上海大学

SHANGHAI UNIVERSITY

课程论文

COOURSE PAPER

题 目: 基于 Django 框架的 CMS 系统基础功能实现

课程名称 生产实习 A

课程编号 0101A007

小 组 程自岚 16121818

单紫叶 16121980

代巧巧 16121457

马竹寒 16121723

基于 Django 框架的 CMS 系统基础功能实现

摘要

为实现通用会议网站CMS系统的构建，我们引入了$Django$框架，利于已经封装好的模块进行操作与补充，利用$SQLite$数据库与$HTML$语言的使用，克服了包括数据库引入、图片保存等一系列困难，并对注册、登入与登出这个极易出错的系统进行了有效的优化，最终实现了包括网站注册、登录与登出、网页访问人数统计、用户访问的到达时间与离开时间与文件和图片的上传保存等一系列操作。

关键词：CMS 系统；Django；Python；HTML；SQLite 数据库等。

Abstract

In order to realize the construction of CMS system of General Conference website, we introduce Django framework, which is beneficial to the operation and supplement of encapsulated modules. By using SQLite database and HTML language, we overcome a series of difficulties including the introduction of database, preservation of pictures and so on. We have been also effectively optimized the error-prone system of register, log in and log out. And finally we realized a series of operations including website registration, login and logout, statistics of page visitors, arrival and departure time of user visits and upload and preservation of files and pictures.

Keywords：CMS System; Django; Python; HTML; SQLite Database etc

目录

1. 选题背景与思路 1
2. 预期基础功能与优化 1
   2.1. 基础功能 1
   2.2. 功能优化 1
3. 基本原理 1
   3.1. 框架——Django 1
      3.1.1. $M—models$ 2
      3.1.2. $T—Templates$ 2
      3.1.3. $V—views$ 3
   3.2. Html 3
   3.3. 数据库——SQLite 4
4. 基础功能 4
   4.1. 完全访问者权限 4
   4.2. 注册、登录与登出功能 4
      4.2.1. 注册 4
      4.2.2. 登录 6
      4.2.3. 登出 7
   4.3. 图片与文件的提交功能 7
   4.4. 到达与离开时间 8
   4.5. 网站访问人数统计 9
   4.6. 优化功能 11
      4.6.1. 未注册用户登录报错 11
      4.6.2. 已注册用户再次注册跳转登录页面 11
5. 障碍与解决 11
   5.1. 数据库引入 11

5.2. 图片上传 12
6. 优点与缺点 12
6.1. 优点 12
6.2. 缺点 12
7. 全部代码 13
目录一览 13
7.1. app 13
7.2. mysite 20
7.3. Templates 23
7.4. manage.py 27

1. 选题背景与思路

在传统的通用会议中，对于会议材料的收集往往采取两种方式，其一是线下的纸质回执等材料收集，其二是通过电子邮箱收集。纸质材料的收集固然是与会期间较为正式的一种材料收集方式，但电子邮箱提交方式，却存在着一系列值得商榷的问题：一方面从会议主办方对邮件的收集难度角度来看，需要打开收件箱一一查看，对文件进行下载与保存，从而浪费了较多人力与时间资源，因此我们希望能够简化这一过程；另一方面由于操作不当或者是邮箱自身的垃圾邮件认定机制，很容易遗漏部分与会人员的提交文件，我们希望能够避免此类失误。

而CMS系统（即$Content\ Management\ System$，内容管理系统）就为我们提供了这样一种问题解决办法，它使用简捷、建设速度快、管理方便，却具有完善的信息管理与发布管理功能。且它基于强健的多层体系架构，遵从开放标准，易于与其他应用集成，实现功能扩展和快速部署。 因此我们希望能够建立一个CMS系统，进行会议信息的采集、整理、分类、审核、发布和管理。

为建立有效的CMS系统，从降低不同代码块间的耦合以降低代码写作难度出发，我们引入了$Django$框架，直接对已被封装的类与方法进行操作与补充，从而避免了过于繁琐复杂的结构写作对于初学者的难度要求。而由于$Django$采用MVT的框架模式，代码与样式完全分离，因此在模板部分，我们需要完全抛弃$Python$语言而通过$html$语言书写。网站访问结果及数据将被保存在$SQL\quad e$数据库中。

2. 预期基础功能与优化

2.1. 基础功能

- 网站可实现注册、登录与登出
- 注册用户可提交图片与文字到网站
- 网站可显示最近访问用户的到达时间与离开时间
- 后台（数据库）可进行网站访问人数统计

2.2. 功能优化

- 未注册用户登录报错
- 已注册用户再次登录跳转登录页面

3. 基本原理

3.1. 框架——$Django$

$Django$是一个开放源代码的Web应用框架，由$python$写成，它用了MVT的框架模

式，即模型M，视图V和模板T。其架构总览图见 **Figure 1**。以下我们将主要介绍框架模式的三个部分。

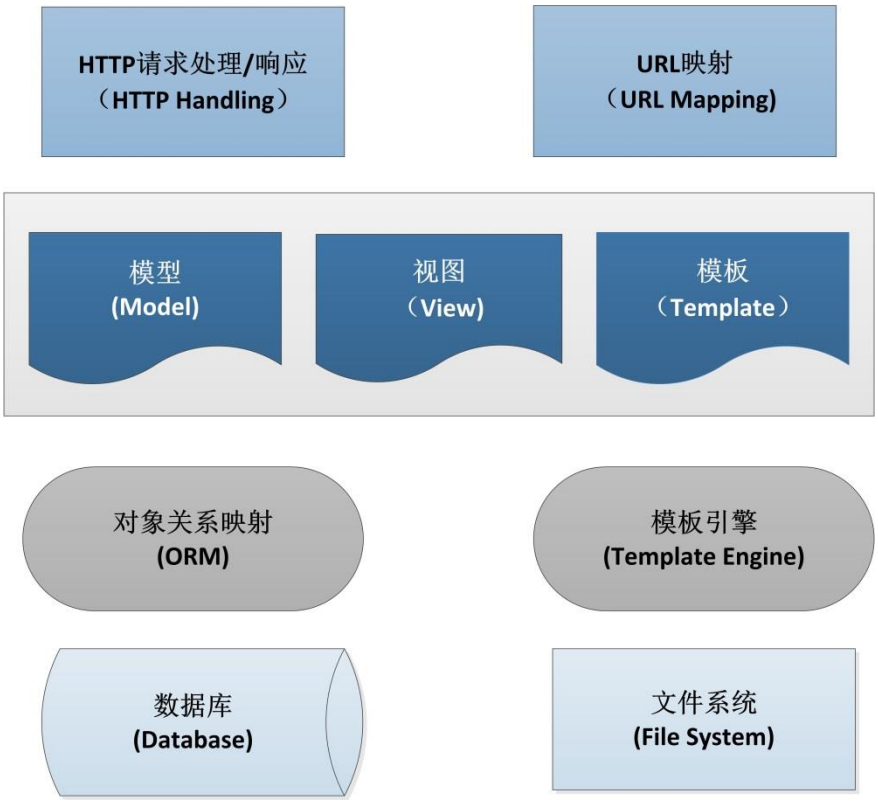

Figure 1 Django 架构总览图

3.1.1. M—$models$

我们需要在$setting.py$ 中的$database$的字典中配置数据库（我们采用的是$Python$自带的$SQLite$）。配置完成后使用$manage.py\ startapp$来创建app在$models$中编写$python$代码描述实体映射。

$models$包含在$django.db$中，里面封装了模型类的通用接口。$CharField()$是创建$varchar$型数据，参数有$max_length, blank, verbose_name$等，分别表示最大长度、是否为空、显示名称。而$def__unicode__$提供了装箱后的默认显示，如果没有设置此函数，默认显示$object$类型。$class\ Meta$则规定了模型的默认排序字段。

3.1.2. T—$Templates$

模版在$Django$中是显示数据的地方，通常为$HTML$格式，在模版中$Django$的处理逻辑要写在{% %}中，而要显示的变量要写在{{ }}中。$Django$的母板页可以用任何文档充当，前提是要用{% $block\ name$ %}{% $endblock$ %}声明要填充或替换的块，而使用时只需{% $extends$ 母版名字 %}然后调用相应的块就可以了。（在我们的模板中关于登录的$l\quad in.html$，关于注册的$register.html$和关于图片与文字提交$profile.html$继承于父类$base.html$，而$index.html$则定义了主页$homepage$）。

3.1.3. $V—views$

$views$是业务逻辑层，在$Django$里面$views$通常是一个$views.py$模块，放在对应的包里。$views.py$里面是具体的逻辑函数，每一个函数对应着一个或多个模版，为了建立模版与视图的联系，还要有一定的路由机制，于是$Django$通常在根目录有一个路由程序$urls.py$。路由则由$patterns$来创建，用正则表达式来描述，从而极大地提高路由机制的灵活性。

在这里$request$参数是必须的，但是可以任意命名，只要符合规范即可，$request$包含页面的请求信息。$render_to_response$在$django.shortcuts$里，所以前面还需要声明$form\ django.shortcuts\ import\ render_to_response$。而$request.MATE$里含有所有的请求界面信息和用户信息。$sort()$是对$list$从小到大排序。返回值的意思就是向html模版提交一个$values$变量。$Urls$的 $patterns$中元组添加了正则的导向规则：除去原地址匹配’^$’者导向home。当然这前提是$views.py$文件与$urls.py$在同一个文件夹里面否则就要引用home的命名空间。

3.2. Html

超文本标记语言（英语：$HyperText\ Markup\ Language$，简称：$HTML$）是一种用于创建网页的标准标记语言。$HTML$可被来建立自己的 WEB 站点，$HTML$ 运行在浏览器上，由浏览器来解析。

$\boldsymbol{Table\ 1}$是本项目中运用的部分标签，也是在web编写过程中常用的一些标签：

Table 1 部分$HTML$标签

标签	作用
$<$ html $>$ 与 $</$html $>$	限定了文档的开始点和结束点
$< title >$	定义文档的标题
$< body >$	定义文档的主体
$< div >$	定义文档中的分区或节
$<$ hr $>$	在 $HTML$ 页面中创建一条水平线
$< a >$	定义超链接，用于从一张页面链接到另一张页面
$<$ h1 $>$	定义最大的标题
$< br >$	可插入一个简单的换行符
$< form >$	用于为用户输入创建 $HTML$ 表单
$< tr >$	定义 $HTML$ 表格中的行
$< td >$	定义 $HTML$ 表格中的标准单元格
$< table >$	定义 $HTML$ 表格
$< strong >$	把文本定义为语气更强的强调的内容

3.3. 数据库——$SQLite$

自几十年前出现商业应用程序以来，数据库就成为软件应用程序的主要组成部分。由于数据库管理系统的关键形，它们变得非常庞大，并占用了相当多的系统资源，从而增加了管理的复杂性。而随着软件应用程序逐渐模块化，一种新型数据库——嵌入式数据库会比大型复杂的传统数据库管理系统更适应。嵌入式数据库直接在应用程序进程中运行，提供了零配置（$zero-configuration$）运行模式，并且资源占用较少。

$SQLite$就是这样的一个开源的嵌入式关系数据库，它减少了应用程序管理数据的开销，且可移植性好，易于使用，小而高效且可靠。

$SQLite$通过嵌入到使用它的应用程序中而共用相同的进程空间，而非单独的进程。

它的数据库权限只依赖于文件系统，没有用户帐户的概念。$SQLite$有数据库级锁定，而没有网络服务器。

4. 基础功能

4.1. 完全访问者权限

在$Django$框架中，由于自动生成的$setting.py$中代码行如下，使得原有的网站访问权限仅限制于本机：

```
ALLOWED_HOST = [‘’]
```

而我们所要建立的是一个通用会议网站，需要接受一定数目的用户访问。故而我们修改了代码行，使访问权限被扩大为所有人：

```
ALLOWED_HOST = [‘*’]
```

4.2. 注册、登录与登出功能

4.2.1. 注册

我们的注册页面由用户名、邮箱地址、密码、图片输入与文字输入共同组成。

```
def register(request):
    # counting visitor
    change_info(request)
```

在注册视图中，我们通过$change_info$函数计数，为访问量统计做准备。

```
    registered = False
    if request.method == 'POST':
        user_form = UserForm(data=request.POST)
        profile_form = UserProfileForm(data=request.POST)
```

首先将注册初始值设为$False$，为后续注册成功与否的判断做准备。

```
        if user_form.is_valid() and profile_form.is_valid():
            user = user_form.save()
            user.set_password(user.password)
            user.save()
```

判断$HTTP$请求方法是否是$POST$，是的话则创建一个表单。

```
            profile = profile_form.save(commit=False)
            profile.user = user
            if 'picture' in request.FILES:
                profile.picture = request.FILES['picture']
                profile.save()
                registered = True
            else:
                print(user_form.errors, profile_form.errors)
```

以上代码用于判断用户是否有上传的照片，若有则加以保存，若无则报错。

```
    else:
        user_form = UserForm()
        profile_form = UserProfileForm()
```

若$HTTP$请求方法并非$POST$，则清空$user_form$和$profile_form$。最后我们得到返回值如下：

```
    return render(request,
                  'app/register.html',
                  {'user_form': user_form,
                   'profile_form': profile_form,
                   'registered': registered
                   })
```

此外，我们在$forms.py$文件中图片输入与文字输入打包，用户名、邮箱与密码打包，生成$form$表单代码和验证表单数据是否合法。

```
class UserProfileForm(forms.ModelForm):
    class Meta:
        model = UserProfile
        fields = ('desciption', 'picture')
class UserForm(forms.ModelForm):
    password = forms.CharField(widget=forms.PasswordInput())
    class Meta:
        model = User
        fields = ('username', 'email', 'password')
```

注册页的网页效果如 **Figure 2** 所示：

Figure 2 网页显示效果：注册

4.2.2. 登录

我们所写的登录页面由用户名输入与密码输入组成。

```
def user_login(request):
    # counting visitor
    change_info(request)
```

登录视图同注册视图一样，$change_into$函数是写在$counting.py$文件中的访问函数。

```
    if request.method == 'POST':
        username = request.POST.get('username')
        password = request.POST.get('password')
        user = authenticate(username=username, password=password)
```

在判断$HTTP$请求办法是否为$POST$之后，我们需要校验登录的用户名与密码是否能够通过，成功则返回$user$对象，失败则返回$None$。

```
        request.session['username'] = username
        if user:
            if user.is_active:
                login(request, user)
                user.last_login = datetime.now()
                user.save(update_fields=['last_login'])
                return HttpResponseRedirect(reverse('profile',kwargs={'username':
username}))
```

如果登录时的$user$处于激活状态，就跳转到$profile$视图。

```
            else:
                # An inactive account was used - no logging in!
                return HttpResponse("Your Rango account is disabled.")
```

否则则返回错误：无效账号。

```
        else:
```

```
            print("Invalid login details: {0}, {1}".format(username, password))
            return HttpResponse("Wrong user name or password.")
    else:
        return render(request, 'app/login.html', {})
```

登录页的网页效果如 **Figure 3** 所示：

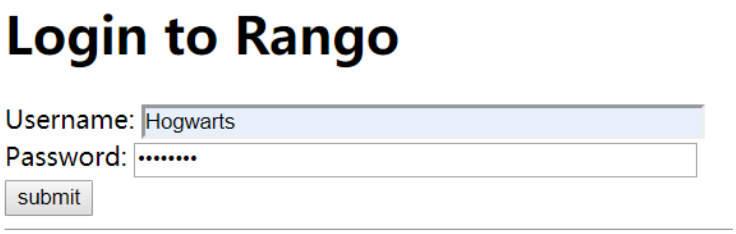

Figure 3 网页显示效果：登录

4.2.3. 登出

Figure 4 网页显示效果：登出

如 **Figure 4** 所示，在$profile$界面，我们提供了$logout$选项，能够有效退出现有的登录情况。

代码行如下：

```
def user_logout(request):
    username = request.session.get('username')
    if username:
        user = User.objects.get(username=username)
        user.date_joined = datetime.now()
        user.save(update_fields=['date_joined'])
```

由于只有用户名是唯一确定的，因此我们以这个$username$作为参数在数据库中进行搜索，将上次登出的时间赋值为现在的时间。

```
    change_info(request)
    # Since we know the user is logged in, we can now just log them out.
    logout(request)
    # Take the user back to the homepage.
    return redirect(reverse('index'))
```

如果退出成功，则能够正确跳转到$index$界面。

4.3. 图片与文件的提交功能

具体的代码解释在 **4.2.1** 中已经加以解释介绍，故此处不再赘述。提交功能的网页效果如 **Figure 5** 与 **Figure 6** 所示：

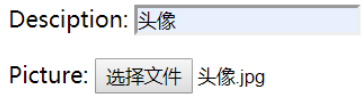

Figure 5 网页显示效果：图片与文字（注册提交）

Figure 6 网页显示效果：图片与文字（登录查看）

4.4. 到达与离开时间

为了记录到达与离开时间，我们建立了$data$和$data1$，分别包括了登入时间和登出时间，通过zip将用户名与之建立联系，从而能够在网站中得以显示访问的到达与离开时间。我们这里可显示最近十位访客的到达与离开情况。代码行如下：

```
# login
    ret_set = User.objects.all().values()
    ret_set = [i for i in ret_set if i['last_login'] is not None]
    ret_set = sorted(ret_set, key=lambda i: i['last_login'])
    ten_last_logins = ret_set[:10]
    last_logins_usernames = [i['username'] for i in ret_set]
    last_logins_logintimes = [i['last_login'] for i in ret_set]
    data = zip(last_logins_usernames, last_logins_logintimes)
```

以上代码用于取得前十个用户的用户名与登录时间。

```
    # logout
    ret_set = User.objects.all().values()
    ret_set = [i for i in ret_set if i['date_joined'] is not None]
```

```
ret_set = sorted(ret_set, key=lambda i: i['date_joined'])
ten_last_logout = ret_set[:10]
last_logouts_usernames = [i['username'] for i in ret_set]
last_logouts_logouttimes = [i['date_joined'] for i in ret_set]
data1 = zip(last_logouts_usernames , last_logouts_logouttimes)
```

而以上代码则用于取得前十个用户的用户名与登出时间。

最终网页的达到与离开时间显示效果如 **Figure 7** 所示：

用户名	登陆时间
BlueOrange	July 8, 2019, 4:06 p.m.
Eileen	July 8, 2019, 10:49 p.m.
Hermione	July 8, 2019, 11:04 p.m.
Ron	July 8, 2019, 11:05 p.m.
Hogwarts	July 9, 2019, 1:14 p.m.

用户名	登出时间
BlueOrange	July 8, 2019, 4:06 p.m.
Harry	July 8, 2019, 4:12 p.m.
Eileen	July 8, 2019, 11:01 p.m.
Hermione	July 8, 2019, 11:04 p.m.
Ron	July 8, 2019, 11:05 p.m.
Hogwarts	July 9, 2019, 12:14 a.m.

Figure 7 网页显示效果：到达与离开时间

4.5. 网站访问人数统计

我们在$models.py$中创建了一个用户的类，此后再用到我们采用的$SQLite$数据库只需直接调用该类。在这个类中，我们将访问网站的ip地址和次数、网站访问总次数和单日访问量统计数据记录在$SQLite$数据库中。代码截取如下：

```
###统计人数
from django.utils import timezone
#  访问网站的 ip 地址和次数
class Userip(models.Model):
    ip = models.CharField(verbose_name='IP 地址', max_length=30)   # ip 地址
    count = models.IntegerField(verbose_name='访问次数', default=0)   #  该 ip 访问次数
    class Meta:
        verbose_name = '访问用户信息'
        verbose_name_plural = verbose_name
    def __str__(self):
        return self.ip
#  网站总访问次数
```

```
class VisitNumber(models.Model):
    count = models.IntegerField(verbose_name='网站访问总次数', default=0)   # 网站访问总次数
    class Meta:
        verbose_name = '网站访问总次数'
        verbose_name_plural = verbose_name
    def __str__(self):
        return str(self.count)
# 单日访问量统计
class DayNumber(models.Model):
    day = models.DateField(verbose_name='日期', default=timezone.now)
    count = models.IntegerField(verbose_name='网站访问次数', default=0)   # 网站访问总次数
    class Meta:
        verbose_name = '网站日访问量统计'
        verbose_name_plural = verbose_name
    def __str__(self):
        return str(self.day)
```

打开根目录中的$db.sqlite3$文件，我们可以看到$SQLite$数据库储存的数据如**Figure 8**：

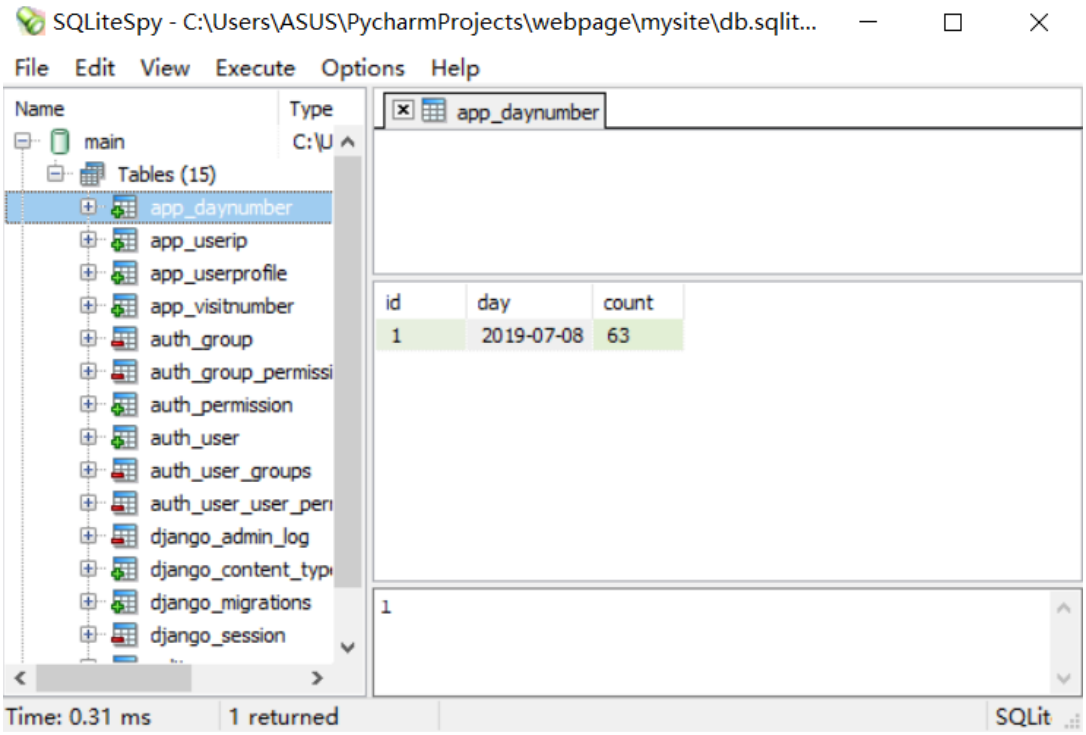

Figure 8 数据库查看访问人数

4.6. 优化功能

4.6.1. 未注册用户登录报错

由于各种现实情况，我们不能够在登录时排除错误输入用户名与密码的情况，因此就不能在数据库中搜索得到对应的数据，可能会造成错误输出，故而我们添加 if 语句进行判断，最后运行结果如 **Figure 9**：

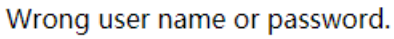

Figure 9 未注册用户登录报错

4.6.2. 已注册用户再次注册跳转登录页面

在现实情况下，常常发生情况如下：已注册用户在较长时间后重新访问网站，却忘记了自己曾经注册过，因而发生了重复注册的情况。我们希望能够对这种情况进行优化，起到提醒用户已注册的作用，因此加入了代码行，实现了已注册用户再次注册跳转登录页面的效果。

代码行与效果如下：

```
if User.objects.filter(username=user_form['username'].data).exists():
    return HttpResponseRedirect(reverse('login'))
```

Figure 10 已注册用户再次登录跳转登录页面

5. 障碍与解决

5.1. 数据库引入

在项目进行前期，我们试图使用的是$MySQL$数据管理系统,在$setting.py$中设置如下：

```
DATABASES = {
    'default': {
        'ENGINE': 'django.db.backends.mysql',
        'NAME': 'mysite',
        'USER': 'root',
        'PASSWORD': 'm',
        'HOST': '127.0.0.1',
```

```
        'PORT': '3306',
    }
}
```

但是由于在首次安装$MYSQL$过程中未保存用户名与密码，卸载重装过程中发现未卸载完全，经过资料查阅发现问题的解决过程过于复杂，因此我们转向了使用轻型的嵌套式数据库$SQLite$，在$setting.py$中设置如下：

```
DATABASES = {
    'default': {
        'ENGINE': 'django.db.backends.sqlite3',
        'NAME': os.path.join(BASE_DIR, 'db.sqlite3'),
    }
}
```

5.2. 图片上传

在简单的文本上传与保存方面，我们遇到的阻碍较小，可以直接存储在数据库等位置。但是对于图片的上传与保存，此类方法就不再适用了。

最终我们选择创建一个$MEDIA_ROOT$，以存放上传的文件。并使用$MEDIA_URL$作为URL的映射。

截取$urls.py$中代码行如下：

```
static(settings.MEDIA_URL, document_root=settings.MEDIA_ROOT)
```

6. 优点与缺点

6.1. 优点

- 实现了通用会议网站的一系列基本功能，能够成功实现图片的上传
- 考虑到注册、登录与登出过程中的一系列可能错误，进行了优化改进与报错

6.2. 缺点

- 由于时间原因，网站设计过于简化，在美工与功能完备化等方面仍然需要改进
- 由于服务器与域名申请的时间跨度问题，该CMS系统暂时只在本机实验
- 采用$Django$框架，过多模块已被封装无法改动，故而一些特定功能的达成较为困难甚至无法实现

7. 全部代码

目录一览

```
——mysite
    ——app
        ——__init__.py
        ——admin.py
        ——apps.py
        ——counting.py
        ——forms.py
        ——models.py
        ——tests.py
        ——urls.py
        ——views.py
    ——media
    ——profile_images
    ——mysite
        ——__init__.py
        ——Settings.py
        ——Urls.py
        ——Wsgi.py
    ——static
    ——templates
    ——app
            ——base.html
            ——index.html
            ——login.html
            ——profile.html
            ——register.html
    ——db.sqlite3
    ——manage.py
```

7.1. app

- **__init__.py**

```
import pymysql
pymysql.install_as_MySQLdb()
```

- **admin.py**

```
from django.contrib import admin
from app.models import UserProfile
```

```
# Register your models here.
admin.register(UserProfile)
```

- **apps.py**

```
from django.apps import AppConfig
class AppConfig(AppConfig):
    name = 'app'
```

- **counting.py**

```
from .models import *
from django.utils import timezone
# 自定义的函数，不是视图
def change_info(request):  # 修改网站访问量和访问 ip 等信息
    # 每一次访问，网站总访问次数加一
    count_nums = VisitNumber.objects.filter(id=1)
    if count_nums:
        count_nums = count_nums[0]
        count_nums.count += 1
    else:
        count_nums = VisitNumber()
        count_nums.count = 1
    count_nums.save()
    # 记录访问 ip 和每个 ip 的次数
    if 'HTTP_X_FORWARDED_FOR' in request.META:  # 获取 ip
        client_ip = request.META['HTTP_X_FORWARDED_FOR']
        client_ip = client_ip.split(",")[0]  # 所以这里是真实的 ip
    else:
        client_ip = request.META['REMOTE_ADDR']  # 这里获得代理 ip
    # print(client_ip)
    ip_exist = Userip.objects.filter(ip=str(client_ip))
    if ip_exist:  # 判断是否存在该 ip
        uobj = ip_exist[0]
        uobj.count += 1
    else:
        uobj = Userip()
        uobj.ip = client_ip
        uobj.count = 1
    uobj.save()
    # 增加今日访问次数
    date = timezone.now().date()
    today = DayNumber.objects.filter(day=date)
    if today:
```

```
            temp = today[0]
            temp.count += 1
        else:
            temp = DayNumber()
            temp.dayTime = date
            temp.count = 1
        temp.save()
```

- **forms.py**

```
from django import forms
from django.contrib.auth.models import User
from app.models import UserProfile
class UserProfileForm(forms.ModelForm):
    class Meta:
        model = UserProfile
        fields = ('desciption', 'picture')
class UserForm(forms.ModelForm):
    password = forms.CharField(widget=forms.PasswordInput())
    class Meta:
        model = User
        fields = ('username', 'email', 'password')
```

- **models.py**

```
from django.db import models
# Create your models here.
from django.db import models
from django.contrib.auth.models import User
# Create your models here.
class UserProfile(models.Model):
    # This line is required. Links UserProfile to a User model instance.
    user = models.OneToOneField(User, on_delete=True)
    # The additional attributes we wish to include.
    desciption = models.CharField(max_length=250)
    picture = models.ImageField(upload_to='profile_images', blank=True)
    # Override the __unicode__() method to return out something meaningful!
    def __str__(self):
        return self.user.username
###统计人数
from django.utils import timezone
#  访问网站的 ip 地址和次数
class Userip(models.Model):
    ip = models.CharField(verbose_name='IP 地址', max_length=30)    # ip 地址
```

```
    count = models.IntegerField(verbose_name='访问次数', default=0)  # 该 ip 访问次数
    class Meta:
        verbose_name = '访问用户信息'
        verbose_name_plural = verbose_name
    def __str__(self):
        return self.ip
# 网站总访问次数
class VisitNumber(models.Model):
    count = models.IntegerField(verbose_name='网站访问总次数', default=0)  # 网站访问总次数
    class Meta:
        verbose_name = '网站访问总次数'
        verbose_name_plural = verbose_name
    def __str__(self):
        return str(self.count)
# 单日访问量统计
class DayNumber(models.Model):
    day = models.DateField(verbose_name='日期', default=timezone.now)
    count = models.IntegerField(verbose_name='网站访问次数', default=0)  # 网站访问总次数
    class Meta:
        verbose_name = '网站日访问量统计'
        verbose_name_plural = verbose_name
    def __str__(self):
        return str(self.day)
```

- **tests.py**

```
from django.test import TestCase
# Create your tests here.
```

- **urls.py**

```
from django.contrib import admin
from django.conf import settings
from django.conf.urls.static import static
from .views import index, register, user_login, restricted, profile, user_logout
from django.urls import path, include
urlpatterns = [
    path('', index, name='index'),
    path('register', register, name='register'),
    path('login', user_login, name='login'),
    path('restricted', restricted, name='restricted'),
    path('logout', user_logout, name='logout'),
```

```
    path(r'profile/<slug:username>', profile, name='profile'),
]+ static(settings.MEDIA_URL, document_root=settings.MEDIA_ROOT)
```

- **views.py**

```
from datetime import timezone
from django.contrib.auth.models import User
from django.shortcuts import render, redirect
# Create your views here.
from django.shortcuts import render
from django.contrib.auth import authenticate, login, logout
from django.http import HttpResponseRedirect, HttpResponse
from django.urls import reverse
from django.contrib.auth.decorators import login_required
from django.contrib.auth import authenticate, login
from django.http import HttpResponseRedirect, HttpResponse
from django.urls import reverse
from datetime import datetime
from app.forms import UserForm, UserProfileForm
from app.models import UserProfile
from .counting import change_info
def index(request):
    # counting visitor
    change_info(request)
    return render(request, 'app/index.html')
def register(request):
    # counting visitor
    change_info(request)
    registered = False
    if request.method == 'POST':
        user_form = UserForm(data=request.POST)
        profile_form = UserProfileForm(data=request.POST)
        if User.objects.filter(username=user_form['username'].data).exists():
            return HttpResponseRedirect(reverse('login'))
        if user_form.is_valid() and profile_form.is_valid():
            user = user_form.save()
            user.set_password(user.password)
            user.save()
            profile = profile_form.save(commit=False)
            profile.user = user
            if 'picture' in request.FILES:
                profile.picture = request.FILES['picture']
                profile.save()
```

```
                registered = True
            else:
                print(user_form.errors, profile_form.errors)
    else:
        user_form = UserForm()
        profile_form = UserProfileForm()
    return render(request,
                  'app/register.html',
                  {'user_form': user_form,
                   'profile_form': profile_form,
                   'registered': registered
                   })
def user_login(request):
    # counting visitor
    change_info(request)
    if request.method == 'POST':
        username = request.POST.get('username')
        password = request.POST.get('password')
        user = authenticate(username=username, password=password)
        request.session['username'] = username
        if user:
            if user.is_active:
                login(request, user)
                user.last_login = datetime.now()
                user.save(update_fields=['last_login'])
                return HttpResponseRedirect(reverse('profile', kwargs={'username':
username}))
            else:
                # An inactive account was used - no logging in!
                return HttpResponse("Your Rango account is disabled.")
        else:
            print("Invalid login details: {0}, {1}".format(username, password))
            return HttpResponse("Wrong user name or password.")
    else:
        return render(request, 'app/login.html', {})
@login_required
def restricted(request):
    change_info(request)
    return HttpResponse("Since you're logged in, you can see this text!")
@login_required
def user_logout(request):
    username = request.session.get('username')
```

```
    if username:
        user = User.objects.get(username=username)
        user.date_joined = datetime.now()
        user.save(update_fields=['date_joined'])
    change_info(request)
    # Since we know the user is logged in, we can now just log them out.
    logout(request)
    # Take the user back to the homepage.
    return redirect(reverse('index'))
@login_required
def profile(request, username):
    # counting visitor
    change_info(request)
    try:
        user = User.objects.get(username=username)
    except User.DoesNotExist:
        return redirect('index')
    userprofile = UserProfile.objects.get_or_create(user=user)[0]
    # login
    ret_set = User.objects.all().values()
    ret_set = [i for i in ret_set if i['last_login'] is not None]
    ret_set = sorted(ret_set, key=lambda i: i['last_login'])
    ten_last_logins = ret_set[:10]
    last_logins_usernames = [i['username'] for i in ret_set]
    last_logins_logintimes = [i['last_login'] for i in ret_set]
    data = zip(last_logins_usernames, last_logins_logintimes)
    # logout
    ret_set = User.objects.all().values()
    ret_set = [i for i in ret_set if i['date_joined'] is not None]
    ret_set = sorted(ret_set, key=lambda i: i['date_joined'])
    ten_last_logout = ret_set[:10]
    last_logouts_usernames = [i['username'] for i in ret_set]
    last_logouts_logouttimes = [i['date_joined'] for i in ret_set]
    data1 = zip(last_logouts_usernames , last_logouts_logouttimes)
    form = UserProfileForm(
        {'desciption': userprofile.desciption, 'picture': userprofile.picture} )
    return render(request, 'app/profile.html', {'userprofile': userprofile, 'selecteduser': user, 'form':
form , 'data':data ,'data1':data1})
```

7.2. mysite

- **__init__.py**

空

- **setting.py**

```
"""
Django settings for mysite project.
Generated by 'django-admin startproject' using Django 2.0.9.
For more information on this file, see
https://docs.djangoproject.com/en/2.0/topics/settings/
For the full list of settings and their values, see
https://docs.djangoproject.com/en/2.0/ref/settings/
"""
import os
# Build paths inside the project like this: os.path.join(BASE_DIR, ...)
# Quick-start development settings - unsuitable for production
# See https://docs.djangoproject.com/en/2.0/howto/deployment/checklist/
# SECURITY WARNING: keep the secret key used in production secret!
SECRET_KEY = '&xd0l4*rpq$qt4l^q%=zi^@5=r1!ubs%f%fc!$a#5&7v59fem6'
# SECURITY WARNING: don't run with debug turned on in production!
DEBUG = True
BASE_DIR = os.path.dirname(os.path.dirname(os.path.abspath(__file__)))
TEMPLATE_DIR = os.path.join(BASE_DIR, 'templates')
STATIC_DIR = os.path.join(BASE_DIR, 'static')
MEDIA_DIR = os.path.join(BASE_DIR, 'media')
ALLOWED_HOSTS = ['*']
# Application definition
INSTALLED_APPS = [
    'django.contrib.admin',
    'django.contrib.auth',
    'django.contrib.contenttypes',
    'django.contrib.sessions',
    'django.contrib.messages',
    'django.contrib.staticfiles',
    # 'app.apps.AppConfig',
    'app',
]
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',
]
ROOT_URLCONF = 'mysite.urls'
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',
            ],
        },
    },
]
WSGI_APPLICATION = 'mysite.wsgi.application'
# Database
# https://docs.djangoproject.com/en/2.0/ref/settings/#databases
DATABASES = {
    'default': {
        'ENGINE': 'django.db.backends.sqlite3',
        'NAME': os.path.join(BASE_DIR, 'db.sqlite3'),
    }
}
# Password validation
# https://docs.djangoproject.com/en/2.0/ref/settings/#auth-password-validators
AUTH_PASSWORD_VALIDATORS = [
    {
        'NAME': 'django.contrib.auth.password_validation.UserAttributeSimilarityValidator',
    },
    {
        'NAME': 'django.contrib.auth.password_validation.MinimumLengthValidator',
        'OPTIONS': {'min_length': 6, }
    },
```

```
    {
        'NAME': 'django.contrib.auth.password_validation.CommonPasswordValidator',
    },
    {
        'NAME': 'django.contrib.auth.password_validation.NumericPasswordValidator',
    },
]
# Internationalization
# https://docs.djangoproject.com/en/2.0/topics/i18n/
LANGUAGE_CODE = 'en-us'
TIME_ZONE = 'UTC'
USE_I18N = True
USE_L10N = True
USE_TZ = True
# Static files (CSS, JavaScript, Images)
# https://docs.djangoproject.com/en/2.0/howto/static-files/
# STATIC_URL = '/static/'
# MEDIA_DIR = os.path.join(BASE_DIR, 'media')
#
# STATIC_DIR = os.path.join(BASE_DIR, 'static')
# MEDIA_URL = '/media/'
# PASSWORD_HASHERS = (
# 'django.contrib.auth.hashers.PBKDF2PasswordHasher',
'django.contrib.auth.hashers.PBKDF2SHA1PasswordHasher',)
# MEDIA_ROOT = MEDIA_DIR
STATICFILES_DIRS = [
    STATIC_DIR,
]
STATIC_URL = '/static/'
# MEDIA_ROOT = MEDIA_DIR
MEDIA_ROOT = os.path.join(BASE_DIR, 'media').replace('\\', '/')
MEDIA_URL = '/media/'
```

- **urls.py**

```
"""mysite URL Configuration
The `urlpatterns` list routes URLs to views. For more information please see:
    https://docs.djangoproject.com/en/2.0/topics/http/urls/
Examples:
Function views
    1. Add an import:  from my_app import views
    2. Add a URL to urlpatterns:  path('', views.home, name='home')
Class-based views
```

```
    1. Add an import:  from other_app.views import Home
    2. Add a URL to urlpatterns:  path('', Home.as_view(), name='home')
Including another URLconf
    1. Import the include() function: from django.urls import include, path
    2. Add a URL to urlpatterns:  path('blog/', include('blog.urls'))
"""
from django.contrib import admin
from django.urls import path, include
from django.conf import settings
from django.conf.urls.static import static
urlpatterns = [
    path('admin/', admin.site.urls),
    path('', include('app.urls')),
] + static(settings.MEDIA_URL, document_root=settings.MEDIA_ROOT)
```

- **wsgi.py**

```
"""
WSGI config for mysite project.
It exposes the WSGI callable as a module-level variable named ``application``.
For more information on this file, see
https://docs.djangoproject.com/en/2.0/howto/deployment/wsgi/
"""
import os
from django.core.wsgi import get_wsgi_application
os.environ.setdefault("DJANGO_SETTINGS_MODULE", "mysite.settings")
application = get_wsgi_application()
```

7.3. Templates

- **base.html**

```
<!DOCTYPE html>
    {% load staticfiles %}
    <html>
    <head>
        <title>
            {% block title_block_ %}
            {% endblock %}
        </title>
    </head>
    <body>
        <div>
        </div>
```

```
        <div>
            {% block body_block %}
            {% endblock %}
        </div>
        <hr />
        <div>
        </div>
    </body>
    </html>
```

- **index.html**

```
<html>
    <head>
    </head>
    <body>
        {% if user.is_authenticated %}
    welcome {{ user.username }}!
            <a href="{% url 'logout' %}">Logout</a>
        {% else %}
        <a href="{% url 'login' %}">login</a>
            <a href="{% url 'register' %}">register</a>
        {% endif %}
    </body>
</html>
```

- **login.html**

```
    {% extends 'app/base.html' %}
    {% load staticfiles %}
    {% block title_block %}
        Login
    {% endblock %}
    {% block body_block %}
    <h1>Login to Rango</h1>
    <form id="login_form" method="post" action="{% url 'login' %}">
        {% csrf_token %}
        Username: <input type="text" name="username" value="" size="50" />
        <br />
        Password: <input type="password" name="password" value="" size="50" />
        <br />
        <input type="submit" value="submit" />
    </form>
    {% endblock %}
```

- **profile.html**

```
{% extends 'app/base.html' %}
{% load staticfiles %}
{% block title %}{{ user.username }} Profile{% endblock %}
{% block body_block %}
    <div class="page-header">
        <h1>{{selecteduser.username}} Profile</h1>
        <h1>{{ MEDIA_URL }}</h1>
        <h1>{{ MEDIA_URL }}</h1>
    </div>
    {% if userprofile.picture %}
    <img src="/media/{{userprofile.picture }}" width="300" height="300"
alt="{{user.username}}" />
    {% else %}
    <h1>you don' have profile</h1>
    {% endif %}
    <br/><br/>
    <hr>
    <h4>desciption</h4>
        {{ form.desciption }}
    <hr>
    <table>
        <tr>
            <td>用户名</td>
            <td>登陆时间</td>
        </tr>
          {% for login_time,username in data%}
        <tr>
            <td>
                {{ login_time }}
            </td>
              <td>
                {{ username }}
            </td>
        </tr>
        {% endfor %}
    </table>
    <hr>
    <table>
        <tr>
            <td>用户名</td>
```

```
            <td>登出时间</td>
        </tr>
    {% for logout_time,username in data1 %}
        <tr>
            <td>
                {{ logout_time }}
            </td>
             <td>
                {{ username }}
            </td>
        </tr>
        {% endfor %}
    </table>
    <a href="{% url 'logout' %}">Logout</a>
{% endblock %}
```

- **register.html**

```
{% extends 'app/base.html' %}
{% load staticfiles %}
{% block title_block %}
    Register
{% endblock %}
{% block body_block %}
    <h1>***Registration***</h1>
    {% if registered %}
        <strong>thank you for registering!</strong>
        <a href="{% url 'index' %}">Return to the homepage.</a><br />
    {% else %}
      <strong>register here!</strong><br />
        <form id="user_form" method="post" action="{% url 'register' %}"
                enctype="multipart/form-data">
        {% csrf_token %}
        <!-- Display each form    -->
        {{ user_form.as_p }}
        {{ profile_form.as_p }}
        <!-- Provide a button to click to submit the form. -->
        <input type="submit" name="submit" value="Register" />
    </form>
    {% endif %}
{% endblock %}
```

7.4. manage.py

```
#!/usr/bin/env python
import os
import sys
if __name__ == "__main__":
    os.environ.setdefault("DJANGO_SETTINGS_MODULE", "mysite.settings")
    try:
        from django.core.management import execute_from_command_line
    except ImportError as exc:
        raise ImportError(
            "Couldn't import Django. Are you sure it's installed and "
            "available on your PYTHONPATH environment variable? Did you "
            "forget to activate a virtual environment?"
        ) from exc
    execute_from_command_line(sys.argv)
```