# 关于django的请求和响应的基本操作:[参考上一个笔记](django学习笔记4-django框架1.docx)

# 4.这里我们来学习用django来操作MySQL数据库

## 前面我们学习了pymysql库,其实我们完全可以用这个库来操作数据库,但是在django中我们一般不那么做,因为我们有更好的方法,django提供了orm框架来操作数据库

## 1>虽然django提供了orm功能,但是他的底层是需要数据库python驱动库的.这里我们需要安装一个叫做mysqlclient的库(新版本的django对pymysql支持不好,所以我们需要使用mysqlclient)

|  |
| --- |
|  |

## 2>然后我们需要用django的orm来配置数据库设置,orm不能够创建数据库,你必须先创建数据库,然后orm可以帮你创建,修改和删除数据表.orm可以对数据表的数据进行增删改查操作.

### 这里我们新建一个叫做day15的数据库: create database day15 default charset utf8 collate utf8\_general\_ci;

|  |
| --- |
|  |

## 3>django连接MySQL数据库

### 1)在settings.py中进行配置,打开settings.py,找到databases一栏.它默认是使用sqlite3数据库的.

|  |
| --- |
|  |

### 2)我们这里使用的是MySQL,所以需要把这个配置覆盖掉,我们可以把它注释掉,然后点击它提供的连接,找一个MySQL的配置,然后填写自己的用户名密码,配置保存后django会自动检测,如果版本不对,会报错.

|  |
| --- |
| DATABASES = {  **"default"**: {  **"ENGINE"**: **"django.db.backends.mysql"**,  **"NAME"**: **"day15"**,  **"USER"**: **"root"**,  **"PASSWORD"**: **"root"**,  **"HOST"**: **"127.0.0.1"**,  **"PORT"**: **"3306"**,  } } |

#### 需要注意:新版本的django不支持MySQL5.x需要使用MySQL8.x我这里使用mysql8.4.6LTS

## 4>配置没有问题的话,我们就可以让django来帮我们操作数据表(创建,修改,删除)

### 1)创建表,需要把代码写在mysiteApp/models.py中,我们打开models.py,需要创建一个类,让他继承自models.Model类,比如我们创建一个UserInfo类

|  |
| --- |
|  |

#### 运行命令后然后django在底层会帮你生成一个数据表

|  |
| --- |
|  |

### 2)此时,表还没有在数据库中生成,我们需要执行2条命令:

#### 1.python manage.py makemigrations

|  |
| --- |
|  |

#### 2. python manage.py migrate

|  |
| --- |
|  |

#### 3.此时你去navicat查看,发现django除了生成我们的数据表以外还生成了一些表格,因为django默认给我们提供了一些app,orm是根据app来创建对于的数据表的

|  |
| --- |
|  |
|  |

##### 我们只需要关心mysitapp\_userinfo,这个才是我们的表,其他表django会在他的内部使用到.

|  |
| --- |
|  |

#### 4.然后我们在models.py里面添加2个类

|  |
| --- |
|  |

##### 如果你又想把这两个类添加到数据库,你必须重新执行上面的命令

|  |
| --- |
|  |
|  |
|  |

#### 5.如果你想删除一个数据表,你只需要把它对于的类的代码注释或者删除,然后在执行上面的代码,比如我们想把Role类对应的表删除

|  |
| --- |
|  |
|  |
|  |
|  |

#### 6.我们也可以把类的某些字段的代码注释了,然后运行上面的代码就可以删除数据表对应的列.

|  |
| --- |
|  |
|  |
|  |
|  |

#### 7.需要注意,如果你在已经有表的类里面添加字段,然后运行上面两个命令会有问题

|  |
| --- |
|  |
|  |

#### 8.此时你需要做一个选择,比如我们选择选项1,他就会使用我们提供的值来填充这个新列的所有数据

|  |
| --- |
|  |
|  |

#### 9.然后我们可以继续执行迁移

|  |
| --- |
|  |
|  |

#### 10.也可以设置默认值.,比如我们把age一列重新添加进来

|  |
| --- |
|  |
|  |
|  |
|  |

#### 11.还可以这么写

|  |
| --- |
|  |
|  |
|  |
|  |

#### 12.我们把size和data列删除,保留age列

|  |
| --- |
|  |
|  |
|  |
|  |

##### 注意:只要我们对models.py里面的类做了任何操作,都必须重新执行这两条命令

## 5>使用django的orm来操作数据表中的数据

### 1)新增数据,比如我们在department表中新增一条数据,我们可以在models.py中这么写

|  |
| --- |
|  |

### 2.我们添加一条orm/路由来练习数据,

|  |
| --- |
|  |

#### 然后在vews.py里面添加一个orm函数,需要先把models.py的类引入进来

|  |
| --- |
|  |

#### 然后我们需要利用效果类的objects.create方法来新建数据

|  |
| --- |
|  |

### 3>运行程序,在浏览器在输入/orm,提示操作成功

|  |
| --- |
|  |

#### 然后我们用navicat查看数据,发现的确添加进来了

|  |
| --- |
|  |

### 4>然后我们修改一下orm函数的代码,把部门数据操作注释了,然后我们来操作用户信息表

|  |
| --- |
|  |

### 5>然后重启服务器,刷新页面,提示添加成功,我们到navicat里面看看,发现数据添加进来了

|  |
| --- |
|  |

### 6>然后我们来删除数据,先把新增数据的代码注释了.然后我们来添加删除数据的代码,在删除之前需要筛选

|  |
| --- |
|  |

#### 重新运行程序,返回操作成功

|  |
| --- |
|  |

#### 然后用navicat查看数据,效果如下

|  |
| --- |
|  |
|  |

### 7>然后我们来学习获取数据,先练习获取所有数据,代码如下

|  |
| --- |
|  |
|  |
|  |

### 8>获取一条数据,代码如下

|  |
| --- |
|  |
|  |
|  |

### 9>如果你非常确定只有一条数据,你可以这么写

|  |
| --- |
|  |
|  |
|  |

### 10>我们还可以实现更新数据,先来更新所有数据

|  |
| --- |
|  |
|  |
|  |

### 11>然后可以更新某一些或者某一个数据

|  |
| --- |
|  |
|  |
|  |

# 5.课堂案例:用户管理

# 还是上面的项目,我们给他添加一个展示所有用户的功能

## 1.在urls.py中添加一个users/路由

|  |
| --- |
|  |

## 2.然后在views.py里面添加一个showAll函数,在里面获取所有用户信息,然后渲染一个all.html模板

|  |
| --- |
|  |

## 3.在template文件夹里面创建一个all.html文件展示数据,使用运行bootstrap的样式

|  |
| --- |
| {% **load static** %} <!DOCTYPE **html**> <**html lang="en"**> <**head**>  <**meta charset="UTF-8"**>  <**title**>用户列表</**title**>  <**link rel="stylesheet" href="**{% **static 'plugins/bootstrap-3.4.1/css/bootstrap.css'** %}**"**> </**head**> <**body**> <**div class="container"**>  <**table class="table table-bordered"**>  <**h2**><**caption**>用户列表</**caption**></**h2**>  <**thead**>  <**tr**>  <**td**>ID</**td**>  <**td**>用户名</**td**>  <**td**>密码</**td**>  <**td**>年龄</**td**>  </**tr**>  </**thead**>  <**tbody**>  {% **for user in users** %}  <**tr**>  <**td**>{{ **user**.**id** }}</**td**>  <**td**>{{ **user**.**name** }}</**td**>  <**td**>{{ **user**.**password** }}</**td**>  <**td**>{{ **user**.**age** }}</**td**>  </**tr**>  {% **endfor** %}  </**tbody**>  </**table**> </**div**> <**script src="**{% **static 'js/jquery3.7.1.min.js'** %}**"**></**script**> <**script src="**{% **static 'plugins/bootstrap-3.4.1/js/bootstrap.js'** %}**"**></**script**> </**body**> </**html**> |

### 效果

|  |
| --- |
|  |

## 4.实现一个添加用户的路由add/,在urls中添加这个路由

|  |
| --- |
|  |

## 5.然后在views.py在添加这个addNew函数,这个函数需要处理get和post请求,一开始在浏览器地址栏在输入路径是get请求,返回一个新增用户页面,然后我们先不写post请求处理,只是返回一个字符串,因为我们需要先做这个模板html文件

|  |
| --- |
|  |

## 6.然后我们在template文件夹里面添加一个新增用户的add\_new.html文件

|  |
| --- |
| <!DOCTYPE **html**> <**html lang="en"**> <**head**>  <**meta charset="UTF-8"**>  <**title**>Title</**title**>  <**style**>  **body** {  **background-color**: **#f2f2f2**;  **font-family**: **Arial**, **sans-serif**;  **display**: **flex**;  **align-items**: **center**;  **justify-content**: **center**;  **height**: 100**vh**;  }   .**login-container** {  **background-color**: **#FFF**;  **border-radius**: 10**px**;  **box-shadow**: 0 4**px** 8**px rgba**(0, 0, 0, 0.1);  **padding**: 30**px**;  **width**: 300**px**;  }   .**login-container h2** {  **color**: **#333**;  **font-size**: 24**px**;  **margin-bottom**: 20**px**;  **text-align**: **center**;  }   .**login-form** {  **margin-bottom**: 20**px**;  }   .**login-form label** {  **display**: **block**;  **color**: **#555**;  **font-size**: 16**px**;  **margin-bottom**: 10**px**;  }   .**login-form input**[**type**=**"text"**],  .**login-form input**[**type**=**"password"**] {  **width**: 100%;  **padding**: 10**px**;  **border**: 1**px solid #DDD**;  **border-radius**: 4**px**;  **outline**: **none**;  }   .**login-button** {  **margin-top**: 10**px**;  **background-color**: **#E25822**;  **color**: **#FFF**;  **border**: **none**;  **border-radius**: 4**px**;  **cursor**: **pointer**;  **padding**: 10**px** 20**px**;  **font-size**: 16**px**;  **transition**: **background-color** 0.3**s ease**;  **width**: 100%;  }   .**login-button**:**hover** {  **background-color**: **#BF4316**;  }  </**style**> </**head**> <**body**>  <**div class="login-container"**>  <**h2**>登录</**h2**>  <**form class="login-form" method="POST" action="/add/"**>  {% **csrf\_token** %}  <**label for="username"**>用户名</**label**>  <**input type="text" name="username" placeholder="请输入用户名"**>  <**label for="password"**>密码</**label**>  <**input type="password" name="password" placeholder="请输入密码"**>  <**label for="username"**>年龄</**label**>  <**input type="text" name="age" placeholder="请输入年龄"**>  <**input type="submit" class="login-button" value="登录"** />  </**form**>    </**div**> </**body**> </**html**> |

## 7.运行程序,尝试一下,发现是能够返回页面的

|  |
| --- |
|  |

## 8.我们填写用户数据,然后点击添加用户信息,然后在addNew函数里面输入表单数据,看看能否获取到数据

|  |
| --- |
|  |

## 9.点击添加按钮后,效果如下

|  |
| --- |
|  |

### 然后控制台也有输入

|  |
| --- |
|  |

## 10,然后我们来获取数据并且把它插入到数据库中

|  |
| --- |
|  |

## 11.然后我们重新启动服务器,进入add路由,填写数据,然后点击添加按钮

|  |
| --- |
|  |
|  |

## 12.然后我们到navicat中去查看数据,发现用户被添加进来了

|  |
| --- |
|  |

### 不知道为什么,老师非常喜欢使用二级路由,可能是由好处的.我们以后要规范一点.

## 13.其实添加数据成功后返回字符串不太好,我们需要把路由重定向到用户列表

|  |
| --- |
|  |

## 14.然后我们重启服务器,填写数据,然后点击添加按钮,效果如下

|  |
| --- |
|  |
|  |

### 发现它跳转了,而且数据有添加进来了

|  |
| --- |
|  |

## 15.然后我们需要把我们的项目优化一下,在用户列表的页面里面添加一个按钮或者超链接,点击按钮会跳转到添加用户信息页面,这样子就很方便了

|  |
| --- |
|  |

## 16.重启服务器进入users/路由,然后点击超链接

|  |
| --- |
|  |
|  |

### 可见,它的确跳转了

## 17. 首先我们还是规范一下路由,改为跟老师的一样.

|  |
| --- |
|  |
|  |
|  |
|  |

## 18.然后我们来练习删除用户功能,我们的设计是这样子的

|  |
| --- |
|  |

### 1>在urls.py中添加一个info/delete/,把它和views.py里面deleteOne函数对应起来

|  |
| --- |
|  |

### 2>在views.py里面定义一个deleteOne函数

|  |
| --- |
|  |

#### 重启服务器,在浏览器在输入/info/delete/?nid=8,然后回车删除了,然后我们输入/info/delete/?nid=7,然后回车

|  |
| --- |
|  |
|  |

#### 发现id为7的用户也被删除了.功能是实现了.但是我们需要优化代码,不能在浏览器里面输入路径来删除,这样子太麻烦.

## 19.我们应该在用户里面的表格里面的每一行添加应该删除链接,然后跳转到上面类似的地址,我们可以修改一下all.html,添加删除操作

|  |
| --- |
|  |

## 20.然后我们点击id为6的用户的删除超链接

|  |
| --- |
|  |

### 发现数据成功删除了

|  |
| --- |
|  |

### 注意:修改功能笔记麻烦,我们现在先不学,以后我们会学习到