《Django Web框架教学笔记》

目录

admin 后台数据库管理

- django 提供了比较完善的后台管理数据库的接口，可供开发过程中调用和测试使用
- django 会搜集所有已注册的模型类，为这些模型类提拱数据管理界面，供开发者使用
- 使用步骤:
  1. 创建后台管理帐号:
     - 后台管理--创建管理员帐号
       - `$ python3 manage.py createsuperuser`
       - 根据提示完成注册,参考如下:

```
$ python3 manage.py createsuperuser
Username (leave blank to use 'tarena'): tarena  # 此处输入用户名
Email address: laowei@tedu.cn  # 此处输入邮箱
Password: # 此处输入密码(密码要复杂些，否则会提示密码太简单)
Password (again): # 再次输入重复密码
Superuser created successfully.
$
```

  2. 用注册的帐号登陆后台管理界面
     - 后台管理的登录地址:
       - http://127.0.0.1:8000/admin/

注册自定义模型类

- 若要自己定义的模型类也能在 `/admin` 后台管理界中显示和管理，需要将自己的类注册到后台管理界面

- 添加自己定义模型类的后台管理数据表的,需要用 `admin.site.register(自定义模型类)` 方法进行注册
  - 配置步骤如下:
    1. 在应用app中的admin.py中导入注册要管理的模型models类, 如:

    ```
    from .models import Book
    ```

    2. 调用 admin.site.register 方法进行注册,如:

    ```
    from django.contrib import admin
    admin.site.register(自定义模型类)
    ```

  - 如: 在 bookstore/admin.py 添加如下代码对Book类进行管理
  - 示例:

```
# file: bookstore/admin.py
from django.contrib import admin
# Register your models here.

from . import models
...
admin.site.register(models.Book)  # 将Book类注册为可管理页面
```

修改自定义模型类的展现样式

- 在admin后台管理数据库中对自定义的数据记录都展示为 `XXXX object` 类型的记录，不便于阅读和判断
- 在用户自定义的模型类中可以重写 `def __str__(self):` 方法解决显示问题,如:
  - 在 自定义模型类中重写 **str**(self) 方法返回显示文字内容:

```
class Book(models.Model):
    ...
    def __str__(self):
        return "书名" + self.title
```

模型管理器类

- 作用:
  - 为后台管理界面添加便于操作的新功能。
- 说明:
  - 后台管理器类须继承自 `django.contrib.admin` 里的 `ModelAdmin` 类
- 模型管理器的使用方法:
  1. 在 `<应用app>/admin.py` 里定义模型管理器类

```
class XXXXManager(admin.ModelAdmin):
    ......
```

  2. 绑定注册模型管理器和模型类

```
from django.contrib import admin
from .models import *
admin.site.register(YYYY, XXXXManager) # 绑定 YYYY 模型类与 管理器类 XXXXManager
```

- 示例:

```
# file : bookstore/admin.py
from django.contrib import admin
from .models import Book

class BookManager(admin.ModelAdmin):
    list_display = ['id', 'title', 'price', 'market_price']

admin.site.register(Book, BookManager)
```

  - 进入http://127.0.0.1:8000/admin/bookstore/book/ 查看显示方式和以前有所不同

- 模型管理器类ModelAdmin中实现的高级管理功能

1. list_display 去控制哪些字段会显示在Admin 的修改列表页面中。
2. list_display_links 可以控制list_display中的字段是否应该链接到对象的"更改"页面。
3. list_filter 设置激活Admin 修改列表页面右侧栏中的过滤器
4. search_fields 设置启用Admin 更改列表页面上的搜索框。
5. list_editable 设置为模型上的字段名称列表，这将允许在更改列表页面上进行编辑。
6. 其它参见https://docs.djangoproject.com/en/2.2/ref/contrib/admin/

再谈Meta类

通过Meta内嵌类 定义模型类的属性

- 模型类可以通过定义内部类class Meta 来重新定义当前模型类和数据表的一些属性信息
- 用法格式如下:

```
class Book(models.Model):
    title = CharField(....)
    class Meta:
        1. db_table = '数据表名'
            - 该模型所用的数据表的名称。(设置完成后需要立马更新同步数据库)
        2. verbose_name = '单数名'
            - 给模型对象的一个易于理解的名称(单数),用于显示在/admin管理界面中
        3. verbose_name_plural = '复数名'
            - 该对象复数形式的名称(复数),用于显示在/admin管理界面中
```

- 练习:
  - 将Author 模型类加入后台管理
  - 制作一个AuthorManager管理器类，让后台管理Authors列表中显示作者的ID、姓名、年龄信息
  - 用后台管理程序 添加三条 Author 记录
  - 修改其中一条记录的年龄 - Author
  - 删除最后一条添加的记录 - Author

数据表关联关系映射

- 常用的表关联方式有三种:
    1. 一对一映射
        - 如: 一个身份证对应一个人
    2. 一对多映射
        - 如: 一个班级可以有多个学生
    3. 多对多映射
        - 如: 一个学生可以报多个课程，一个课程可以有多个学生学习

一对一映射

- 一对一是表示现实事物间存在的一对一的对应关系。
- 如:一个家庭只有一个户主，一个男人有一个妻子，一个人有一个唯一的指纹信息等

语法

```
class A(model.Model):
    ...
class B(model.Model):
    属性 = models.OneToOneField(A, on_delete=xxx)
```

外键类字段选项

- 特殊字段参数【必须项】:
    - on_delete
        1. models.CASCADE 级联删除。 Django模拟SQL约束ON DELETE CASCADE的行为，并删除包含ForeignKey的对象。
        2. models.PROTECT 抛出ProtectedError 以阻止被引用对象的删除;[等同于mysql默认的RESTRICT]
        3. models.SET_NULL 设置ForeignKey null；需要指定null=True
        4. models.SET_DEFAULT 将ForeignKey设置为其默认值；必须设置ForeignKey的默认值。
        5. ... 其它参请参考文档 https://docs.djangoproject.com/en/2.2/ref/models/fields/#foreignkey
- 其余常用的字段选项【非必须项】；如:
    1. null
    2. unique 等

用法示例

1. 创建作家和作家妻子类

```
# file : xxxxxxxx/models.py
from django.db import models

class Author(models.Model):
    '''作家模型类'''
    name = models.CharField('作家', max_length=50)

class Wife(models.Model):
    '''作家妻子模型类'''
    name = models.CharField("妻子", max_length=50)
    author = models.OneToOneField(Author, on_delete=models.CASCADE)  # 增加一对一属性
```

2. 创建一对一的数据记录

```
from .models import *
author1 = Author.objects.create(name='王老师')
wife1 = Wife.objects.create(name='王夫人', author=author1)  # 关联王老师
author2 = Author.objects.create(name='小泽老师')  # 一对一可以没有数据对应的数据
```

3. 数据查询

    1. 正向查询

        - 直接通过关联属性查询即可

        ```
        # 通过 wife 找 author
        from .models import Wife
        wife = Wife.objects.get(name='王夫人')
        print(wife.name, '的老公是', wife.author.name)
        ```

    2. 反向查询

        - 通过反向关联属性查询
        - 反向关联属性为`实例对象.引用类名(小写)`，如作家的反向引用为`作家对象.wife`
        - 当反向引用不存在时，则会触发异常

        ```
        # 通过 author.wife 关联属性 找 wife,如果没有对应的wife则触发异常
        author1 = Author.objects.get(name='王老师')
        print(author1.name, '的妻子是', author1.wife.name)
        author2 = Author.objects.get(name='小泽老师')
        try:
            print(author2.name, '的妻子是', author2.wife.name)
        except:
            print(author2.name, '还没有妻子')
        ```

一对多映射

- 一对多是表示现实事物间存在的一对多的对应关系。
- 如:一个学校有多个班级,一个班级有多个学生, 一本图书只能属于一个出版社,一个出版社允许出版多本图书

1. 语法

    - 当一个A类对象可以关联多个B类对象时

    ```
    class A(model.Model):
        ...

    class B(model.Model):
        属性 = models.ForeignKey("一"的模型类, on_delete=xx)
    ```

2. 用法示例

    - 有二个出版社对应五本书的情况.

        1. `清华大学出版社`有书

            1. C++

    2. Java
    3. Python
2. `北京大学出版社` 有书
    1. 西游记
    2. 水浒

3. 创建模型类

```
# file: otm/models.py
from django.db import models

class Publisher(models.Model):
    '''出版社【一】'''
    name = models.CharField('名称', max_length=50, unique=True)

class Book(models.Model):
    '''书【多】'''
    title = models.CharField('书名', max_length=50)
    publisher = ForeignKey(Publisher, on_delete=models.CASCADE)
```

4. 创建数据

```
#先创建 '一' ，再创建 '多'
from .models import *
pub1 = Publisher.objects.create(name='清华大学出版社')
Book.objects.create(title='C++', publisher=pub1)
Book.objects.create(title='Java', publisher_id=1)

#高级创建 - 利用 反向属性
pub2 = Publisher.objects.create(name='北京大学出版社')
pub2.book_set.create(title='西游记')
```

5. 数据查询

通过 Book 查询 Publisher【正向】

```
通过 publisher 属性查询即可
book.publisher

abook = Book.objects.get(id=1)
print(abook.title, '的出版社是:', abook.publisher.name)
```

通过 Publisher 查询 对应的所有的 Book 【反向】

```
Django会在Publisher中增加一个属性来表示对对应的Book们的查询引用
属性:book_set  等价于 objects

# 通过出版社查询对应的书
pub1 = Publisher.objects.get(name='清华大学出版社')
books = pub1.book_set.all()  # 通过book_set 获取pub1对应的多个Book数据对象
#books = Book.objects.filter(publisher=pub1)  # 也可以采用此方式获取
print("清华大学出版社的书有:")
for book in books:
   print(book.title)
```

多对多映射

- 多对多表达对象之间多对多复杂关系，如: 每个人都有不同的学校(小学，初中，高中,...),每个学校都有不同的学生...

1. 语法
    - 在关联的两个类中的任意一个类中,增加:

```
属性 = models.ManyToManyField(MyModel)
```

2. 用法示例
    - 一个作者可以出版多本图书
    - 一本图书可以被多名作者同时编写

```
class Author(models.Model):
    ...

class Book(models.Model):
    ...
    authors = models.ManyToManyField(Author)
```

3. 创建模型类

```
class Author(models.Model):
    '''作家模型类'''
    name = models.CharField('作家', max_length=50)
    def __str__(self):
        return self.name

class Book(models.Model):
    '''书模型类'''
    title = models.CharField('书名', max_length=50)
    authors = models.ManyToManyField(Author)
    def __str__(self):
        return self.title
```

4. 创建数据

```
方案1 先创建 author 再关联 book
    author1 = Author.objects.create(name='吕老师')
    author2 = Author.objects.create(name='王老师')
    # 吕老师和王老师同时写了一本Python
    book11 = author1.book_set.create(title="Python")
    author2.book_set.add(book11)

方案2 先创建 book 再关联 author
    book = Book.objects.create(title='python1')
    #郭小闹和吕老师都参与了 python1 的 创作
    author3 = book.authors.create(name='guoxiaonao')
    book.authors.add(author1)
```

5. 数据查询
    1. 通过 Book 查询对应的所有的 Author【正向】

```
book.authors.all() -> 获取 book 对应的所有的author的信息
book.authors.filter(age__gt=80) -> 获取book对应的作者中年龄大于80岁的作者的信息
```

    2. 通过 Author 查询对应的所有的Book【反向】
    - Django会生成一个反向属性 book_set 用于表示对对应的book的查询对象相关操作

```
author.book_set.all()
author.book_set.filter()
```

cookies 和 session

- 会话 - 从打开浏览器访问一个网站，到关闭浏览器结束此次访问，称之为一次会话
- HTTP协议是无状态的，导致会话状态难以保持
- 试想一下，如果不保持会话状态，在电商网站购物的场景
  体验?

Cookies和Session就是为了保持会话状态而诞生的两个存储技术

cookies

- cookies是保存在客户端浏览器上的存储空间
- Chrome 浏览器 可能通过开发者工具的 `Application` >> `Storage` >> `Cookies` 查看和操作浏览器端所有的 Cookies 值
- 火狐浏览器 可能通过开发者工具的 存储 -> Cookie
    - cookies 在浏览器上是以键-值对的形式进行存储的，键和值都是以ASCII字符串的形存储(不能是中文字符串)
    - cookies中的数据是按域存储隔离的，不同的域之间无法访问
    - cookies 的内部的数据会在每次访问此网址时都会携带到服务器端，如果cookies过大会降低响应速度
- 在Django 设置浏览器的COOKIE 必须通过 HttpResponse 对象来完成
    - 添加、修改COOKIE
        - HttpResponse.set_cookie(key, value='', max_age=None, expires=None)

        - key:cookie的名字
        - value:cookie的值
        - max_age:cookie存活时间，秒为单位
        - expires:具体过期时间
        - 当不指定max_age和expires 时,关闭浏览器时此数据失效
    - 删除COOKIE
        - HttpResponse.delete_cookie(key)
        - 删除指定的key 的Cookie。 如果key 不存在则什么也不发生。
    - 获取cookie
        - 通过 request.COOKIES 绑定的字典(dict) 获取客户端的 COOKIES数据

```
value = request.COOKIES.get('cookies名', '默认值')
print("cookies名 = ", value)
```

- 示例
    - 以下示例均在视图函数中调用
    - 添加cookie

```
# 为浏览器添加键为 my_var1,值为123，过期时间为1个小时的cookie
responds = HttpResponse("已添加 my_var1,值为123")
responds.set_cookie('my_var1', 123, 3600)
return responds
```

    - 修改cookie

```
# 为浏览器添加键为 my_var1,修改值为456，过期时间为2个小时的cookie
responds = HttpResponse("已修改 my_var1,值为456")
responds.set_cookie('my_var1', 456, 3600*2)
return responds
```

    - 删除cookie

```
# 删除浏览器键为 my_var1的cookie
responds = HttpResponse("已删除 my_var1")
responds.delete_cookie('my_var1')
return responds
```

    - 获取cookie

```
# 获取浏览器中 my_var变量对应的值
value = request.COOKIES.get('my_var1', '没有值!')
print("cookie my_var1 = ", value)
return HttpResponse("my_var1:" + value)
```

session

- session又名会话控制，是在服务器上开辟一段空间用于保留浏览器和服务器交互时的重要数据
- 实现方式
    - 使用 session 需要在浏览器客户端启动 cookie，且用在cookie中存储sessionid

    - 每个客户端都可以在服务器端有一个独立的Session
    - 注意：不同的请求者之间不会共享这个数据，与请求者一一对应
- Django中配置Session
    - 在 settings.py 文件中
    - 向 INSTALLED_APPS 列表中添加：

```
INSTALLED_APPS = [
    # 启用 sessions 应用
    'django.contrib.sessions',
]
```

    - 向 MIDDLEWARE 列表中添加：

```
MIDDLEWARE = [
    # 启用 Session 中间件
    'django.contrib.sessions.middleware.SessionMiddleware',
]
```

- session的基本操作:
    - session对于象是一个类似于字典的SessionStore类型的对象, 可以用类拟于字典的方式进行操作
    - session 只能够存储能够序列化的数据,如字典，列表等。
    1. 保存 session 的值到服务器
        - `request.session['KEY'] = VALUE`
    2. 获取session的值
        - `VALUE = request.session['KEY']`
        - `VALUE = request.session.get('KEY', 缺省值)`
    3. 删除session的值
        - del request.session['KEY']`
- 在 settings.py 中有关 session 的设置
    1. SESSION_COOKIE_AGE
        - 作用: 指定sessionid在cookies中的保存时长(默认是2周)，如下:
        - `SESSION_COOKIE_AGE = 60 * 60 * 24 * 7 * 2`
    2. SESSION_EXPIRE_AT_BROWSER_CLOSE = True
       设置只要浏览器关闭时,session就失效(默认为False)
- 注: 当使用session时需要迁移数据库,否则会出现错误

```
python3 manage.py migrate
```

```
django 原生session 问题：
1，django_session表是 单表设计； 且该表数据量持续增持【浏览器故意删掉sessionid&过期数据未删除】
2，可以每晚执行 python3 manage.py clearsessions 【该命令可删除已过期的session数据】
```

Cookies vs session

存储位置：
C- 浏览器中　　s- 服务器中【mysql】
安全性：
C - 不安全　　s- 相对安全一些

不管C还是S ， 不要存储敏感数据 【密码】