django

1. 简述http协议和常用请求头

```
http协议：（基于TCP/IP通信协议来传递数据（HTML 文件， 图片文件， 查询结果等））
    HTTP协议是Hyper Text Transfer Protocol（超文本传输协议）的缩写,是用于从万维网（WWW:World
Wide Web ）服务器传输超文本到本地浏览器的传送协议
```

```
HTTP是无连接：无连接的含义是限制每次连接只处理一个请求。服务器处理完客户的请求，并收到客户的应答后，即断开连接。采用这种方式可以节省传输时间。
HTTP是媒体独立的：这意味着，只要客户端和服务器知道如何处理的数据内容，任何类型的数据都可以通过HTTP发送。客户端以及服务器指定使用适合的MIME-type内容类型。
HTTP是无状态：HTTP协议是无状态协议。无状态是指协议对于事务处理没有记忆能力。缺少状态意味着如果后续处理需要前面的信息，则它必须重传，这样可能导致每次连接传送的数据量增大。另一方面，在服务器不需要先前信息时它的应答就较快
```

常用请求头

```
Accept :指定客户端能够接收的内容类型    |  Accept: text/plain, text/html
Cache-Control: 指定请求和响应遵循的缓存机制   | Cache-Control: no-cache
Cookie :HTTP请求发送时，会把保存在该请求域名下的所有cookie值一起发送给web服务器 | Cookie: $Version=1;
Skin=new;
Content-Type:请求的与实体对应的MIME信息 | Content-Type: application/x-www-form-urlencoded
Host: 指定请求的服务器的域名和端口号
Referer: 先前网页的地址，当前请求网页紧随其后,即来路
User-Agent : 发起请求的用户的身份标识
```

2. 列举常用的请求方法

```
GET:请求指定的页面信息,并返回实体主体
HEAD:类似于get请求,只不过返回的响应中没有具体内容,用于获取报头
POST:向指定的资源提交数据进行请求处理,数据被包含在请求体中,POST请求可能导致新的资源的建立或已有资源的修改
put:从客户端向服务器传送的数据取代指定的文档的内容
DELETE:请求服务器删除指定的内容
CONNECT:Http/1.1 协议中预留给能将连接改为管道方式的代理服务器
OPTIONS:允许客户端查看服务器的性能
TRACE:回显服务器收到的请求,主要用于测试和诊断
```

3.列举常见的http状态码

```
1xx：指示信息–表示请求已接收，继续处理。
2xx：成功–表示请求已被成功接收、理解、接受。
3xx：重定向–要完成请求必须进行更进一步的操作。
4xx：客户端错误–请求有语法错误或请求无法实现。
5xx：服务器端错误–服务器未能实现合法的请求
```

```
200 OK: 客户端请求成功,一般用于GET和POST请求
400 Bad Request: 客户端请求有语法错误，不能被服务器所理解。
301 Moved Permanently:永久移动,请求的资源已被永久移动到新url,返回信息会包含新的url,浏览器会自动定向到新url
401 Unauthorized: 请求未经授权，这个状态代码必须和WWW-Authenticate报头域一起使用。
403 Forbidden: 服务器收到请求，但是拒绝提供服务。
404 Not Found: 请求资源不存在，举个例子：输入了错误的URL。
500 Internal Server Error: 服务器发生不可预期的错误。
502 Bad Gateway: 充当网关或代理的服务器,从远端接收到一个无效的请求
503 Server Unavailable: 服务器当前不能处理客户端的请求，一段时间后可能恢复正常
```

4. http与https的区别

```
http:
    超文本传输协议,明文传输
    80端口
    连接简单且是无状态的
https:
    需要到ca申请证书,要费用
    具有安全性的ssl加密传输协议
    端口是443
    https协议是有ssl+http协议构建的可进行加密传输,身份认证的网络协议,安全
```

5.websocket协议以及原理

```
WebSocket 是 HTML5 开始提供的一种在单个 TCP 连接上进行全双工通讯的协议。


在WebSocket API 中，浏览器和服务器只需要完成一次握手，两者之间就直接可以创建持久性的连接，
并进行双向数据传输。
旧--很多网站为了实现推送技术，所用的技术都是 Ajax 轮询。轮询是在特定的的时间间隔（如每1秒），由浏览器对服务器发出HTTP请求，然后由服务器返回最新的数据给客户端的浏览器。这种传统的模式带来很明显的缺点，即浏览器需要不断的向服务器发出请求，然而HTTP请求可能包含较长的头部，其中真正有效的数据可能只是很小的一部分，显然这样会浪费很多的带宽等资源。

新---HTML5 定义的 WebSocket 协议，能更好的节省服务器资源和带宽，并且能够更实时地进行通讯。浏览器通过 JavaScript 向服务器发出建立 WebSocket 连接的请求，连接建立以后，客户端和服务器端就可以通过 TCP 连接直接交换数据。
```

6.django如何实现websocket

7.python web开发中跨域问题的解决思路

```
使用JSONP
    Web页面上调用js的script脚本文件不受跨域影响
    创建一个回调函数，然后在远程服务上调用这个函数并且将JSON 数据形式作为参数传递，完成回调
    jquery中可以使用jquery提供的getJSON方法来发送JSONP请求获取数据 （随机产生一个函数名随请求送到服务器,服务器将JSON数据填充进回调函数,JSONP请求结束后执行回调函数）
缺点: 前后端都要支持,只能发个get请求
```

CORS解决跨域
    处理简单请求: 只需要在返回的响应头中加上 Access-Control-Allow-Origin 字段.并且把该字段的值设置为跨域请求的来源地址或简单的设置为 *
    处理非简单请求:自定义一个中间件,在后端简单的给响应对象添加上 常用请求方法（PUT、DELETE）的支持

```
from django.utils.deprecation import MiddlewareMixin
class CorsMiddleware(MiddlewareMixin):
    def process_response(self, request, response):
        # 给响应头加上 Access-Control-Allow-Origin 字段 并简单的设置为 *
        response['Access-Control-Allow-Origin'] = '*'
        if request.method == 'OPTIONS':
            # 允许发送 PUT 请求
            response['Access-Control-Allow-Methods'] = 'PUT, DELETE'
            # 允许在请求头中携带 Content-type字段，从而支持发送json数据
            response['Access-Control-Allow-Headers'] = 'Content-type'
        return response
```

CORS解决方案,使用django-cors-headers 已封装,在settings文件中进行相关配置

8.请简述http缓存机制

强制缓存，服务器通知浏览器一个缓存时间，在缓存时间内，下次请求，直接用缓存，不在时间内，执行比较缓存策略。
比较缓存，将缓存信息中的Etag和Last-Modified通过请求发送给服务器，由服务器校验，返回304状态码时，浏览器直接使用缓存。

9.谈谈你所知道的python web框架

django,falsk,tornado

10.django,flask,tornado的比较

django 大而全,内部集成了许多的组件和功能 适用于大型的项目
flask 轻量级且第三方组件丰富

11.什么是Wsgi

wsgi server（比如uWSGI） 要和 wsgi application（比如django ）交互，uwsgi需要将过来的请求转给django 处理，那么uWSGI 和 django的交互和调用就需要一个统一的规范，这个规范就是WSGI WSGI（Web Server Gateway Interface）
    WSGI，全称 Web Server Gateway Interface，或者 Python Web Server Gateway Interface ，是为 Python 语言定义的 Web 服务器和 Web 应用程序或框架之间的一种简单而通用的接口

    WSGI 的官方定义是，the Python Web Server Gateway Interface。从名字就可以看出来，这东西是一个 Gateway，也就是网关。网关的作用就是在协议之间进行转换

    WSGI 是作为 Web 服务器与 Web 应用程序或应用框架之间的一种低级别的接口

12.列举django的内置组件

13.简述django的缓存机制

缓存是将一个或某个views的返回值保存至内存或者memcache等中,一定时间后当再有人来访问时候,则不再去执行view中的操作,而是直接从内存或者redis中之前缓存的内容拿到,并返回
django中的6种缓存机制
    开发调试
    内存
    文件
    数据库
    Memcache缓存（python-memcached模块）
    Memcache缓存（pylibmc模块
    另外：基于redis实现的缓存

应用方面：
    全站应用
    单独视图的缓存
    局部视图的应用

14.django中model的SlugField类型字段有什么用途

15.Django常见的线上部署方式

Nginx+uwsgi
    nginx作为服务器最前端，负责接收client的所有请求，统一管理。静态请求由Nginx自己处理。
    非静态请求通过uwsgi传递给Django，由Django来进行处理，从而完成一次WEB请求

16.django对数据查询结果排序怎么做,降序怎么做

```
user = Users.objects.order_by('id')
user = Users.objects.order_by('id')[0:1]
# 如果需要逆序 在字段前加负号 例  ('-id')
```

17.django中使用memcached作为缓存的具体方法,优缺点说明

优点: Memcached是Django原生支持的缓存系统，速度快，效率高

缺点:基于内存的缓存系统有个明显的缺点就是断电数据丢失

18.django中orm如何查询id不等于5 的元素

```
User.objects.filter().exclude(id=5)     # 查询id不为5的用户
```

19.把sql语句转化成python代码

```
select * from company where title like "%abc%" or mecount>999 order by createtime desc;
```

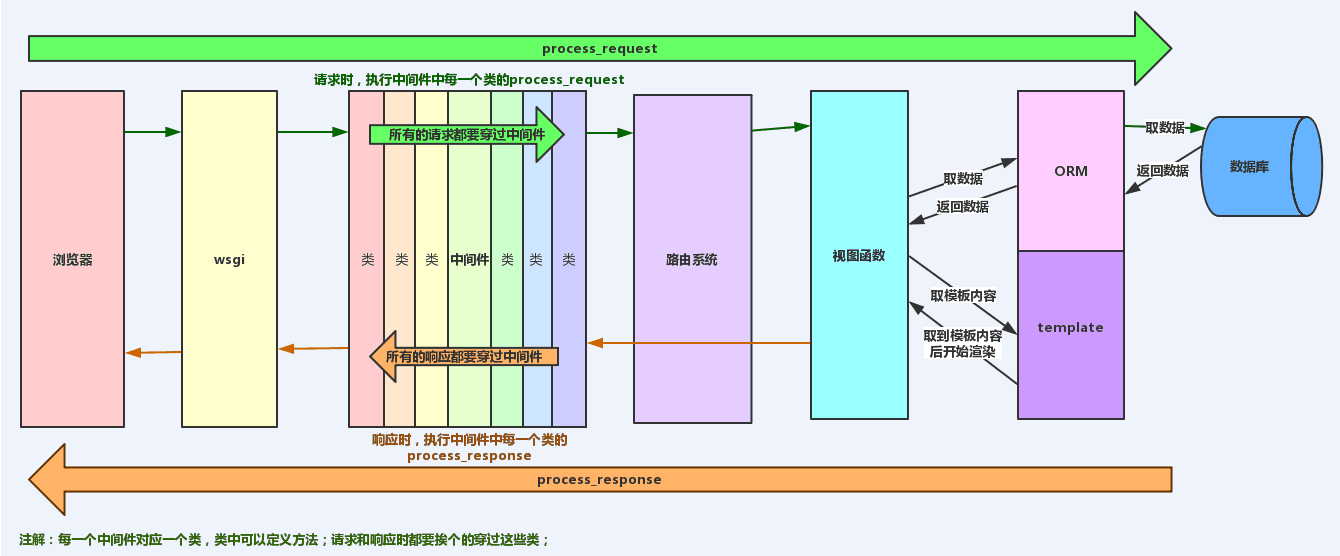
```
objects.filter(Q(title__contains='abc')|Q(mecount__gh=999)).order_by(-createtime)
```

20.从输入http://www.baidu.com到页面返回,中间都是发生了什么?

浏览器向 DNS 服务器请求解析该 URL中的域名所对应的 IP 地址;
解析出 IP 地址后，根据该 IP 地址和默认端口 80，和服务器建立TCP连接;
浏览器发出读取文件(URL中域名后面部分对应的文件)的HTTP 请求，该请求报文作为 TCP 三次握手的第三个报文的数据发送给服务器;
服务器对浏览器请求作出响应，并把对应的 html 文本发送给浏览器;
释放 TCP连接,四次挥手;
浏览器将该 html 文本并显示内容;

21.django 请求的生命周期

wsgi—中间件—路由----视图—中间件—wsgi



22.django中如何在 model保存前做一定的固定操作,比如写一句日志?

使用django信号实现:

23.简述django的中间件以及应用场景

```
中间件的概述:
中间件是一个轻量,低级别的的插件系统,用于在全局范围内改变django的输入和输出,实际作用就是在视图函数之执行之前后执行之后做一些额外的操作。本质上是一个自定义类,类中定义几个方法,Django框架会在处理请求的特定的时间去执行这些方法

应用场景:
    django项目中默认启用了csrf保护,每次请求时通过CSRF中间件检查请求中是否有正确的token值
    当用户在页面上发送请求时,通过自定义的认证中间件,判断用户是否已登录,未登录跳转登录页面
    当有用户请求过来时，判断用户是否在白名单或者在黑名单里
    非同源策略中跨域问题,使用CORS解决跨域问题时候,需要自己定义中间件或者使用 django-cors-headers需要在中间中进行配置
    当要全站使用缓存的情形,需要在中间件中进行配置
    .......
```

24.简述django的FBV和CBV

```
FBV:function base view  函数处理请求
    通过在view.py里面定义一个函数，然后通过函数的request参数获取method的类型，然后根据提交的不同方式进行不同的处理。
```

```
CBV:class base view 类处理请求
    通过自定义一个子类继承这个类，我们可以利用View里面的dispatch () 方法来判断请求是get还是post。dispatch方法的本质是一个反射器，可以根据名字来调用同名的方法

注:子类里面可以重新定义了dispatch这个方法，同时使用super () 来调用父类的构造方法，这样的好处是功能不变，但是可以灵活的添加新的功能
class Login(View):

    def dispatch(self, request, *args, **kwargs):
        print('before')
        obj = super(Login,self).dispatch(request, *args, **kwargs)
        print('after')
        return obj
```

25.如何给django CBV的函数设置添加装饰器

```
from django.utils.decorators import method_decorator
#将装饰器加在某个get/post的方法上
@method_decorator(xxx)
def get(self, request):
```

```
#将装饰器加在自己定义的的dispatch方法上
@method_decorator(xxx)
def dispatch(self, request, *args, **kwargs):
```

```
@method_decorator(xxx, name='post')
@method_decorator(xxx, name='get')
class AddPublisher(View):
```

```
#或者可以在url上加装饰器 -- 会为类视图中的所有请求方法都加上装饰器行为
urlpatterns = [
  url(r'^demo/$', xxx(DemoView.as_view()))
]
```

26.django如何连接多个数据库并实现读写分离?

多个数据库

python manage.py makemigrations

python manage.py migrate -- database db2 //settings配置多个数据库

读写分离

- 手动

```
s = models.Student.objects.using('default').all()

models.Student.objects.using('db2').create(name='xxx',)
```

- 自动

```
# myrouter
class Router:
    def db_for_read(self, model, **kwargs):
        return 'default'

    def db_for_write(self, model, **kwargs):
        return 'db2'
```

settings中配置

DATABASE_ROUTERS = ['myrouter.Router']

一主多从

```
# 一主多从
import random


class Router:
    def db_for_read(self, model, **kwargs):
        return random.choices(['db1', 'db2', 'db3'])

    def db_for_write(self, model, **kwargs):
        return 'db'
```

分库分表

```
# 分库分表

# app01  db1     app02 db2
```

```
class Router:
    def db_for_read(self, model, **kwargs):

        if model._meta.app_label == 'app01':
            return 'db1 '
        elif model._meta.app_label == 'app02':
            return 'db2'

    def db_for_write(self, model, **kwargs):
        if model._meta.app_label == 'app01':
            return 'db1 '
        elif model._meta.app_label == 'app02':
            return 'db2'
```

27.列举 django orm中你了解的所有方法?

```
---------必知必会13条----------
#获取对象列表
    all()            --获取所有数据 (对象列表)
    filter()         --获取满足条件的所有对象(对象列表)    #获取一个对象,没有或者是多个的时候会报错
    exclude()        --获取不满足条件的所有对象 (对象列表)
    values()         --获取对象的字段名和值
    values_list()    --获取对象的值
    order_by()       --排序,默认升序,加 - 降序 (指定多个排序,按第一个升序,遇到相同时候再使用第二个)
    reverse()        --给已经排好序的结果倒叙排序   -->前提是得使用order_by后
    distinct()       -- 会对完全相同的内容去重(但不是某一字段相同就去重)
#获取对象:
    get()
    first()          --取第一个
    last()           --最后一个
#数字
    count()          --计数

#布尔值
    exists()         --判断是否存在,返回bool
```

```
---------单表双下划线----------
    __gt                  大于
    __gte                 大于等于
    __lt                  小于
    __lte                 小于等于
    __isnull              为空
    __in =[  ]            在列表中的所有对象
    __range=[1,10]        字段值的范围,左右都包含
    __contains            包含某一字母
    __icontains           大小写都会包含
    __startswith          以..开头
    __istartswith         以..开头,忽略大小写
    __endswith            以..结尾
    __isendswith          以..结尾,忽略大小写
```

```
#---------外键操作----------                    #描述多对一的关系
class Publisher(models.Model):
```

```
    name = models.CharField(max_length=32)
class Book(models.Model):
    publisher=models.ForeignKey('Publisher',on_delete=models.CASCADE,related_name='books)

#-------------------------------------------------------------------------------
#基于对象的查询
     #正向查询  -> 有外键的一侧,查关联对象
          book_obj = models.Book.objects.get(id =1)
          #book_obj.publisher          -> 关联的对象
          #book_obj.publisher_id       -> 关联对象的id
          #book_obj.publisher.id       -> 关联对象的id
      #反向查询     被关联对象,查有外键的一侧
           publiser_obj = models.Publisher.objects.get(id=1)
           #publisher_obj.book_set      ->管理对象（所有书籍）
           #publisher_obj.book_set.all()      ->出版社所管理的所有书籍对象（所有书籍）
           #另外:当指定 relate_name = books"   publisher_obj.books      ->管理对象（所有书籍）

# 基于字段的查询
       models.Book.objects.filter(publisher__name ='xxxx')
```

```
#---------多对多----------
方式一
class Author():
    name = models.CharField(max_length=32)
    books = models.ManyToManyField('Book')

方式二
class Author(models.Model):
    name = models.CharField(max_length=32)

class Author_Book(models.Model):
    author = models.ForeignKey('Author')
    book = models.ForeignKey('Book')
    date = models.DateTimeField()

方式三
class Author(models.Model):
    name = models.CharField(max_length=32)
    books = models.ManyToManyField('Book', through='Author_Book')

class Author_Book(models.Model):
    author = models.ForeignKey('Author')
    book = models.ForeignKey('Book')
    date = models.DateTimeField()

# ------------------------------ 查询方法 ------------------------------
#基于对象的查询
    #正向查询:
        author_obj = models.Author.objects.get(id=2)
        #author_obj.books          -> 管理对象
        #author_obj.books.all()    -> 该作者写的所有书籍对象
   #反向查询
        book_obj = models.Book.objects.get(id=2)
        #不指定related_name ='authors'
```

```
        # print(book_obj.author_set.all())

        #指定related_name ='authors'
        # print(book_obj.authors.all())   [<Author: Author object>, <Author: Author
object>]>  因为作者没有定义str方法
# ------------------------------ 其他方法 ----------------------------------

#set 设置多对多关系
#全部重新设置id= 2 的三条数据全部清空
author_obj.books.set([])
author_obj.books.set([1,2]) #y要关联对象的id（就是给id为1和id为2 的两本书设置多对多）

#add 添加对对多关系
author_obj.books.add(3)   #要关联对象的id
author_obj.books.add(models.Book.objects.get(id=3))   #要关联的对象

#remove 删除多对多的关系
author_obj.books.remove(3)   #要关联对象的id
author_obj.books.remove(models.Book.objects.get (id=2))  #要关联的对象

#clear()  #清空当前(id=2也就是author_obj对象)的多对多的关系
author_obj.books.clear()

#create()
author_obj.books.create(title='楚留香传奇')
book_obj.authors.create(name='古龙2')
```

28.django中的F的作用?

```
from django.db.models import F
#动态获取字段的值，可以在一条记录中比较两个字段的value



#查询库存大于50的书籍
ret = models.Book.objects.filter(store__gt=50).values()
# for i in ret:
#     print(i)

#第一种用法,动态的获取想要获取的字段（可以比较两列的值）
# 获取库存大于售出的书籍
ret2 = models.Book.objects.filter(store__gt=F('sale')).values()
for i in ret2:
    print(i)

#第二种用法  (使用update更新)
#库存不变时候,售出数量乘2
models.Book.objects.all().update(sale=F('sale')*2)

#注: update 和更新后使用save的区别
# update 它可以更新指定字段,可以是多个，而后者会将所有的字段做更新
```

29.django中的Q的作用?

```
from django.db.models import Q
ret = models.Book.objects.filter(Q(~Q(id__lt=3) | Q(id__gt=5))&Q(id__lt=4))
print(ret)

#    ~  取反
#    |  或
#    & 与 AND
```

30.django中如何执行原生SQL?

```
在Django中使用原生Sql主要有以下几种方式:

extra:结果集修改器，一种提供额外查询参数的机制(依赖model)
raw:执行原始sql并返回模型实例(依赖model)
直接执行自定义sql(不依赖model)
```

```
# ---------使用extra ----------
Book.objects.filter(publisher__name='广东人员出版社').extra(where=['price>50'])
Book.objects.filter(publisher__name='广东人员出版社',price__gt=50)
Book.objects.extra(select={'count':'select count(*) from hello_Book'})
```

```
#---------使用raw ----------
Book.objects.raw('select * from hello_Book')
Book.objects.raw("insert into hello_author(name) values('测试')")
rawQuerySet为惰性查询，只有在使用时生会真正执行
```

```
#--------自定义查询 ----------
执行自定义sql:
from django.db import connection
cursor=connection.cursor()
#插入操作
cursor.execute("insert into hello_author(name) values('郭敬明')")
#更新操作
cursor.execute('update hello_author set name='abc' where name='bcd'')
#删除操作
cursor.execute('delete from hello_author where name='abc'')
#查询操作
cursor.execute('select * from hello_author')
raw=cursor.fetchone() #返回结果行游标直读向前，读取一条
cursor.fetchall() #读取所有
```

31.only和 defer的区别?

```
only(field)  : 定义需要的字段(id保留)

defer(field): 排除不需要的字段  (id保留)
```

32.select_related和 prefetch_related的区别?

33.django中 values和 values list的区别?

```
values 字典列表,ValuesQuerySet查询集
<QuerySet [{'id': 2, 'name': '作者2', 'age': 24}, {'id': 3, 'name': '作者3', 'age': 35}]>

values_list 返回元祖列表,字段值
<QuerySet [(2, '作者2', 24), (3, '作者3', 35)]>
```

34.如何使用django orm批量创建数据?

```
objs_list = []
for i in range(100):
    obj = People('name%s'%i,18)  #models里面的People表
    objs_list.append(obj)  #添加对象到列表

People.objects.bulk_create(objs_list,100)  #更新操作
```

35.django的Form和 Mode Form的作用?

```
form  生成页面可用的HTML标签 | 对用户提交的数据进行校验 | 保留上次输入内容
Model Form 利用 Django 的 ORM 模型model创建Form(表单跟model关系密切)
```

36.django的Fom组件中,如果字段中包含 choices参数,请使用两种方式 实现数据源实时更新。

在使用选择标签时，需要注意choices的选项可以从数据库中获取，但是由于是静态字段 ***获取的值无法实时更新***，那么需要自定义构造方法从而达到此目的。

```
#重写构造函数
from django.forms import Form
from django.forms import widgets
from django.forms import fields

class MyForm(Form):
    user = fields.ChoiceField(
        # choices=((1, '上海'), (2, '北京'),),
        initial=2,
        widget=widgets.Select
    )

    def __init__(self, *args, **kwargs):
        super(MyForm,self).__init__(*args, **kwargs)
        # self.fields['user'].choices = ((1, '上海'), (2, '北京'),)
        # 或
        self.fields['user'].choices = models.Classes.objects.all().values_list('id','caption')
```

```
#利用ModelChoiceField字段,参数为queryset对象
from django import forms
from django.forms import fields
from django.forms import models as form_model

class FInfo(forms.Form):
    authors = form_model.ModelMultipleChoiceField(queryset=models.NNewType.objects.all())  # 多选
    # authors = form_model.ModelChoiceField(queryset=models.NNewType.objects.all())  # 单选
```

37.django的 Model I中的 ForeignKey字段中的 on_delete参数有什么作用?

```
class Author(models.Model):
    author = models.CharField(max_length=250)

class Books(models.Model):
    book = models.ForeignKey(Author,on_delete=models.CASCADE)
```

```
CASCADE：删除作者信息一并删除作者名下的所有书的信息；
PROTECT：删除作者的信息时，采取保护机制，抛出错误：即不删除Books的内容
SET_NULL：只有当null=True才将关联的内容置空；
SET_DEFAULT：设置为默认值；
SET()：括号里可以是函数，设置为自己定义的东西；
DO_NOTHING：字面的意思，啥也不干，你删除你的干我毛线
```

38.django中csrf的实现机制?

```
第一步：django第一次响应来自某个客户端的请求时,后端随机产生一个token值，把这个token保存在SESSION状态中;同时,后端把这个token放到cookie中交给前端页面；
第二步：下次前端需要发起请求（比如发帖）的时候把这个token值加入到请求数据或者头信息中,一起传给后端；Cookies: {csrftoken:xxxxx}
第三步：后端校验前端请求带过来的token和SESSION里的token是否一致。
```

39.基于 django使用ajax发送post请求时,有哪种方法携带 csrf token?

```
//在HTML页面中使用 {% csrf_token %}，给POST提交数据中添加 csrfmiddlewaretoken的键值对
 data :{
      'csrfmiddlewaretoken':$('[name="csrfmiddlewaretoken"]').val(),
       i1:$('[name="i1"]').val(),
       i2:$('[name="i2"]').val()
},
```

```
//在HTML页面中使用 {% csrf_token %}，在ajax中添加X-csrftoken的请求头
headers:{
  'X-csrftoken': $('[name="csrfmiddlewaretoken"]').val(),
},
```

40.django的缓存能使用redis吗?如果可以的话,如何配置?

```
依赖：pip3 install django-redis
```

```
from django_redis import get_redis_connection
conn = get_redis_connection("default")
```

41.django路由系统中name的作用?

```
url的命名:用于反向解析
```

```
#以命名user_list为例
url(r'/user_list/',view,name='user_list')

from django.urls import reverse
reverse('user_list')   --》 '/app01/user_list/'
#模板中使用
{% url 'user_list'  %}   --》 '/app01/user_list/'
```

```
url(r'/edit_user/(\d+)',view,name='edit_user')

reverse('edit_user',args=('1',))   --》 '/edit_user/1/'
{% url 'edit_user' 1  %}   --》'/edit_user/1/'
```

```
url(r'/edit_user/(?P<pk>\d+)',view,name='edit_user')

reverse('edit_user',args=('1',))   --》 '/edit_user/1/'
{% url 'edit_user' 1  %}   --》'/edit_user/1/'

reverse('edit_user',kwargs={'pk':2}   --》 '/edit_user/2/'
{% url 'edit_user' pk=1  %}   --》'/edit_user/1/'
```

42.django的模板中 filter、 simpletag, inclusiontag的区别?

```
from django import  template
register = template.Library()
```

```
@register.filter
def addxx(value,arg=None):
    return 'xxxxx'

@register.simple_tag
def simple(*args,**kwargs):
    return 'cccccc'

@register.inclusion_tag('li.html')
def show_li(*args,**kwargs):
    return {'k1':v1}
```

```
{% load my_tags %}
{{ 'alex'|addxx:'sb' }}
{% simple 'v1' 'v2'  k1='v3' %}

{% show_li  %}
```

43.django- debug-toolbar的作用?

```
显示django运行中的数据，操作记录等，方便调试
```

44.django中如何实现单元测试?

45.解释orm中 db first和 code first的含义?

```
db first: 先创建数据库，再更新表模型
code first: 先写表模型，再更新数据库
```

46.django中如何根据数据库表生成 model类?

```
1、修改seting文件，在setting里面设置要连接的数据库类型和名称、地址
2、运行下面代码可以自动生成models模型文件
        - python manage.py inspectdb
3、创建一个app执行下下面代码:
        - python manage.py inspectdb > app/models.py
```

47.使用orm和原生sql的优缺点?

```
SQL:
优点:
    执行速度快
缺点:
    编写复杂，开发效率不高
-----------------------------------------------------------------
ORM:
优点:
    让用户不再写SQL语句，提高开发效率
    可以很方便地引入数据缓存之类的附加功能
缺点:
    在处理多表联查、where条件复杂查询时，ORM的语法会变得复杂。
    没有原生SQL速度快
```

48.简述MVC和MTV

```
MVC，全名是Model View Controller，是软件工程中的一种软件架构模式，把软件系统分为三个基本部分:

模型(Model)                        -- 存取数据
视图(View)                         -- 信息展示
控制器(Controller)                -- 逻辑的控制
具有耦合性低、重用性高、生命周期成本低等优点
```

Django框架的设计模式借鉴了MVC框架的思想，也是分成三部分，来降低各个部分之间的耦合性。

Django框架的不同之处在于它拆分的三部分为：Model（模型）、Template（模板）和View（视图），也就是MTV框架

    Model(模型)：负责业务对象与数据库的对象(ORM)
    Template(模版)：负责如何把页面展示给用户
    View(视图)：负责业务逻辑，并在适当的时候调用Model和Template
此外，Django还有一个urls分发器，它的作用是将一个个URL的页面请求分发给不同的view处理，view再调用相应的Model和Template

49.django的 contenttype组件的作用?

contenttype是django的一个组件(app)，它可以将django下所有app下的表记录下来
可以使用他再加上表中的两个字段,实现一张表和N张表动态创建FK关系。
    - 字段：表名称
    - 字段：数据行ID
应用：路飞表结构优惠券和专题课和学位课关联