南开大学滨海学院

作

业

报

告

系别：计算机科学系

专业：数字媒体技术专业

姓名：庞怡文

学号：18990160

目录

第一章 绪论

第一节 国内外发展现状

HTML5 是构建 Web 内容的一种语言描述方式。HTML5 是互联网的下一代标准，是构建以及呈现互联网内容的一种语言方式．被认为是互联网的核心技术之一。HTML 产生于 1990 年，1997 年 HTML4 成为互联网标准，并广泛应用于互联网应用的开发。

在前五年（1990-1995），HTML 经历了多次修订并经历了许多扩展，主要是在欧洲核子研究中心首先托管，然后是 IETF。

随着 W3C 的创建，HTML 的发展再次改变了场地。 1995 年第一次尝试在 HTML 3.0 中扩展 HTML，然后在 1997 年完成了一种称为 HTML 3.2 的更实用的方法。同一年晚些时候，HTML 4.01 很快就出现了。

第二年，W3C 成员决定停止发展 HTML，而是开始研究基于 XML 的等价物，称为 XHTML。这项工作始于 XML 中的 HTML 4.01 重新编写，称为 XHTML 1.0，除了新的序列化之外没有添加任何新功能，并且在 2000 年完成。在 XHTML 1.0 之后，W3C 的重点转向使其他工作组更容易在 XHTML 模块化的旗帜下扩展 XHTML。与此同时，W3C 还开发了一种与早期 HTML 和 XHTML 语言不兼容的新语言，称之为 XHTML 2.0。

大约在 1998 年停止 HTML 演变的时候，浏览器供应商开发的 HTML 部分 API 被命名并以 DOM Level 1（1998 年）和 DOM Level 2 Core 和 DOM Level 2 HTML（从 2000 年开始）发布。最终于 2003 年）。这些努力随后逐渐消失，2004 年发布了一些 DOM Level 3 规范，但工作组在所有 3 级草案完成之前就已关闭。

2003 年，作为下一代 Web 表单定位的技术 XForms 的出版引发了对 HTML 本身发展的新兴趣，而不是寻找它的替代品。这种兴趣来自于认识到 XML 作为 Web 技术的部署仅限于全新技术（如 RSS 和后来的 Atom），而不是替代现有的已部署技术（如 HTML）。

一个概念证明，可以扩展 HTML 4.01 的表单，提供 XForms 1.0 引入的许多功能，而不需要浏览器实现与现有 HTML 网页不兼容的渲染引擎，这是第一个结果。重新兴趣。在早期阶段，虽然草案已经公开发布，并且已经从所有来源征求意见，但该规范仅受 Opera Software 的版权保护。

在 2004 年的 W3C 研讨会上测试了 HTML 应该重新开放的想法，其中提出了 HTML 工作的一些原则（如下所述），以及上述早期草案提案，仅涉及与表单相关的功能，由 Mozilla 和 Opera 联合推出的 W3C。该提案被驳回，理由是该提案与之前选择的网络发展方向相冲突； W3C 的工作人员和成员投票决定继续开发基于 XML 的替代品。

此后不久，Apple，Mozilla 和 Opera 联合宣布他们打算在一个名为 WHATWG 的新场地的保护下继续努力。创建了一个公共邮件列表，草案已移至 WHATWG 站点。随后将版权修改为由所有三个供应商共同拥有，并允许重复使用该规范。

WHATWG 基于几个核心原则，特别是技术需要向后兼容，规范和实现需要匹配，即使这意味着更改规范而不是实现，并且规范需要足够详细，实现可以实现完整的互操作性，无需相互逆向工程。后一要求特别要求 HTML 规范的范围包括先前在三个单独的文档中指定的内容：HTML 4.01，XHTML 1.1 和 DOM Level 2 HTML。它还意味着包含比以前被认为是标准更多的细节。

2006 年，W3C 表示有兴趣参与 HTML 5.0 的开发，并于 2007 年组建了一个工作组，专门与 WHATWG 合作开发 HTML 规范。 Apple，Mozilla 和 Opera 允许 W3C 在 W3C 版权下发布规范，同时保留 WHATWG 网站上限制较少的许可版本。多年来，两个小组在同一编辑下共同工作：Ian Hickson。在 2011 年，小组得出的结论是，他们有不同的目标：W3C 希望

为 HTML 5.0 推荐的功能划清界限，而 WHATWG 希望继续致力于 HTML 的生活标准，不断维护规范和添加新功能。 2012 年中期，W3C 推出了一个新的编辑团队，负责创建 HTML 5.0 推荐标准，并为下一个 HTML 版本准备工作草案。

第二节 研究方法

以 Windows 10 为操作系统，Pycharm 与 Django 和 Bootstrap 结合，以 Html5 和 Python 为基本语言，搭建出一款具有前后台、数据库存储与读取功能的网站。

第三节 研究内容

Python 编程语言的使用，Html5 网页语言的使用，Pycharm 编译器的使用，Django 环境的配置，Bootstrap 前端的应用。

第四节 文章结构

本文章进行了用户需求的分析，对项目进行了系统的设计和利用相关技术进行了实现，同时穿插了用例图和数据库表进行解释和演示。

第二章 相关技术

第一节 Python

Python 是一种跨平台的计算机程序设计语言。是一个高层次的结合了解释性、编译性、互动性和面向对象的脚本语言。最初被设计用于编写自动化脚本(shell)，随着版本的不断更新和语言新功能的添加，越多被用于独立的、大型项目的开发。

第二节 Django

Django 是一个开放源代码的 Web 应用框架，由 Python 写成。采用了 MTV 的框架模式，即模型 M，视图 V 和模版 T。它最初是被开发来用于管理劳伦斯出版集团旗下的一些以新闻内容为主的网站的，即是 CMS（内容管理系统）软件。并于 2005 年 7 月在 BSD 许可证下发布。这套框架是以比利时的吉普赛爵士吉他手 Django Reinhardt 来命名的。

第三节 Pycharm

PyCharm 是一种 Python IDE，带有一整套可以帮助用户在使用 Python 语言开发时提高其效率的工具，比如调试、语法高亮、Project 管理、代码跳转、智能提示、自动完成、单元测试、版本控制。此外，该 IDE 提供了一些高级功能，以用于支持 Django 框架下的专业 Web 开发。

第四节 Bootstrap

Bootstrap 是美国 Twitter 公司的设计师 Mark Otto 和 Jacob Thornton 合作基于 HTML、CSS、JavaScript 开发的简洁、直观、强悍的前端开发框架，使得 Web 开发更加快捷。Bootstrap 提供了优雅的 HTML 和 CSS 规范，它即是由动态 CSS 语言 Less 写成。Bootstrap 一经推出后颇受欢迎，一直是 GitHub 上的热门开源项目，包括 NASA 的 MSNBC（微软全国广播公司）的 Breaking News 都使用了该项目。国内一些移动开发者较为熟悉的框架，如 WeX5 前端开源框架等，也是基于 Bootstrap 源码进行性能优化而来。

第三章 需求分析

第一节 服务器运行需求

利用 Django 平台，创建一个小型服务器，然后再上面调试所产生的各种路由是否通畅，方法是否得到了实现，以及网页的渲染效果是否符合需求（如图 3-1）

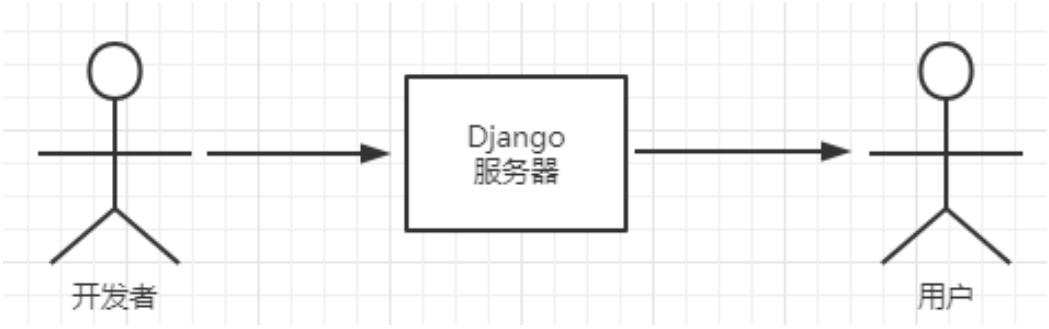

图 3-1 服务器运行

第二节 前端需求

前端页面要显示 LOGO、文章分类链接和图片等看起来花里胡哨的东西，点击文章分类超链接要能显示这个文章分类下的文章标题，不能出现其他的东西。（如图 3-2）

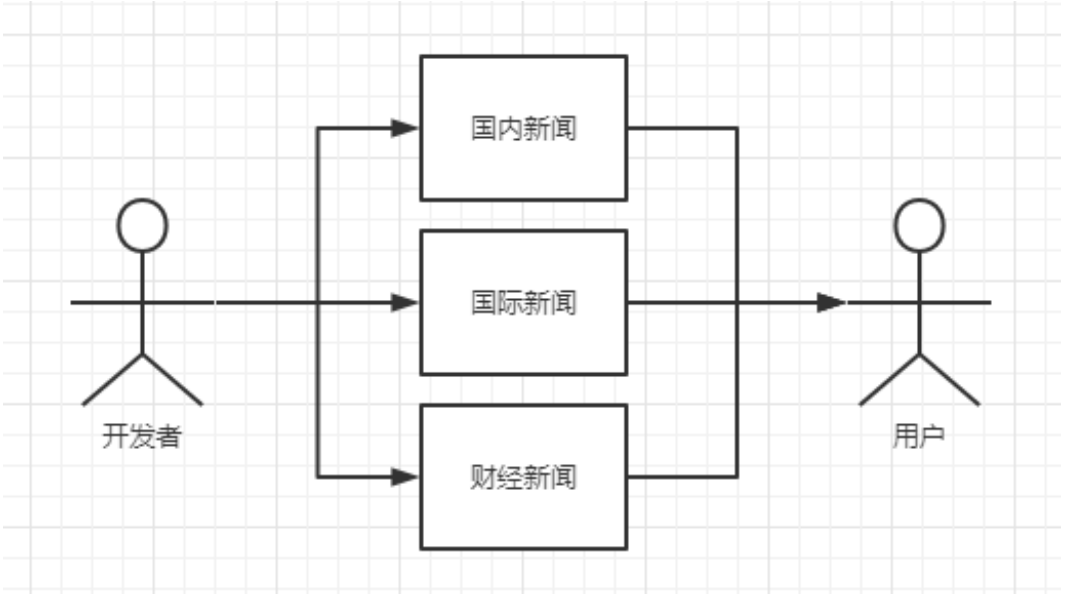

图 3-2 前端

第三节 前端文章显示需求

进入文章分类超链接后，将会看到这个文章分类下的文章标题。点击文章标题可以看到文章内容。（如图 3-3）

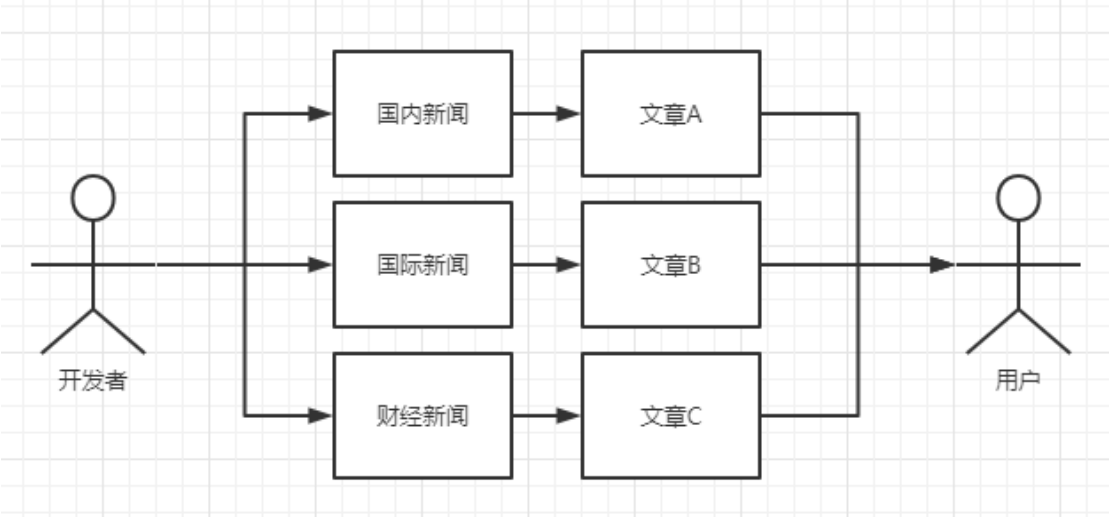

图 3-3 前端文章显示

第四节 后端总体需求

后端管理着文章，内部可以对文章进行增删改查功能，同时不允许用户没登陆就进入这个后台管理系统，所以要有安全机制，并且只需点击就可进行文章的一系列操作，而且界面人性化，没有复杂的操作，需要适合电脑初学者使用。(如图 3-4)

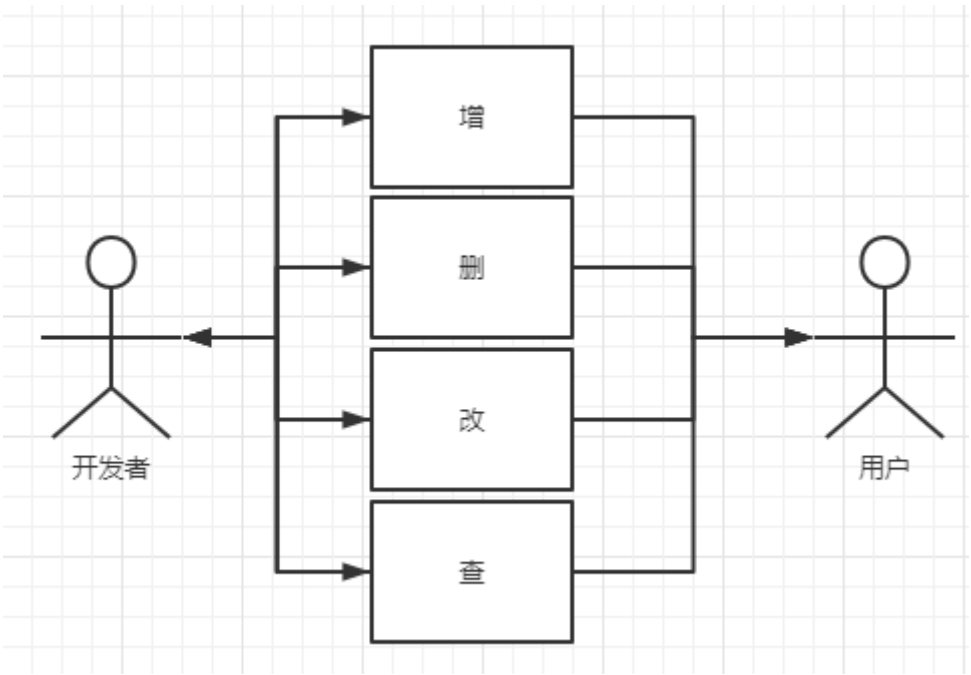

图 3-4 后端

第五节 后台登录需求

要有两种登陆方式，普通的登录方式和 Ajax 登陆方式，普通登录方式在登陆成功后将会进入后台界面，登陆失败就会进入一个错误提示界面，但 Ajax 并不会进入错误界面，而是在当前页面直接弹出错误信息，这样做的好处就是能够保留已经输入过的信息，用户无需

再次输入信息。(如图 3-5)

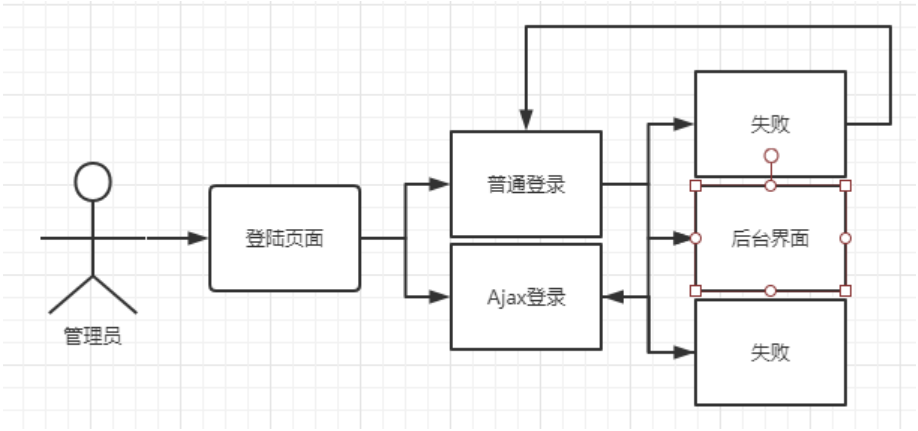

图 3-5 后台登陆

第六节 后台保护需求

这个功能主要是为了防止不登陆就能直接操纵后台的情况，就像防火墙一样，防止非法用户侵入。(如图 3-6)

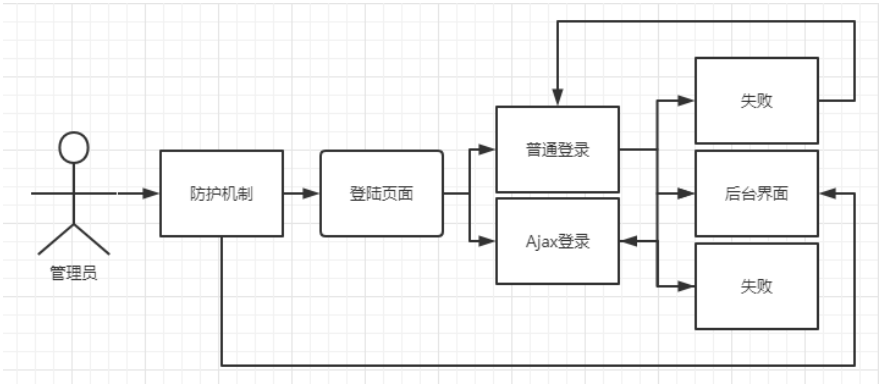

图 3-6 后台保护

第七节 后台资讯增加需求

用于增加文章，应该具有文章标题和文章内容可以填写，文章类型可以选择。(如图 3-7)

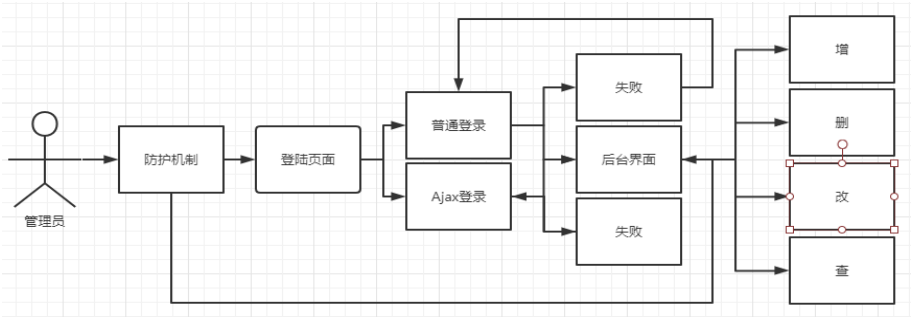

图 3-7 增

第八节 后台资讯删除需求

用于删除文章，但是这个功能仅能用于已经存在的文章，点击了文章控制栏上的垃圾桶图标就会显示是否删除的文本框，再确认后才能删除文章，基于用户一定的反悔权。（如图 3-8）

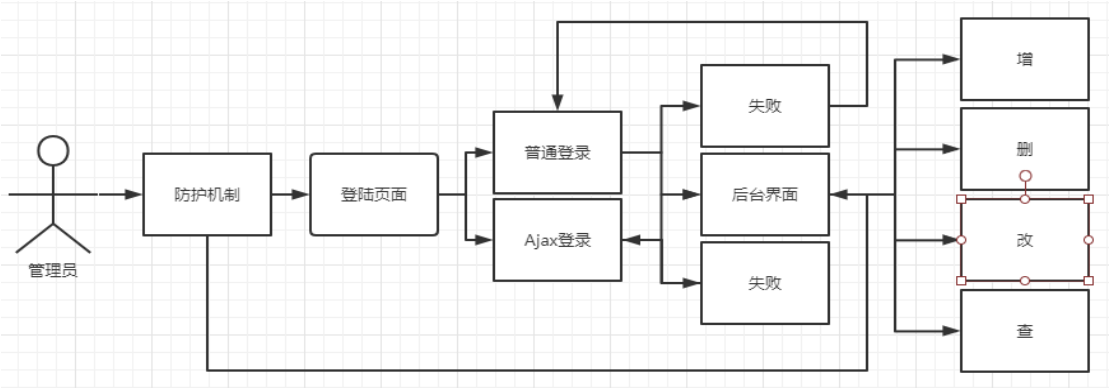

图 3-8 删

第九节 后台资讯修改需求

用于修改文章内容，在打开时应该显示这个文章的标题、类型和文章内容。（如图 3-9）

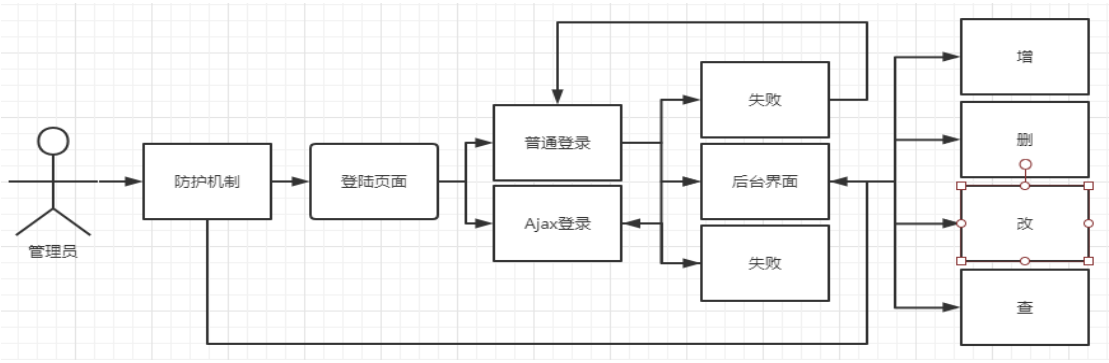

图 3-9 改

第十节 后台资讯查询需求

在后台显示出所有的文章。（如图 3-10）

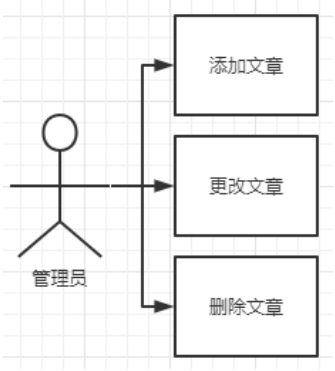

图 3-10 查

第十一节 后台路由需求

用于在输入网址后进行页面的渲染与后台代码的实现。(如图 3-11)

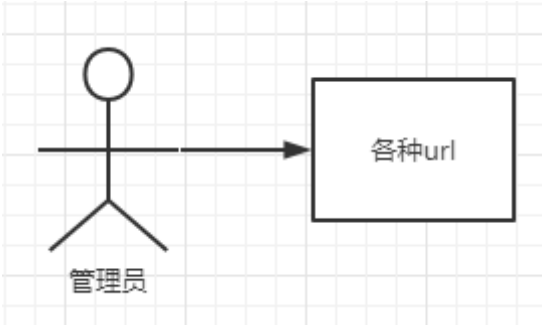

图 3-11 **后台路由**

第四章 系统设计

第一节 服务器运行程序设计

使用 Django 来实现，基于 Python 代码来实现，安装和配置 Django 项目环境后，加入可以拦截非法用户的安全中间件，并指定哪些 URL 可以绕过中间件。（如图 4-1）

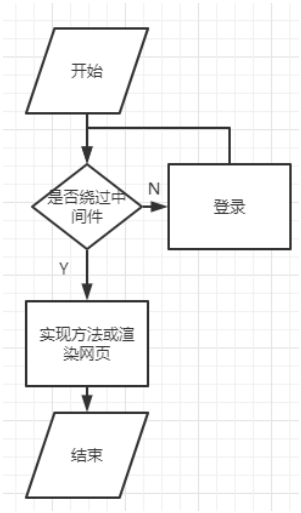

图 4-1 服务器运行

第二节 前端程序设计

前端作为模板放进 Django 环境中，使用 HTML5 语言进行编写，作为展示使用的前端是不需要中间件的保护的，任何用户都能够进行访问。（如图 4-2）

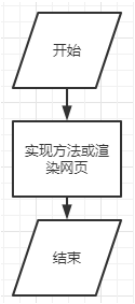

图 4-2 前端

第三节 前端文章显示程序设计

点击主页面上的文章类型后，可以显示出该类型下的文章，需要在 HTML5 代码内写出判断方法和显示方法。（如图 4-3）

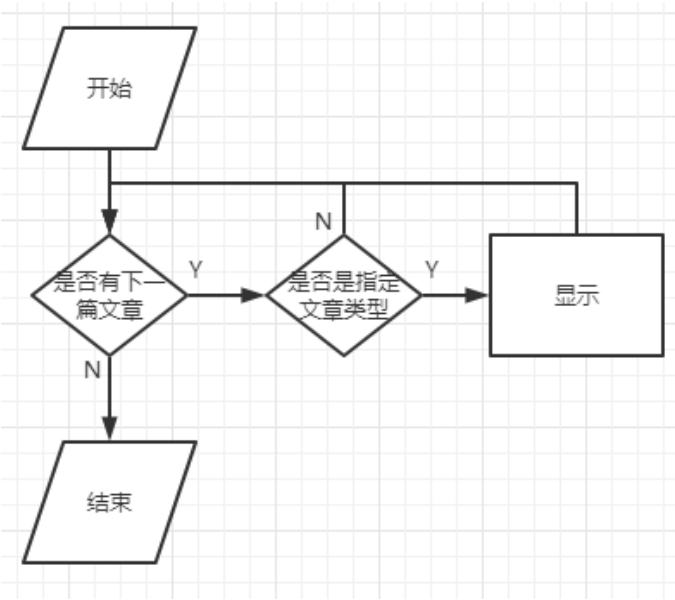

图 4-3 前端文章显示

第四节 后端总体设计

主要显示的是后台的管理界面，但是除了登陆界面外不允许绕过中间件，检测到没有登陆就进入登陆界面。（如图 4-4）

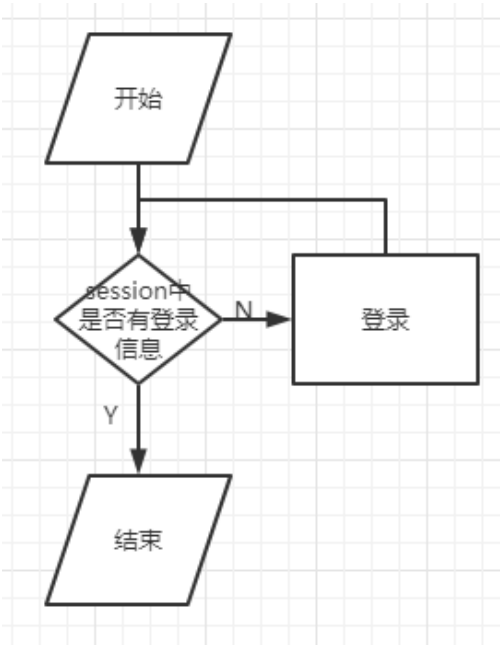

图 4-4 后端

第五节 后台登录程序设计

如果在 session 中检测到了登录信息，那就直接跳转至后台界面，如果没有则跳转至登陆界面，在登陆界面中需要有普通的 POST 和 GET 登录方式，也要有 AJAX 的直接提示登陆方式，普通的 POST 和 GET 登陆方式登陆成功后进入后台管理界面，否则跳转至提示失败的

界面，AJAX 登录成功显示“1”字符，否则显示登陆失败提示。（如图 4-5）

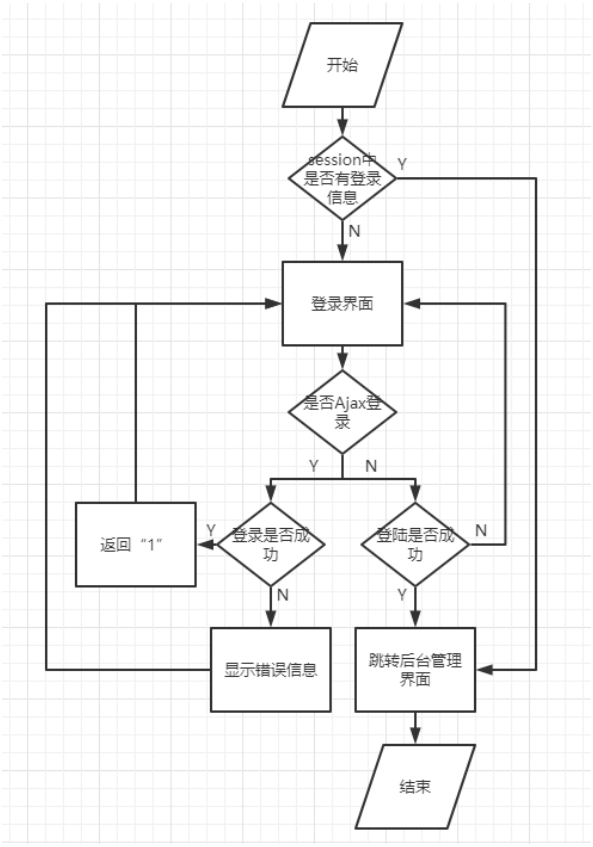

图 4-5 后台登陆

第六节 后台资讯添加程序设计

使用 HTML55 语言编写该界面，然后使用 GET 和 POST 方法进行数据的传输，使用数据库储存信息。（如图 4-6）

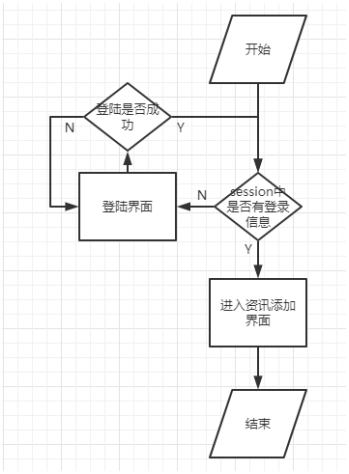

图 4-6 增

第七节 后台资讯删除程序设计

使用 Javascript 在后台管理界面实现该功能，使用 URL 控制数据库删除该条记录。（如图 4-7）

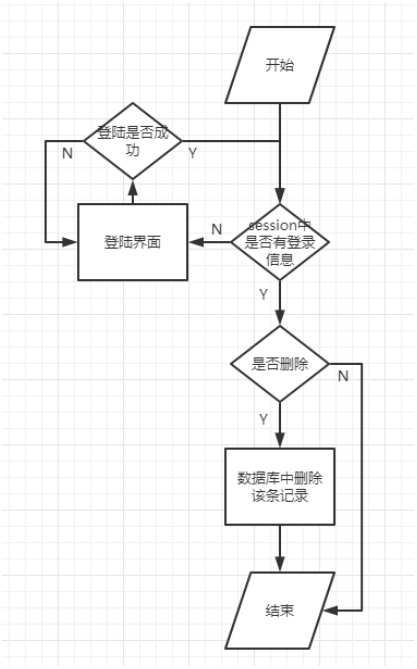

图 4-7 删

第八节 后台资讯修改程序设计

原理与添加资讯相同，但是需要 GET 和 POST 进行传参，参数主要是文章标题、文章类型和文章内容，对修改后的信息进行数据库信息修改即可。（如图 4-8）

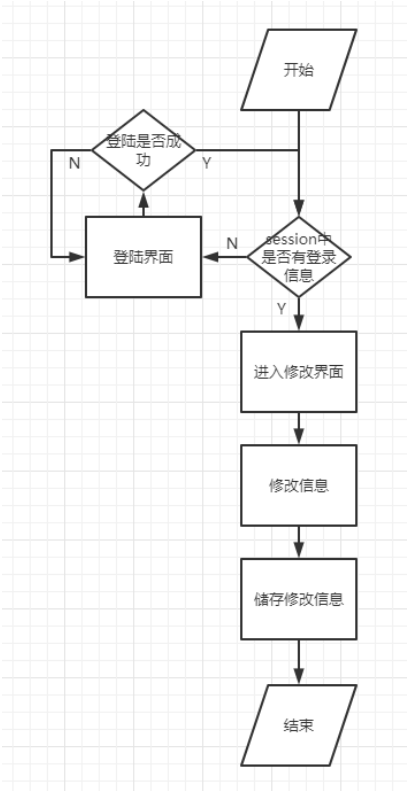

图 4-8 改

第九节 后台资讯查询程序设计

利用 Django 在前端的查看某类文章类型的文章时所用，用 for 循环遍历整个文章数据库，然后用 if 语句筛选出符合文章类型条件的文章并显示。(如图 4-9)

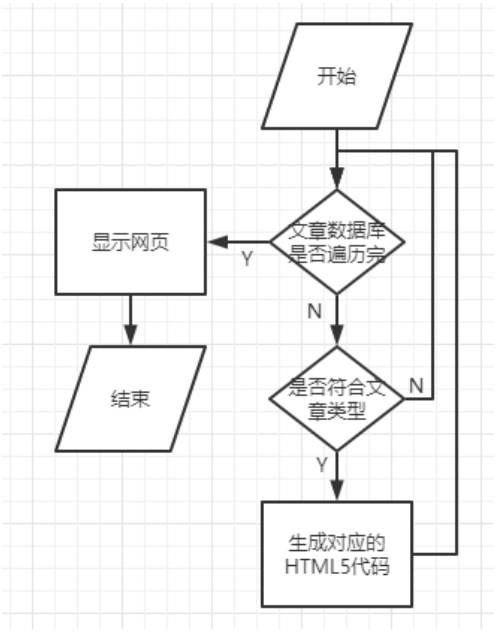

图 4-9 查

第十节 后台路由跳转程序设计

利用 URL 和 Views 进行路由的匹配和方法的实现。(如图 4-10)

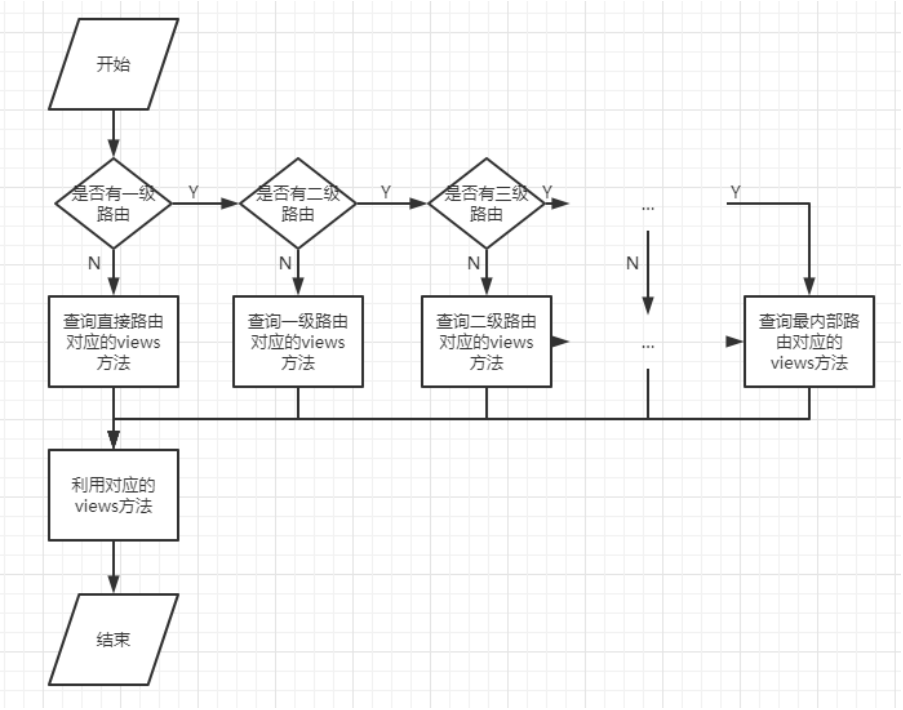

图 4-10 后台路由

第五章 系统实现

第一节 创建 Python3 虚拟环境

在命令提示符下进入指定目录，使用如下代码格式即可进行虚拟环境初始化：

```
python -m venv venv
```

等待一段时间，工作完成后，Python3 虚拟环境创建成功；当然，你可以使用曾经安装过的 Python3 环境，例如 Anaconda3 或者 Python3.x，这样无需使用以上命令。

第二节 下载 Django

打开 Pycharm，进入虚拟环境文件夹所对应的目录，在 Terminal 中输入如下命令：

```
pip install django
```

这将下载最新版本的 Django，当工作完成后，Django 就被部署在虚拟环境中；当然，如果您在您的环境中部署过 Django，无需使用以上代码。

第三节 配置 Django 项目

在 Terminal 中输入如下命令：

```
django manage.py startproject mysite
```

等待工作完成，名为 mysite 的 Django 项目就创建好了。

第四节 配置 Django 项目内 app

使用如下代码创建 login、article、front 三个 app

```
python manage.py startapp login
python manage.py startapp article
python manage.py startapp front
```

等待工作完成，三个 app 就创建完毕；因为接下来的工作需要，需要在三个 app 所对应的文件夹中创建 urls.py。

第五节 更改 setting.py 文件内容

对 mysite 文件夹下的 setting.py 内的部分语句改为如下内容：

```
INSTALLED_APPS = [
    'django.contrib.admin',
    'django.contrib.auth',
    'django.contrib.contenttypes',
    'django.contrib.sessions',
    'django.contrib.messages',
    'django.contrib.staticfiles',
    'login',
    'article',
    'front'
]
```

```
MIDDLEWARE = [
    'django.middleware.security.SecurityMiddleware',
    'django.contrib.sessions.middleware.SessionMiddleware',
    'django.middleware.common.CommonMiddleware',
    # 'django.middleware.csrf.CsrfViewMiddleware',
    'django.contrib.auth.middleware.AuthenticationMiddleware',
    'django.contrib.messages.middleware.MessageMiddleware',
    'django.middleware.clickjacking.XFrameOptionsMiddleware',
    'middleware.login.LoginMiddleware'
]
TEMPLATES = [
    {
        'BACKEND': 'django.template.backends.django.DjangoTemplates',
        'DIRS': [os.path.join(BASE_DIR, 'templates')],
        'APP_DIRS': True,
        'OPTIONS': {
            'context_processors': [
                'django.template.context_processors.debug',
                'django.template.context_processors.request',
                'django.contrib.auth.context_processors.auth',
                'django.contrib.messages.context_processors.messages',
            ],
        },
    },
]
STATIC_URL = '/static/'
STATICFILES_DIRS = [
    os.path.join(BASE_DIR, 'static')
]
X_FRAME_OPTIONS = "SAMEORGIN"
```

INSTALLED_APPS 内写入了刚创建的三个 app 的名字；MIDDLEWARE 中写入了所需的中间件，其中'middleware.login.LoginMiddleware'是我们将要编写的关于登陆的中间件，起到了中间保护的作用；TEMPLATES 使模板路径对应了'templates'文件夹；STATIC_URL 和 STATICFILES_DIRS 定义了静态文件所在的位置，这里包含了 css 文件、js 文件和图片等文件；定义 X_FRAME_OPTIONS 为"SAMEORGIN"。

第六节 设置 mysite 路由

mysite 中的路由定义如下：

```
from django.contrib import admin
from django.urls import path, include
urlpatterns = [
    path('admin/', admin.site.urls),
    path('login/', include('login.urls')),
```

```
    path('article/', include('article.urls')),
    path('front/', include('front.urls'))
]
```

这里这三个 app 都使用 include 方法定义了自己的二级路由，那么我们就在接下来的工作中定义他们的二级路由。

第七节 设置 login 路由

login 这个 app 是后台管理界面的主界面，定义了以下 URL:

```
from django.urls import path
from login import views
urlpatterns = [
    path('login/', views.login),
    path('init/', views.init),
    path('index/', views.index),
    path('exit/', views.exit),
    path('welcome/', views.welcome),
    path('ajaxlogin/', views.ajaxlogin)
]
```

其中'login/'对应了其 views 中的 login 方法；'init/'对应了其 views 中的 init 方法；'index/'对应了其 views 中的 index 方法；'exit/'对应了其 views 中的 exit 方法；'welcome/'对应了其 views 中的 welcome 方法；'ajaxlogin/'对应了其 views 中的 ajaxlogin 方法.

第八节 设置 article 路由

article 这个 app 是文章的管理、增加和修改界面，定义了以下 URL:

```
from django.urls import path
from article import views
urlpatterns = [
    path('articlelist/', views.articlelist),
    path('articleadd/', views.articleadd),
    path('typeinit/', views.typeinit),
    path('articledel/', views.articledel),
    path('articleedit/', views.articleedit)
]
```

其中'articlelist/'对应了其 views 中的 articlelist 方法；'articleadd/'对应了其 views 中的 articleadd 方法；'typeinit/'对应了其 views 中的 typeinit 方法；' articledel /'对应了其 views 中的 articledel 方法；' articleedit /'对应了其 views 中的 articleedit 方法。

第九节 设置 front 路由

front 这个 app 是前端界面，定义了以下 URL:

```
from django.urls import path
from front import views
```

```
urlpatterns = [
    path('', views.default),
    path('types/', views.types),
    path('articles/', views.articles)
]
```

其中直接输入其一级目录将会进入其 views 中的 default 方法；'types/'对应了其 views 中的 types 方法；'articles/'对应了其 views 中的 articles 方法。

第十节 设置 login 所对应的 views.py 文件内容

login 这个 app 主要管理了登陆界面和后台管理界面的主界面，以下是其 views 实现方法的代码：

```
from django.db import IntegrityError
from django.http import HttpResponse
from django.shortcuts import render, redirect
from login.models import Users
def login(request):
    if request.session.get('id', None):
        return redirect('/login/index/')
    if request.method == "GET":
        return render(request, 'login.html')
    else:
        username = request.POST.get("username")
        password = request.POST.get("password")
        info = ""
        if res:
            request.session['id'] = res[0].id
            request.session['username'] = res[0].username
            return redirect('/login/index/')
        else:
            info = "登录失败！"
        return render(request, 'temp.html', {"info": info})
def init(request):
    username = "admin"
    password = "1234"
    users = Users()
    users.username = username
    users.password = password
    try:
        users.save()
    except IntegrityError:
        return HttpResponse("用户名重复！")
    return HttpResponse("ok")
def index(request):
```

```
    if request.session.get('id', None):
        return render(request, 'index.html')
    else:
        return HttpResponse("请重新登录！")
def exit(request):
    if request.session.get('id', None):
        del request.session['id']
    if request.session.get('username', None):
        del request.session['username']
    return redirect('/login/login/')
def welcome(request):
    return render(request, 'welcome.html')
def ajaxlogin(request):
    username = request.POST.get("username")
    password = request.POST.get("password")
    res = Users.objects.filter(username=username, password=password)
    if res:
        return HttpResponse("1")
    else:
        return HttpResponse("0")
```

login 方法进行了一些判断，如果已经登陆过了，将会走路径/login/index/，否则就进入 login.html 文件；没有登陆的情况下登陆成功将会走路径/login/index/，否则进入 temp.html 文件提示错误信息，传参为 info。（如图 5-1 为登陆界面）

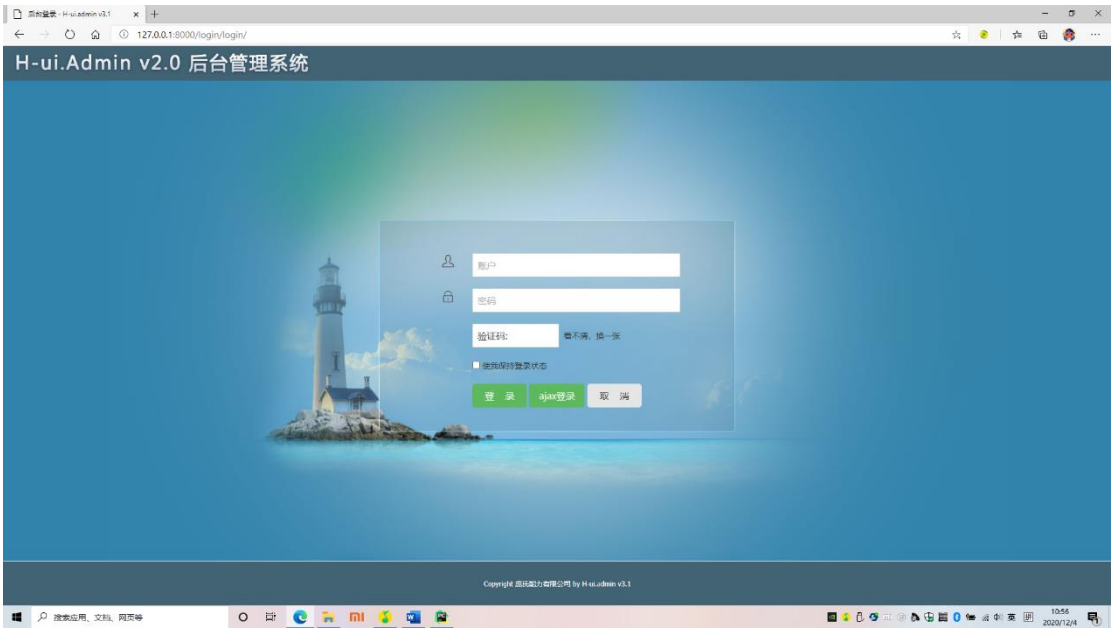

图 5-1 登录界面

init 方法相当于模拟了一个注册的工作，username 指定了用户名为 admin，password 指定了密码为 1234，如果遇到了相同的用户名，数据库将无法写入这个用户（如图 5-2），因为用户名为 unique，否则返回一个含有“ok”的页面。

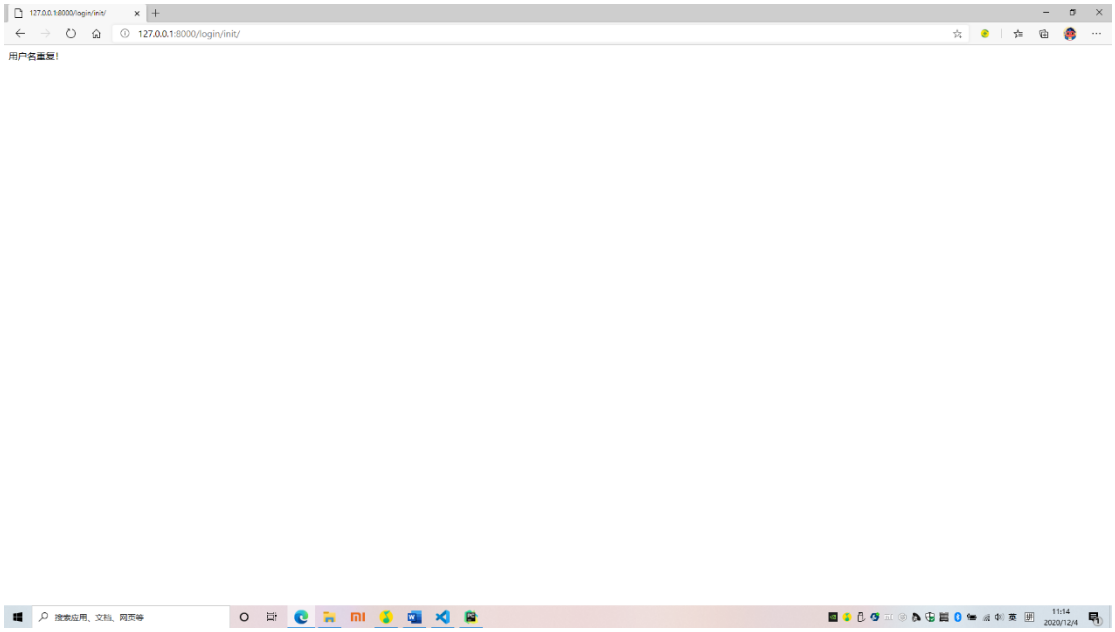

图 5-2 模拟注册

exit 方法用与页面中的“退出”按钮（如图 5-3），点击按钮后会在 session 中删除 id 和 username，就相当于推出登录状态，同时页面跳转至路径/login/login/。

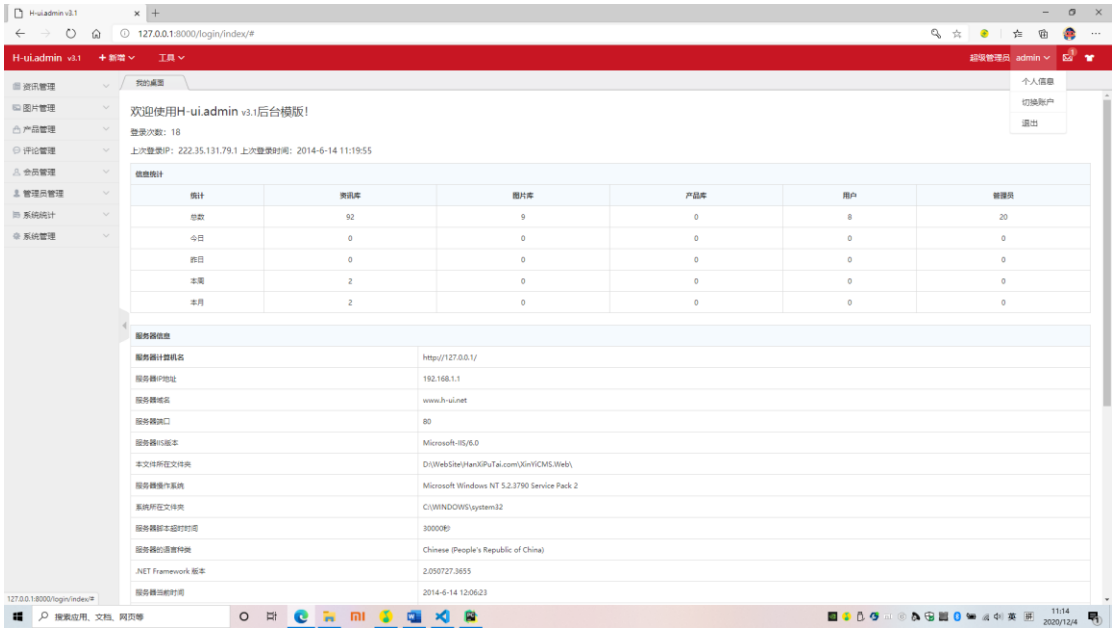

图 5-3 退出

welcome 方法主要是渲染 welcome.html 这个界面，这个界面位于后台登陆界面的数据块中，为了显示数据使用。（同上图）

ajaxlogin 方法是为了测试 ajax 登陆方法所设立的，没有实际登陆意义，这样做的好处就是保留了登录错误时的登陆页面状态，用户输入过的信息被保留了下来，不被重置。（如图 5-4）

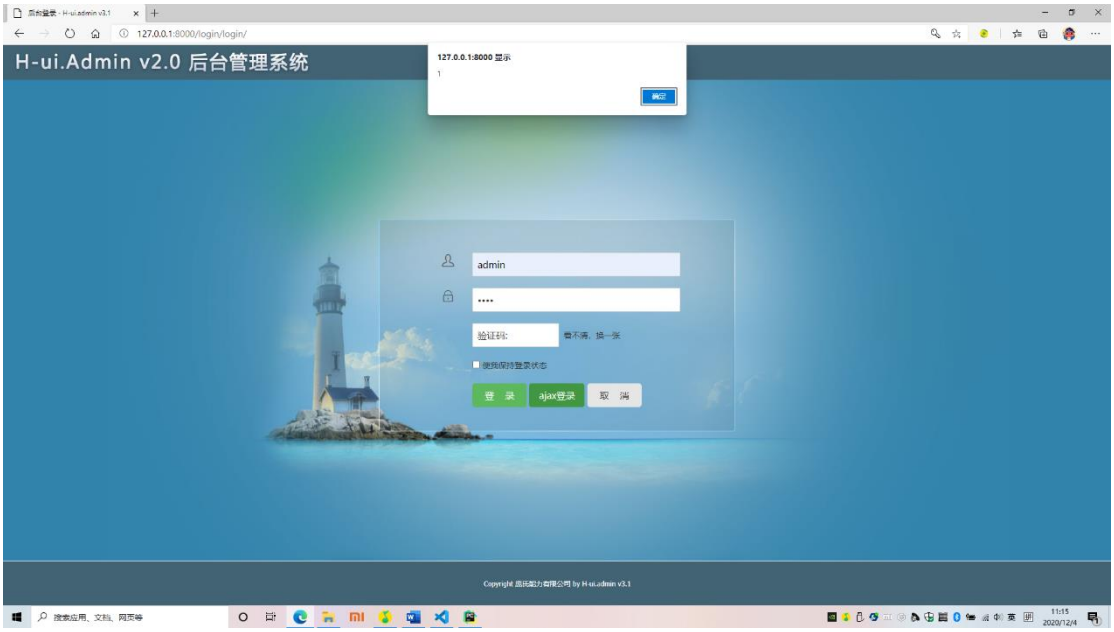

图 5-4 登陆界面

第十一节 设置 article 所对应的 views.py 文件内容

article 这个 app 主要管理了资讯管理、资讯添加和资讯修改界面，以下是其 views 实现方法的代码:

```
from django.http import HttpResponse
from django.shortcuts import render, redirect
from article.models import Types, Articles
def articlelist(request):
    articles = Articles.objects.all()
    return render(request, 'article-list.html', {'articles': articles})
def articleadd(request):
    if request.method == 'GET':
        types = Types.objects.all()
        print(types)
        return render(request, 'article-add.html', {'types': types})
    else:
        title = request.POST.get("title")
        type = request.POST.get("type")
        content = request.POST.get("content")
        author = request.session.get("id")
        article = Articles(title=title, content=content, type_id=type,
author_id=author)
        article.save()
        return HttpResponse("文章添加成功！")
def typeinit(request):
    type = Types(title='国内新闻')
```

```
    type.save()
    type = Types(title='国际新闻')
    type.save()
    type = Types(title='财经新闻')
    type.save()
    return render(request, 'temp.html', {"info": "文章类型初始化完成！"})
def articledel(request):
    id = request.POST.get('id')
    Articles.objects.filter(id=id)[0].delete()
    return HttpResponse("1")
def articleedit(request):
    if request.method == 'GET':
        id = request.GET.get('id')
        article = Articles.objects.filter(id=id)[0]
        types = Types.objects.all()
        return render(request, 'article-
edit.html', {'article': article, 'types': types})
    else:
        title = request.POST.get("title")
        type = request.POST.get("type")
        content = request.POST.get("content")
        id = request.POST.get("id")
        author = request.session.get("id")
        article = Articles.objects.filter(id=id)[0]
        article.title = title
        article.type_id = type
        article.author_id = author
        article.content = content
        article.save()
        return HttpResponse("修改成功！")
```

articlelist 方法将所有资讯信息传参给 article-list.html 并渲染。（如图 5-5）

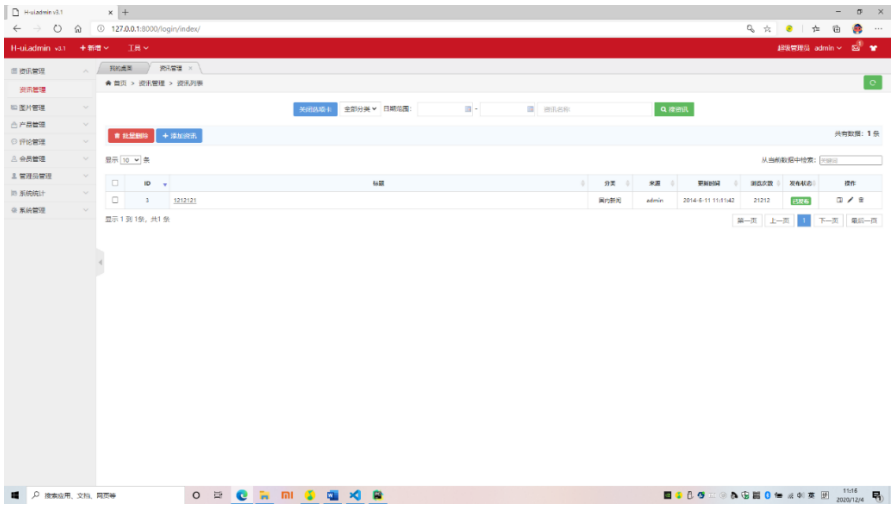

图 5-5 资讯页面

articleadd 方法是用于添加资讯的方法，使用 GET 和 POST 方法进行传参，同时涉及到了数据库，如果文章添加成功，则会提示“文章添加成功！”。（如图 5-6）

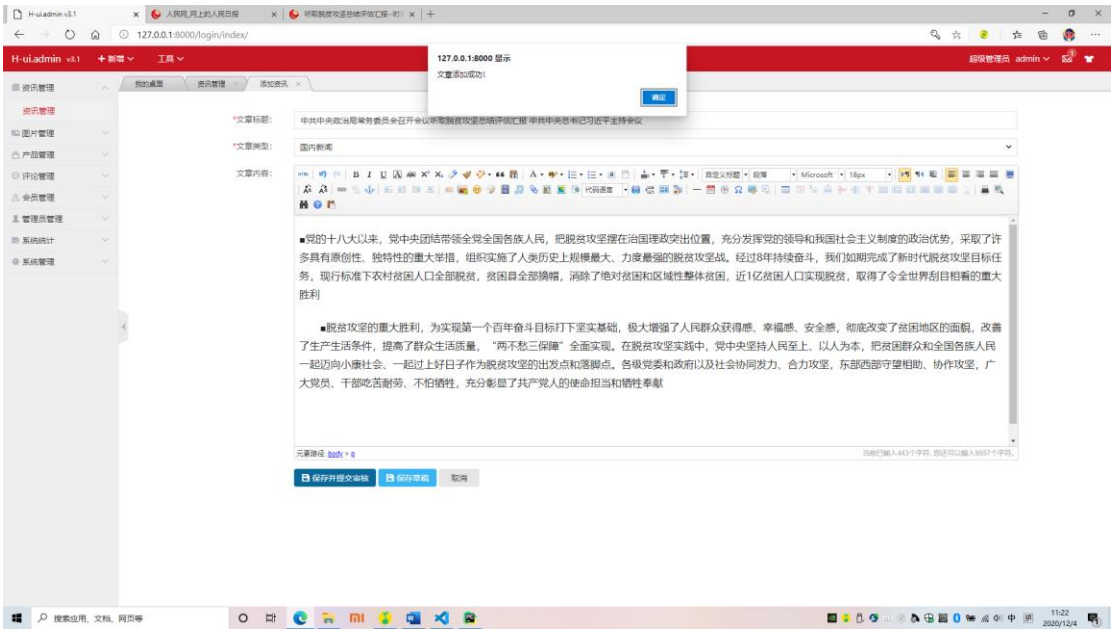

图 5-6 资讯增加

typeinit 方法主要是初始化资讯类型。

articledel 方法实现了文章的删除，找到对应 id 后再数据库中执行删除操作，并返回“1”。（如图 5-7）

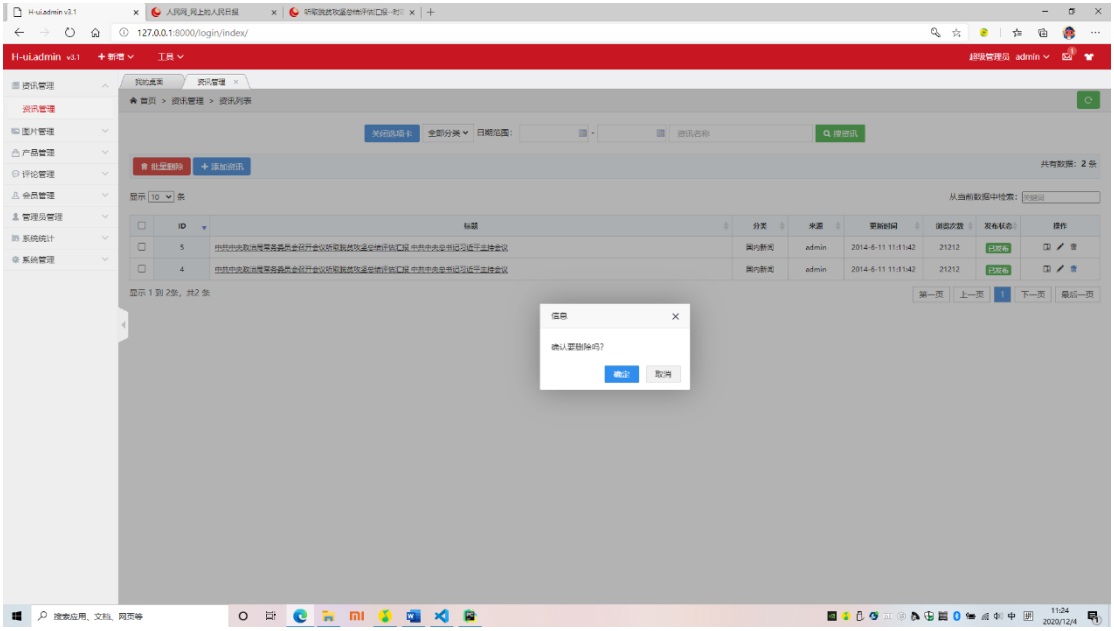

图 5-7 资讯删除

artileedit 方法用于修改资讯，其原理和 articleadd 方法差不多，不过在使用时使用 GET 和 POST 方法进行传参，使页面显示出原始资讯的标题、类型和内容。（如图 5-8）

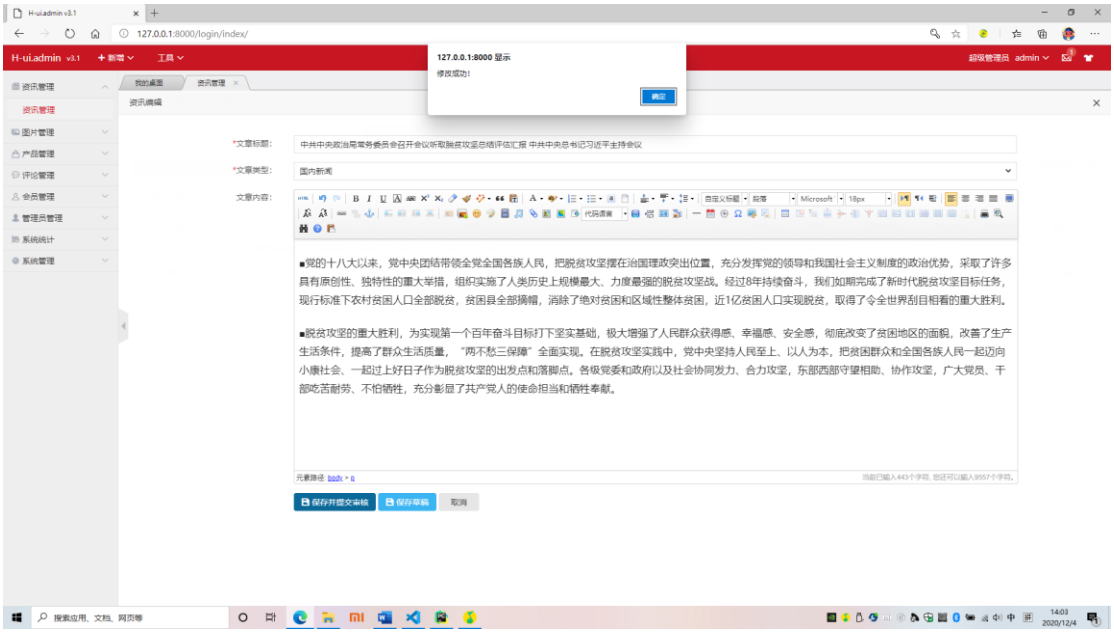

图 5-8 资讯修改

第十二节 设置 front 所对应的 views.py 文件内容

```
from django.shortcuts import render
from article.models import Types, Articles
def default(request):
    types = Types.objects.all()
    return render(request, 'default.html', {'types': types})
def types(request):
    id = request.GET.get('id')
    articles = Articles.objects.filter(type_id=id)
    return render(request, 'front-types.html', {'articles': articles})
def articles(request):
    id = request.GET.get('id')
    article = Articles.objects.filter(id=id)[0]
    return render(request, 'font-article.html', {'article': article})
```

default 方法是整个前台的主页（如图 5-9），传了参 types，主要是显示出所有的文章类型，其对应的 HTML5 代码为：

```
{% load static %}
<!DOCTYPE html>
<html lang="en">
<head>
    <meta charset="UTF-8">
    <title>Title</title>
    <link rel="stylesheet" href="{% static "bootstrap/css/bootstrap.css
" %}">
```

```
    <style>
        .top {
            height: 200px;
            background-color: #00a0e9;
            margin: 0px;
        }
        .top img {
            height: 180px;
            margin-top: 10px;
        }
        .copyright {
            height: 30px;
            background-color: grey;
        }
        #mynav li:first-child {
            margin-left: 2px !important;
        }
        .setbg {
            background: url("{% static "images/Explosion_4k.jpg" %}");
            filter: "progid:DXImageTransform.Microsoft.AlphaImageLoader
(sizingMethod='scale')";
            -moz-background-size: 100% 100%;
            background-size: 100% 100%;
        }
    </style>
</head>
<body>
<div class="row justify-content-center container-fluid top">
    <img src="{% static "images/logo.png" %}" alt="">
</div>
<div class="container-fluid setbg" style="height: 800px">
    <nav aria-label="breadcrumb" class="row justify-content-center">
        <ol class="breadcrumb" id="mynav">
            {% for type in types %}
                <li class="ml-
3"><a href="/front/types/?id={{ type.id }}">{{ type.title }}</a></li>
            {% endfor %}
        </ol>
    </nav>
</div>
<div class="copyright">
    <div class="row justify-content-center">
        <span>庞怡文有部分代码的版权信息（当然框架还是从马老师那里改的哈哈哈
哈哈哈哈哈哈）</span>
```

```
    </div>
</div>
</body>
<script src="{% static "jquery.js" %}"></script>
<script src="{% static "popper.min.js" %}"></script>
<script src="{% static "bootstrap/js/bootstrap.min.js" %}"></script>
</html>
```

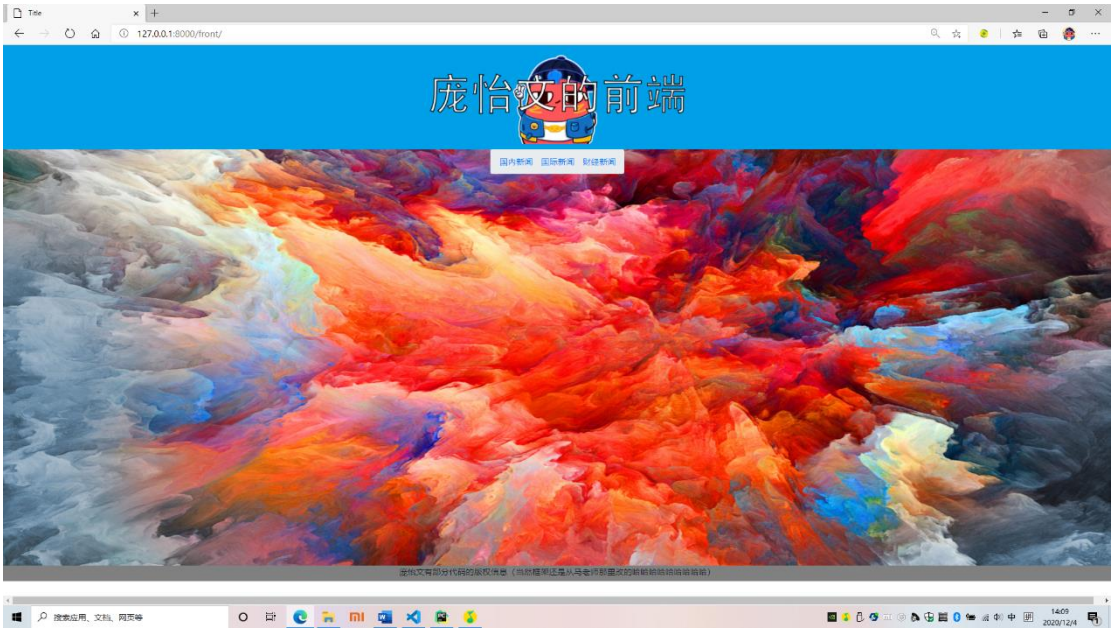

图 5-9 前端页面

types 方法是获取对应的资讯类型下的文章，不能显示别的类型下的资讯（如图 5-10），对应的 HTML5 代码为：

```
{% load static %}
<!DOCTYPE html>
<html lang="en">
<head>
    <meta charset="UTF-8">
    <title>Title</title>
    <link rel="stylesheet" href="{% static "bootstrap/css/bootstrap.css
" %}">
</head>
<body>
<div>
    {% for article in articles %}
        <a href="/front/articles/?id={{ article.id }}"
           class="list-group-item list-group-item-
action">{{ article.title }}</a>
    {% endfor %}
</div>
<script src="{% static "jquery.js" %}"></script>
```

```
<script src="{% static "popper.min.js" %}"></script>
<script src="{% static "bootstrap/js/bootstrap.min.js" %}"></script>
</body>
</html>
```

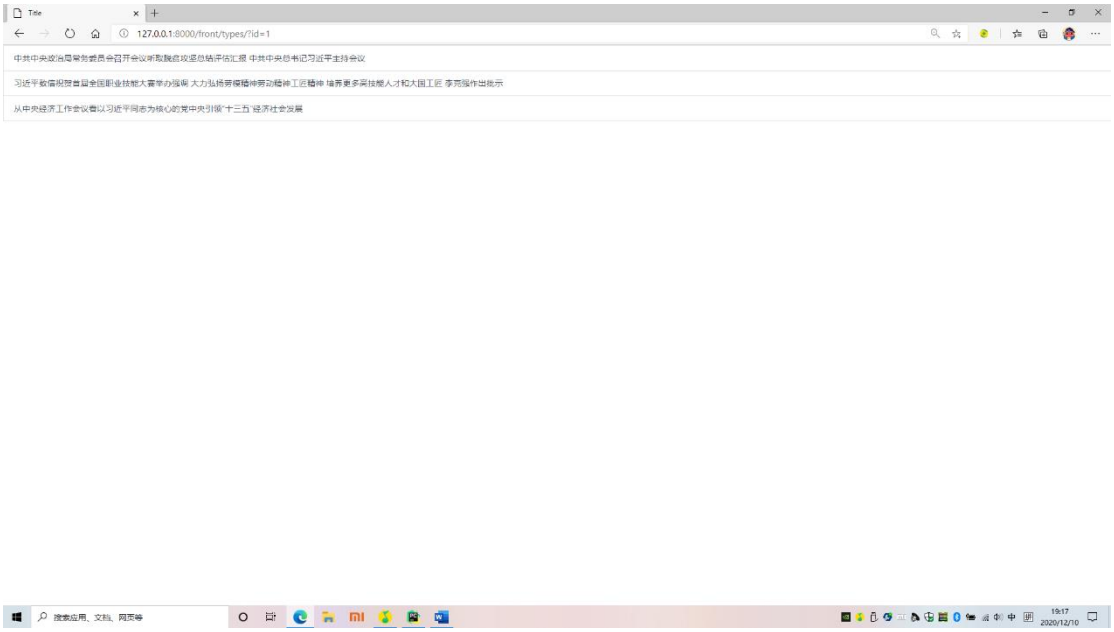

图 5-10 资讯标题

第十三节 模版内的静态文件适配

静态文件主要靠两个代码来实现:

```
{% load static %}
{% static "文件路径" %}
```

这样就能实现对其静态文件的解析。

第十四节 模板文件的动态内容适配

动态内部主要是使用了 for 和 if 与 else 来实现动态内容的生成，例如项目内的以下部分代码中进行了使用:

```
<div class="row cl">
    <label class="form-label col-xs-4 col-sm-2"><span class="c-
red">*</span>文章类型：</label>
    <div class="formControls col-xs-8 col-sm-9"> <span class="select-
box">
        <select id="articletype" name="articletype" class="select">
            {% for type in types %}
                {% if type.id == article.type.id %}
                    <option selected="selected" value="{{ type.id }}">{
{ type.title }}</option>
                {% else %}
```

```
                    <option value="{{ type.id }}">{{ type.title }}</opt
ion>
                {% endif %}
            {% endfor %}
        </select>
    </span>
    </div>
</div>
```

第十五节 中间件 LoginMiddleware 的编写

中间件是为了防止没有登陆后台的情况下进入后台进行操作，其实现代码如下所示：

```
from django.http import request
from django.shortcuts import redirect
class LoginMiddleware:
    def __init__(self, get_response):
        self.get_response = get_response
        # One-time configuration and initialization.
    def __call__(self, request):
        # Code to be executed for each request before
        # the view (and later middleware) are called.
        urls = [
            '/login/login/',
            '/login/ajaxlogin/',
            '/front/',
            '/front/types/',
            '/front/articles/'
        ]
        if request.session.get('id') or request.path_info in urls:
            response = self.get_response(request)
            return response
        else:
            return redirect('/login/login/')
```

其中 urls 列表中的五个字符串规定了绕开此中间件的几个路由，如果不满足的话将会跳转至路径/login/login/进行的登录操作。

第十六节 数据库的构造

对于 Uesrs 数据表的构建代码如下：

```
from django.db import models
class Users(models.Model):
    username = models.CharField(max_length=50, unique=True)
    password = models.CharField(max_length=50)
```

对于 Types 数据表的构建代码如下：

```
from django.db import models
from login.models import Users
class Types(models.Model):
    title = models.CharField(max_length=50)
class Articles(models.Model):
    title = models.CharField(max_length=100)
    content = models.TextField(default="这个人很懒，什么都没有写！")
    type = models.ForeignKey(Types, on_delete=models.CASCADE)
    author = models.ForeignKey(Users, on_delete=models.CASCADE)
```

Users 的数据库如下图 5-11:

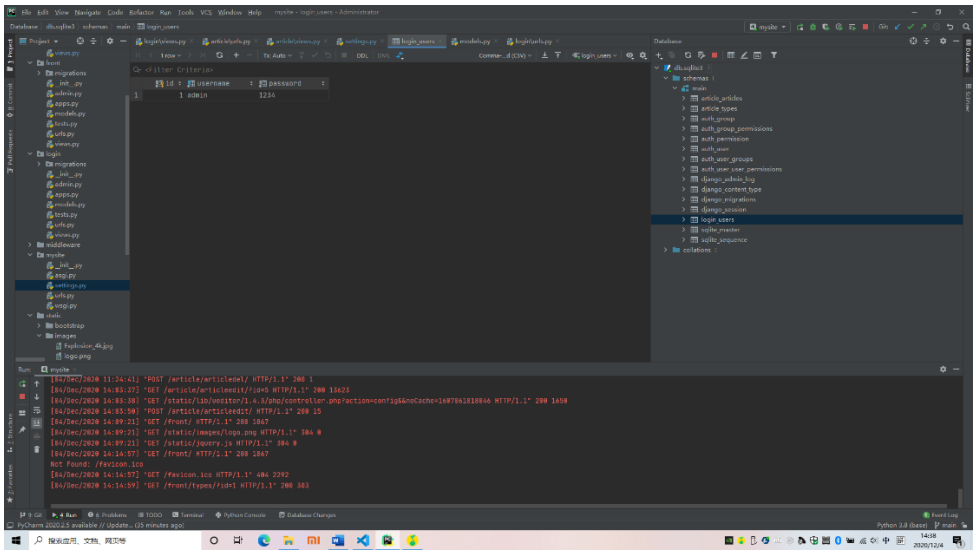

图 5-11 Users 数据库

Types 的数据库如下图 5-12:

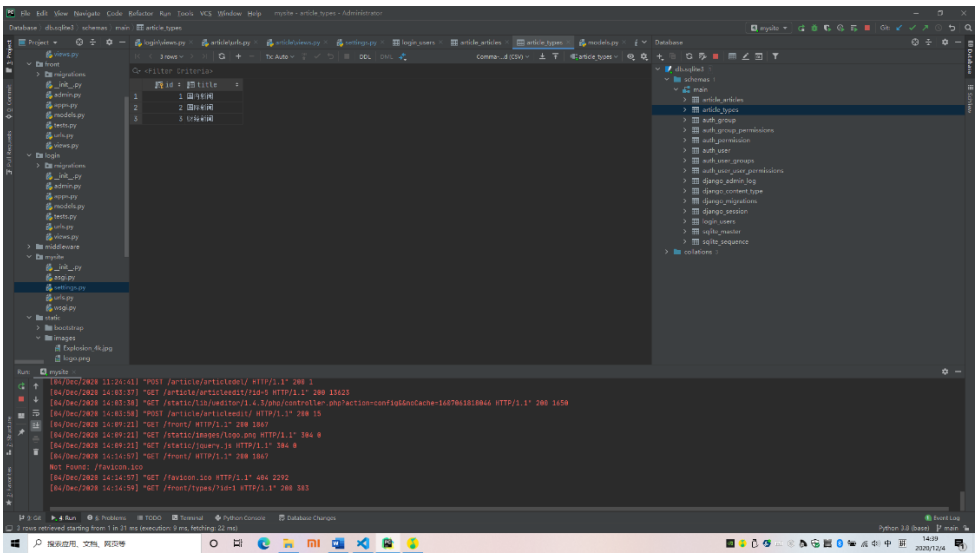

图 5-12 Types 数据库

第十七节 数据迁移

这个过程只需在 Terminal 中输入两行代码即可完成：

```
python manage.py makemigrations
python manage.py migrate
```

就完成了数据迁移。

第十八节 数据库的创建

数据库配置如下图 5-13：

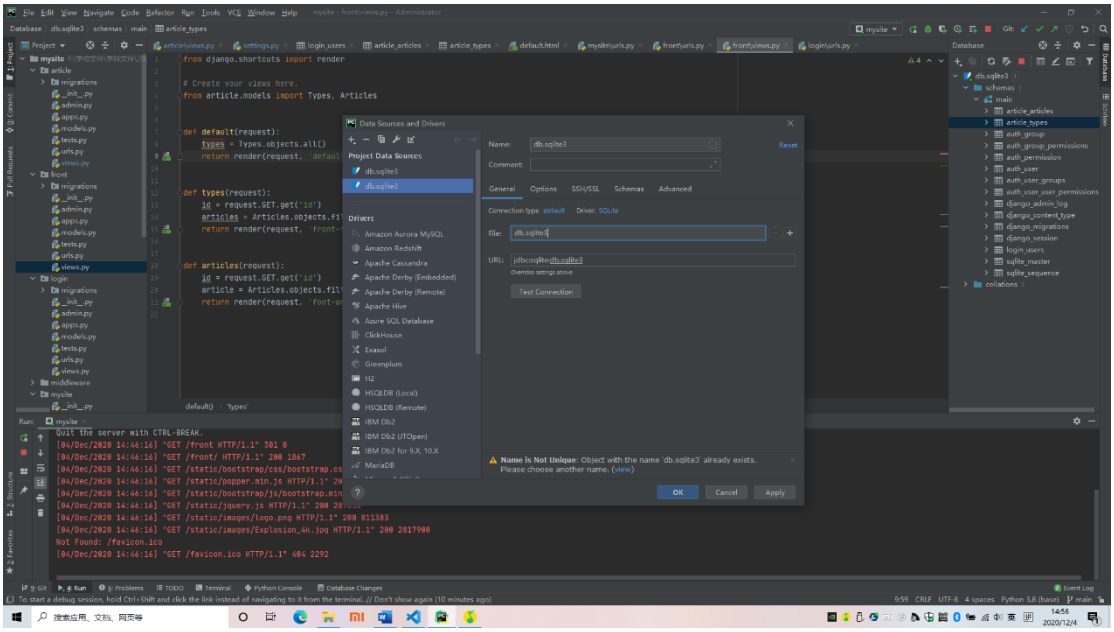

图 5-13 数据库配置

第十九节 数据库的初始化

运行 127.0.0.1:8000/login/init 和 1270.0.1:8000/article/typeinit 即可（前提是服务器已运行，具体运行服务器方法请看下方“服务器测试”。）。

第二十节 服务器测试

在 PyCharm 里直接运行 Django 或者在 Terminal 里输入以下代码运行：

```
python manage.py runserver 8000
```