绪论 2

网站选题背景 2

网站实现目标 2

网站可行性分析 2

需求分析 3

业务分析 3

功能需求分析（数据流图、数据字典或者是用例图、交互图等） 3

系统性能需求 5

系统设计 5

网站功能模块设计 5

数据库设计 6

网站软件和硬件选择 8

其它设计 9

系统实现 12

设置模块 12

用户登录模块 13

添加用户图书模块 16

数据库的访问 17

前端模块 20

网站的部署与发布 23

网站功能测试 24

测试的方法 24

测试中的发现问题及解决方案 25

测试的感想 26

总结 26

附录：源程序清单 27

参考文献 29

# 绪论

## 网站选题背景

计算机和网络技术的应用推进了信息社会的进程，而随着人们环保意识的增强，以及各行业对办公模式需求的不断升级，现代化、信息化建设步伐的加快，无纸化办公已经由概念逐渐应用到多个行业领域中。无纸化办公是指在不用纸张办公，在无纸化办公环境中进行的一种工作方式，无纸化办公需要硬件、软件与通信网络协力才能达到的办公体验。

很多公司为加强企业文化建设，进一步丰富员工业余文化生活，提升员工知识储备，和文化素养，好多企业都建立了图书馆，或场地或电子阅览室。可满足员工的多样化的需求，规范图书馆的管理是重要的一个环节，因为只有良好的后续管理，图书馆才能真正发挥作用。我找了一下市面上的图书管理软件发现，绝大多数都是公共图书馆的管理系统，并且费用不菲。而且他们更关心图书的管理、索引和统计，却并不太关心对借阅人而言，能不能简单的操作，系统是否好用。我们公司图书馆在借书方面用的日事请，日事清的计划功能则可以比较好的胜任。让员工操作起来简单好用。

本毕业设计论文就从以上所诉需求角度，结合本人的工作内容，设计并实现一个基于数据库的公司图书馆管理网站系统。

## 网站实现目标

公司图书管理网站系统主要实现用户登入，登出，用户信息管理和图书录入，出借等信息维护功能。该系统要做到，成本投入低，操作简单方便，易学好用。

## 网站可行性分析

该网站主要服务于公司内部图书管理业务，访问量和数据量不会太大，这就使得公司投入的研发，硬件，系统日常维护等成本不会太高，但是给员工带来的巨大方便是不言而喻的，具有很高的性价比。

软件方面，采用开源的Flask和AdminLTE框架，几乎零成本。Flask是一个基于Python语言开发的轻量级后段框架，开源，免费，简单易用。AdminLTE是一套开源的前端框架，提供了清晰，简介的UI，方便灵活的控件。基于这两者框架，可以快速的开发功能强大的网站系统。

硬件方面，市面上有很多云服务提供商都提供虚拟主机（VPS）业务，例如，阿里云，腾讯云等。虚拟主机一般基于Linux系统构建，系统稳定，硬件可拓展性高，整体更稳定，性价比高。以阿里云为例，一台1GB内存，双核2.0G，100GB存储的虚拟主机售价为550RMB/年。

基于以上分析，设计和实现一个基于数据库的公司图书馆管理网站系统是可行的。

# 需求分析

## 业务分析

从业务角度来看，该图书管理系统主要需要实现用户管理和图书管理这两大块业务。

用户管理：

用户管理模块主要提供用户登录验证的功能。该系统用户管理设计成主要有两大角色，管理员和普通用户。用户打开浏览器，输入网站网址，系统会进入用户登录界面，用户用管理员账号(默认为admin)登入，完成添加普通账户的操作，将生成的用户名和密码以邮件的形式发送给员工。员工使用该用户名和密码登入系统完成图书的借阅操作。

图书管理

图书管理模块主要提供图书的录入，搜索，出借，归还等功能。由管理员登录系统，完成对图书的录入操作。普通用户登录后，可以搜索自己需要的图书，以及查看图书的库存，出借情况。

## 功能需求分析

从功能角度来看，该图书管理系统主要需要实现用户权限管理和图书管理这两大功能。该功能主要逻辑如下：

用户管理：

管理员用户名和密码在系统上线后设置，原则上有原图书管理员持有，使用管理员账户登录后，管理员根据公司员工的入职信息，帮其建立普通账户，通过邮件的形式发送给各员工。

如图1所示，用户登陆管理模块要做到对不合法或者错误的输入给出友好提示功能，前端页面中JavaScript代码主要处理明显的输入问题，比如输入框为空，用户名，密码过长或过短。经过初步过滤后发送到后端进行用户名密码有效性检查，根据返回的结果执行下一步操作。

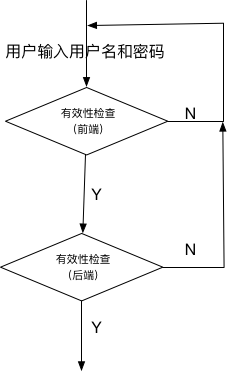


图1 登陆有效性验证流程

图书管理

管理员账户具有图书管理功能，主要包括，录入图书信息，例如，书名，作者，版次，ISBN等信息，由系统维护借阅信息。普通账户有记录自己借阅书籍情况和借阅历史记录。

系统用例图

如图2，图3所示。

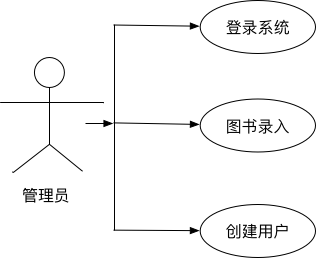


图2 管理员用例图

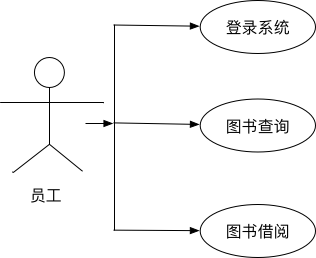


图2 员工用例图

## 系统性能需求

该系统面向公司全体员工，访问量和数据量均不会太大，按照公司员工数为200人的一个中型公司，日活跃用户数为20%为例，其日活跃人数为40人左右。这种访问量市面上的提供的普通虚拟主机均可满足。普遍情况下，网站时间特性可以根据国际标准3-5-8原则来确定：

登录时间最长不超过5秒。

检索时间最长不超过5秒。

页面之间跳转不超过3秒。

操作平均时间在3～5秒以为。

# 系统设计

## 网站功能模块设计

用户登录模块

如图所示，管理员和普通用户由统一界面登录。前端JavaScript代码负责初步检查用户输入的账号和密码，比如，用户名和密码不能为空。然后将表单通过HTTP POST请求发送给后端服务，来验证用户名和密码的有效性，根据后端的返回结果，来显示出错信息或者允许登录。

管理员账号创建用户模块

如图所示，管理员账号登录后，选择左侧的用户管理导航栏，根据员工的入职情况，填入用户的基本信息，完成用户的创建，主要关键的字段：员工用户名和默认密码。

管理员账号图书录入模块

如图所示，管理员账号登录后，选择左侧的图书管理导航栏，根据公司的藏书情况，录入图书的基本信息，包括书名，作者，ISBN等信息。

普通账号图书检索模块

如图所示，员工登录后，选择左侧的图书管理导航栏，可以查看图书列表，当图书数量过多，可以使用顶部的检索功能，主要可以根据书名，作者进行模糊检索，根据ISBN进行精确检索。

普通账号图书借阅模块

普通账号登录后，选择左侧的图书管理导航栏，检索到心仪的图书后，可以查看图书的借阅状态，如果该图书状态为“未借阅”则可以单击“借阅”后取走该图书。此后，该图书状态变为“已借阅”。用户可以在自己“已借阅”列表里面看到该图书的借阅记录。

## 数据库设计

* + 1. ORM

对象-关系映射（OBJECT/RELATIONALMAPPING，简称ORM），是随着面向对象的软件开发方法发展而产生的。用来把对象模型表示的对象映射到基于SQL 的关系模型数据库结构中去。这样，我们在具体的操作实体对象的时候，就不需要再去和复杂的 SQ L 语句打交道，只需简单的操作实体对象的属性和方法ORM 技术是在对象和关系之间提供了一条桥梁，前台的对象型数据和数据库中的关系型的数据通过这个桥梁来相互转化。本网站系统中，主要采用了SQLAlchemy这一ORM框架。

* + 1. 数据库设计

根据前述业务需求，如表所示，主要包含两个数据库表：user表主要负责记录用户信息，admin\_user表用了记录管理员信息，book表主要用来维护图书信息。

|  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- |
| user表 | | | | | | | | |
| 是否主键 | 字段名 | 字段描述 | 数据类型 | 长度 | 可空 | 约束 | 缺省值 | 备注 |
| 是 | id | 用户编号 | Integer |  | 否 |  |  | 自动增长 |
|  | name | 用户名 | VARCHAR | 32 | 否 |  |  |  |
|  | password | 密码 | VARCHAR | 32 | 否 |  | 111111 |  |
|  | phone | 手机号码 | VARCHAR | 11 | 否 |  |  |  |
|  | number | 工号 | VARCHAR | 16 | 否 |  |  |  |
|  | active | 是否激活 | Boolean |  | 否 |  | 是 |  |
|  | created\_at | 创建时间 | DateTime |  | 否 |  | CURRENTTIME |  |

表 1 User表

|  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- |
| admin\_user表 | | | | | | | | |
| 是否主键 | 字段名 | 字段描述 | 数据类型 | 长度 | 可空 | 约束 | 缺省值 | 备注 |
| 是 | id | 用户编号 | Integer |  | 否 |  |  | 自动增长 |
|  | username | 用户名 | VARCHAR | 32 | 否 |  |  |  |
|  | password | 密码 | VARCHAR | 32 | 否 |  | 111111 |  |
|  | active | 是否激活 | Boolean |  | 否 |  | 是 |  |
|  | created\_at | 创建时间 | DateTime |  | 否 |  | CURRENTTIME |  |

表2 admin\_user表

|  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- |
| book表 | | | | | | | | |
| 是否主键 | 字段名 | 字段描述 | 数据类型 | 长度 | 可空 | 约束 | 缺省值 | 备注 |
| 是 | id | 图书编号 | Integer |  | 否 |  |  | 自动增长 |
|  | name | 书名 | VARCHAR | 64 | 否 |  |  |  |
|  | author | 作者 | VARCHAR | 32 | 否 |  |  |  |
|  | price | 价格 | VARCHAR | 11 | 否 |  | 0.0 |  |
|  | isbn | ISBN | VARCHAR | 32 | 否 |  |  |  |
|  | borrowed\_by | 借阅人 | VARCHAR | 32 | 是 |  |  |  |
|  | borrowed\_phone | 借阅人电话 | VARCHAR | 16 | 是 |  |  |  |
|  | active | 是否激活 | Boolean | 1 | 否 |  | 是 |  |
|  | created\_at | 创建时间 | DateTime |  | 否 |  | CURRENTTIME |  |

表3 book表

## 网站软件和硬件选择

* + 1. 网站软件

本网站软件环境主要采用Linux+Nginx+uwsgi套件。总所周知，Linux是一套免费使用和自由传播的类Unix操作系统，是一个基于POSIX和UNIX的多用户、多任务、支持多线程和多CPU的操作系统。它能运行主要的UNIX工具软件、应用程序和网络协议。它支持32位和64位硬件。Linux继承了Unix以网络为核心的设计思想，是一个性能稳定的多用户网络操作系统。其中Ubuntu Server作为一个优秀的Linux发行版，广泛应用于服务器领域。

Nginx是一个高性能的HTTP和反向代理服务器, 它是一款轻量级的Web服务器/反向代理服务器及电子邮件（IMAP/POP3）代理服务器，并在一个BSD-like 协议下发行。其特点是占有内存少，并发能力强，事实上nginx的并发能力确实在同类型的网页服务器中表现较好，其将源代码以类BSD许可证的形式发布，因它的稳定性、丰富的功能集、示例配置文件和低系统资源的消耗而闻名。

uWSGI是一个Web服务器，它实现了WSGI协议、uwsgi、http等协议。Nginx中HttpUwsgiModule的作用是与uWSGI服务器进行交换。WSGI是一种Web服务器网关接口。它是一个Web服务器（如nginx，uWSGI等服务器）与web应用（如用Flask框架写的程序）通信的一种规范。

综合考虑，软件环境采用Ubuntu+Nginx+uwsgi这一流行的网络套件。

* + 1. 硬件选择

过去，企业如果想要搭建自己的网站，需要自主购买机器搭建机房，导致成本高，灵活性差，整个网站运行期间需要专人维护。目前流行的建站方式是不在自己搭建机房，而是采用成熟企业提供的云服务，比如AWS，阿里云等。本网站主要为某公司员工服务，访问量不大，可以选择较廉价的云主机，如图所示：

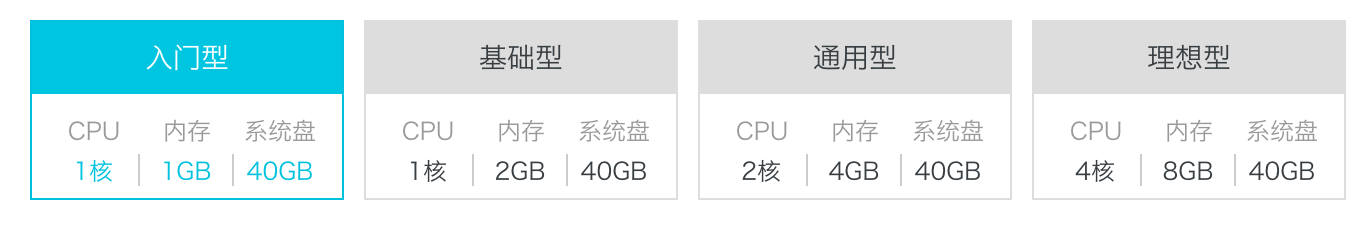


图 3 阿里云提供的云主机



图4 带宽

## 其它设计

* + 1. 登陆界面设计

当用户输入网址：http://domain/[admin|client]/signin，会打开如图5所示的登陆界面。



图 5 登陆界面

其中网址中的admin表示管理员登陆链接，client表示普通用户的登陆链接，两个链接复用一套登陆代码。

* + 1. 整体界面设计

整体界面设计采用简洁流畅的风格，操作简单，易上手，如图6所示：



图6 整体界面设计

* + 1. 增加用户界面设计

当管理员点击“添加用户”后，会打开添加用户界面，如图7所示：



图 7 添加用户界面

* + 1. 添加图书界面设计

当管理员点击“添加图书”后，会打开添加图书界面，如图8所示：



图8 添加图书界面

* + 1. 图书搜索界面设计

当用户点击“全部图书”后，除了会显示图书信息外，还会在顶部有一个“条件筛选”模块，主要用了根据书名，作者等信息搜索图书，如图9所示：

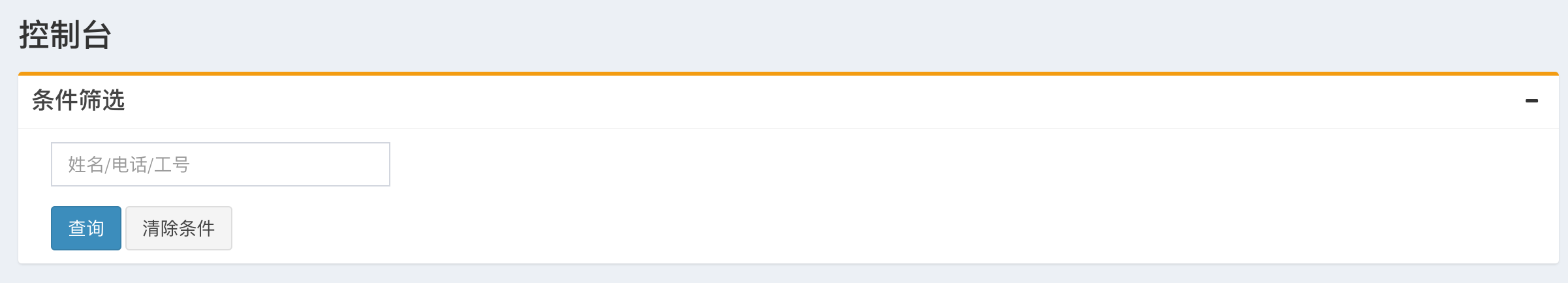


图9 图书搜索界面

* + 1. 用户搜索界面设计

当管理员点击“全部用户”后，除了会显示用户信息外，还会在顶部有一个“条件筛选”模块，主要用了根据用户姓名，手机号等信息搜索用户，如图10所示：

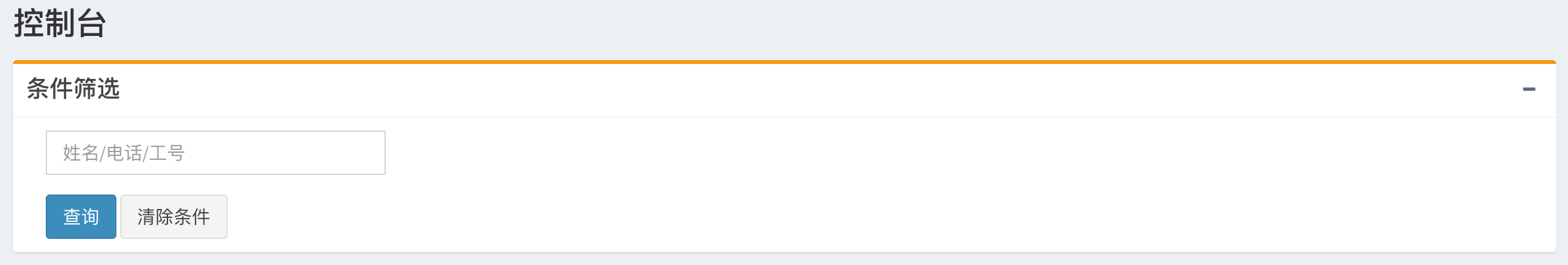


图 10 用户搜索界面

# 系统实现

## 设置模块

设置模块，主要用来整体设置系统运行环境，包括配置开发／线上环境，数据库🔗环境等。从网站类项目开发流程上来分，主要包括两个环境，开发环境（DEV）和正式环境（PROD）。所以在代码设计上，这两个环境从一个公共类（Config）继承，分别配置自己需要的各种环境参数，主要代码如下：

import os

import yaml

class Config(object):

def \_\_init\_\_(self):

yaml\_file = os.path.join(os.path.dirname(\_\_file\_\_), self.\_\_class\_\_.\_\_name\_\_ + '.yaml')

if os.path.exists(yaml\_file):

yaml\_obj = yaml.load(open(yaml\_file, 'r'))

for key in yaml\_obj.keys():

self.\_\_setattr\_\_(key, yaml\_obj[key])

DEFAULT\_USER\_PWD = '111111'

SECRET\_KEY = "xxxxyadfadkfjaldjfalkdjfalkd"

class DevConfig(Config):

DEBUG = True

SQLALCHEMY\_ECHO = True

SQLALCHEMY\_TRACK\_MODIFICATIONS = True

SQLALCHEMY\_DATABASE\_URI = "mysql+mysqlconnector://root:a1b2c3@A@localhost/bms?charset=utf8"

class ProdConfig(Config):

DEBUG = False

def get\_config\_obj():

env = os.environ.get('BMS\_ENV', "dev")

env\_map = {

'dev': DevConfig,

"prod": ProdConfig

}

return env\_map[env]()

系统初始化时，会调用get\_config\_obj来选择运行环境，所以在部署的时候，通过设置BMS\_ENV为dev或者prod来选择让系统运行在什么环境。

## 用户登录模块

用户登录模块使用了Flask-Login插件，它为 Flask 提供了用户会话管理。它处理了日常的登入，登出并且长时间记住用户的会话。

它会:

在会话中存储当前活跃的用户 ID，让你能够自由地登入和登出。

让你限制登入(或者登出)用户可以访问的视图。

处理让人棘手的 “记住我” 功能。

帮助你保护用户会话免遭 cookie 被盗的牵连。

可以与以后可能使用的 Flask-Principal 或其它认证扩展集成。

但是，它不会:

限制你使用特定的数据库或其它存储方法。如何加载用户完全由你决定。

限制你使用用户名和密码，OpenIDs，或者其它的认证方法。

处理超越 “登入或者登出” 之外的权限。

处理用户注册或者账号恢复。

配置你的应用

对一个使用 Flask-Login 的应用最重要的一部分就是 LoginManager 类。你应该在你的代码的某处为应用创建一个，像这样:

login\_manager = LoginManager()

登录管理(login manager)包含了让你的应用和 Flask-Login 协同工作的代码，比如怎样从一个 ID 加载用户，当用户需要登录的时候跳转到哪里等等。

一旦实际的应用对象创建后，你能够这样配置它来实现登录:

login\_manager.init\_app(app)

它是如何工作

你必须提供一个 user\_loader 回调。这个回调用于从会话中存储的用户 ID 重新加载用户对象。它应该接受一个用户的 unicode ID 作为参数，并且返回相应的用户对象。比如:

你的用户类

你用来表示用户的类需要实现这些属性和方法:

is\_authenticated

当用户通过验证时，也即提供有效证明时返回 True 。（只有通过验证的用户会满足 login\_required 的条件。）

is\_active

如果这是一个活动用户且通过验证，账户也已激活，未被停用，也不符合任何你 的应用拒绝一个账号的条件，返回 True 。不活动的账号可能不会登入（当然， 是在没被强制的情况下）。

is\_anonymous

如果是一个匿名用户，返回 True 。（真实用户应返回 False 。）

get\_id()

返回一个能唯一识别用户的，并能用于从 user\_loader 回调中加载用户的 unicode 。注意着 必须 是一个 unicode —— 如果 ID 原本是 一个 int 或其它类型，你需要把它转换为 unicode 。

要简便地实现用户类，你可以从 UserMixin 继承，它提供了对所有这些方法的默认 实现。（虽然这不是必须的。）

主要核心登录代码：

from flask\_login import UserMixin

from flask\_login import login\_user

from flask\_login import logout\_user

from flask\_login import login\_required

from core.extension import login\_manager

from core.views.common import render\_json

from core.models import AdminUser

class LoginUser(UserMixin):

def \_\_init\_\_(self, user):

self.user = user

def get\_id(self):

return unicode(self.user.id)

@login\_manager.user\_loader

def load\_user(userid):

user = AdminUser.get\_by\_id(int(userid))

return LoginUser(user)

@admin.route('/signin', methods=['GET', 'POST'])

def signin():

if request.method == 'POST':

user = AdminUser.query.filter\_by(

active=True,

username=request.form['username'],

password=request.form['pwd']

).first()

if not user:

return render\_json(1, {'err\_no': 'pwd\_error', 'input': 'pwd'})

login\_user(LoginUser(user))

next = request.form.get('next', '')

if next:

next = urllib.unquote(next)

return render\_json(0, {'href': next, 'delaySuccess': True})

return render\_json(0, {'href': '/admin/dashboard', 'delaySuccess': True})

return render\_template('/admin/signin.html')

## 添加用户图书模块

Flask 用 蓝图（blueprints） 的概念来在一个应用中或跨应用制作应用组件和支持通用的模式。蓝图很好地简化了大型应用工作的方式，并提供给 Flask 扩展在应用上注册操作的核心方法。一个 Blueprint 对象与 Flask 应用对象的工作方式很像，但它确实不是一个应用，而是一个描述如何构建或扩展应用的 蓝图 。

为什么使用蓝图？

Flask 中的蓝图为这些情况设计:

把一个应用分解为一个蓝图的集合。这对大型应用是理想的。一个项目可以实例化一个应用对象，初始化几个扩展，并注册一集合的蓝图。

以 URL 前缀和/或子域名，在应用上注册一个蓝图。 URL 前缀/子域名中的参数即成为这个蓝图下的所有视图函数的共同的视图参数（默认情况下）。

在一个应用中用不同的 URL 规则多次注册一个蓝图。

通过蓝图提供模板过滤器、静态文件、模板和其它功能。一个蓝图不一定要实现应用或者视图函数。

初始化一个 Flask 扩展时，在这些情况中注册一个蓝图。

Flask 中的蓝图不是即插应用，因为它实际上并不是一个应用——它是可以注册，甚至可以多次注册到应用上的操作集合。为什么不使用多个应用对象？你可以做到那样 （见 应用调度 ），但是你的应用的配置是分开的，并在 WSGI 层管理。

蓝图作为 Flask 层提供分割的替代，共享应用配置，并且在必要情况下可以更改所注册的应用对象。它的缺点是你不能在应用创建后撤销注册一个蓝图而不销毁整个应用对象。

蓝图的设想

蓝图的基本设想是当它们注册到应用上时，它们记录将会被执行的操作。 当分派请求和生成从一个端点到另一个的 URL 时，Flask 会关联蓝图中的视图函数。

主要核心代码：

from flask import Blueprint

admin = Blueprint('admin', \_\_name\_\_, static\_folder='static', template\_folder='templates', url\_prefix='/admin')

from . import admin

from flask import request

from core.models import User

from flask import url\_for

from flask import redirect

from flask\_login import login\_required

from flask import render\_template

from sqlalchemy import or\_

from core.views.common import render\_json

@admin.route('/users', methods=['GET'])

@login\_required

def company\_users():

keyword = request.args.get('keyword', '')

user\_query = User.query

if keyword:

keyword = "%%%s%%" % keyword

user\_query = user\_query.filter(or\_(

User.name.like(keyword),

User.phone.like(keyword),

User.number.like(keyword)

))

users = user\_query.filter\_by(active=True).all()

return render\_template("admin/user.html", users=users, count=len(users))

## 数据库的访问

本项目中使用Flask-SQLAlchemy这一流行的ORM框架，它 使用起来非常有趣，对于基本应用十分容易使用，并且对于大型项目易于扩展。常见情况下对于只有一个 Flask 应用，所有您需要做的事情就是创建 Flask 应用，选择加载配置接着创建 SQLAlchemy 对象时候把 Flask 应用传递给它作为参数。一旦创建，这个对象就包含 sqlalchemy 和 sqlalchemy.orm 中的所有函数和助手。此外它还提供一个名为 Model 的类，用于作为声明模型时的 delarative 基类:在本项目中采用了它来简化数据库的操作，使得操作数据库不在使用原生的SQL语句。根据之前的数据库表设计，对应的核心代码如下：

user表代码：

# -\*- coding: utf-8 -\*-

import datetime

from core.database import db

from core.database import Model

from core.database import SurrogatePK

class User(Model, SurrogatePK):

\_\_tablename\_\_ = "user"

name = db.Column(db.String(32), nullable=False)

password = db.Column(db.String(32), nullable=False)

gender = db.Column(db.Integer, nullable=False)

number = db.Column(db.String(16), nullable=False)

phone = db.Column(db.String(16), nullable=True)

department = db.Column(db.String(32), nullable=True)

active = db.Column(db.Boolean(), default=True)

created\_at = db.Column(db.DateTime, default=datetime.datetime.now)

admin\_user表代码：

# -\*- coding: utf-8 -\*-

import datetime

from core.database import db

from core.database import Model

from core.database import SurrogatePK

class AdminUser(Model, SurrogatePK):

\_\_tablename\_\_ = 'admin\_user'

username = db.Column(db.String(32), nullable=False)

password = db.Column(db.String(32), nullable=False)

active = db.Column(db.Boolean(), default=True)

create\_at = db.Column(db.DateTime, default=datetime.datetime.now())

book表代码：

import datetime

from core.database import db

from core.database import Model

from core.database import SurrogatePK

class Book(Model, SurrogatePK):

\_\_tablename\_\_ = "book"

name = db.Column(db.String(64), nullable=False)

author = db.Column(db.String(32), nullable=False)

price = db.Column(db.String(8), nullable=False)

publish = db.Column(db.String(32), nullable=False)

isbn = db.Column(db.String(32), nullable=True)

borrowed\_by = db.Column(db.String(32), nullable=True)

borrowed\_phone = db.Column(db.String(32), nullable=True)

active = db.Column(db.Boolean(), default=True)

created\_at = db.Column(db.DateTime, default=datetime.datetime.now)

## 前端模块

本项目采用了AdminLTE作为前端模版，它是一个完全的响应式管理模板，基于Bootstrap3框架，高度可定制，易于使用。适合从小型移动设备到大型台式机的屏幕分辨率。AdminLTE框架在网页架构与设计上，有很大的辅助作用，尤其是前端架构设计师，用好AdminLTE 不但美观，而且可以免去写很大CSS与JS的工作量。主要核心代码：

菜单模块核心代码：

<aside class="main-sidebar">

<section class="sidebar">

<ul class="sidebar-menu">

<li class="header">控制台</li>

<li class="header">图书管理</li>

<li class=""><a href="/client/book/mine"><i class="fa fa-cloud"></i> <span>我的图书</span></a></li>

<li class=""><a href="/client/books"><i class="fa fa-cloud"></i> <span>图书借阅</span></a></li>

<li class="header">我的账户</li>

<li><a href="/client/password/change"><i class="fa fa-circle-o text-aqua"></i> <span>修改密码</span></a></li>

<li><a href="/client/signout"><i class="fa fa-circle-o text-aqua"></i> <span>退出登录</span></a></li>

</ul>

<p>&nbsp;</p>

<p>&nbsp;</p>

<p>&nbsp;</p>

</section>

</aside>

搜索模块核心代码：

<div class="col-md-12">

<div class="box box-warning collapsed-box">

<div class="box-header with-border">

<h3 class="box-title">条件筛选</h3>

<div class="box-tools pull-right">

<button class="btn btn-box-tool" data-widget="collapse"><i class="fa fa-plus"></i></button>

</div>

</div>

<div class="box-body">

<form class="container-fluid" action="?" method="get">

<div class="row">

<div class="col-xs-3">

<div class="form-group">

<input name="keyword" type="text" class="form-control" placeholder="书名/作者/出版社/ISBN">

</div>

</div>

</div>

<div class="row">

<div class="col-xs-3">

<button type="submit" class="btn btn-primary">查询</button>

<a href="?" class="btn btn-default">清除条件</a>

</div>

</div>

</form>

</div>

</div>

</div>

登陆模块核心代码：

<div class="login-box-body">

<p class="login-box-msg">图书借阅管理平台</p>

<form action="/client/signin" method="post">

<div class="form-group has-feedback">

<input name="name" type="text" class="form-control" placeholder="登录名">

<span class="fa fa-mobile form-control-feedback"></span>

<div class="help-block">&nbsp;</div>

</div>

<div class="form-group has-feedback">

<input name="password" type="password" class="form-control" placeholder="密码">

<span class="fa fa-lock form-control-feedback"></span>

<div class="help-block">&nbsp;</div>

</div>

<div class="form-group">

<button type="button" class="btn btn-submit btn-primary btn-block">

登录

<span class="label label-success hide">操作成功</span>

</button>

</div>

</form>

</div>

</div>

编辑模块核心代码：

<form action="/admin/book/edit?id={{book.id}}" method="post">

<div class="box-header with-border">

<h3 class="box-title">图书编辑</h3>

</div>

<div class="box-body box-profile">

<div class="form-group">

<label>书名</label>

<input name="name" type="text" class="form-control" value={{ book.name }}>

</div>

<div class="form-group">

<label>作者</label>

<input name="author" type="text" class="form-control" value={{ book.author }}>

</div>

<div class="form-group">

<label>价格</label>

<input name="price" type="text" class="form-control" value={{ book.price }}>

</div>

<div class="form-group">

<label>出版社</label>

<input name="publish" type="text" class="form-control" value={{ book.publish }}>

</div>

<div class="form-group">

<label>ISBN</label>

<input name="isbn" type="text" class="form-control" value={{ book.isbn }}>

</div>

</div>

<div class="box-footer">

<button type="button" class="btn btn-submit btn-primary btn-block">

更新

<span class="label label-success hide">操作成功</span>

</button>

</div>

</form>

# 网站的部署与发布

本网站的部署使用ansible作为部署工具，ansible是新出现的自动化运维工具，基于Python开发，集合了众多运维工具（puppet、cfengine、chef、func、fabric）的优点，实现了批量系统配置、批量程序部署、批量运行命令等功能。主要核心部署代码如下，

- hosts: 'prod-srv'

gather\_facts: yes

remote\_user: "{{ user if user is defined else 'ubuntu' }}"

vars\_files:

- roles/BMS/files/prod.yml

roles:

- {role: BMS}

- hosts: 'dev-srv'

gather\_facts: yes

remote\_user: "{{ user if user is defined else 'ubuntu' }}"

vars\_files:

- roles/BMS/files/dev.yml

roles:

- {role: BMS}

部署命令：

ansible-playbook –l dev –t update,restart

ansible-playbook –l prod –t update,restart

# 网站功能测试

## 测试的方法

按是否查看程序内部结构分为：

黑盒测试（black-box testing）：只关心输入和输出的结果

白盒测试（white-box testing）：去研究里面的源代码和程序结构

按是否运行程序分为：

静态测试（static testing）：是指不实际运行被测软件，而只是静态地检查程序代码、界面或文档可能存在的错误的过程。

静态测试包括：

对于代码测试，主要是测试代码是否符合相应的标准和规范。

对于界面测试，主要测试软件的实际界面与需求中的说明是否相符。

对于文档测试，主要测试用户手册和需求说明是否真正符合用户的实际需求。

动态测试（dynamic testing），是指实际运行被测程序，输入相应的测试数据，检查输出结果和预期结果是否相符的过程

按阶段划分：

单元测试（unit testing），是指对软件中的最小可测试单元进行检查和验证。

桩模块（stud）是指模拟被测模块所调用的模块，驱动模块（driver）是指模拟被测模块的上级模块，驱动模块用来接收测试数据，启动被测模块并输出结果。

集成测试（integration testing），是单元测试的下一阶段，是指将通过测试的单元模块组装成系统或子系统，再进行测试，重点测试不同模块的接口部门。

集成测试就是用来检查各个单元模块结合到一起能否协同配合，正常运行。

系统测试（system testing），指的是将整个软件系统看做一个整体进行测试，包括对功能、性能，以及软件所运行的软硬件环境进行测试。

## 测试中的发现问题及解决方案

* + 1. 数据库方面

在测试的过程中，发现当输入中文时程序会出错，主要原因是在创建数据库的时候没有指定数据库的字符集。MySQL默认采用的字符集是：latin1，所以在插入中文字符时会报错，解决方法，在创建数据库和表时指定字符集为utf-8：

SQLALCHEMY\_DATABASE\_URI = "mysql+mysqlconnector://root:a1b2c3@A@localhost/bms?charset=utf8"

* + 1. 代码方面

在添加用户模块，填写完对应的字段，点击确定后，网站无响应，经检查页面post数据后，HTTP请求返回的结果是404，原因为flask的request对象在获取form表单值时key写错，解决方法，替换使用key来获取字典value，使用get，并提供默认值。

## 测试的感想

体会一：软件测试在整个软件周期中的重要性。

它存在于整个项目周期，在项目开始之初需求调研的时候就开始了，在形成需求规格说明书的时候就需要针对文档进行测试。这个环节在后续整个项目中占了很大的比重，能主导整个项目的走向，成败与否全在于开始阶段的决策。

体会二：软件测试的真正意义在于发现错误，而不在于验证软件是正确的。

再严密的测试也不能完全发现软件当中所有的错误，但是测试还是能发现大部分的错误，能确保软件基本是可用的，所以在后续使用的过程中还需要加强快速响应的环节。结合软件测试的理论，故障暴露在最终客户端之前及时主动的去发现并解决。

# 总结

在学习过程中，毕业设计是一个重要的环节，是我们步入社会参与实际项目的规划建设的一次极好的演示。毕业设计是学习的总结和提高，和做科研开发工作一样，要有严谨求实的科学态度。毕业设计有一定的学术价值和实用价值，能反映出作者所具有的专业基础知识和分析解决问题的能力。从最初的选题，开题到分析调查、编码直到完成设计。其间，查找资料，老师指导，与同学交流，反复修改代码，每一个过程都是对自己能力的一次检验和充实。在毕业设计期间，尽可能多的阅读文献资料是很重要的，一方面是为毕业设计做技术准备，另一方面是学习做毕业设计的方法。一次优秀的设计对启发我们的思维，掌握设计的规范、流程、具体操作都很有帮助.但是毕业设计也暴露出自己专业基础的很多不足之处。比如缺乏综合应用专业知识的能力，对材料的不了解，对具体设计涉及到的规范要求的不熟悉等等，需要在做的过程中需要去不断的翻阅相关的资料和书籍，这降低了自己的速度和设计的进程，但这个过程对我来说是对自己知识的不足处的一个很好的补充和对已学过知识的一个巩固。这个过程虽然是有一定的难度但还是通过自己的慢慢的摸索和老师的指导下从熟悉到上手，经过这次努力对自己的信心很好的提高。通过这样的一个自己从开始到结束全程自己参与的设计来说对知识的了解和掌握是纯理论的学习远远达不到的效果。这次实践是对自己大学所学的一次大检阅，使我明白自己知识还很浅薄，虽然马上要毕业了，但是自己的求学之路还很长，以后更应该在工作中学习，努力使自己 成为一个可以参与工作能独立完成设计的人。

# 附录：源程序清单

源代码托管地址：

源代码目录结构：

1. ~/bms $ tree -I static
2. .
3. ├── LICENSE
4. ├── README.md
5. ├── core
6. │   ├── \_\_init\_\_.py
7. │   ├── app.py
8. │   ├── database.py
9. │   ├── extension.py
10. │   ├── models
11. │   │   ├── \_\_init\_\_.py
12. │   │   ├── admin\_user.py
13. │   │   ├── book.py
14. │   │   └── user.py
15. │   ├── settings.py
16. │   ├── templates
17. │   │   ├── admin
18. │   │   │   ├── base.html
19. │   │   │   ├── book.html
20. │   │   │   ├── book\_add.html
21. │   │   │   ├── book\_edit.html
22. │   │   │   ├── dashboard.html
23. │   │   │   ├── menus.html
24. │   │   │   ├── operator.html
25. │   │   │   ├── signin.html
26. │   │   │   ├── user.html
27. │   │   │   ├── user\_add.html
28. │   │   │   └── user\_edit.html
29. │   │   └── user
30. │   │   ├── base.html
31. │   │   ├── book.html
32. │   │   ├── book\_order.html
33. │   │   ├── dashboard.html
34. │   │   ├── menus.html
35. │   │   ├── password.html
36. │   │   ├── signin.html
37. │   │   └── student.html
38. │   ├── utils
39. │   │   ├── \_\_init\_\_.py
40. │   │   └── vcode.py
41. │   └── views
42. │   ├── \_\_init\_\_.py
43. │   ├── admin
44. │   │   ├── \_\_init\_\_.py
45. │   │   ├── book.py
46. │   │   ├── login.py
47. │   │   └── user.py
48. │   ├── common
49. │   │   └── \_\_init\_\_.py
50. │   └── user
51. │   ├── \_\_init\_\_.py
52. │   ├── book.py
53. │   ├── signin.py
54. │   └── user.py
55. ├── manage.py
56. └── requirements
57. └── requirements.txt
58. 11 directories, 44 files

# 参考文献

O'Reilly：Python学习手册（第4版）[美] 鲁特兹（Mark Lutz） 著；李军 等 译

MySQL技术内幕（第5版）[美] 保罗·迪布瓦（Paul DuBois） 著；张雪平，何莉莉，陶虹 译

HTML5权威指南 [The Definitive Guide To HTML5] [美] 弗里曼（Adam Freeman） 著；谢廷晟，牛化成，刘美英 译

http://flask.pocoo.org/

http://flask-login.readthedocs.io/en/latest/

http://flask-sqlalchemy.pocoo.org/2.1/

https://docs.python.org/2.7/