primary key） |
| ArticleName | 文章的名字 | Varchar(30) | 否 |  |
| year | 年份 | Int(8) | 否 |  |
| Date | 详细日期 | Varchar(20) | 否 |  |
| address | 文章对应的地址 | Varchar(20) | 否 |  |
| categoryId | 文章所属类别的id | Int(11) | 否 |  |

1. 所有类别的表格

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| 字段名称 | 字段含义 | 类型（精度） | 是否可以为空 | 约束条件 |
| categoryId | 类别的id | Int(11) | 否 | 主键（primary key) |
| CategoryName | 类别的名称 | Varchar(30) | 否 |  |

1. 用户留言的表格

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| 字段名称 | 字段含义 | 类型（精度） | 是否可以为空 | 约束条件 |
| messageId | 留言的id | Int(11) | 否 | 主键（primary key) |
| useName | 留言者的名字 | Vercahr(30) | 否 |  |
| Image | 留言者的头像 | Varchar(60) | 否 |  |
| Content | 留言的内容 | Varchar(255) | 否 |  |
| Date | 留言的时间 | datetime | 否 |  |

1. 用户评论

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| 字段名称 | 字段含义 | 类型（精度） | 是否可以为空 | 约束条件 |
| CommentId | 评论id | Int(11) | 否 | 主键(primary key) |
| UserName | 评论者的名字 | Varchar(45) | 否 |  |
| Image | 评论者的头像 | Varchar(60) | 否 |  |
| content | 评论的内容 | Varchar(255) | 否 |  |
| Date | 评论的时间 | Datetime | 否 |  |

1. 图片的表格

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| 字段名称 | 字段含义 | 类型（精度） | 是否可以为空 | 约束条件 |
| imageId | 图片的id | Int(11) | 否 | 主键（primary key) |
| address | 图片的存储地址 | Varchar(45) | 否 |  |

## 系统的详细设计

### 系统分析

本系统采用的是前后端分离的开发模式。前端负责交互与展示，后端处理数据与业务。很多的php或者jsp开发过程中，是后端给什么前端就渲染什么。这样也有很多的好处，比如前端开发的时候只要本地起一个tomcat或者apache就能开发，只要业务并不太复杂就依然可以顺畅的进行调试。但是后面当项目越来越大，参与开发的人员越来越多的时候前端搭建本地开发环境将变得比较困难，同时service互相调用越来越多前端的开发更加困难。所以为了解决这个问题，我采用了前后端分离的模式。前端使用css，html来开发页面，javascript来控制页面，完成交互需求。后端使用node.js完成对业务逻辑的开发，和提供用于ajax交互的api，这样来完成前后端的数据的传递和处理。

系统主要分为前端页面，后端node.js的业务处理与数据服务的api，由node.js进行读取、写入、删除、修改操作的mysql数据库。

前端又由两个部分组成，一个是管理员账户：管理员通过点击头像进入admin页面，然后输入密码和账号进行登录。如果账号和密码正确则进入user页面，可以行驶管理员权限，执行删除文章或者添加文章，添加或者删除图片，增加或者减少文章所属类别的操作。

另一个是访客身份游览博客：可以通过首页的各种链接进入不同的博主公开的各种主题页面，有博主公开的相册，对博主进行留言的页面，阅读博主发布的文章。

### 主要功能的实现

首页实现

<div class = "home\_box">

<h2>杨国栋的个人博客&nbsp

<a href="/admin">

<img src = "/img/11.jpg" width = 50px height = 50px;>

</a>

</h2>

<h3>millionbug's Personal Website</h3>

<p class = "move1">»

<a href = "/blog">博客</a>/

<a href = "/message">留言</a>/

<a href = "/aboutMe">个人随笔</a>/

<a href = "/about">特别鸣谢</a>

</p>

<p class = "move2">»

<a href = "https://github.com/millionbug">我的Github</a>/

<a href = "/img">个人相册</a>

</p>

</div>

上面的每一个带有href属性的a标签代表一个链接，用鼠标点击都会向后端发起一个get请求，当后端的接收到相应的后端请求的时候便会根据路由中定义好的各种路由规则返回相应的页面，然后前端渲染收到的页面。但是其中一个<a href = "https://github.com/millionbug">我的Github</a> 的href属性为https://github.com/millionbug属于超链接，浏览器会直接跳转不会向后端发起get请求。

这里的首页用到了对于整个页面的垂直水平居中定位。对于整个屏幕的水平垂直居中定位有很多方法，这里采用的是将需要居中的元素用一个div给包裹起来。然后因为最顶级的div是相对于屏幕的宽高都是百分之一百的。

因此只需要该div给一个宽高的值，

然后将其定为设置位position: absolute;这样就是绝对定位，相对于父级（中最近的位relative或者absolute的元素，如果没有则是相对于body元素）的绝对定位。

再将left和top都为百分之五十。这样元素的左上角丁点便在整个页面居中。

margin设置为左边距和上边距都为自身宽高的一半，这样将自身偏移量给拉回来。

博客页面的实现

首先会进入一个有所有博客链接的页面，类似百度搜索得到的搜索结果总览

<div class = "content">

<h2>

<a href = "/" class="index">首页</a>»

<a href = "/aboutMe" class="index">杨国栋同学</a>»

<a href = "/message" class = "index">给他留言</a>»

<a>他的博客</a>

</h2>

<div class ="blog\_category">

<p class = "cate">博客类别</p>

</div>

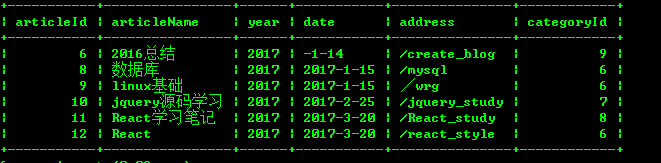
<div class = "main">

<!-- 此处通过js进行请求文章的名字，时间，类别等数据然后填充 -->

</div>

</div>

其中的<div class=”main”></div>，的内容则是通过javascript动态请求数据库存储的数据，然后渲染的博主的所有文章的总览。例如从数据库中读取到当前有这些文章及相关数据



则通过以下代码进行渲染出相应的dom结构

for(var i = 0,len1 = rows.length;i<len1;i++){

for(var j= 0,len2 = yearTotal.length;j<len2;j++){

if(rows[i].year == yearTotal[j]){

//此处根据从数据库读取到的所有的文章相关信息和所记录的每一篇文章所在的地址，来渲染出能够链接到相应的文章的a标签集合

//其中需要渲染的有文章名称，发布文章的时间，还有根据类别的id来查询得到所属类别的名称。

$("#"+yearTotal[j]).append(html)

}

}

}

根据数据最终得到的一个有关所有文章数据与文章地址的列表的dom结构大概是这样的



然后则可以点击进入文章详情页面



留言页面的实现

<div class = "message">

<img src="/img/11.jpg" alt="头像">

<input type="text" placeholder="客官名" class = "form-control" name = "userName" id = "userName">

<textarea type="text" placeholder="和杨国栋同学聊聊天～" class = "form-control" name="messageContent" id="messageContent"></textarea>

<button id = "btn"><a>提交留言</a></button>

<span>不要跑堂叻，给博客的主人留句话吧((^\_^))</span>

<hr>

</div>

这里实现了留言编辑框的dom结构，加上合适的css代码就是合格的留言的UI。通过下列javascript代码进行控制，访客，填写自己的名称，还有填写留言的内容。同时，对博客主人的留言不得为空，而且访客昵称也不得为空采用了校验，当检测到没有填写内容即取到的element的value为空的时候禁止提交，并且提示“昵称与留言内容不得为空”。然后点击提交。将访客名称，与留言内容通过json数据格式，使用post请求传输给后端。我们都知道javascript是事件驱动的，ajax请求是异步的，所以下面在完成了向后端传输数据，同时接收到后端完成插入数据的操作之后咋会执行所定义的回调函数。这里的回掉函数只用执行了

Location.reload()，这里让页面重新加载了，这样便会将新添加成功的评论给显示到页面上。

$.postJSON(

"/SubmitMessage", //请求的url链接

{

"userName": name, //前端传来访客昵称

"messageContent": content //真正的留言内容

},

Function(data){

If (data != "1") return

location.reload(); //将页面刷新，这样重新加载留言数据，将新添加的留言更新到页面

}

);

已有留言的展示是通过发起的ajax请求拉取到所有的已有留言的数据，然后使用下列代码进行遍历然后插入到页面的

var rows = data.result;

var total = data.total;

var html = "";

for(var i = 0;i<total;i++){

//这里通过拼接html结构的字符串

}

$(".message").append(html);

Node.js后端的server开发

在项目的main.js文件中，我们先导入express，和在另一个文件配置的mysql规则。最后

用app.listen(8080);来启动服务。这样就可以在浏览器端即客户端通过所给出的主机地址和端口号访问到我们的博客系统。

var express = require('express');

var Mysql = require("./Mysql/Mysql");

var Router = require("./Router/Router");

var formidable = require("formidable");

var fs = require("fs");

var app = express();

var server = app.listen(8080, function () {

var host = server.address().address;

var port = server.address().port;

console.log(`应用实例，访问地址为 http://${host}:${port}`);

});

不同的链接进入不同的页面的实现

app.get('/admin',function(request,response){

response.sendFile(\_\_dirname+"/html/admin.html");

});

app.get('/img',function(request,response){

response.sendFile(\_\_dirname+"/html/image.html");

});

当后端接受到前端传来的get请求时，将url与路由中定义的规则进行匹配，然后发现比如是/admin，就执行/admin后面的回调函数，response.sendFile(\_dirname+”/html/admin.html”)将admin.html文件传输给前端，这样前端就进入了管理员的登录界面了。

给主人留言的后端实现

当前段通过点击留言按钮发送了留言者的昵称和留言内容时，后端接受请求，并且根据 "/SubmitMessage"，匹配到相应的处理controller函数Router.SubmitMessage

app.get("/SubmitMessage",Router.SubmitMessage);

Router.SubmitMessage函数的内部实现

function(request,response){

var date = new Date(), param = request.query, time = date.getFullYear() + "-" + (date.getMonth() + 1); //生成所需要有关评论的时间的相关数据

var query = "insert into Message(userName,image,content,date) values(? ,? ,? ,?)";

var mysql = new Mysql.createMysql({

query:query,

response:response,

param:[param.userName,"/img/12.jpg",param.messageContent,time],

});

mysql.Add();

};

管理员登录的实现

在首页的头像，就是那个可爱的河马的地方，点击可以跳转到admin页面，就是管理员的登录的页面。页面中的输入框输入账号和密码，然后通过点击按钮触发javascript代码中的click事件最后如果如果成功登录则跳转到管理员的/user页面。

Dom结构

<label for="name">用户名</label>

<input type="text" class = "form-control" id = "name" name = "name">

<label for="pws">密码</label>

<input type="password" class = "form-control" id = "pws" name = "pws">

<input type="button" id = "btn" value = "登录" class = "btn btn-success">

根据文章的类别id查询文章

首先时在所有的博客文章总览的页面有一个导航头，有着各个文章类别，然后点击之后就会跳转到一个包含此类别的所有文章链接的页面下。导航头的dom结构则因为文章的类别时可以添加的，所以必须也是根据数据库中的所有包含的类别进行读取大所有的类别然后进行渲染的。

导航头的dom结构由下列javascript代码渲染：

$.getJSON(

"/Category",

function(data){

If(!data.result)return;

var rows = data.result, html;

for(var i = 0,len = rows.length;i<len;i++){

//此处通过拼接html字符串然后将其添加到目标element中进行渲染到页面中

}

Html += "</ul>";

$(".blog\_category").append(html);

}

);

最终得到的dom结构



当点击代表某一个类别的标签时向后端发起查询请求，然后后端执行查询请求的代码

function(request,response){

var param = request.query;

var query = "select \* from Article,Category where Article.categoryId=Category.categoryId and Category.categoryId=? order by articleId DESC;";

var mysql = new Mysql.createMysql({

query:query,

response:response,

config:["articleId","articleName","year","date","address","category","categoryName"],

param:[param.categoryId]

});

mysql.Query();

};

管理员权限的行使

当管理员成功登录之后就自动跳转到管理员页面，管理员页面可以执行添加或者删除文章，添加或者删除对文章进行分类的类别选项，增加或者删除相册中的照片。因为三个功能的实现方式都很类似，所以下面以添加或者删除的代码实现为例：

点击添加文章之后，页面会出现表单，然后进行填入需要添加的文章的信息。例如：文章名称，文章的地址，文章所属分类。

填写完成之后可以选择提交，则回将信息传输给服务器。服务器接受到信息，确认传输的数据项正确然后正确的插入到数据库中之后则回返回给浏览器true或者false为结果，然后页面执行刷新，我们就可以看到列表中出现了新添加的文章以及许多信息。

1. 或者可以选择取消添加，则不会向服务器发起请求与传送数据。

添加文章时填写表单的dom结构：

<div class = "add" style = "display: none">

<label for="name">文章名：</label>

<input type="text" class = "form-control" id = "name" style = "width:300px;height:22px;">

<label for="name">地址：</label>

<input type="text" class = "form-control" id = "address" style = "width:300px;height:22px;">

<label for="name">分类：</label>

<select name="category" id="category" style = "width:150px;height:22px;">

</select>

<input type="button" id = "btn" value = "提交">

<input type="button" class = "clo" value = "关闭">

</div>

点击提交按钮时执行的主要javascript代码：

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

$.getJSON(

"/appendArticle",

{

"date":time, //需要向后端传输的有关时间，文章名称，文章所属类别，文章地址

"year":year,

"articleName":name,

"address":address,

"categoryId":categroy

},

function(data){

if(data == "1"){ //当请求的到响应之后判断返回结果是否正确，如果返回结果正确

alert("添加成功！"); //对用户提示“添加成功”

location.reload(); //页面重新加载

}

}

)

});

后端接收到此添加文章的请求时执行的代码：

exports.addCategory = function(request,response){

var param = request.query;

var query = "insert into Category(categoryName) values(?)";

var mysql = new Mysql.createMysql({

query:query,

response:response,

param:[param.name],

});

mysql.Add();

};

删除文章的操作

点击显示所有的文章的list的每一项后面的删除按钮，触发javascript中定义给删除按钮的点击事件：

function del(obj){

var id = obj.parentNode.getAttribute("class")|0;

ajax("GET","/DelArticle?id="+id,function(data){

alert("删除成功！");

location.reload();

})

}

同时后端根据”/DelArticle”匹配到相应的controller函数，函数内部再读取query的参数id，通过id得知，具体是要删除那一篇文章。然后执行删除数据库中相应的那一条文章的数据。最后将执行的结果返回给前端，前端刷新页面得到删除后的数据展示。

exports.DelArticle = function(request,response){

var param = request.query;

var query = "delete from Article where articleId=?";

var mysql = new Mysql.createMysql({

query:query,

response:response,

param:[param.id],

});

mysql.Delete();

};

删除文章的类别的功能的实现

点击添加按钮之后弹出表单来，然后填写类别名称，点击提交按钮。触发javascript中定义的点击事件，然后发送ajax请求，当node.js后端接受到请求之后调用定义好的向数据库插入类别的数据。最后后端返回执行结果，前端收到执行结果之后自动刷新页面，这样新的类别的数据也就更新到页面了。

据库对数据库的增删改查操作的实现

页面中的很多操作，涉及到数据的都是需要读写数据库的。当前端发起请求后，后端接收到请求然后判断是否需要进行对数据库的读写，这时就有连接mysql数据库，根据参数删除数中的某一条数据，根据传来的数据向数据库写如一条数据，修改数据库中某一条已经存在的数据等操作。所有的操作都被简单封装在一个Mysql对象中。

连接数据库，先有Mysql对象中的clientConnect方法进行登录mysql，因为clientConnect方法中包含了登录mysql所需要的用户名称与密码，然后再是调用this.connectMysql方法进入blog的database。并且每一步操作都尽可能的避免了因为其他原因导致出错而添加了catch错误的方法：

clientConnect:function(){

this.client = this.mysql.createConnection({

user:"root",

password:"double"

});

this.client.connect();

}

connectMysql:function(target){

var def = this.Q.defer();

this.client.query("use blog");

return def.promise;

}

更新数据库中的某一条数据操作，先调用this.clientConnect()方法登录mysql然后调用this.connectMysql方法进入blog的database。同时this.connectMysql方法会返回一个Q库的defer方法生成的promise对象因此可以进行链式的方法调用：

Update:function(){

this.clientConnect();

var me = this;

this.connectMysql(this.target).

then(function(target){

return me.queryData(target);

}).done(function(results,field){

me.client.end();

me.target.response.send("1");

},console.error)

}

删除数据库中的某一条数据的操作，和update更新方法一样需要先连接mysql并且进入blog数据库，同样调用前面路由系统中定义sql语句，只是不同的操作返回给前端的成功执行的数据不一样。例如：update方法是返回1，delete方法是返回2，

Delete:function(){

this.connectMysql(this.target).

then((target)=> {

return me.queryData(target);

}).done(function (results) {

me.target.response.send("2");

},console.error);

}

文章下的进入下一篇文章的链接

为了增加用户体验，每一篇文章下面都有一个链接到下一篇文章的链接。

可以在文章的底部增加以下的html代码：

<a href="/stydy\_bind" class = "next">

<b>下一篇:</b>

<b>深入学习对bind的实现</b>

</a>

Dom中a标签包含链接到下一篇文章的href属性，点击之后自动跳转页面到下一篇文章。

### 系统的部署与测试

系统的测试是软件系统的开发过程中必不可少的一步，是防止在交付产品时出现错误或者不利的缺陷和避免。软件测试是为利益相关者提供有关被测产品或服务质量的信息进行的调查。[1]软件测试还可以提供客观，独立的软件视图，使企业能够理解和了解软件实施的风险。测试技术包括执行程序或应用程序的过程，目的是查找软件错误（错误或其他缺陷），以及验证软件产品是否适合使用。 软件测试涉及执行软件组件或系统组件以评估感兴趣的一个或多个属性。通常，这些属性表示被测组件或系统的程度：

符合指导设计开发的要求，

正确响应各种输入，

在可接受的时间内执行其功能，

足够可用，

可以在其预期的环境中安装和运行，

实现其利益相关者的一般希望。

本系统采用白盒测试与黑盒测试：白盒测试是在已知道程序的功能与内部逻辑，然后按照说明书上的说明来验证是否如说明书的每个动作正常工作。但是因为软件开发者自己总会有盲点。所以也采用了黑盒测试，具体方法就是邀请同学进行在不清楚软件的内部逻辑的情况下进行穷举测试的过程。

测试用例的摘要

首页功能的测试

|  |  |  |  |
| --- | --- | --- | --- |
| 测试编号 | 输入 | 预期结果 | 实际结果 |
| Index1 | 当访问者进入博客首页的时候，页面UI是否正确。 | 在浏览器输入网址，进入首页。页面UI是否正确。 | 首页UI正确，与预期一致 |
| Index2 | 点击头像，跳转到管理员登录界面，然后输入错误的账号密码，和再输入正确的账号和密码。 | 当点击头像之后，页面正确跳转到管理员登录页面，同时是否能够正确登录。输入错误的账号和密码的时候，页面提示“账号密码错误”，输入正确的账号密码的时候，页面提示“正确登录”然后跳转到管理员页面。 | 符合期望输出 |
| Index3 | 用鼠标点击“博客” | 当用户点击“博客”时，页面正确跳转到所有博客文章分类的页面。 | 符合期望输出 |
| Index4 | 用鼠标点击“留言” | 当用户点击“留言”时，页面是否正确跳转到给博客主人进行留言的页面 | 符合期望输出 |

留言页面的留言功能的测试

|  |  |  |  |
| --- | --- | --- | --- |
| 测试编号 | 输入 | 预期结果 | 实际结果 |
| Message1 | 不输入访客昵称，也不输入留言内容，点击提交留言按钮，查看是否可以给主人添加留言。 | 点击提交留言按钮页面提示“请输入访客昵称与留言内容” | 符合预期结果 |
| Message2 | 正确输入访客昵称，正确输入留言内容，点击提交留言按钮 | 点击提交留言之后，提示留言成功，页面自动刷新。留言出现在页面中 | 提示留言成功,页面并没有自动刷新。手动刷新之后，新的留言也没有出现在页面中  解决方案：（1）通过debug打上断点对前端的ajax请求进行调试发现前端发送的请求没有问题，传送的数据格式正确，请求路径正确，请求方法正确。排除是前端的错误。（2）在后端找到路由系统，定位到路由系统定义的controller方法，在代码中进行console方法打印出传送的参数。（3）检查向数据库插入留言的sql语句。检查到错误是sql语句拼写错误。 |
| Message3 | 用鼠标点击“我的github” | 点击“我的github”是否正确新开页面打开我的github主页 | 页面发生跳转到我的github主页，但是没有新开页面。  解决方案：给a标签增加target=”\_blank”属性。然后在此点击“我的github”成功打开新的页面并且跳转到我的github主页。 |

博客总览文章的页面测试

|  |  |  |  |
| --- | --- | --- | --- |
| 测试编号 | 输入 | 预期结果 | 实际结果 |
| BlogAll1 | 点击博客总览页面懂得顶部的，博客的分类标签 | 跳转到具有该分类的所有博客总览的页面。类似百度搜索的结果展示 | 符合预期结果 |
| BlogAll2 | 点击每一篇文章的标题链接 | 跳转到该文章页面 | 符合预期结果 |
| BlogAll3 | 点击文章详情下面的向前或者向后的链接 | 点击向前时，跳转到前一篇文章页面。点击向后时，跳转到后一篇文章页面 | 符合预期结果 |

相册页面的测试

|  |  |  |  |
| --- | --- | --- | --- |
| 测试编号 | 输入 | 预期结果 | 实际结果 |
| photo1 | 点击进入相册页面 | 进入相册页面显示相册第一张图片，并且图片两边出现想左或者向右的按钮 | 符合预期结果 |
| Photo2 | 点击图片的左边的按钮 | 点击之后，页面的图片切换到前一张图片。并且出现过渡的动画效果 | 符合预期结果 |
| Photo3 | 点击图片右边的按钮 | 点击之后，页面的图片切换到后一张图片。并且出现过渡的动画效果 | 符合预期结果 |