

一，创建项目

1.1 创建一个项目和应用

```
django-admin startproject user_project
```

```
cd cd user_project
```

```
python manage.py startapp user
```

```
F:\Python\Django\code>workon drfw
(drfw) F:\Python\Django\code>django-admin startproject user_project

(drfw) F:\Python\Django\code>cd user_project

(drfw) F:\Python\Django\code\user_project>python manage.py startapp user

(drfw) F:\Python\Django\code\user_project>_
```

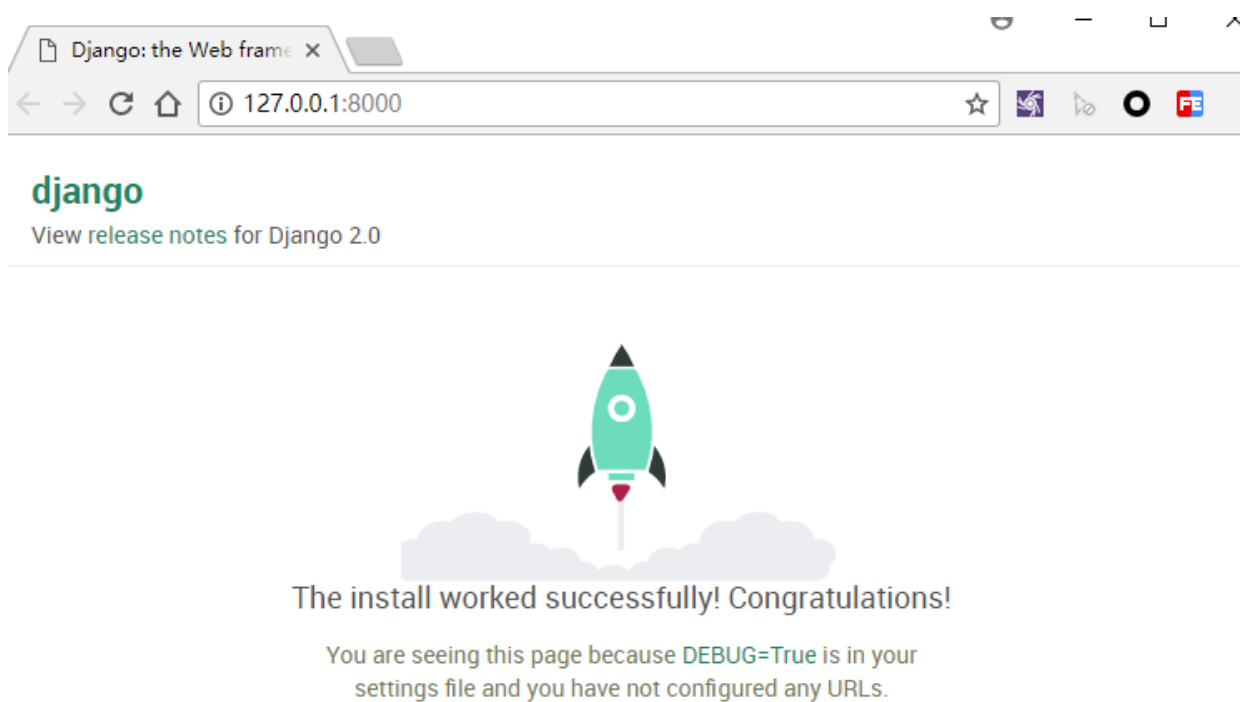
1.2 运行测试

```
python manage.py runserver
```

```
(drfw) F:\Python\Django\code\user_project>python manage.py runserver
Performing system checks...

System check identified no issues (0 silenced).

You have 14 unapplied migration(s). Your project may not work properly until you apply the migrations for app(s): admin, auth, co
s.
Run 'python manage.py migrate' to apply them.
April 10, 2018 - 16:56:09
Django version 2.0.3, using settings 'user_project.settings'
Starting development server at http://127.0.0.1:8000/
Quit the server with CTRL-BREAK.
[10/Apr/2018 16:56:18] "GET / HTTP/1.0" 200 16348
```



二，设置数据模型

2.1.数据库模型设计

作为一个用户登录和注册项目，需要保存的都是各种用户的相关信息。很显然，我们至少需要一张用户表User，在用户表里需要保存下面的信息：

- 用户名

- 密码
- 邮箱地址
- 性别
- 创建时间

进入user/models.py, 代码如下:

```
from django.db import models

class User(models.Model):
    "用户表"
    gender = (
        ('male', '男'),
        ('female', '女'),
    )
    name = models.CharField(max_length=128, unique=True)
    #unique=True, 表示唯一不可重复, 默认为False
    password = models.CharField(max_length=256)
    email = models.EmailField(unique=True)
    sex = models.CharField(max_length=32, choices=gender, default='男')
    c_time = models.DateTimeField(auto_now_add=True)

    def __str__(self):
        return self.name

    class Meta:
        ordering = ['-c_time']
        verbose_name = '用户'
        verbose_name_plural = '用户'
```

各字段含义:

- name必填, 最长不超过128个字符, 并且唯一, 也就是不能有相同姓名;
- password必填, 最长不超过256个字符 (实际可能不需要这么长);
- email使用Django内置的邮箱类型, 并且唯一;
- 性别使用了一个choice, 只能选择男或者女, 默认为男;
- 使用__str__帮助人性化显示对象信息;
- 元数据里定义用户按创建时间的反序排列, 也就是最近的最先显示;

注意: 这里的用户名指的是网络上注册的用户名, 不要等同于现实中的真实姓名, 所以采用了唯一机制。如果是现实中可以重复的人名, 那肯定是不能设置unique的。

2.2. 设置数据库为Mysql

在settings.py修改

```
DATABASES = {
    'default': {
        'ENGINE': 'django.db.backends.mysql', # 更改为mysql引擎
        'NAME': 'user', # 数据库名字
        'USER': 'root', # 用户名
        'PASSWORD': 'admin', # 密码
        'HOST': '127.0.0.1', # ip
        'PORT': 3306, # 端口
    }
}
```

init.py里面导入pymysql模块

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

2.3.数据库迁移

注册app

```
INSTALLED_APPS = [
    'django.contrib.admin',
    'django.contrib.auth',
    'django.contrib.contenttypes',
    'django.contrib.sessions',
    'django.contrib.messages',
    'django.contrib.staticfiles',
    'user', # 注册userapp
]
```

进入数据库，创建表user

```
(drfw) F:\Python\Django\code\user_project>mysql -uroot -p
Enter password: *****
Welcome to the MySQL monitor.  Commands end with ; or \g.
Your MySQL connection id is 9
Server version: 5.7.21-log MySQL Community Server (GPL)

Copyright (c) 2000, 2018, Oracle and/or its affiliates. All rights reserved.

Oracle is a registered trademark of Oracle Corporation and/or its
affiliates. Other names may be trademarks of their respective
owners.

Type 'help;' or '\h' for help. Type '\c' to clear the current input statement.

mysql> show databases;

+-----+
| Database |
+-----+
| information_schema |
| mxshop      |
| mysql       |
| performance_schema |
| sakila      |
| sys         |
| world       |
+-----+
7 rows in set (0.00 sec)

mysql> create database user charset='utf8';
Query OK, 1 row affected (0.00 sec)
```

第一步

第二步

第三步

迁移到数据库

python manage.py makemigrations

python manage.py migrate

```
(drfw) F:\Python\Django\code\user_project>python manage.py makemigrations
Migrations for 'user':
  user\migrations\0001_initial.py
    - Create model User

(drwf) F:\Python\Django\code\user_project>python manage.py migrate
```

生成迁移脚本

执行迁移

```
Operations to perform:
  Apply all migrations: admin, auth, contenttypes, sessions, user
Running migrations:
  Applying contenttypes.0001_initial... OK
  Applying auth.0001_initial... OK
  Applying admin.0001_initial... OK
  Applying admin.0002_logentry_remove_auto_add... OK
  Applying contenttypes.0002_remove_content_type_name... OK
  Applying auth.0002_alter_permission_name_max_length... OK
  Applying auth.0003_alter_user_email_max_length... OK
  Applying auth.0004_alter_user_username_opts... OK
  Applying auth.0005_alter_user_last_login_null... OK
  Applying auth.0006_require_contenttypes_0002... OK
  Applying auth.0007_alter_validators_add_error_messages... OK
  Applying auth.0008_alter_user_username_max_length... OK
  Applying auth.0009_alter_user_last_name_max_length... OK
  Applying sessions.0001_initial... OK
  Applying user.0001_initial... OK
```

三、admin后台

3.1.在admin中注册模型

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

# Register your models here.
admin.site.register(models.User)
```

3.2.创建超级管理员

python manage.py createsuperuser

```
F:\Python\Django\code\user_project>python manage.py createsuperuser
System check identified some issues:

WARNINGS:
?: (1_7.W001) MIDDLEWARE_CLASSES is not set.
  HINT: Django 1.7 changed the global defaults for the MIDDLEWARE_CLASSES. django.contrib.sessions.middleware.SessionMiddleware, django.contrib.auth.middleware.AuthenticationMiddleware, and django.contrib.messages.middleware.MessageMiddleware were removed from the default MIDDLEWARE_CLASSES. If you need these middleware then you should configure this setting.

Username (leave blank to use 'admin'): admin
Email address:
Password:
Password (again):
Superuser created successfully.
```

输入用户名和密码: admin admin

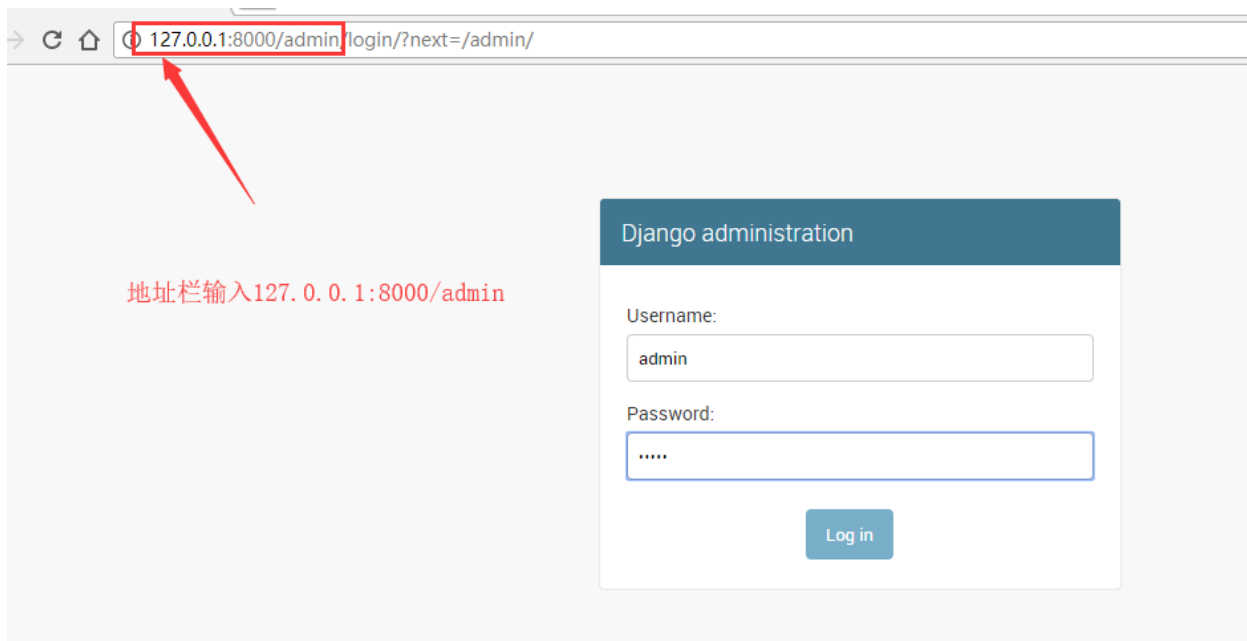
运行并登录

python manage.py runserver

```
(drfw) F:\Python\Django\code\user_project>python manage.py runserver
Performing system checks...

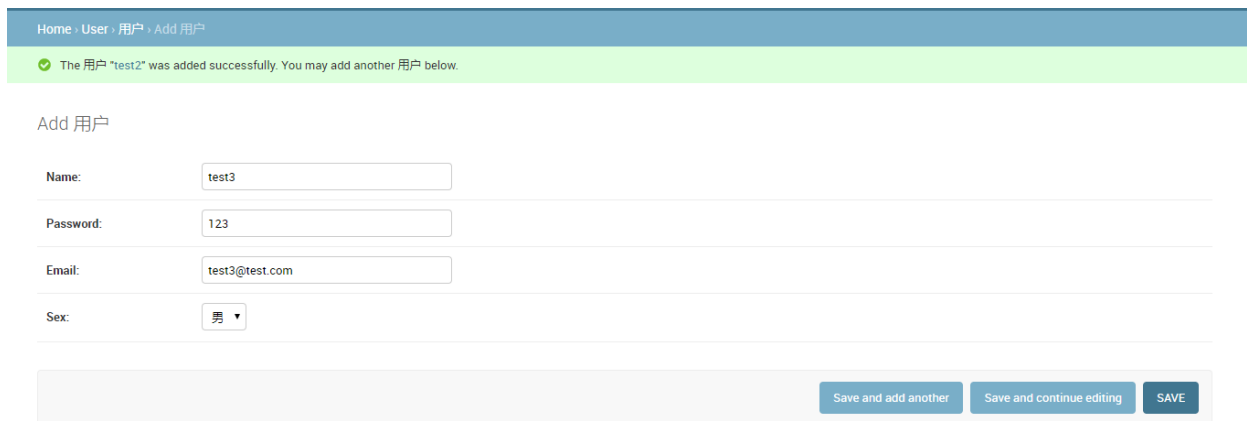
System check identified no issues (0 silenced).
April 11, 2018 - 11:33:52
Django version 2.0.3, using settings 'user_project.settings'
Starting development server at http://127.0.0.1:8000/
Quit the server with CTRL-BREAK.
[11/Apr/2018 11:33:56] "GET /admin HTTP/1.0" 301 0
[11/Apr/2018 11:33:56] "GET /admin/ HTTP/1.0" 302 0
[11/Apr/2018 11:33:56] "GET /admin/login/?next=/admin/ HTTP/1.0" 200 1855
[11/Apr/2018 11:33:56] "GET /static/admin/css/login.css HTTP/1.0" 200 1203
[11/Apr/2018 11:33:56] "GET /static/admin/css/base.css HTTP/1.0" 200 16106
[11/Apr/2018 11:33:56] "GET /static/admin/css/responsive.css HTTP/1.0" 200 17894
Not Found: /favicon.ico
[11/Apr/2018 11:33:57] "GET /favicon.ico HTTP/1.0" 404 1978
[11/Apr/2018 11:35:08] "POST /admin/login/?next=/admin/ HTTP/1.0" 302 0
[11/Apr/2018 11:35:08] "GET /admin/ HTTP/1.0" 200 3603
[11/Apr/2018 11:35:08] "GET /static/admin/css/dashboard.css HTTP/1.0" 200 412
[11/Apr/2018 11:35:09] "GET /static/admin/img/icon-changelink.svg HTTP/1.0" 200 380
[11/Apr/2018 11:35:09] "GET /static/admin/img/icon-addlink.svg HTTP/1.0" 200 331
```

地址栏输入127.0.0.1:8000/admin

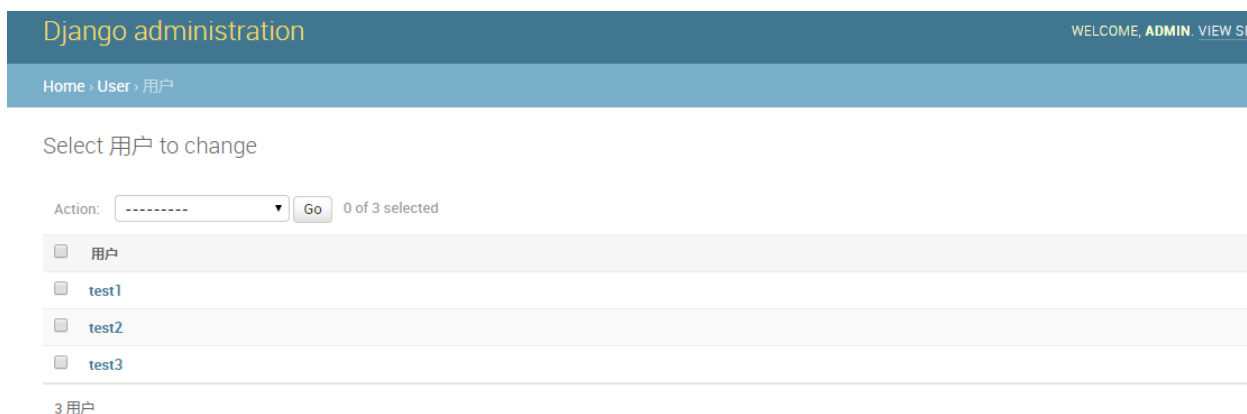


然后再增加几个测试用户

1,



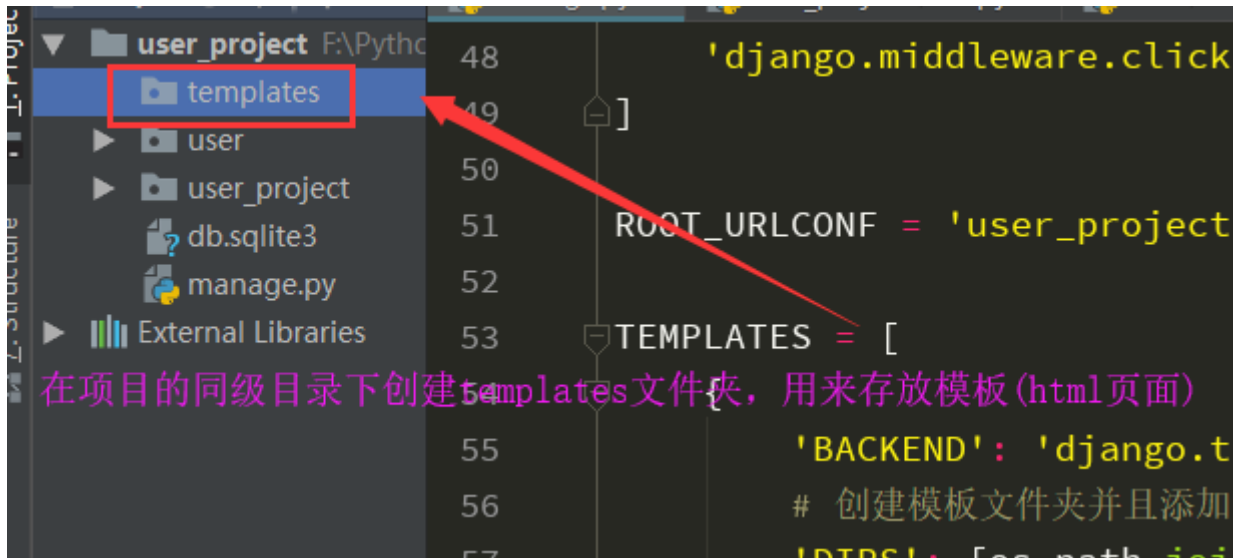
2,



四、url路由和视图

前面我们已经创建好数据模型了，并且在admin后台中添加了一些测试用户。下面我们就要设计好站点的url路由、对应的处理视图函数以及使用的前端模板了。

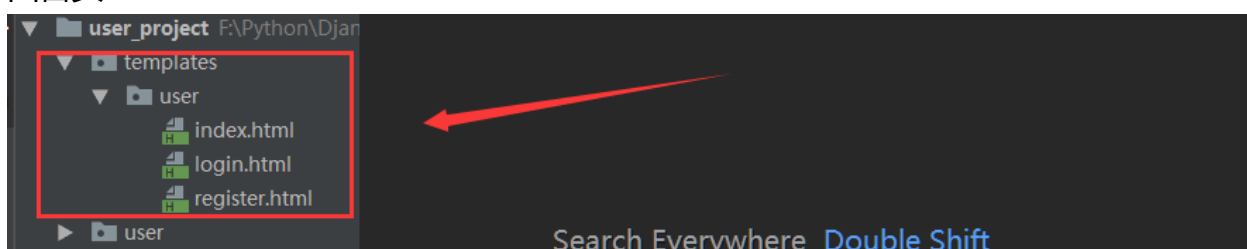
4.1 创建templates文件夹，并添加到根路径



修改user_project/setting.py



在templates/user文件夹下创建三个空的html页面(名字跟路由设计的对应得上)ps:注销返回首页



4.2 视图设计

```

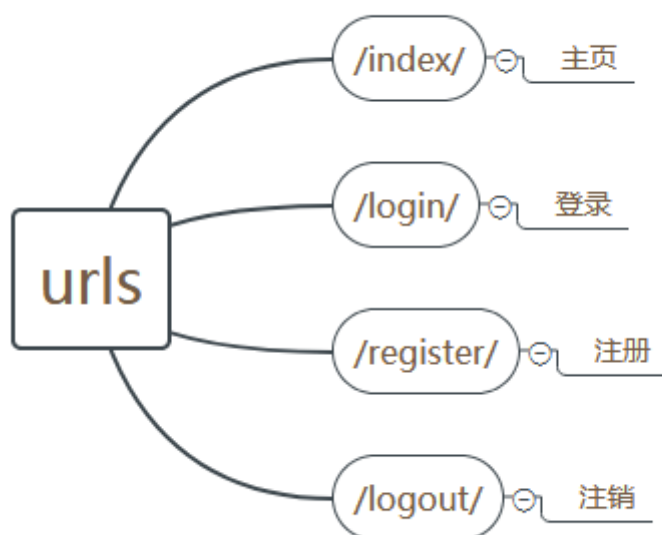
1  from django.shortcuts import render, redirect
2  # Create your views here.
3  def index(request):
4      pass
5      return render(request, 'user/index.html')
6  def register(request):
7      pass
8      return render(request, 'user/register.html')
9  def login(request):
10     pass
11     return render(request, 'user/login.html')
12 def logout(request):
13     pass
14     return redirect(request, 'user/index.html')

```

我们先不着急完成视图内部的具体细节，而是把框架先搭建起来。

4.3.路由设计

初步设想需要下面的四个URL：



考虑到登录系统属于站点的一级功能，为了直观和更易于接受，这里没有采用二级路由的方式，而是在根路由下直接编写路由条目，同样也没有使用反向解析名（name参数）。

1, 在应用下创建一个urls.py文件

user/urls.py

```

from django.urls import path
from . import views
urlpatterns = [
    path('index/', views.index),
    path('register/', views.register),
    path('login/', views.login),
    path('logout/', views.logout),
]

```

2, 在项目下的根路由urls.py添加应用的子路由

user_project/urls.py

```

from django.contrib import admin
from django.urls import path, include

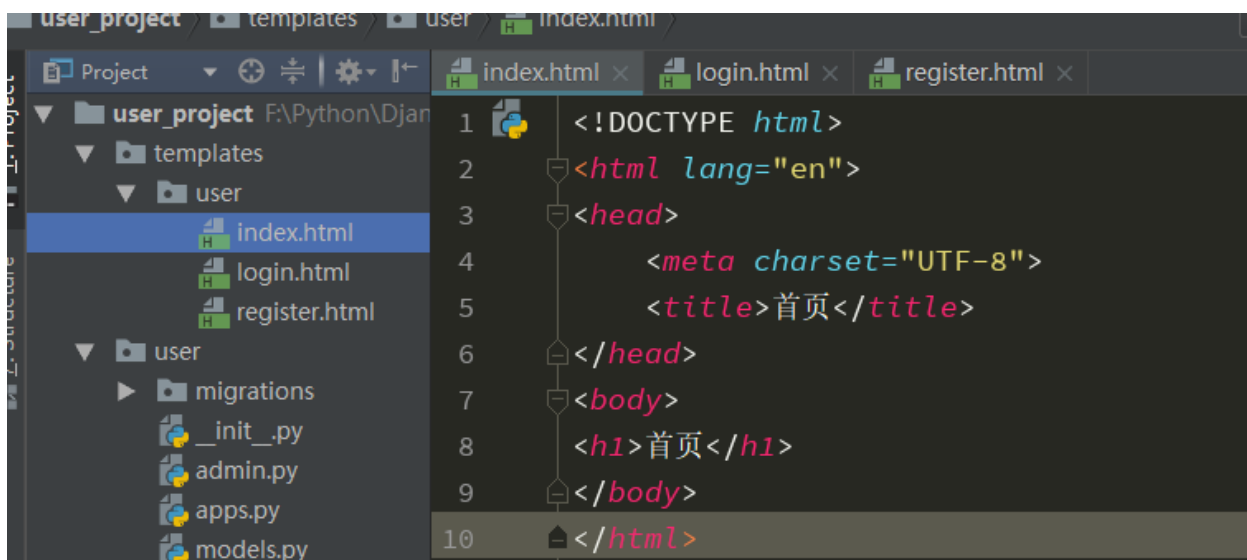
urlpatterns = [
    path('admin/', admin.site.urls),
    path('', include('user.urls'))
]

```

4.4.修改HTML页面文件

在templates目录中的三个文件index.html、login.html以及register.html，并写入如下的代码：

index.html

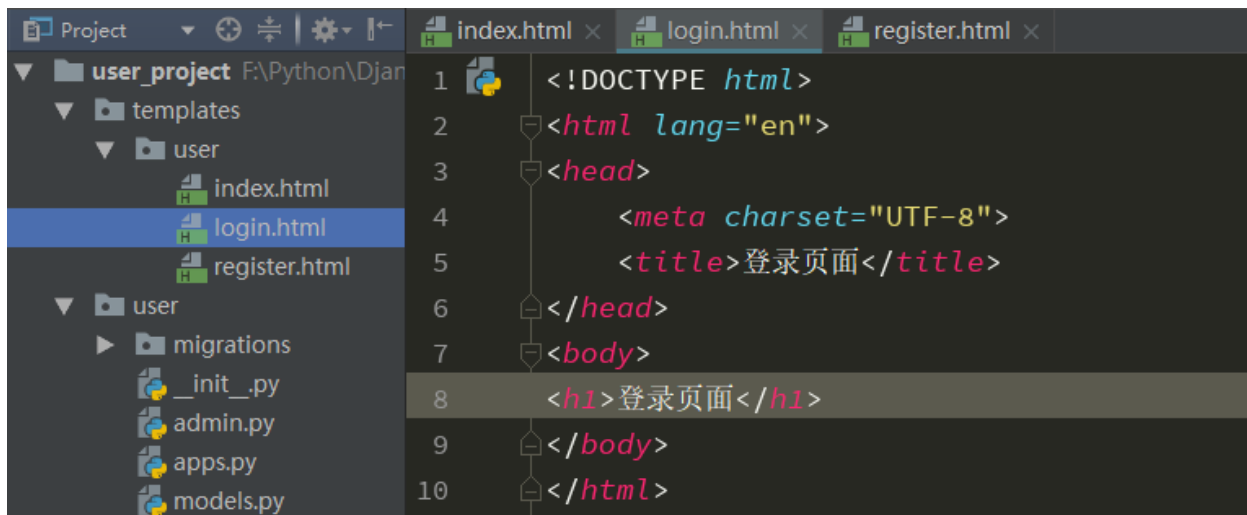


```

1 <!DOCTYPE html>
2 <html lang="en">
3 <head>
4     <meta charset="UTF-8">
5     <title>首页</title>
6 </head>
7 <body>
8     <h1>首页</h1>
9 </body>
10 </html>

```

login.html



```
1 <!DOCTYPE html>
2 <html lang="en">
3 <head>
4     <meta charset="UTF-8">
5     <title>登录页面</title>
6 </head>
7 <body>
8     <h1>登录页面</h1>
9 </body>
10 </html>
```

register.html



```
1 <!DOCTYPE html>
2 <html lang="en">
3 <head>
4     <meta charset="UTF-8">
5     <title>注册</title>
6 </head>
7 <body>
8     <h1>注册页面</h1>
9 </body>
10 </html>
```

五、前端页面设计

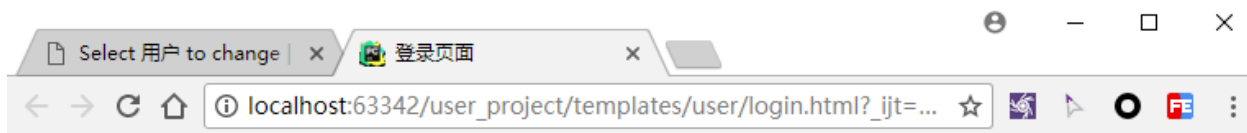
5.1.原生HTML页面

login.html文件中的内容，写入下面的代码：



```
<!DOCTYPE html>
<html lang="en">
<head>
    <meta charset="UTF-8">
    <title>登录页面</title>
</head>
<body>
<div style="margin: 15% 40%">
    <h2>欢迎登录! </h2>
    <form action="/login/" method="post">
        <p>
            <label for="id_username">用户名: </label>
            <input type="text" id="id_username" name="username" placeholder="用户名" autofocus required/>
        </p>
        <p>
            <label for="id_password">密码: </label>
            <input type="text" id="id_password" placeholder="密码" name="password" required>
        </p>
        <input type="submit" value="登录">
    </form>
</div>
</body>
</html>
```

可以看到如下图的页面：



欢迎登录!

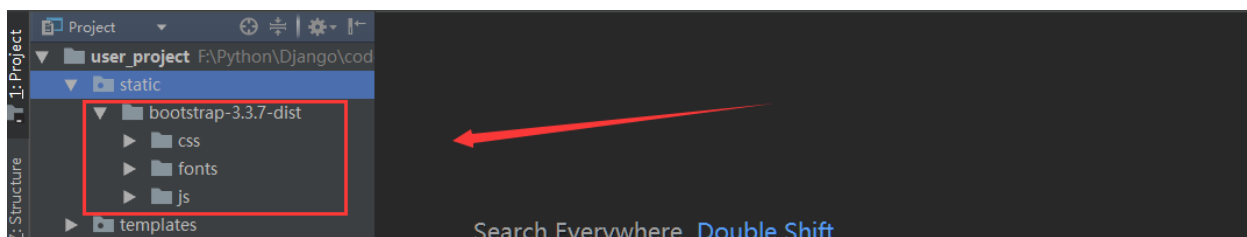
用户名:

密码:

5.2.引入Bootstrap

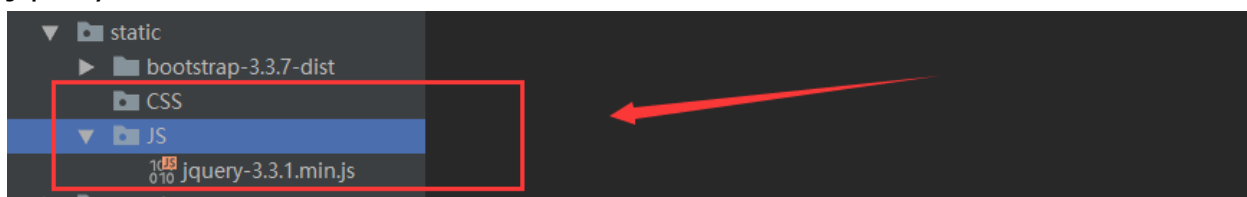
[Bootstrap3.3.7下载地址](#)

根目录下新建一个static目录，并将解压后的bootstrap-3.3.7-dist目录，整体拷贝到static目录中，如下图所示：

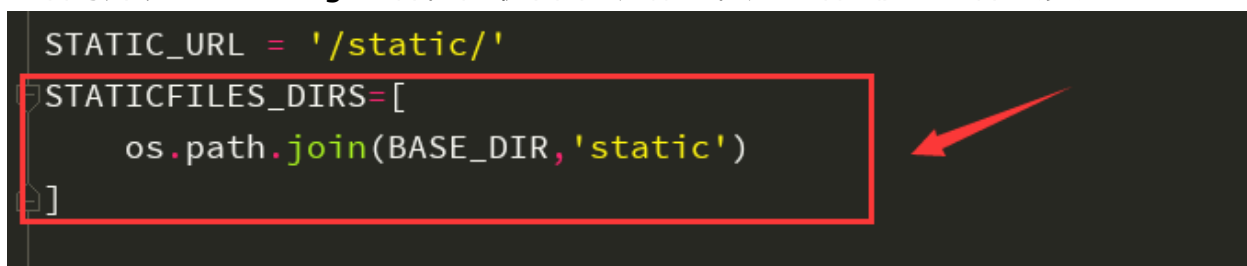


由于Bootstrap依赖jQuery，所以我们需要提前下载并引入jQuery：[下载地址](#)

在static目录下，新建一个css和js目录，作为以后的样式文件和js文件的存放地，将我们的jquery文件拷贝到static/js目录下。



然后打开项目的settings文件，在最下面添加配置，用于指定静态文件的搜索目录：



5.3.创建base.html模板

既然要将前端页面做得像个样子，那么就不能和前面一样，每个页面都各写各的，单打独斗。一个网站有自己的统一风格和公用部分，可以把这部分内容集中到一个基础模板

base.html中。现在，在根目录下的templates中新建一个base.html文件用作站点的基础模板。

在Bootstrap文档中，为我们提供了一个非常简单而又实用的基本模板，代码如下：

```
<!DOCTYPE html>
<html lang="zh-CN">
  <head>
    <meta charset="utf-8">
    <meta http-equiv="X-UA-Compatible" content="IE=edge">
    <meta name="viewport" content="width=device-width, initial-scale=1">
    <!-- 上述3个meta标签*必须*放在最前面，任何其他内容都*必须*跟随其后！ -->
    <title>Bootstrap 101 Template</title>

    <!-- Bootstrap -->
    <link href="css/bootstrap.min.css" rel="stylesheet">

    <!-- HTML5 shim and Respond.js for IE8 support of HTML5 elements and media queries -->
    <!-- WARNING: Respond.js doesn't work if you view the page via file:// -->
    <!--[if lt IE 9]>
      <script src="https://cdn.bootcss.com/html5shiv/3.7.3/html5shiv.min.js"></script>
      <script src="https://cdn.bootcss.com/respond.js/1.4.2/respond.min.js"></script>
    <![endif]-->
  </head>
  <body>
    <h1>你好，世界！</h1>

    <!-- jQuery (necessary for Bootstrap's JavaScript plugins) -->
    <script src="https://cdn.bootcss.com/jquery/1.12.4/jquery.min.js"></script>
    <!-- Include all compiled plugins (below), or include individual files as needed -->
    <script src="js/bootstrap.min.js"></script>
  </body>
</html>
```

将它整体拷贝到base.html文件中。

5.4.创建页面导航条

Bootstrap提供了现成的导航条组件

```
<nav class="navbar navbar-default">
  <div class="container-fluid">
    <!-- Brand and toggle get grouped for better mobile display -->
    <div class="navbar-header">
      <button type="button" class="navbar-toggle collapsed" data-toggle="collapse"
data-target="#bs-example-navbar-collapse-1" aria-expanded="false">
        <span class="sr-only">Toggle navigation</span>
        <span class="icon-bar"></span>
        <span class="icon-bar"></span>
        <span class="icon-bar"></span>
      </button>
      <a class="navbar-brand" href="#">Brand</a>
    </div>
```

```

<!-- Collect the nav links, forms, and other content for toggling -->
<div class="collapse navbar-collapse" id="bs-example-navbar-collapse-1">
  <ul class="nav navbar-nav">
    <li class="active"><a href="#">Link <span class="sr-only">(current)</span></a>
  </li>
    <li><a href="#">Link</a></li>
    <li class="dropdown">
      <a href="#" class="dropdown-toggle" data-toggle="dropdown" role="button"
aria-haspopup="true" aria-expanded="false">Dropdown <span class="caret"></span>
    </a>
      <ul class="dropdown-menu">
        <li><a href="#">Action</a></li>
        <li><a href="#">Another action</a></li>
        <li><a href="#">Something else here</a></li>
        <li role="separator" class="divider"></li>
        <li><a href="#">Separated link</a></li>
        <li role="separator" class="divider"></li>
        <li><a href="#">One more separated link</a></li>
      </ul>
    </li>
  </ul>
  <form class="navbar-form navbar-left">
    <div class="form-group">
      <input type="text" class="form-control" placeholder="Search">
    </div>
    <button type="submit" class="btn btn-default">Submit</button>
  </form>
  <ul class="nav navbar-nav navbar-right">
    <li><a href="#">Link</a></li>
    <li class="dropdown">
      <a href="#" class="dropdown-toggle" data-toggle="dropdown" role="button"
aria-haspopup="true" aria-expanded="false">Dropdown <span class="caret"></span>
    </a>
      <ul class="dropdown-menu">
        <li><a href="#">Action</a></li>
        <li><a href="#">Another action</a></li>
        <li><a href="#">Something else here</a></li>
        <li role="separator" class="divider"></li>
        <li><a href="#">Separated link</a></li>
      </ul>
    </li>
  </ul>
</div><!-- /.navbar-collapse -->
</div><!-- /.container-fluid -->
</nav>

```

实例：

Brand	Link	Link	Dropdown ▼	<input type="text" value="Search"/>	<input type="submit" value="Submit"/>	Link	Dropdown ▼
-------	------	------	------------	-------------------------------------	---------------------------------------	------	------------

其中有一些部分，比如搜索框是我们目前还不需要的，需要将多余的内容裁剪掉。同时，有一些名称和url地址等需要按我们的实际内容修改。最终导航条的代码如下：

```
<nav class="navbar navbar-default">
  <div class="container-fluid">
    <!-- Brand and toggle get grouped for better mobile display -->
    <div class="navbar-header">
      <button type="button" class="navbar-toggle collapsed" data-toggle="collapse"
data-target="#my-nav" aria-expanded="false">
        <span class="sr-only">切换导航条</span>
        <span class="icon-bar"></span>
        <span class="icon-bar"></span>
        <span class="icon-bar"></span>
      </button>
      <a class="navbar-brand" href="#">Mysite</a>
    </div>

    <!-- Collect the nav links, forms, and other content for toggling -->
    <div class="collapse navbar-collapse" id="my-nav">
      <ul class="nav navbar-nav">
        <li class="active"><a href="/index/">主页</a></li>
      </ul>
      <ul class="nav navbar-nav navbar-right">
        <li><a href="/login/">登录</a></li>
        <li><a href="/register/">注册</a></li>
      </ul>
    </div> <!-- /.navbar-collapse -->
  </div> <!-- /.container-fluid -->
</nav>
```

5.5.使用Bootstrap静态文件

{% static '相对路径' %} 这个Django为我们提供的静态文件加载方法，可以将页面与静态文件链接起来

最后，base.html内容如下：

```
{% load staticfiles %}

<!DOCTYPE html>
<html lang="zh-CN">
  <head>
    <meta charset="utf-8">
    <meta http-equiv="X-UA-Compatible" content="IE=edge">
    <meta name="viewport" content="width=device-width, initial-scale=1">
    <!-- 上述3个meta标签*必须*放在最前面，任何其他内容都*必须*跟随其后！ -->
    <title>{% block title %}base{% endblock %}</title>

    <!-- Bootstrap -->
    <link href="{% static 'bootstrap-3.3.7-dist/css/bootstrap.min.css' %}" rel="stylesheet">

    <!-- HTML5 shim and Respond.js for IE8 support of HTML5 elements and media queries -
-->
    <!-- WARNING: Respond.js doesn't work if you view the page via file:// -->
```

```

<!--[if lt IE 9]>
  <script src="https://cdn.bootcss.com/html5shiv/3.7.3/html5shiv.min.js"></script>
  <script src="https://cdn.bootcss.com/respond.js/1.4.2/respond.min.js"></script>
<![endif]-->
{% block css %}{% endblock %}
</head>
<body>
  <nav class="navbar navbar-default">
    <div class="container-fluid">
      <!-- Brand and toggle get grouped for better mobile display -->
      <div class="navbar-header">
        <button type="button" class="navbar-toggle collapsed" data-toggle="collapse"
data-target="#my-nav" aria-expanded="false">
          <span class="sr-only">切换导航条</span>
          <span class="icon-bar"></span>
          <span class="icon-bar"></span>
          <span class="icon-bar"></span>
        </button>
        <a class="navbar-brand" href="#">Mysite</a>
      </div>

      <!-- Collect the nav links, forms, and other content for toggling -->
      <div class="collapse navbar-collapse" id="my-nav">
        <ul class="nav navbar-nav">
          <li class="active"><a href="/index/">主页</a></li>
        </ul>
        <ul class="nav navbar-nav navbar-right">
          <li><a href="/login/">登录</a></li>
          <li><a href="/register/">注册</a></li>
        </ul>
      </div><!-- /.navbar-collapse -->
    </div><!-- /.container-fluid -->
  </nav>

  {% block content %}{% endblock %}

  <!-- jQuery (necessary for Bootstrap's JavaScript plugins) -->
  <script src="{% static 'js/jquery-3.2.1.js' %}"></script>
  <!-- Include all compiled plugins (below), or include individual files as needed -->
  <script src="{% static 'bootstrap-3.3.7-dist/js/bootstrap.min.js' %}"></script>
</body>
</html>

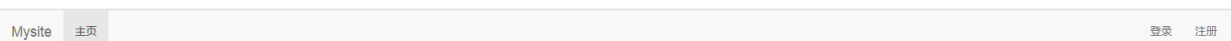
```

简要说明：

- 通过页面顶端的{% load staticfiles %}加载后，才可以使用static方法；
- 通过{% block title %}base{% endblock %}，设置了一个动态的页面title块；
- 通过{% block css %}{% endblock %}，设置了一个动态的css加载块；
- 通过{% block content %}{% endblock %}，为具体页面的主体内容留下接口；

- 通过`{% static 'bootstrap-3.3.7-dist/css/bootstrap.min.css' %}`将样式文件指向了我们的实际静态文件，下面的js脚本也是同样的道理。

看下效果



5.6.设计登录页面

Bootstarp提供了一个基本的表单样式，代码如下：

```
<form>
  <div class="form-group">
    <label for="exampleInputEmail">Email address</label>
    <input type="email" class="form-control" id="exampleInputEmail"
placeholder="Email">
  </div>
  <div class="form-group">
    <label for="exampleInputPassword1">Password</label>
    <input type="password" class="form-control" id="exampleInputPassword1"
placeholder="Password">
  </div>
  <div class="form-group">
    <label for="exampleInputFile">File input</label>
    <input type="file" id="exampleInputFile">
    <p class="help-block">Example block-level help text here.</p>
  </div>
  <div class="checkbox">
    <label>
      <input type="checkbox"> Check me out
    </label>
  </div>
  <button type="submit" class="btn btn-default">Submit</button>
</form>
```

如下：

实例：

Email address

Password

File input

 未选择文件。
Example block-level help text here.

☐ Check me out

我们结合Bootstrap和前面自己写的form表单，修改`login/templates/login/login.html`成符合项目要求的样子：

```
{% extends '../base.html' %}
```

```

{% load staticfiles %}

{% block title %}登录{% endblock %}

{% block css %}
    <link rel="stylesheet" href="{% static 'css/login.css' %}" >
{% endblock %}

{% block content %}
    <div class="container">
        <div class="col-md-4 col-md-offset-4">
            <form class='form-login' action="/login/" method="post">
                <h2 class="text-center">欢迎登录</h2>
                <div class="form-group">
                    <label for="id_username">用户名: </label>
                    <input type="text" name='username' class="form-control" id="id_username"
placeholder="Username" autofocus required>
                </div>
                <div class="form-group">
                    <label for="id_password">密码: </label>
                    <input type="password" name='password' class="form-control"
id="id_password" placeholder="Password" required>
                </div>
                <button type="reset" class="btn btn-default pull-left">重置</button>
                <button type="submit" class="btn btn-primary pull-right">提交</button>
            </form>
        </div>
    </div> <!-- /container -->
{% endblock %}

```

说明:

- 通过{% extends 'base.html' %}继承了'base.html'模板的内容;
- 通过{% block title %}登录{% endblock %}设置了专门的title;
- 通过block css引入了针对性的login.css样式文件;
- 主体内容定义在block content内部
- 添加了一个重置按钮。

在static/css目录中新建一个login.css样式文件, 这里简单地写了点样式,

```

body {
    background-color: #eee;
}

.form-login {
    max-width: 330px;
    padding: 15px;
    margin: 0 auto;
}

.form-login .form-control {
    position: relative;
    height: auto;
}

```



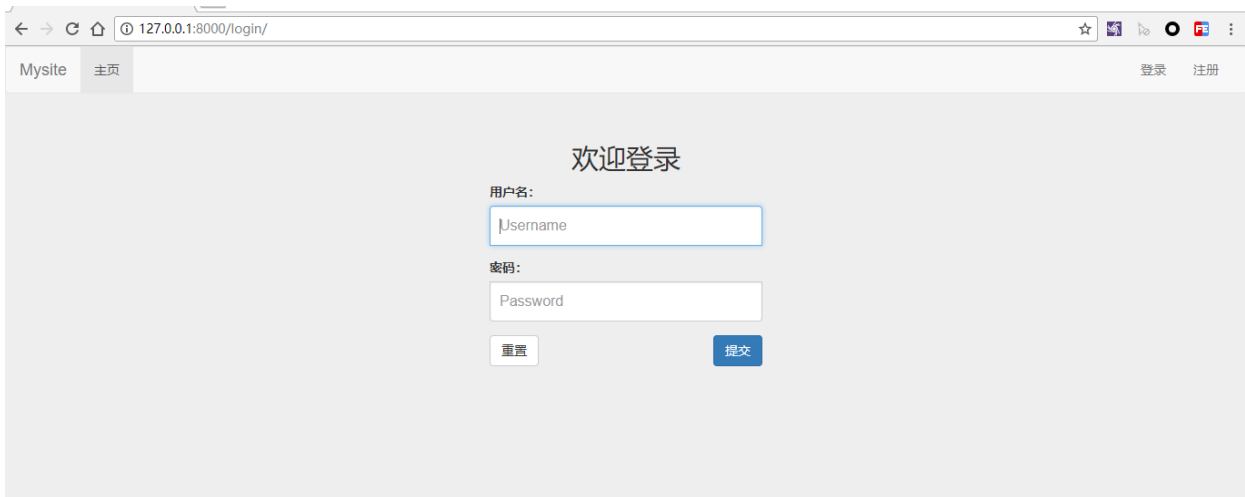
```

-webkit-box-sizing: border-box;
-moz-box-sizing: border-box;
box-sizing: border-box;
padding: 10px;
font-size: 16px;
}
.form-login .form-control:focus {
z-index: 2;
}
.form-login input[type="text"] {
margin-bottom: -1px;

border-bottom-right-radius: 0;
border-bottom-left-radius: 0;
}
.form-login input[type="password"] {
margin-bottom: 10px;
border-top-left-radius: 0;
border-top-right-radius: 0;
}

```

python manage.py runserver运行最后效果：



六、登录视图

6.1. 登录视图

根据我们在路由中的设计，用户通过login.html中的表单填写用户名和密码，并以POST的方式发送到服务器的/login/地址。服务器通过login/views.py中的login()视图函数，接收并处理这一请求。

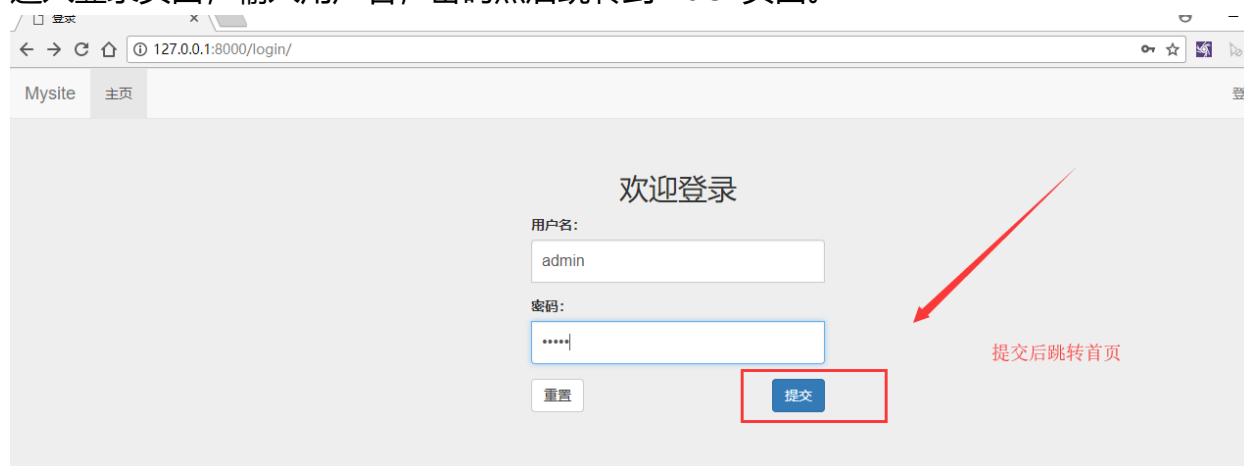
我们可以通过下面的方法接收和处理请求：

```
def login(request):
    if request.method == 'POST':
        username = request.POST.get('username')
        password = request.POST.get('password')
        return redirect('/index/')
    return render(request, 'user/login.html')
```

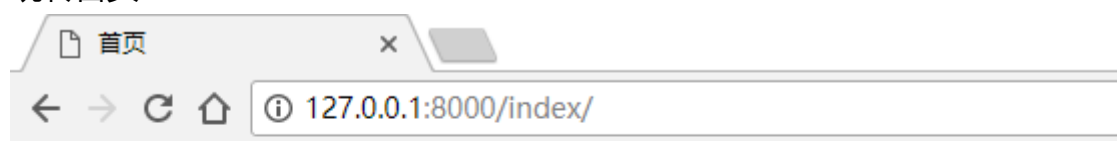
还需要在前端页面的form表单内添加一个{% csrf_token %} 标签：

```
<form class='form-login' action="/login/" method="post">
    {% csrf_token %}
    <h2 class="text-center">欢迎登录</h2>
    <div class="form-group">
        <label for="id_username">用户名: </label>
        <input type="text" name='username' class="form-control"
```

进入登录页面，输入用户名，密码然后跳转到index页面。



跳转首页



首页

6.2.数据验证

通过唯一的用户名，使用Django的ORM去数据库中查询用户数据，如果有匹配项，则进行密码对比，如果没有匹配项，说明用户名不存在。如果密码对比错误，说明密码不正确。

```
def login(request):
    if request.method == "POST":
        username = request.POST.get('username', None)
        password = request.POST.get('password', None)
        if username and password: # 确保用户名和密码都不为空
            username = username.strip()
            # 用户名字符合法性验证
            # 密码长度验证
            # 更多的其它验证.....
            try:
                user = models.User.objects.get(name=username)
            except:
                return render(request, 'user/login.html')
            if user.password == password:
                return redirect('/index/')
        return render(request, 'user/login.html')
```

6.3.添加提示信息

上面的代码还缺少很重要的一部分内容，提示信息！无论是登录成功还是失败，用户都没有得到任何提示信息，这显然是不行的。

修改一下login视图：

```
def login(request):
    if request.method == 'POST':
        username = request.POST.get('username', None)
        password = request.POST.get('password', None)
        if username and password: # 确保用户名和密码都不为空
            # 用户名字符合法性验证
            # 密码长度验证
            # 更多的其它验证.....
            try:
                user = models.User.objects.get(name=username)
                if user.password == password:
                    return redirect('/index/')
                else:
                    message = '密码不正确！'
            except:
                message = '用户名不存在！'
        return render(request, 'user/login.html', {'message': message})
    return render(request, 'user/login.html')
```

增加了message变量，用于保存提示信息。当有错误信息的时候，将错误信息打包成一个字典，然后作为第三个参数提供给render()方法。这个数据字典在渲染模板的时候会传递到模板里供你调用。

为了在前端页面显示信息，还需要对login.html进行修改：

```
{% extends './base.html' %}
{% load staticfiles %}
{% block title %}登录{% endblock %}
{% block css %}
    <link rel="stylesheet" href="{% static 'css/login.css' %}">
{% endblock %}

{% block content %}
```

```

<div class="container">
  <div class="col-md-4 col-md-offset-4">
    <form class='form-login' action="/login/" method="post">

      {% if message %}
        <div class="alert alert-warning">{{ message }}</div>
      {% endif %}

      {% csrf_token %}
      <h2 class="text-center">欢迎登录</h2>
      <div class="form-group">
        <label for="id_username">用户名: </label>
        <input type="text" name='username' class="form-control"
id="id_username" placeholder="Username"
          autofocus required>
        </div>
        <div class="form-group">
          <label for="id_password">密码: </label>
          <input type="password" name='password' class="form-control"
id="id_password" placeholder="Password"
            required>
          </div>
          <button type="reset" class="btn btn-default pull-left">重置</button>
          <button type="submit" class="btn btn-primary pull-right">提交</button>
        </form>
      </div>
    </div> <!-- /container -->
  {% endblock %}

```

把index.html主页模板也修改一下，删除原有内容，添加下面的代码：

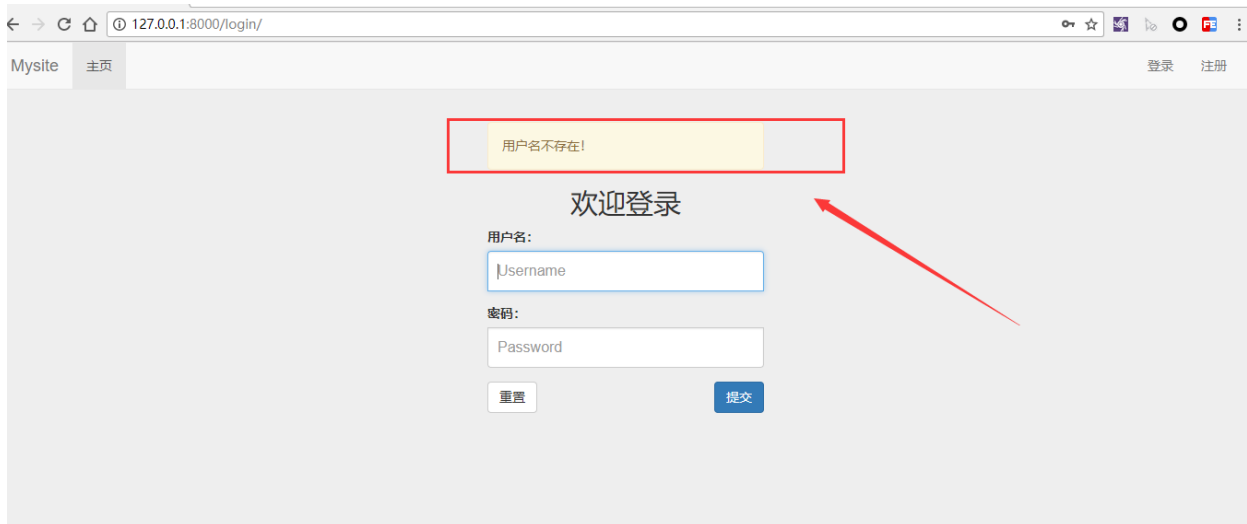
```

{#login/templates/login/index.html#}

{% extends '../base.html' %}
{% block title %}主页{% endblock %}
{% block content %}
  <h1>欢迎回来! </h1>
{% endblock %}

```

运行效果：



七、Django表单

Django的表单给我们提供了下面三个主要功能：

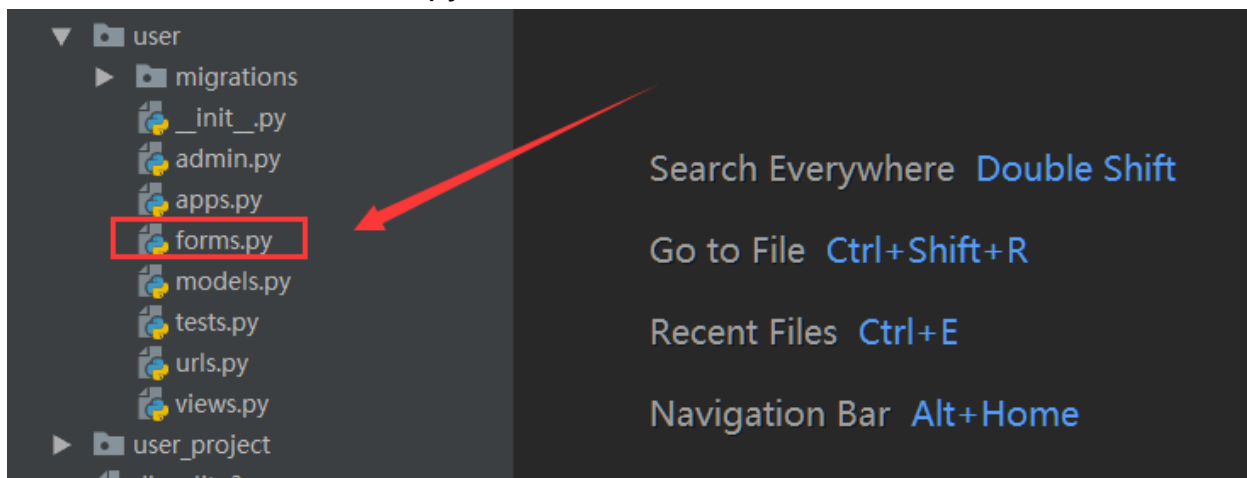
准备和重构数据用于页面渲染；

为数据创建HTML表单元素；

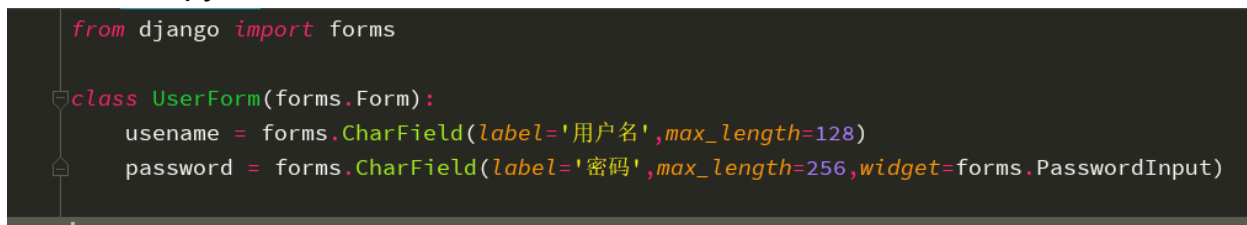
接收和处理用户从表单发送过来的数据

7.1.创建表单模型：

在user目录下新建一个forms.py



user/forms.py



说明：

- 要先导入forms模块
- 所有的表单类都要继承forms.Form类

- 每个表单字段都有自己的字段类型比如CharField，它们分别对应一种HTML语言中<form>内的一个input元素。这一点和Django模型系统的设计非常相似。
- label参数用于设置<label>标签
- max_length限制字段输入的最大长度。它同时起到两个作用，一是在浏览器页面限制用户输入不可超过字符数，二是在后端服务器验证用户输入的长度也不可超过。
- widget=forms.PasswordInput用于指定该字段在form表单里表现为<input type='password' />，也就是密码输入框。

7.2修改视图

user/views.py

```
def login(request):
    if request.method == 'POST':
        login_form = UserForm(request.POST)
        message = '请检查填写的内容!'
        if login_form.is_valid():
            username = login_form.cleaned_data['username']
            password = login_form.cleaned_data['password']
            try:
                user = models.User.objects.get(name=username)
                if user.password == password:
                    return redirect('/index/')
                else:
                    message = '密码不正确!'
            except:
                message = '用户不存在!'
            return render(request, 'user/login.html', locals())
        login_form = UserForm()
    return render(request, 'user/login.html', locals())
```

说明：

- 对于非POST方法发送数据时，比如GET方法请求页面，返回空的表单，让用户可以填入数据；
- 对于POST方法，接收表单数据，并验证；
- 使用表单类自带的is_valid()方法一步完成数据验证工作；
- 验证成功后可以从表单对象的cleaned_data数据字典中获取表单的具体值；
- 如果验证不通过，则返回一个包含先前数据的表单给前端页面，方便用户修改。也就是说，它会帮你保留先前填写的数据内容，而不是返回一个空表！

另外，这里使用了一个小技巧，Python内置了一个locals()函数，它返回当前所有的本地变量字典，我们可以偷懒的将这作为render函数的数据字典参数值，就不用费劲去构造一个形如{'message':message, 'login_form':login_form}的字典了。这样做的好处当然是大大方便了我们，但是同时也可能往模板传入了一些多余的变量数据，造成数据冗余降低效率。

7.3.修改login界面

Django的表单很重要的一个功能就是自动生成HTML的form表单内容。现在，我们需要修改一下原来的login.html文件：

```
{% extends '../base.html' %}
{% load staticfiles %}
{% block title %}登录{% endblock %}
{% block css %}<link href="{% static 'css/login.css' %}" rel="stylesheet"/>{% endblock %}

{% block content %}
    <div class="container">
        <div class="col-md-4 col-md-offset-4">
            <form class="form-login" action="/login/" method="post">

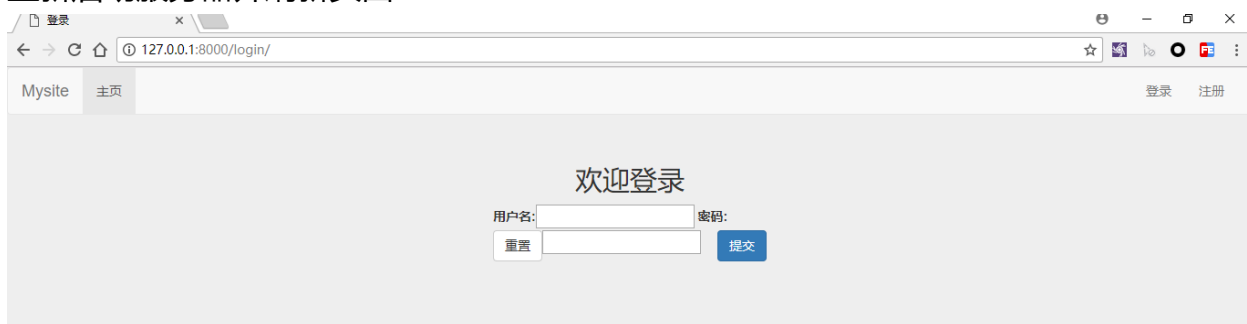
                {% if message %}
                    <div class="alert alert-warning">{{ message }}</div>
                {% endif %}
                {% csrf_token %}
                <h2 class="text-center">欢迎登录</h2>

                {{ login_form }}

                <button type="reset" class="btn btn-default pull-left">重置</button>
                <button type="submit" class="btn btn-primary pull-right">提交</button>

            </form>
        </div>
    </div> <!-- /container -->
{% endblock %}
```

重新启动服务器并刷新页面：



浏览器生成的源码：

```
<!DOCTYPE html>
<html lang="zh-CN">
  <head>
    <meta charset="utf-8">
    <meta http-equiv="X-UA-Compatible" content="IE=edge">
    <meta name="viewport" content="width=device-width, initial-scale=1">
    <!-- 上述3个meta标签*必须*放在最前面，任何其他内容都*必须*跟随其后！ -->
    <title>登录</title>
```

```

<!-- Bootstrap -->
<link href="/static/bootstrap-3.3.7-dist/css/bootstrap.min.css" rel="stylesheet">

<!-- HTML5 shim and Respond.js for IE8 support of HTML5 elements and media queries -
->
<!-- WARNING: Respond.js doesn't work if you view the page via file:// -->
<!--[if lt IE 9]>
  <script src="https://cdn.bootcss.com/html5shiv/3.7.3/html5shiv.min.js"></script>
  <script src="https://cdn.bootcss.com/respond.js/1.4.2/respond.min.js"></script>
<![endif]-->
<link href="/static/css/login.css" rel="stylesheet"/>

<!-- Ad Muncher content start --> <script type="text/javascript"
src="http://interceptedby.admuncher.com/B05E79FA11E181C7/helper.js#0.3146.0"
id="XZwz_MainScript"></script> <link rel="stylesheet"
href="http://interceptedby.admuncher.com/B05E79FA11E181C7/helper.css"
type="text/css" media="all" /> <!-- Ad Muncher content end -->

</head>
<body>
  <nav class="navbar navbar-default">
    <div class="container-fluid">
      <!-- Brand and toggle get grouped for better mobile display -->
      <div class="navbar-header">
        <button type="button" class="navbar-toggle collapsed" data-toggle="collapse"
data-target="#my-nav" aria-expanded="false">
          <span class="sr-only">切换导航条</span>
          <span class="icon-bar"></span>
          <span class="icon-bar"></span>
          <span class="icon-bar"></span>
        </button>
        <a class="navbar-brand" href="#">Mysite</a>
      </div>

      <!-- Collect the nav links, forms, and other content for toggling -->
      <div class="collapse navbar-collapse" id="my-nav">
        <ul class="nav navbar-nav">
          <li class="active"><a href="/index/">主页</a></li>
        </ul>
        <ul class="nav navbar-nav navbar-right">
          <li><a href="/login/">登录</a></li>
          <li><a href="/register/">注册</a></li>
        </ul>
      </div> <!-- /.navbar-collapse -->
    </div> <!-- /.container-fluid -->
  </nav>

  <div class="container">
    <div class="col-md-4 col-md-offset-4">

```



```

<form class='form-login' action="/login/" method="post">

    <input type='hidden' name='csrfmiddlewaretoken'
value='3fx6wPzFYj5px8Rkv1OXclguwfkMPAaCPTJCZqXknAmGy4nTj8HhPSzekN3rC8fl'
/>

    <h2 class="text-center">欢迎登录</h2>

    <tr><th><label for="id_username">用户名:</label></th><td><input
type="text" name="username" maxlength="128" required id="id_username" /></td>
</tr>

<tr><th><label for="id_password">密码:</label></th><td><input type="password"
name="password" maxlength="256" required id="id_password" /></td></tr>

    <button type="reset" class="btn btn-default pull-left">重置</button>
    <button type="submit" class="btn btn-primary pull-right">提交</button>

</form>
</div>
</div> <!-- /container -->

<!-- jQuery (necessary for Bootstrap's JavaScript plugins) -->
<script src="/static/js/jquery-3.2.1.js"></script>
<!-- Include all compiled plugins (below), or include individual files as needed -->
<script src="/static/bootstrap-3.3.7-dist/js/bootstrap.min.js"></script>
</body>
</html>

```

7.4.手动渲染表单

直接`{{ login_form }}`虽然好，啥都不用操心，但是界面真的很丑，往往并不是你想要的，如果你要使用CSS和JS，比如你要引入Bootstrap框架，这些都需要对表单内的input元素进行额外控制，那怎么办呢？手动渲染字段就可以了。

可以通过`{{ login_form.name_of_field }}`获取每一个字段，然后分别渲染，如下例所示：

可以通过`{{ login_form.name_of_field }}`获取每一个字段，然后分别渲染，如下例所示：

```

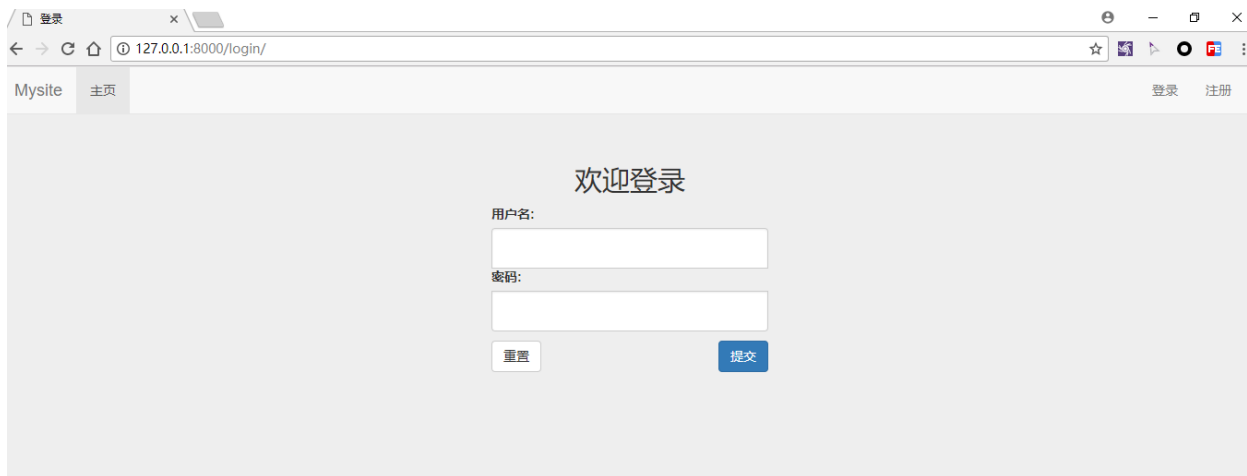
<div class="form-group">
    {{ login_form.username.label_tag }}
    {{ login_form.username }}
</div>
<div class="form-group">
    {{ login_form.password.label_tag }}
    {{ login_form.password }}
</div>

```

然后，在form类里添加attr属性即可，如下所示修改`user/forms.py`

```
views.py x forms.py x login.html x
1 from django import forms
2
3
4 class UserForm(forms.Form):
5     username = forms.CharField(label="用户名", max_length=128, widget=forms.TextInput(attrs={'class': 'form-control'}))
6     password = forms.CharField(label="密码", max_length=256, widget=forms.PasswordInput(attrs={'class': 'form-control'}))
7
```

再次刷新页面，就显示正常了！



八、图片验证码

为了防止机器人频繁登录网站或者破坏分子恶意登录，很多用户登录和注册系统都提供了图形验证码功能。

验证码（CAPTCHA）是“Completely Automated Public Turing test to tell Computers and Humans Apart”（全自动区分计算机和人类的图灵测试）的缩写，是一种区分用户是计算机还是人的公共全自动程序。可以防止恶意破解密码、刷票、论坛灌水，有效防止某个黑客对某一个特定注册用户用特定程序暴力破解方式进行不断的登陆尝试。图形验证码的历史比较悠久，到现在已经有点英雄末路的味道了。因为机器学习、图像识别的存在，机器人已经可以比较正确的识别图像内的字符了。但不管怎么说，作为一种防御手段，至少还是可以抵挡一些低级入门的攻击手段，抬高了攻击者的门槛。

在Django中实现图片验证码功能非常简单，有现成的第三方库可以使用，我们不必自己开发（也要能开发得出来，囧）。这个库叫做django-simple-captcha。

8.1.安装captcha

直接安装：pip install django-simple-captcha

```
(drfw) F:\Python\Django\code\user_project>pip install django-simple-captcha
Collecting django-simple-captcha
  Downloading https://files.pythonhosted.org/packages/4d/46/44aff307e370e873ebb44dd8e9a1b0de10ddd1e55779361a5654d273d939/django-simple-captcha-0.5.6.zip (226kB)
    100% |#####| 235kB 7.4kB/s
Requirement already satisfied: setuptools in d:\env\drfw\lib\site-packages (from django-simple-captcha)
Collecting six>=1.2.0 (from django-simple-captcha)
  Downloading https://files.pythonhosted.org/packages/67/4b/141a581104b1f6397bfa78ac9d43d8ad29a7ca43ea90a2d863fe3056e86a/six-1.11.0-py3-none-any.whl
Requirement already satisfied: Django>=1.7 in d:\env\drfw\lib\site-packages (from django-simple-captcha)
Collecting Pillow>=2.2.2 (from django-simple-captcha)
  Downloading https://files.pythonhosted.org/packages/a4/86/283719dac6309cf483452abb09759be9b2c0974435ed608dc67949127e13/Pillow-5.1.0-cp36-cp36m-win_amd64.whl (1.6MB)
    100% |#####| 1.6MB 4.9kB/s
Collecting django-ranged-response==0.2.0 (from django-simple-captcha)
  Retrying (Retry(total=4, connect=None, read=None, redirect=None, status=None)) after connection broken by 'ConnectTimeoutError(<pip._vendor.urllib3.connection.VerifiedHTTPSConnection object at 0x0000020A10423048>, 'Connection to pypi.python.org timed out. (connect timeout=15)')': /simple/django-ranged-response/
  Retrying (Retry(total=3, connect=None, read=None, redirect=None, status=None)) after connection broken by 'ConnectTimeoutError(<pip._vendor.urllib3.connection.VerifiedHTTPSConnection object at 0x0000020A1136EF98>, 'Connection to pypi.python.org timed out. (connect timeout=15)')': /simple/django-ranged-response/
  Downloading https://files.pythonhosted.org/packages/70/e3/9372fedca8e9c3205e7979528cd1a14354a9a24d38efff11c1846ff8bf1/django-ranged-response-0.2.0.tar.gz
Requirement already satisfied: pytz in d:\env\drfw\lib\site-packages (from Django>=1.7->django-simple-captcha)
Building wheels for collected packages: django-simple-captcha, django-ranged-response
  Running setup.py bdist_wheel for django-simple-captcha ... done
  Stored in directory: C:\Users\85753\AppData\Local\pip\Cache\wheels\97\29\48\84a2fe8a2a9751781847b68970a14ff66d6b0e879d407510e8
  Running setup.py bdist_wheel for django-ranged-response ... done
  Stored in directory: C:\Users\85753\AppData\Local\pip\Cache\wheels\16\43\50\05353cd3f784bf657c48c608aa2f2a65c37f1cb362c9249774
Successfully built django-simple-captcha django-ranged-response
Installing collected packages: six, Pillow, django-ranged-response, django-simple-captcha
Successfully installed Pillow-5.1.0 django-ranged-response-0.2.0 django-simple-captcha-0.5.6 six-1.11.0
```

Django自动帮我们安装了相关的依赖库six、olefile和Pillow，其中的Pillow是大名鼎鼎的绘图模块。

注册captcha

在settings中，将 'captcha' 注册到app列表里：

```
INSTALLED_APPS = [
    'django.contrib.admin',
    'django.contrib.auth',
    'django.contrib.contenttypes',
    'django.contrib.sessions',
    'django.contrib.messages',
    'django.contrib.staticfiles',
    'user', # 注册userapp
    'captcha', # 注册captcha
]
```

captcha需要在数据库中建立自己的数据表，所以需要执行migrate命令生成数据表：

python manage.py migrate

```
(drfw) F:\Python\Django\code\user_project>python manage.py migrate
Operations to perform:
  Apply all migrations: admin, auth, captcha, contenttypes, sessions, user
Running migrations:
  Applying captcha.0001_initial... OK
```

8.2.添加url路由

根目录下的urls.py文件中增加captcha对应的网址：

```
from django.contrib import admin
from django.urls import path, include

urlpatterns = [
    path('admin/', admin.site.urls),
    path('', include('user.urls')),
    path('capctha', include('captcha.urls'))
]
```

8.3.修改forms.py

如果上面都OK了，就可以直接在我们的forms.py文件中添加CaptchaField了。

user.forms.py

```
1 from django import forms
2 from captcha.fields import CaptchaField
3
4 class UserForm(forms.Form):
5     username = forms.CharField(label="用户名", max_length=128, widget=forms.TextInput(attrs={'class': 'form-control'}))
6     password = forms.CharField(label="密码", max_length=256, widget=forms.PasswordInput(attrs={'class': 'form-control'}))
7     captcha = CaptchaField(label='验证码')
```

需要提前导入from captcha.fields import CaptchaField，然后就像写普通的form字段一样添加一个captcha字段就可以了！

8.4.修改login.html

由于我们前面是手动生成的form表单，所以还要修改一下，添加captcha的相关内容，如下所示：

```
{% extends '../base.html' %}
{% load staticfiles %}
{% block title %}登录{% endblock %}
{% block css %}
    <link rel="stylesheet" href="{% static 'css/login.css' %}">
{% endblock %}

{% block content %}
    <div class="container">
        <div class="col-md-4 col-md-offset-4">
            <form class="form-login" action="/login/" method="post">

                {% if message %}
                    <div class="alert alert-warning">{{ message }}</div>
                {% endif %}
                {% csrf_token %}

                <h2 class="text-center">欢迎登录</h2>
                <div class="form-group">
                    {{ login_form.username.label_tag }}
                    {{ login_form.username }}
                </div>
                <div class="form-group">
                    {{ login_form.password.label_tag }}
                    {{ login_form.password }}
                </div>

                <div class="form-group">
                    {{ login_form.captcha.errors }}
                    {{ login_form.captcha.label_tag }}
                    {{ login_form.captcha }}
                </div>

            </form>
        </div>
    </div>
{% endblock %}
```

```

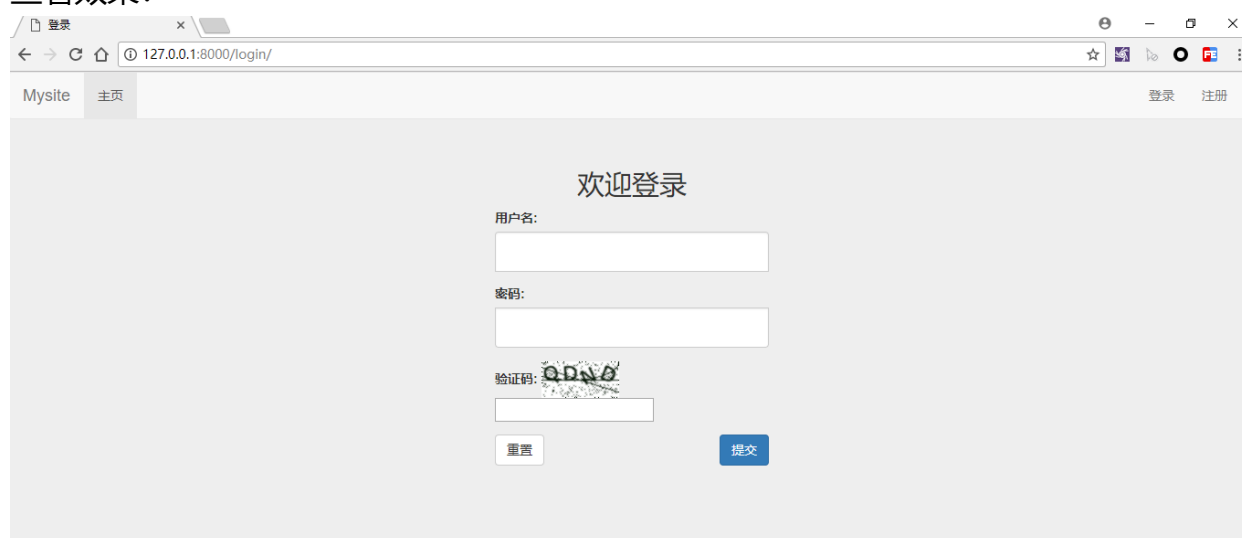
        <button type="reset" class="btn btn-default pull-left">重置
    </button>
        <button type="submit" class="btn btn-primary pull-right">提交
    </button>

    </form>
</div>
</div> <!-- /container -->
{% endblock %}

```

这里额外增加了一条`{{ login_form.captcha.errors }}`用于明确指示用户，你的验证码不正确

查看效果：



其中验证图形码是否正确的工作都是在后台自动完成的，只需要使用`is_valid()`这个forms内置的验证方法就一起进行了，完全不需要在视图函数中添加任何的验证代码，非常方便快捷！

九、session会话

因为因特网HTTP协议的特性，每一次来自于用户浏览器的请求（request）都是无状态的、独立的。通俗地说，就是无法保存用户状态，后台服务器根本就不知道当前请求和以前及以后请求是否来自同一用户。对于静态网站，这可能不是个问题，而对于动态网站，尤其是京东、天猫、银行等购物或金融网站，无法识别用户并保持用户状态是致命的，根本就无法提供服务。你可以尝试将浏览器的cookie功能关闭，你会发现将无法在京东登录和购物。

为了实现连接状态的保持功能，网站会通过用户的浏览器在用户机器内被限定的硬盘位置中写入一些数据，也就是所谓的Cookie。通过Cookie可以保存一些诸如用户名、浏览记录、表单记录、登录和注销等各种数据。但是这种方式非常不安全，因为Cookie保存在用户的机器上，如果Cookie被伪造、篡改或删除，就会造成极大的安全威胁，因此，现代网站设

计通常将Cookie用来保存一些不重要的内容，实际的用户数据和状态还是以Session会话的方式保存在服务器端。

Session依赖Cookie！但与Cookie不同的地方在于Session将所有数据都放在服务器端，用户浏览器的Cookie中只会保存一个非明文的识别信息，比如哈希值。

Django提供了一个通用的Session框架，并且可以使用多种session数据的保存方式：

- 保存在数据库内
- 保存到缓存
- 保存到文件内
- 保存到cookie内

通常情况，没有特别需求的话，请使用保存在数据库内的方式，尽量不要保存到Cookie内。

Django的session框架默认启用，并已经注册在app设置内，如果真的没有启用，那么参考下面的内容添加有说明的那两行，再执行migrate命令创建数据表，就可以使用session了。

user_project/settings.py

```
INSTALLED_APPS = [  
    'django.contrib.admin',  
    'django.contrib.auth',  
    'django.contrib.contenttypes',  
    'django.contrib.sessions', # 这一行  
    'django.contrib.messages',  
    'django.contrib.staticfiles',  
    'user', # 注册userapp  
    'captcha', # 注册captcha  
]
```

user_project/settings.py

```
MIDDLEWARE = [  
    'django.middleware.security.SecurityMiddleware',  
    'django.contrib.sessions.middleware.SessionMiddleware', # 这一行  
    'django.middleware.common.CommonMiddleware',  
    'django.middleware.csrf.CsrfViewMiddleware',  
    'django.contrib.auth.middleware.AuthenticationMiddleware',  
    'django.contrib.messages.middleware.MessageMiddleware',  
    'django.middleware.clickjacking.XFrameOptionsMiddleware',  
]
```

当session启用后，传递给视图request参数的HttpRequest对象将包含一个session属性，就像一个字典对象一样。你可以在Django的任何地方读写request.session属性，或者多次编辑使用它。

下面是session使用参考：

```
class backends.base.SessionBase
    # 这是所有会话对象的基类，包含标准的字典方法：
    __getitem__(key)
        Example: fav_color = request.session['fav_color']
    __setitem__(key, value)
        Example: request.session['fav_color'] = 'blue'
    __delitem__(key)
        Example: del request.session['fav_color'] # 如果不存在会抛出异常
    __contains__(key)
        Example: 'fav_color' in request.session
    get(key, default=None)
        Example: fav_color = request.session.get('fav_color', 'red')
    pop(key, default=__not_given)
        Example: fav_color = request.session.pop('fav_color', 'blue')
# 类似字典数据类型的内置方法
    keys()
    items()
    setdefault()
    clear()

    # 它还有下面的方法：
    flush()
        # 删除当前的会话数据和会话cookie。经常用在用户退出后，删除会话。

    set_test_cookie()
        # 设置一个测试cookie，用于探测用户浏览器是否支持cookies。由于cookie的工作机制，你只有在下次用户请求的时候才可以测试。
    test_cookie_worked()
        # 返回True或者False，取决于用户的浏览器是否接受测试cookie。你必须在之前先调用set_test_cookie()方法。
    delete_test_cookie()
        # 删除测试cookie。
    set_expiry(value)
        # 设置cookie的有效期。可以传递不同类型的参数值：
        • 如果值是一个整数，session将在对应的秒数后失效。例如request.session.set_expiry(300)将在300秒后失效。
        • 如果值是一个datetime或者timedelta对象，会话将在指定的日期失效
        • 如果为0，在用户关闭浏览器后失效
        • 如果为None，则将使用全局会话失效策略
        失效时间从上一次会话被修改的时刻开始计时。

    get_expiry_age()
        # 返回多少秒后失效的秒数。对于没有自定义失效时间的会话，这等同于SESSION_COOKIE_AGE。
        # 这个方法接受2个可选的关键字参数
        • modification: 会话的最后修改时间（datetime对象）。默认是当前时间。
        • expiry: 会话失效信息，可以是datetime对象，也可以是int或None

    get_expiry_date()
        # 和上面的方法类似，只是返回的是日期

    get_expire_at_browser_close()
        # 返回True或False，根据用户会话是否是浏览器关闭后就结束。

    clear_expired()
        # 删除已经失效的会话数据。
    cycle_key()

    # 创建一个新的会话密钥用于保持当前的会话数据。django.contrib.auth.login() 会调用这
```


个方法。

9.1.使用session

首先，修改user/views.py中的login()视图函数：

```
def login(request):
    if request.session.get('is_login',None):
        return redirect('/index')

    if request.method == "POST":
        login_form = UserForm(request.POST)
        message = "请检查填写的内容！"
        if login_form.is_valid():
            username = login_form.cleaned_data['username']
            password = login_form.cleaned_data['password']
            try:
                user = models.User.objects.get(name=username)
                if user.password == password:
                    request.session['is_login'] = True
                    request.session['user_id'] = user.id
                    request.session['user_name'] = user.name
                    return redirect('/index/')
            except:
                message = "密码不正确！"
        except:
            message = "用户不存在！"
        return render(request, 'user/login.html', locals())

    login_form = UserForm()
    return render(request, 'user/login.html', locals())
```

通过下面的if语句，我们不允许重复登录：

```
if request.session.get('is_login',None):
    return redirect("/index/")
```

通过下面的语句，我们往session字典内写入用户状态和数据：

```
request.session['is_login'] = True
request.session['user_id'] = user.id
request.session['user_name'] = user.name
```

你完全可以往里面写任何数据，不仅仅限于用户相关！

既然有了session记录用户登录状态，那么就可以完善我们的登出视图函数了：

user/views.py


```
def logout(request):
    if not request.session.get('is_login', None):
        # 如果本来就未登录，也就没有登出一说
        return redirect("/index/")
    request.session.flush()
    # 或者使用下面的方法
    # del request.session['is_login']
    # del request.session['user_id']
    # del request.session['user_name']
    return redirect("/index/")
```

flush()方法是比较安全的一种做法，而且一次性将session中的所有内容全部清空，确保不留后患。但也有不好的地方，那就是如果你在session中夹带了一点‘私货’，会被一并删除，这一点一定要注意。

9.2.完善页面

有了用户状态，就可以根据用户登录与否，展示不同的页面，比如导航条内容：

首先，修改base.html文件：

```
<!-- Collect the nav links, forms, and other content for toggling -->
<div class="collapse navbar-collapse" id="my-nav">
  <ul class="nav navbar-nav">
    <li class="active"><a href="/index/">主页</a></li>
  </ul>
  <ul class="nav navbar-nav navbar-right">
    {% if request.session.is_login %}
      <li><a href="#">当前在线: {{ request.session.user_name }}</a></li>
      <li><a href="/logout/">登出</a></li>
    {% else %}
      <li><a href="/login/">登录</a></li>
      <li><a href="/register/">注册</a></li>
    {% endif %}
  </ul>
</div><!-- /.navbar-collapse -->
</div><!-- /.container-fluid -->
```

通过if判断，当登录时，显示当前用户名和登出按钮。未登录时，显示登录和注册按钮。注意其中的模板语言，{{ request }}这个变量会被默认传入模板中，可以通过圆点的调用方式，获取它内部的{{ request.session }}，再进一步的获取session中的内容。其实{{ request }}中的数据远不止此，例如{{ request.path }}就可以获取先前的url地址。

再修改一下index.html页面，根据登录与否的不同，显示不同的内容：

看下效果：

Mysite 主页 当前已登录: test 登出

你好,test！欢迎回来！

Mysite 主页 登录 注册

你尚未登录，只能访问公开内容！

十、注册视图

10.1.创建forms

在/login/forms.py中添加一个新的表单类：

```
class RegisterForm(forms.Form):
    gender = (
        ('male', "男"),
        ('female', "女"),
    )
    username = forms.CharField(label="用户名", max_length=128, widget=forms.TextInput(attrs={'class': 'form-control'}))
    password1 = forms.CharField(label="密码", max_length=256, widget=forms.PasswordInput(attrs={'class': 'form-control'}))
    password2 = forms.CharField(label="确认密码", max_length=256,
                                widget=forms.PasswordInput(attrs={'class': 'form-control'}))
    email = forms.EmailField(label="邮箱地址", widget=forms.EmailInput(attrs={'class': 'form-control'}))
    sex = forms.ChoiceField(label='性别', choices=gender)
    captcha = CaptchaField(label='验证码')
```

说明：

- gender和User模型中的一样，其实可以拉出来作为常量共用，为了直观，特意重写一遍；
- password1和password2，用于输入两遍密码，并进行比较，防止误输密码；
- email是一个邮箱输入框；
- sex是一个select下拉框；

10.2.完善register.html

同样地，类似login.html文件，我们在register.html中编写forms相关条目：

```
{% extends '../base.html' %}

{% block title %}注册{% endblock %}
{% block content %}
    <div class="container">
        <div class="col-md-4 col-md-offset-4">
            <form class='form-register' action="/register/" method="post">

                {% if message %}
                    <div class="alert alert-warning">{{ message }}</div>
                {% endif %}

                {% csrf_token %}

                <h2 class="text-center">欢迎注册</h2>
                <div class="form-group">
                    {{ register_form.username.label_tag }}
                    {{ register_form.username }}
                </div>
                <div class="form-group">
                    {{ register_form.password1.label_tag }}
                    {{ register_form.password1 }}
                </div>
                <div class="form-group">
                    {{ register_form.password2.label_tag }}
                    {{ register_form.password2 }}
                </div>
                <div class="form-group">
                    {{ register_form.email.label_tag }}
                    {{ register_form.email }}
```

```

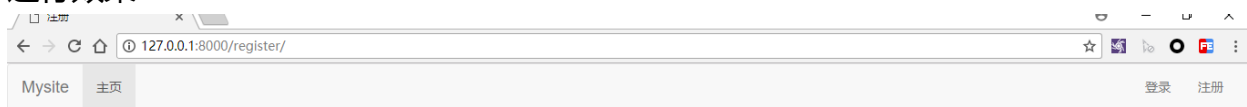
</div>
<div class="form-group">
    {{ register_form.sex.label_tag }}
    {{ register_form.sex }}
</div>
<div class="form-group">
    {{ register_form.captcha.errors }}
    {{ register_form.captcha.label_tag }}
    {{ register_form.captcha }}
</div>

<button type="reset" class="btn btn-default pull-left">重置</button>
<button type="submit" class="btn btn-primary pull-right">提交</button>

</form>
</div>
</div> <!-- /container -->
{% endblock %}

```

运行效果：



10.3.注册视图

进入/user/views.py文件，现在来完善我们的register()视图：

```

def register(request):
    if request.session.get('is_login', None):
        # 登录状态不允许注册。你可以修改这条原则！
        return redirect("/index/")
    if request.method == "POST":
        register_form = RegisterForm(request.POST)
        message = "请检查填写的内容！"
        if register_form.is_valid(): # 获取数据
            username = register_form.cleaned_data['username']
            password1 = register_form.cleaned_data['password1']
            password2 = register_form.cleaned_data['password2']
            email = register_form.cleaned_data['email']
            sex = register_form.cleaned_data['sex']
            if password1 != password2: # 判断两次密码是否相同
                message = "两次输入的密码不同！"
                return render(request, 'user/register.html', locals())
            else:
                same_name_user = models.User.objects.filter(name=username)
                if same_name_user: # 用户名唯一
                    message = '用户已经存在，请重新选择用户名！'
                    return render(request, 'user/register.html', locals())
                same_email_user = models.User.objects.filter(email=email)
                if same_email_user: # 邮箱地址唯一
                    message = '该邮箱地址已被注册，请使用别的邮箱！'
                    return render(request, 'user/register.html', locals())

                # 当一切都OK的情况下，创建新用户

                new_user = models.User.objects.create()
                new_user.name = username
                new_user.password = password1

```

```
new_user.email = email
new_user.sex = sex
new_user.save()
return redirect('/login/') # 自动跳转到登录页面
register_form = RegisterForm()
return render(request, 'user/register.html', locals())
```

从大体逻辑上，也是先实例化一个RegisterForm的对象，然后使用is_valid()验证数据，再从cleaned_data中获取数据。

重点在于注册逻辑，首先两次输入的密码必须相同，其次不能存在相同用户名和邮箱，最后如果条件都满足，利用ORM的API，创建一个用户实例，然后保存到数据库内。

看一下注册的页面：



注册页面截图：

浏览器地址栏显示：127.0.0.1:8000/register/

页面标题：欢迎注册

注册表单包含以下字段：

- 用户名：
- 密码：
- 确认密码：
- 邮箱地址：
- 性别：男
- 验证码：YIAK

底部按钮：重置、提交

注册成功在admin后台可以看到注册的用户 (user:testuser,password:testuser)

欢迎注册

用户名:

testuser

密码:

确认密码:

邮箱地址:

testuser@test.com

性别: 男 ▼

验证码:



Y1AK

重置

提交

进入127.0.0.1:8000/admin(user:admin,password:admin ps:这是超级用户)

Site administration | Django

127.0.0.1:8000/admin/

Django administration

Site administration

AUTHENTICATION AND AUTHORIZATION	
Groups	+ Add Change
Users	+ Add Change

USER	
用户	+ Add Change

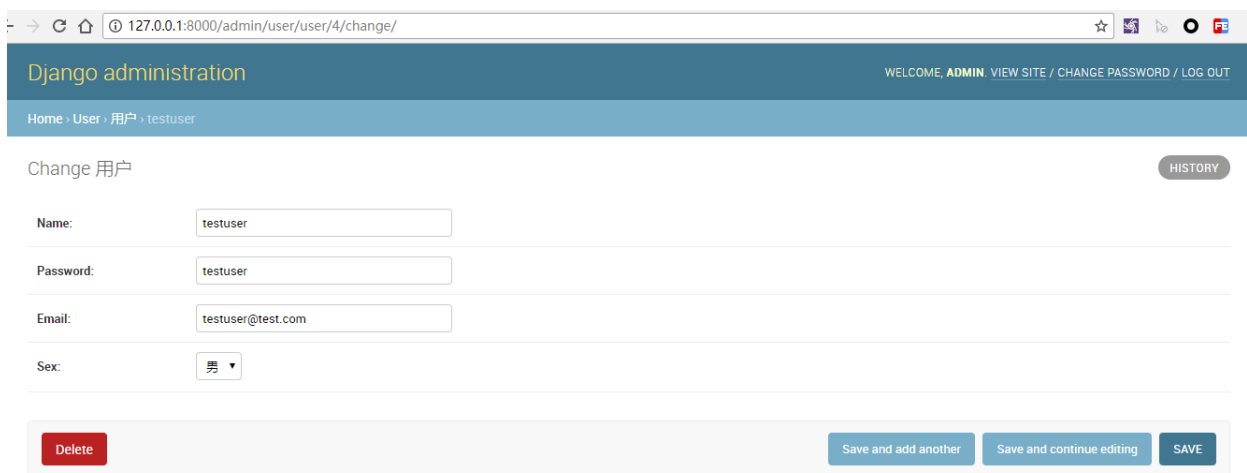
Recent actions

My actions

- + test3 用户
- + test2 用户
- + test1 用户



点击进去



10.4.密码加密

用户注册的密码应该加密才对

对于如何加密密码，有很多不同的途径，其安全程度也高低不等。这里我们使用Python内置的hashlib库，使用哈希值的方式加密密码，可能安全等级不够高，但足够简单，方便使用，不是吗？

首先在user/views.py中编写一个hash函数：

```
import hashlib

# Create your views here.
def hash_code(s, salt='mysite'):# 加点盐
    h = hashlib.sha256()
    s += salt
    h.update(s.encode()) # update方法只接收bytes类型
    return h.hexdigest()
```

然后，我们还要对login()和register()视图进行一下修改：

```
#login.html

if user.password == hash_code(password): # 哈希值和数据库内的值进行比对

#register.html

new_user.password = hash_code(password1) # 使用加密密码
```

user.views.py --def login(request):

```
def login(request):
    if request.session.get('is_login', None):
        return redirect('/index')

    if request.method == "POST":
        login_form = UserForm(request.POST)
        message = "请检查填写的内容!"
        if login_form.is_valid():
            username = login_form.cleaned_data['username']
            password = login_form.cleaned_data['password']
            try:
                user = models.User.objects.get(name=username)
                if user.password == hash_code(password): # 哈希值和数据库内的值进行比对
                    request.session['is_login'] = True
                    request.session['user_id'] = user.id
                    request.session['user_name'] = user.name
                    return redirect('/index/')
            except:
                pass
```

user.views.py --def register(request):

```
return render(request, 'user/register.html', locals())
else:
    same_name_user = models.User.objects.filter(name=username)
    if same_name_user: # 用户名唯一
        message = '用户已经存在, 请重新选择用户名!'
        return render(request, 'user/register.html', locals())
    same_email_user = models.User.objects.filter(email=email)
    if same_email_user: # 邮箱地址唯一
        message = '该邮箱地址已被注册, 请使用别的邮箱!'
        return render(request, 'user/register.html', locals())

    # 当一切都OK的情况下, 创建新用户

    new_user = models.User.objects.create()
    new_user.name = username
    new_user.password = hash_code(password1) # 使用加密密码
    new_user.email = email
    new_user.sex = sex
```

在后台删掉我们刚才注册的用户：



重启服务器,

```
(drfw) F:\Python\Django\code\user_project>python manage.py runserver
Performing system checks...

System check identified no issues (0 silenced).
April 13, 2018 - 01:25:04
Django version 2.0.3, using settings 'user_project.settings'
Starting development server at http://127.0.0.1:8000/
Quit the server with CTRL-BREAK.
```

进入注册页面, 新建一个用户,

欢迎注册

用户名:

testuser

密码:

.....

确认密码:

.....

邮箱地址:

testuser@test.com

性别: 男 ▼

验证码:  VDTU

重置

提交

注册成功自动跳转登陆页面：

欢迎登录

用户名:

密码:

验证码:



重置

提交

然后进入admin后台，查看用户的密码情况：

Django administration

WELCOME, ADMIN. VIEW SITE / CHANGE PASSWORD / LOG OUT

Home > User > 用户 > testuser

Change 用户

HISTORY

Name:

testuser

Password:

b4db40330ee7d7f848f492eef336a1ecde40b>

Email:

testuser@test.com

Sex:

男

Delete

Save and add another

Save and continue editing

SAVE

再使用该用户登录一下，

请检查填写的内容!

欢迎登录

用户名:

testuser

密码:

.....

- Invalid CAPTCHA

验证码:

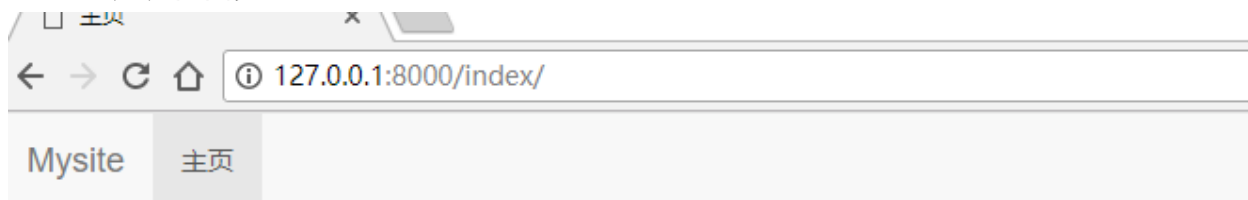


WIBX|

重置

提交

登录成功跳转首页



你好,testuser! 欢迎回来!

end!.....