# 《Django 教程》

- 讲师: 魏明择
- 时间: 2019

## 目录

### 数据表关联关系映射 Relationship Map

- 在关系型数据库中，通常不会把所有数据都放在同一张表中，这样做会额外占用内存空间，
- 在关系列数据库中通常用表关联来解决数据库。
- 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

1. 语法

```
在关联的两个类中的任何一个类中:
class A(model.Model):
    ...

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

2. 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)  # 增加一对一属性
```

2. 查询
    - 在 Wife 对象中,通过 author 属性找到对应的author对象
    - 在 Author 对象中,通过 wife 属性找到对应的wife对象
3. 创始一对一的数据记录

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

4. 一对一数据的相互获取
    1. 正向查询
        - 直接通过关联属性查询即可

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

    2. 反向查询
        - 通过反向引用属性查询
        - 反向引用属性为实例对象.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 引用属性 找 wife,如果没有对应的wife刚触发异常
author1 = models.Author.objects.get(name='王老师')
print(author1.name, '的妻子是', author1.wife.name)
author2 = models.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')
```

- 作用:
    - 主要是解决常用数据不常用数据的存储问题,把经常加载的一个数据放在主表中，不常用数据放在另一个副表中，这样在访问主表数据时不需要加载副表中的数据以提高访问速度提高效率和节省内存空间,如经常把书的内容和书名建成两张表，因为在网站上经常访问书名等信息，但不需要得到书的内容。

- 练习:

    1. 创建一个Wife模型类,属性如下
        1. name
        2. age
    2. 在Wife类中增加一对一关联关系,引用 Author
    3. 同步回数据库并观察结果

**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. 用法语法

    - 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(多对一中"一"的模型类, ...)
```

2. 外键类ForeignKey

    - 构造函数:

```
ForeignKey(to, on_delete, **options)
```

    - 常用参数:
        - 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;
            3. SET_NULL 设置ForeignKey null；只有null是True才有可能。
            4. 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://yiyibooks.cn/xx/Django_1.11.6/ref/index.html ForeignKey部分
        - `**options` 可以是常用的字段选项如:
            1. null
            2. unique等
            3. ...

3. 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

1. 定义一对多类

```
# file: myorm/models.py
from django.db import models
class Publisher(models.Model):
    '''出版社'''
    name = models.CharField('名称', max_length=50, unique=True)

class Book(models.Model):
    title = models.CharField('书名', max_length=50)
    publisher = models.ForeignKey(Publisher, null=True)
```

- 创建一对多的对象

```
# file: xxxxx/views.py
from . import models
pub1 = models.Publisher.objects.create(name='清华大学出版社')
models.Book.objects.create(title='C++', publisher=pub1)
models.Book.objects.create(title='Java', publisher=pub1)
models.Book.objects.create(title='Python', publisher=pub1)

pub2 = models.Publisher.objects.create(name='北京大学出版社')
models.Book.objects.create(title='西游记', publisher=pub2)
models.Book.objects.create(title='水浒', publisher=pub2)
```

- 查询:
    - 通过多查一

```
# 通过一本书找到对应的出版社
abook = models.Book.objects.get(id=1)
print(abook.title, '的出版社是:', abook.publisher.name)
```

    - 通过一查多

```
# 通过出版社查询对应的书
pub1 = models.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 = models.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)
```

- 练习:
    1. 完成Book 和 Publisher 之间的一对多
    2. 查看数据库效果
    3. 登录到后台,查看Book实体

3. 数据查询
    1. 通过 Book 查询 Publisher

```
通过 publisher 属性查询即可
练习:
    查询 西游记 对应的出版社信息,打印在终端上
```

    2. 通过 Publisher 查询 对应的所有的 Books

```
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(Entry.objects)
```

**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(Entry)
```

2. 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):
```

```
    xxxx xxxx

class Book(models.Model):
    xxxx xxxx

    authors = models.ManyToManyField(Author)
```

3. 数据查询

    1. 通过 Book 查询对应的所有的 Authors

    ```
    可以通过authors表示对应所有Author的查询对象

    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Books

    ```
    Django会生成一个属性 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()
    author.book_set.create(...)  # 创建新书并联作用author
    author.book_set.add(book)   # 添加已有的书为当前作者author
    author.book_set.clear()  # 删除author所有并联的书
    author.book_set.remove()  # 删除所author所有并联的书
    ```

4. 示例:

    - 多对多模型

```
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):
    title = models.CharField('书名', max_length=50)
    author = models.ManyToManyField(Author, null=True)
    def __str__(self):
        return self.title
```

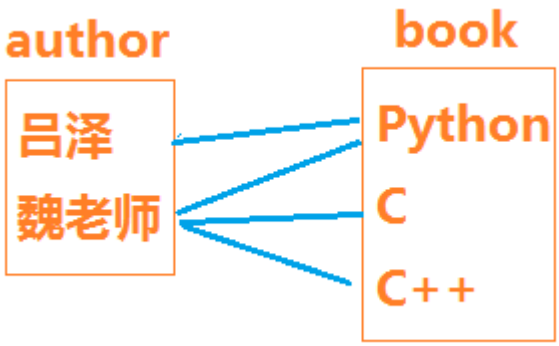
    - 多对多视图操作

```
from django.http import HttpResponse

from . import models

def many2many_init(request):
    # 创建两人个作者
    author1 = models.Author.objects.create(name='吕泽')
    author2 = models.Author.objects.create(name='魏老师')

    # 吕择和魏老师同时写了一本Python
    book11 = author1.book_set.create(title="Python")
    author2.book_set.add(book11)  #

    # 魏老师还写了两本书
    book21 = author2.book_set.create(title="C")  # 创建一本新书"C"
    book22 = author2.book_set.create(title="C++")  # 创建一本新书"C++"

    return HttpResponse("初始化成功")

def show_many2many(request):
    authors = models.Author.objects.all()
    for auth in authors:
        print("作者:", auth.name, '发出版了', auth.book_set.count(), '本
书: ')
        for book in books:
            print('    ', book.title)
    print("----显示书和作者的关系----")
    books = models.Book.objects.all()
    for book in books:
        auths = book.author.all()
        print(book.title, '的作者是:', '、'.join([str(x.name) for x in
auths]))
    return HttpResponse("显示成功，请查看服务器端控制台终端")
```

- 多对多最终的SQL结果

```
mysql> select * from myorm2_author;
+----+-----------+
| id | name      |
+----+-----------+
| 11 | 吕泽      |
| 12 | 魏老师    |
+----+-----------+
2 rows in set (0.00 sec)

mysql> select * from myorm2_book;
+----+--------+
| id | title  |
+----+--------+
| 13 | Python |
| 14 | C      |
```

```
| 15 | C++    |
+----+--------+
3 rows in set (0.00 sec)

mysql> select * from myorm2_book_author;
+----+---------+-----------+
| id | book_id | author_id |
+----+---------+-----------+
| 17 |      13 |        11 |
| 20 |      13 |        12 |
| 18 |      14 |        12 |
| 19 |      15 |        12 |
+----+---------+-----------+
4 rows in set (0.00 sec)
```




- 示例示意图

## cookies 和 session(会话)

**cookies**

- cookies是保存在客户端浏览器上的存储空间，通常用来记录浏览器端自己的信息和当前连接的确认信息
- 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)
- 在Django 服务器端来设置 设置浏览器的COOKIE 必须通过 HttpResponse 对象来完成
- HttpResponse 关于COOKIE的方法
  - 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

```
- 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。
```

- Django中的cookies

- 使用 响应对象HttpResponse 等 将cookie保存进客户端
  1. 方法1

```
from django.http import HttpResponse
resp = HttpResponse()
resp.set_cookie('cookies名', cookies值, 超期时间)
```

     - 如:

```
resp = HttpResponse()
resp.set_cookie('myvar', "weimz", 超期时间)
```

  2. 方法二, 使用render对象

```
from django.shortcuts import render
resp = render(request,'xxx.html',locals())
resp.set_cookie('cookies名', cookies值, 超期时间)
```

  3. 方法三, 使用redirect对象

```
from django.shortcuts import redirect
resp = redirect('/')
resp.set_cookie('cookies名', cookies值, 超期时间)
```

  3. 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', '没有值!')
print("cookies名 = ", value)
```

  4. 注:
     - 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

- cookies 示例

```
# file : <项目名>/urls.py
from . import views

urlpatterns = [
    url(r'^admin/', admin.site.urls),
    # 增删改cookie
```

```
    url(r'^add_cookie', views.add_cookie),
    url(r'^mod_cookie/(\d+)', views.mod_cookie),
    url(r'^del_cookie', views.del_cookie),
    url(r'^show_cookie', views.show_cookie),
]

# file : <项目名>/views.py
from . import views
from django.http import HttpResponse
def add_cookie(request):
    responds = HttpResponse("已添加mycookie_var1,值为123")
    responds.set_cookie('mycookie_var1', 123, 3600)
    return responds

def mod_cookie(request, new_value):
    responds = HttpResponse("已修改mycookie_var1,新值为"+new_value)
    responds.set_cookie('mycookie_var1', new_value, 3600)
    return responds

def del_cookie(request):
    responds = HttpResponse("已删除mycookie_var1")
    responds.delete_cookie('mycookie_var1')
    return responds

def show_cookie(request):
    value = request.COOKIES.get('mycookie_var1', '没有值!')
    print("cookie mycookie_var1 = ", value)
    return HttpResponse("mycookie_var1:" + value)
```

**session 会话**

- session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
- 每个客户端都可以在服务器端有一个独立的Session
- http协议是无状态的：每次请求都是一次新的请求，不会记得之前通信的状态
- 客户端与服务器端的一次通信，就是一次会话
- 实现状态保持的方式：在客户端或服务器端存储与会话有关的数据
- 推荐使用sesison方式，所有数据存储在服务器端，在客户端cookie中存储session_id
- 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
- 什么是session
    - session - 会话
    - 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
- Django启用Session
    - 在 settings.py 文件中

    - 项INSTALLED_APPS列表中添加:

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

    - 项MIDDLEWARE_CLASSES列表中添加:

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

- session的基本操作:
    - session对于象是一个在似于字典的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。
    - 保存 session 的值到服务器
        - `request.session[键] = 值`
        - 如: `request.session['KEY'] = VALUE`
    - 获取session的值
        - `VALUE = request.session['KEY']`
        - 或
        - `VALUE = request.session.get('KEY', 缺省值)`
    - 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
           SESSION_COOKIE_AGE = 60*30
        2. SESSION_EXPIRE_AT_BROWSER_CLOSE = True 设置只要浏览器关闭时,session就失效
- 注: 当使用session时需要迁移数据库,否则会出现错误

```
$ python3 manage.py makemigrations
$ python3 manage.py migrate
```

- session 示例

```
# file : <项目名>/urls.py
from . import  views

urlpatterns = [
    url(r'^admin/', admin.site.urls),
    # 增删改session
    url(r'^add_session', views.add_session),
    url(r'^mod_session/(\d+)', views.mod_session),
    url(r'^del_session', views.del_session),
    url(r'^show_session', views.show_session),
]

# file : <项目名>/views.py
from . import views
from django.http import HttpResponse
def add_session(request):
    request.session['mysession_var'] = 100
    responds = HttpResponse("添加session")
    return responds
def mod_session(request, new_value):
    request.session['mysession_var'] = new_value
    responds = HttpResponse("修改session成功")
    return responds
def del_session(request):
    try:
        del request.session['mysession_var']
        responds = HttpResponse("删除session成功")
    except:
        responds = HttpResponse("删除session失败")
    return responds
def show_session(request):
    mysession_var = request.session.get('mysession_var', '没有值!')
    print("mysession_var = ", mysession_var)
    return HttpResponse("mysession_var = " + str(mysession_var))
```

## 中间件 Middleware

- 中间件是 Django 请求/响应处理的钩子框架。它是一个轻量级的、低级的“插件”系统，用于全局改变 Django 的输入或输出。
- 每个中间件组件负责做一些特定的功能。例如，Django 包含一个中间件组件 AuthenticationMiddleware，它使用会话将用户与请求关联起来。
- 他的文档解释了中间件是如何工作的，如何激活中间件，以及如何编写自己的中间件。Django 具有一些内置的中间件，你可以直接使用。它们被记录在 built-in middleware reference 中。
- 中间件类:
    - 中间件类须继承自 `django.utils.deprecation.MiddlewareMixin`类
    - 中间件类须实现下列五个方法中的一个或多个:

        - `def process_request(self, request):` 执行视图之前被调用，在每个请求上调用，返回None或HttpResponse对象
        - `def process_view(self, request, callback, callback_args, callback_kwargs):` 调用视图之前被调用，在每个请求上调用，返回None或HttpResponse对象
        - `def process_response(self, request, response):` 所有响应返回浏览器之前被调用，在每个请求上调用，返回HttpResponse对象
        - `def process_exception(self, request, exception):` 当处理过程中抛出异常时调用，返回一个HttpResponse对象
        - `def process_template_response(self, request, response):` 在视图刚好执行完毕之后被调用，在每个请求上调用，返回实现了render方法的响应对象
    - 注： 中间件中的大多数方法在返回None时表示忽略当前操作进入下一项事件，当返回HttpResponese对象时表示此请求结果，直接返回给客户端

- 编写中间件类:

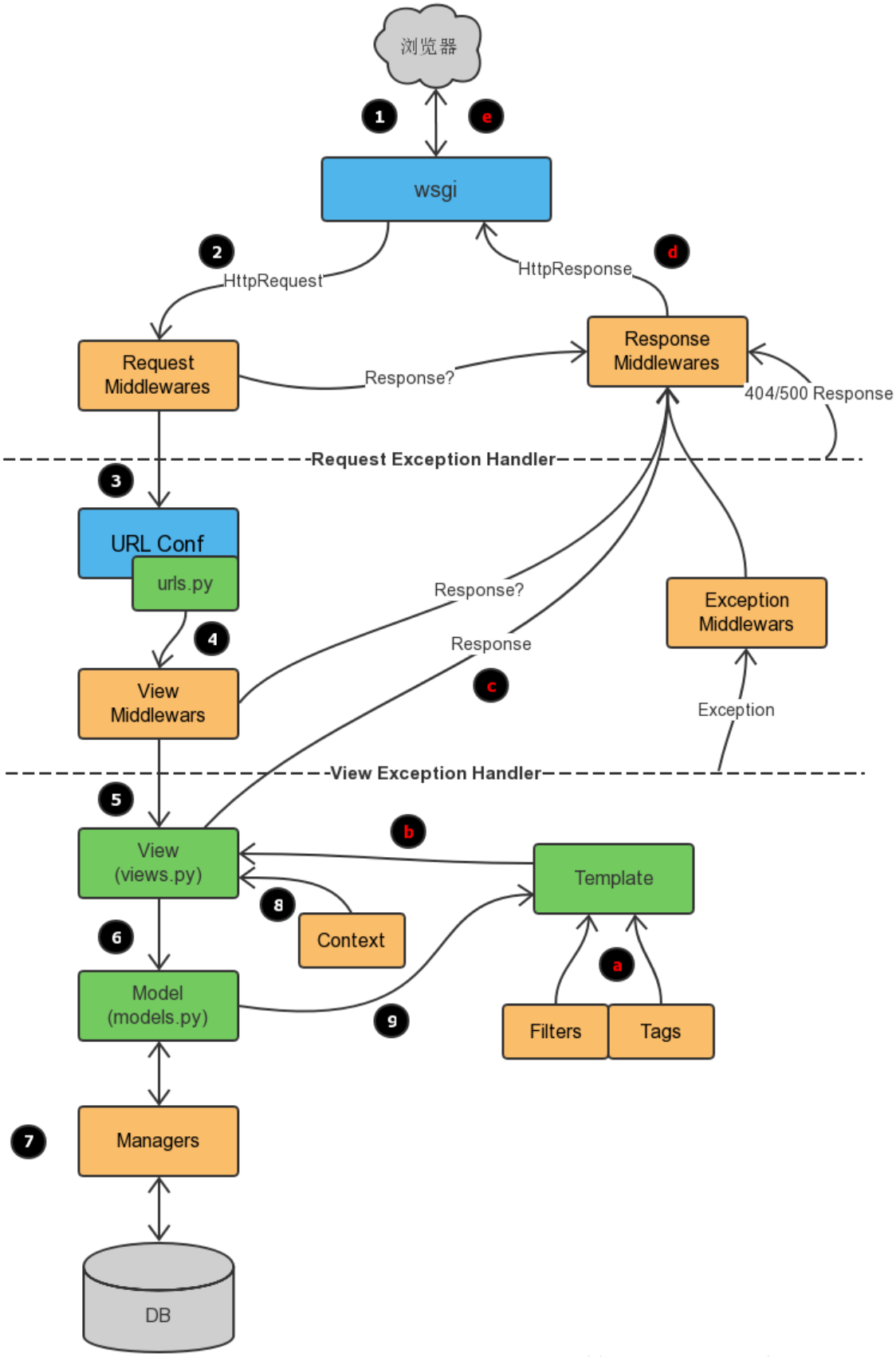
```
# file : middleware/mymiddleware.py
from django.http import HttpResponse, Http404
from django.utils.deprecation import MiddlewareMixin

class MyMiddleWare(MiddlewareMixin):
    def process_request(self, request):
        print("中间件方法 process_request 被调用")

    def process_view(self, request, callback, callback_args,
callback_kwargs):
        print("中间件方法 process_view 被调用")

    def process_response(self, request, response):
        print("中间件方法 process_response 被调用")
        return response

    def process_exception(self, request, exception):
        print("中间件方法 process_exception 被调用")

    def process_template_response(self, request, response):
        print("中间件方法 process_template_response 被调用")
        return response
```

- 注册中间件:

```
# file : settings.py
MIDDLEWARE = [
    ...
```

```
    'middleware.mymiddleware.MyMiddleWare',
]
```

- 中间件的执行过程



- 练习
  - 用中间件实现强制某个IP地址只能向/test 发送一次GET请求
  - 提示:
    - request.META['REMOTE_ADDR'] 可以得到远程客户端的IP地址
    - request.path_info 可以得到客户端访问的GET请求路由信息
  - 答案:

```
from django.http import HttpResponse, Http404
from django.utils.deprecation import MiddlewareMixin
import re
class VisitLimit(MiddlewareMixin):
    '''此中间件限制一个IP地址对应的访问/user/login 的次数不能改过10次,超过后禁止使用'''
    visit_times = {}  # 此字典用于记录客户端IP地址有访问次数
    def process_request(self, request):
        ip_address = request.META['REMOTE_ADDR']  # 得到IP地址
        if not re.match('^/test', request.path_info):
            return
        times = self.visit_times.get(ip_address, 0)
        print("IP:", ip_address, '已经访问过', times, '次!:', request.path_info)
        self.visit_times[ip_address] = times + 1
        if times < 5:
            return

        return HttpResponse('你已经访问过' + str(times) + '次，您被禁止了')
```

**跨站请求伪造保护 CSRF**

- 跨站请求伪造攻击
  - 某些恶意网站上包含链接、表单按钮或者JavaScript，它们会利用登录过的用户在浏览器中的认证信息试图在你的网站上完成某些操作，这就是跨站请求伪造。
- CSRF

```
Cross-Site Request Forgey
跨      站点    请求     伪装
```

- 说明:
  - CSRF中间件和模板标签提供对跨站请求伪造简单易用的防护。
- 作用:
  - 不让其它表单提交到此 Django 服务器
- 解决方案:
  1. 取消 csrf 验证(不推荐)
     - 删除 settings.py 中 MIDDLEWARE 中的 `django.middleware.csrf.CsrfViewMiddleware` 的中间件

2. 开放验证

```
在视图处理函数增加: @csrf_protect
@csrf_protect
def post_views(request):
    pass
```

3. 通过验证

```
需要在表单中增加一个标签
{% csrf_token %}
```

- 练习: 项目的注册部分
    1. 创建一个数据库 - FruitDay
    2. 创建实体类 - Users
        1. uphone - varchar(11)
        2. upwd - varchar(50)
        3. uemail - varchar(245)
        4. uname - varchar(20)
        5. isActive - tinyint 默认值为1 (True)
    3. 完善注册 - /register/
        1. 如果是get请求,则去往register.html
        2. 如果是post请求,则处理请求数据 将提交的数据保存回数据库

## Django中的forms模块

- 在Django中提供了 forms 模块,用forms 模块可以自动生成form内部的表单控件,同时在服务器端可以用对象的形式接收并操作客户端表单元素，并能对表单的数据进行服务器端验证

1. forms模块的作用
    - 通过 forms 模块,允许将表单与class相结合， 允许通过 class 生成表单
2. 使用 forms 模块的步骤
    1. 在应用中创建 forms.py
    2. 导入 django 提供的 forms
        - from django import forms
    3. 创建class,一个class会生成一个表单
        - 定义表单类

```
class ClassName(forms.Form):
    ...
```

4. 在 class 中创建类属性
    - 一个类属性对应到表单中是一个控件
5. 利用Form 类型的对象自动成表单内容
6. 读取form表单并进行验证数据

3. forms.Form 的语法
    - 属性 = forms.Field类型(参数)

1. 类型

```
class XXX(froms.Form):
    forms.CharField() : 文本框 <input type="text">
    forms.ChoiceField() : 下拉选项框 <select>
    forms.DateField() : 日期框 <input type="date">
    ... ...
```

    - 参见:<>
2. 参数
    1. label
        - 控件前的文本
    2. widget
        - 指定小部件
    3. initial
        - 控件的初始值(主要针对文本框类型)
    4. required
        - 是否为必填项，值为(True/False)

- form 表单示例
    - 手动实现Form 表单

```
<form action="/test_form1" method="post">
    <div>
        <label for="id_input_text">请输入内容:</label> <input
type="text" name="input_text" id="id_input_text" />
    </div>
    <button type="submit">提交</button>
</form>
```

    - Django Form 实现 Form 表单

```
class MySearch(forms.Form):
    input_text = forms.CharField(label = '请输入内容')
```

4. 在模板中解析form对象

    1. 方法

        1. 需要自定义
        2. 表单中的按钮需要自定义

    2. 解析form对

```
在 视图中创建form对象并发送到模板中解析.
ex:
    form = XXXForm()
    return render(request,'xx.html',locals())
```

        1. 手动解析 {% for field in form %} field : 表示的是form对象中的每个属性(控件) {{field.label}} : 表示的是label参数值 {{field}} : 表示的就是控件 {% endfor %}

    3. 自动解析

        1. {{form.as_p}}
           将 form 中的每个属性(控件/文本)都使用p标记包裹起来再显示
        2. {{form.as_ul}}

```
将 form 中的每个属性(控件/文本)都使用li标记包裹起来再显示
注意:必须手动提供ol 或 ul 标记
```

        3. {{form.as_table}}

```
将 form 中的每个属性(控件/文本)都使用tr标记包裹起来再显示
注意:必须手动提供table标记
```

    - 练习:
        1. 创建一个注册Form类 - RegisterForm
            - username - 用户名称
            - password - 用户密码(文本框)
            - password2 - 重复用户密码(文本框)
            - phonenumber - 用户年龄(数字框)
            - email - 电子邮箱 2.创建 register 路由
            - get 请求 :
                - 创建 RegisterForm 对象并发送到 模板register.html中显示
            - post 请求:
                - 接收13-register.html 中的数据并输出
    5. 通过 forms 对象获取表单数据
        1. 通过 forms.Form 子类的构造器来接收 post 数据

        - form = XXXForm(request.POST)
    2. 必须是 form 通过验证后,才能取值
        - form.is_valid()
            - 返回True:通过验证,可以取值
            - 返回False:暂未通过验证,则不能取值
    3. 通过 form.cleaned_data 字典的属性接收数据
        - form.cleaned_data : dict 类型

7. Field 内置小部件 - widget

    1. 什么是小部件
        - 表示的是生成到网页上的控件以及一些其他的html属性

    ```
    message=forms.CharField(widget=forms.Textarea)
    upwd=forms.CharField(widget=forms.PasswordInput)
    ```

    2. 常用的小部件类型

| **widget名称** | **对应和type类值** |
| --- | --- |
| TextInput | type='text' |
| PasswordInput | type='password' |
| NumberInput | type="number" |
| EmailInput | type="email" |
| URLInput | type="url" |
| HiddenInput | type="hidden" |
| CheckboxInput | type="checkbox" |
| CheckboxSelectMultiple | type="checkbox" |
| RadioSelect | type="radio" |
| Textarea | textarea标记 |
| Select | select标记 |
| SelectMultiple | select multiple 标记 |

8. 小部件的使用

    1. 继承自forms.Form
        1. 基本版
            1. 语法

    ```
    属性 = forms.CharField() #无预选值使用
        text,password,email,url,textarea,checkbox
    属性 = forms.ChoiceField() #有预选值使用
    ```

```
    checkbox,radio,select

属性 = forms.CharField(
    label='xxx',
    widget=forms.小部件类型
)
```

2. 示例:

```
upwd = forms.CharField(
    label='用户密码',
    widget=forms.PasswordInput
)

message = forms.CharField(
    label='评论内容',
    widget=forms.Textarea
)
```

2. 高级版
    1. 特征
        - 在指定控件类型的基础之上还能指定控件的一些html属性值
    2. 语法

```
    属性 = forms.CharField(
        label='xxx',
        widget=forms.小部件类型(
            attrs={
                'html属性名':'值',
                'html属性名':'值',
            }
        )
    )
```

- 文档参见https://yiyibooks.cn/xx/Django_1.11.6/topics/forms/index.html#forms-in-django

**Django之form表单验证**

- django form 提供表单和字段验证
- 当在创建有不同的多个表单需要提交的网站时，用表单验证比较方便验证的封装
- 当调用form.is_valid() 返回True表示当前表单合法，当返回False说明表单验证出现问题
- 验证步骤:
    1. 先对form.XXXField() 参数值进行验证，比如:min_length,max_length, validators=[...],如果不符合form.is_valid()返回False

2. 对各自from.clean_zzz属性名(self): 方法对相应属性进行验证,如果验证失败form.is_valid()返回False
3. 调胳form.clean(self): 对表单的整体结构进行验证， 如果验证失败form.is_valid()返回False
4. 以上验证都成功 form.is_valid()返回True

- 验证方法:
  - validators = [验证函数1, 验证函数1]
    - 验证函数验证失败抛出forms.ValidationError
    - 验证成功返回None
  - def clean_xxx属性(self):
    - 验证失败必须抛出forms.ValidationError
    - 验证成功必须返回xxx属性的值
  - def clean(self):
    - 验证失败必须抛出forms.ValidationError
    - 验证成功必须返回 self.cleaned_data
- 文档参见https://yiyibooks.cn/xx/Django_1.11.6/topics/forms/index.html#forms-in-django
- 验证示例

```
from django import forms
import re

mobile_re = re.compile(r'^(13[0-9]|15[012356789]|17[678]|18[0-9]|14[57])
[0-9]{8}$')
def mobile_validate(value):
    if not mobile_re.match(value):
        raise forms.ValidationError('手机号码格式错误')

class RegisterForm(forms.Form):
    username = forms.CharField(label='用户名')
    password = forms.CharField(label='请输入密码',
widget=forms.PasswordInput)
    password2 = forms.CharField(label='再次输入新密码',
widget=forms.PasswordInput)
    mobile = forms.CharField(label='电话号码', validators=[mobile_validate])

    def clean(self):
        pwd1 = self.cleaned_data['password']
        pwd2 = self.cleaned_data['password2']
        if pwd1 != pwd2:
            raise forms.ValidationError('两次密码不一致!')
        return self.cleaned_data  # 必须返回cleaned_data

    def clean_username(self):
        username = self.cleaned_data['username']
        if len(username) < 6:
            raise forms.ValidationError("用户名太短")
        return username
```

- 练习，写一个RegisterForm表单类型,要求如下四个属性:
  - username - 用户名称
    - 用户名只能包含[a-zA-Z_0_9]范围内的5~30个英文字符
  - password - 用户密码(文本框)
    - 任意字符，不能少于6个字符
  - password2 - 重复用户密码(文本框)
    - 任意字符，不能少于6个字符且必须与 password一致
  - phonenumber - 用户年龄(数字框)
    - 必须符合`r'^(13[0-9]|15[012356789]|17[678]|18[0-9]|14[57])[0-9]{8}$'`正规表达式