1.简述http协议及常用请求头
2.列举常用的请求方法
3.列举常见的状态码
4.http和https的区别
5.简述websocket协议及实现原理
6.django中如何实现websocket
7.python web开发中，跨域问题的解决思路是
8.请简述http缓存机制
9.谈谈你所知道的python web框架
10.http和https的区别
11.django、flask、tornado框架的比较
12.什么是wsgi
13.列举django的内置组件
14.简述django下的(内建的)缓存机制
15.django中的model的SlugField类型字段有什么用途
16.django中想要验证表单提交是否格式正确需要用到form中的哪个方法
17.django中常见的线上部署方式有哪几种
18.django对数据查询结果排序怎么做，降序怎么做
19.下面关于http协议中的get和post方式区别，哪些是正确的?
20.django中使用memcashed作为缓存的具体方法？优缺点说明?
21.django的orm中如何查询id不等于5的元素
22.使用django中model filter条件过滤方法，把下面的sql语句转换为python代码
23.从输入http://www.baidu.com/到页面返回，中间都发生了什么?
24.django请求的生命周期
25.django中如何在model保存前做一定的固定操作，比如写一句日志?
26.简述django中间件机及应用场景
27.简述django中FBV和CBV
28.如何给django CBV 的函数设置添加装饰器
29.django如何链接多个数据库并进行读写分离
30.列举django中orm中你了解的所有方法
31.django中F的作用
32.django中Q的作用
33.django中如何执行原生sql
34.only和defer的区别
35.selectrelated和prefetchrelated的区别
36.django中filter和exclude的区别
37.django中values和value_list的区别
38.如何使用django中orm批量创建数据
39.django中Form和ModelForm的作用
40.django的Form组件中，如果字段中包含choice参数，请使用两种方式实现数据源实时更新
41.django中的Model中的ForeignKey字段中的on_delete字段有什么作用
42.django中csrf的实现机制
43.django中如何实现websocket
44.基于django使用ajax发送post请求时，有哪种方法带csrf token
45.django中缓存如何设置
46.django中的缓存能使用redis吗？如果可以的话如何配置?
47.django的路由系统中name的作用
48.django模板中的filter、simpletag、inclusiontag的区别

## 1.简述http协议及常用请求头

参考：https://mubu.com/doc/hm1CpZzvaP
http协议：超文本传输协议，是基于tcp协议之上的应用层协议
http协议基于响应、请求的模式，是无状态保存、无连接以及媒体独立的
http协议包含了请求协议（请求行、请求头、请求体）、响应协议（状态行、响应头、响应体）等

常用请求头：
USER-AGENT\COOKIE\CONNECTION:keep_alive\CONTENT-TYPE( json/application
 text/html)\host等

## 2.列举常用的请求方法

http请求中的8种请求方法
1、opions　返回服务器针对特定资源所支持的HTML请求方法　或web服务器发送*测试服务器功能（允许客户端查看服务器性能）
2、Get　向特定资源发出请求（请求指定页面信息，并返回实体主体）
3、Post　向指定资源提交数据进行处理请求（提交表单、上传文件），又可能导致新的资源的建立或原有资源的修改
4、Put　向指定资源位置上上传其最新内容（从客户端向服务器传送的数据取代指定文档的内容）
5、Head　与服务器索与get请求一致的相应，响应体不会返回，获取包含在小消息头中的原信息（与get请求类似，返回的响应中没有具体内容，用于获取报头）
6、Delete　请求服务器删除request-URL所标示的资源*（请求服务器删除页面）
7、Trace　回复服务器收到的请求，用于测试和诊断
8、Connect　HTTP/1.1协议中能够将连接改为管道方式的代理服务器

## 3.列举常见的状态码

200 OK：客户端请求成功。
400 Bad Request：客户端请求有语法错误，不能被服务器所理解。
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。
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，举个例子：HTTP/1.1 200 OK（CRLF）。

| | 类别 | 原因短语 |
| --- | --- | --- |
| 1XX | Informational（信息性状态码） | 接收的请求正在处理 |
| 2XX | Success（成功状态码） | 请求正常处理完毕 |
| 3XX | Redirection（重定向状态码） | 需要进行附加操作以完成请求 |
| 4XX | Client Error（客户端错误状态码） | 服务器无法处理请求 |
| 5XX | Server Error（服务器错误状态码） | 服务器处理请求出错 |

## 4.http和https的区别

HTTPS和HTTP的区别主要如下：

1、https协议需要到ca申请证书，一般免费证书较少，因而需要一定费用。
2、http是超文本传输协议，信息是明文传输，https则是具有安全性的ssl加密传输协议。
3、http和ttps使用的是完全不同的连接方式，用的端口也不一样，前者是80，后者是443。
4、http的连接很简单，是无状态的；HTTPS协议是由SSL+HTTP协议构建的可进行加密传输、身份认证的网络协议，比http协议安全。

## 5.简述websocket协议及实现原理

WebSocket是HTML5下一种新的协议。它实现了浏览器与服务器全双工通信，能更好的节省服务器资源和带宽并达到实时通讯的目的。

它与HTTP一样通过已建立的TCP连接来传输数据，但是它和HTTP最大不同是：WebSocket是一种双向通信协议。在建立连接后，WebSocket服务器端和客户端都能主动向对方发送或接收数据，就像Socket一样；WebSocket需要像TCP一样，先建立连接，连接成功后才能相互通信。

相对于传统HTTP每次请求-应答都需要客户端与服务端建立连接的模式，WebSocket是类似Socket的TCP长连接通讯模式。一旦WebSocket连接建立后，后续数据都以帧序列的形式传输。在客户端断开WebSocket连接或Server端中断连接前，不需要客户端和服务端重新发起连接请求。在海量并发及客户端与服务器交互负载流量大的情况下，极大的节省了网络带宽资源的消耗，有明显的性能优势，且客户端发送和接受消息是在同一个持久连接上发起，实时性优势明显。

链接：https://www.zhihu.com/question/20215561/answer/157908509

---其他知识：

HTTP是应用层协议，定义的是传输数据的内容以及格式的规范。
TCP是底层通讯协议，定义的是数据传输和连接方式的规范。
Socket可以支持不同的传输层协议（TCP或UDP），当使用TCP协议进行连接时，该Socket连接就是一个TCP连接，Socket是发动机，提供了网络通信的能力

## 6.django中如何实现websocket

```
https://www.cnblogs.com/sui776265233/p/10176275.html
    django实现websocket大致上有两种方式，一种channels，一种是dwebsocket。
channels依赖于redis，twisted等，相比之下使用dwebsocket要更为方便一些
#dwebsocket有两种装饰器：require_websocket和accept_websocekt，使用
require_websocket装饰器会导致视图函数无法接收导致正常的http请求，一般情况使用
accept_websocket方式就可以了，
#
# dwebsocket的一些内置方法：
#
# request.is_websocket（）：判断请求是否是websocket方式，是返回true，否则返回
false
# request.websocket：  当请求为websocket的时候，会在request中增加一个websocket属
性，
# WebSocket.wait（）  返回客户端发送的一条消息，没有收到消息则会导致阻塞
# WebSocket.read（）  和wait一样可以接受返回的消息，只是这种是非阻塞的，没有消息返回
None
# WebSocket.count_messages（）返回消息的数量
# WebSocket.has_messages（）返回是否有新的消息过来
# WebSocket.send（message）像客户端发送消息，message为byte类型
```

## 7.python web开发中，跨域问题的解决思路是

```
https://www.cnblogs.com/yzxing/p/9440191.html
django-cors-headers
```

## 8.请简述http缓存机制

```
http://www.cnblogs.com/chenqf/p/6386163.html
```

## 9.谈谈你所知道的python web框架

```
Django\flask\tornado\twisted\synic
```

## 10.http和https的区别

```
见第4题
```

## 11.django、flask、tornado框架的比较

Django：Python 界最全能的 web 开发框架，battery-include 各种功能完备，可维护性和开发速度一级棒。常有人说 Django 慢，其实主要慢在 Django ORM 与数据库的交互上，所以是否选用 Django，取决于项目对数据库交互的要求以及各种优化。而对于 Django 的同步特性导致吞吐量小的问题，其实可以通过 Celery 等解决，倒不是一个根本问题。Django 的项目代表：Instagram，Guardian。

Tornado：天生异步，性能强悍是 Tornado 的名片，然而 Tornado 相比 Django 是较为原始的框架，诸多内容需要自己去处理。当然，随着项目越来越大，框架能够提供的功能占比越来越小，更多的内容需要团队自己去实现，而大项目往往需要性能的保证，这时候 Tornado 就是比较好的选择。Tornado项目代表：知乎。

Flask：微框架的典范，号称 Python 代码写得最好的项目之一。Flask 的灵活性，也是双刃剑：能用好 Flask 的，可以做成 Pinterest，用不好就是灾难（显然对任何框架都是这样）。Flask 虽然是微框架，但是也可以做成规模化的 Flask。加上 Flask 可以自由选择自己的数据库交互组件（通常是 Flask-SQLAlchemy），而且加上 celery +redis 等异步特性以后，Flask 的性能相对 Tornado 也不逞多让，也许Flask 的灵活性可能是某些团队更需要的。

链接：https://www.jianshu.com/p/9960a9667a5c

## 12.什么是wsgi

WSGI：Web Server Gateway Interface，wsgi是说在做Web应用的时候，需要去处理HTTP请求、响应。但是我们不会自己去实现这些底层的东西，而希望专注于业务代码的撰写。因此，需要一个统一的接口，它来帮我们处理http相关的协议，这个接口就是wsgi。

## 13.列举django的内置组件

https://www.cnblogs.com/zjchao/p/9073137.html
1. forms组件
forms组件主要功能是检验字段的功能，校验表单中的键值对的功能
2. 中间件
3，分页器
4.序列化器

## 14.简述django下的(内建的)缓存机制

缓存是将一些常用的数据保存内存或者memcache中,在一定的时间内有人来访问这些数据时,则不再去执行数据库及渲染等操作,而是直接从内存或memcache的缓存中去取得数据,然后返回给用户.django提供了6种内建缓存机制，分别为：

开发调试缓存（为开发调试使用，实际上不使用任何操作）；

内存缓存（将缓存内容缓存到内存中）；

文件缓存（将缓存内容写到文件 ）；

数据库缓存（将缓存内容存到数据库）；

memcache缓存（包含两种模块，python-memcached或pylibmc）。

以上缓存均提供了三种粒度的应用。

## 15.django中的model的SlugField类型字段有什么用途

slug是一个新闻行业的术语。一个slug就是一个某种东西的简短标签，包含字母、数字、下划线或者连接线，通常用于URLs中。可以设置max_length参数，默认为50。
https://stackoverflow.com/questions/427102/what-is-a-slug-in-django

## 16.django中想要验证表单提交是否格式正确需要用到form中的哪个方法

```
  A.form.save()
  B.form.save(commit=False)
  C.form.verity()
✓ D.form.is_valid()
```

## 17.django中常见的线上部署方式有哪几种

https://www.cnblogs.com/DjangoBlog/p/3934647.html

9种 链接：https://www.cnblogs.com/DjangoBlog/p/3934647.html
我们学过的nginx + uwsgi +Django：
https://blog.csdn.net/wl21787/article/details/80066616

## 18.django对数据查询结果排序怎么做，降序怎么做

```
Query_set.order_by("-age")
```

## 19.下面关于http协议中的get和post方式区别，哪些是正确的?

```
  A.他们都可以被收藏以及缓存
✓ B.get请求参数放在url中
  C.get只用于查询请求，不能用于数据请求
✓ D.get不应该处理敏感数据请求
```

## 20.django中使用memcashed作为缓存的具体方法？优缺点说明?

https://blog.csdn.net/zl834205311/article/details/51217760

```
https://blog.csdn.net/zl834205311/article/details/51217760
```

## 21.django的orm中如何查询id不等于5的元素

```
xxx.exclude(id=5)
```

## 22.使用django中model filter条件过滤方法，把下面的sql语句转换为python代码

```
1.select * from company where title like "%adc%" or mecount>999;
-- model.Company.filter(Q(title__contains='adc')|Q(mecount__gt=999))

2.order by createtime desc;  asc
-- xxx.order_by('-createtime')
```

## 23.从输入http://www.baidu.com/到页面返回，中间都发生了什么?

//将域名解析为ip

1.浏览器会查询浏览器的缓存内有无该网址对应的ip地址，若没有则查询/etc/hosts文件下有无该域名对应的ip地址。

2.如果还是没有则访问本地DNS服务器有无DNS缓存，若还没有，则向根服务器一步一步向上查询（例如从.com开始查询）最终可以获得ip地址。

//向服务器建立tcp连接

3.自然这里肯定是先三次握手建立tcp连接。形成客户端到服务端的稳定通道。

4.建立完tcp连接后就可以向服务器发送http请求了。

//后端处理请求

5.后端处理这个请求，处理完返回一个渲染结果给浏览器。完成一次查询。

这整个过程涉及到三个主题：DNS服务器，浏览器，服务器。

## 24.django请求的生命周期

首先，浏览器发来的http请求经过Django中的wsgi被解析生成request对象,再经过Django的中间件,之后根据url对应路由映射表,在路由中一条一条进行匹配,一旦其中一条匹配成功就执行对应的视图函数,后面的路由就不再继续匹配了。
视图函数根据客户端的请求查询相应的数据.返回给Django,然后Django把客户端想要的数据（页面）做为一个Httpresponse对象返回给客户端.

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

在定义model类的时候定义django.db.models.signals.pre_save方法，信号;具体见下方博客:
https://blog.csdn.net/qq_37049050/article/details/81746046

## 26.简述django中间件机及应用场景

1、process_request(self, request) : 请求进来时,权限认证 。

2、process_view(self, request, view_func, view_args, view_kwargs) : 路由匹配之后,能够得到视图函数

3、process_exception(self, request, exception) : 异常时执行

4、process_template_response(self, request, response) : 模板渲染时执行

5、process_response(self, request, response) : 请求有响应时执行

## 27.简述django中FBV和CBV

python视图中定义视图的两种方式，FBV代表函数视图，CBV代表的是类视图。
在FBV中，函数名要与路由中的视图名对应，第一个参数是request对象，通过request对象的method属性判断请求方法。
在CBV中，类名与路由中的视图名对应，类继承views.View的类，视图通过as_view方法调用类中的dispatch方法（继承自View类，也可以自己自定义）进行路由分发，类中的方法名按照http请求方法定义，如get方法就在类中定义get的方法。

## 28.如何给django CBV 的函数设置添加装饰器

```
from django.utils.decorators import method_decorator

# @method_decorator(cookie,name='dispatch')    # dispatch的便捷写法
class CBVtest(views.View):
    @method_decorator(cookie)   # 给dispatch方法添加装饰器，那么下面所有的get，post都会添加
    def dispatch(self, request, *args, **kwargs):
        return super(CBVtest,self).dispatch(request,*args,**kwargs)
    # @method_decorator(cookie)  # 单独添加
    def get(self,request):
         u = request.get_signed_cookie('username',salt='user',default=None)
         return render(request,'houtai.html',{'user':u})
```

需要导入django.utils.decorators 中的method_decorator
第一种方式，可以直接在类上添加，@method_decorator,name为dispatch
第二种方式，直接重写dispatch方法，在这个方法上面添加
第三种方式，在单独的函数上添加

## 29.django如何链接多个数据库并进行读写分离

1.在settings文件里面配置多个数据库的配置

```
    DATABASES = {
    'default': {
        'ENGINE': 'django.db.backends.sqlite3',
        'NAME': os.path.join(BASE_DIR, 'db.sqlite3'),
    },
    'db2': {
        'ENGINE': 'django.db.backends.sqlite3',
        'NAME': os.path.join(BASE_DIR, 'db2.sqlite3'),
    },
}
```

2.迁移默认有参数python manage.py migrate --database default ，在默认数据库上创建表。因此完成以上迁移后，执行python manage.py --database db2，再迁移一次，就可以在db2上创建相同的表。

3.读写分离

1.手动读写分离

在使用数据库时，通过.using(db_name)来手动指定要使用的数据库

2.自动读写分离

自定义文件

```
  class Router:
def db_for_read(self, model, **hints):
    return 'db2'

def db_for_write(self, model, **hints):
    return 'default'
```

配置Router

settings.py中指定DATABASE_ROUTERS

```
DATABASE_ROUTERS = ['myrouter.Router',]
```

## 30.列举django中orm中你了解的所有方法

| all() | 查询所有结果 |
| --- | --- |
| filter(**kwargs) | 它包含了与所给筛选条件相匹配的对象。获取不到返回None |
| get(**kwargs) | 返回与所给筛选条件相匹配的对象，返回结果有且只有一个。获取不到会抱胸 #如果符合筛选条件的对象超过一个或者没有都会抛出错误 |
| exclude(**kwargs) | 它包含了与所给筛选条件不匹配的对象 |
| order_by(*field) | 对查询结果排序 |
| reverse() | 对查询结果反向排序，在已经排序好的基础上进行操作 |
| count() | 返回数据库中匹配查询(QuerySet)的对象数量 |
| first() | 返回第一条记录 |
| last() | 返回最后一条记录 |
| exists() | 如果QuerySet包含数据，就返回True，否则返回False |
| values(*field) | 返回一个特殊的QuerySet，运行后得到是一个可迭代的字典序列 |
| values_list(*field) | 它与values()非常相似，它返回的是一个元组序列，values返回的是一个字典序列 |
| distinct() | 从返回结果中剔除重复纪录 |

## 31.django中F的作用

```
##### F比较两个字段之间的关系及对一个字段的数据进行加减乘除等运算
from django.db.models import F
ret =models.Book.objects.filter(sale__gt=F('kucun')).values()    # 查询销量大于库存的图书
models.Book.objects.all().update(sale=F('sale') * 2)  #更新销量是原来的两倍
```

## 32.django中Q的作用

```
#Q可以同时查询多个条件出来
 ~Q表示非的意思
| 表示或的意思
&表示and的意思
from django.db.models import F, Q
ret = models.Book.objects.filter(~Q(Q(id__lt=2) | Q(id__gt=4)) &
Q(id__gt=3))
print(ret)

执行结果：结果是只有ID为4的对象
<QuerySet [<Book: <Book object: 4 跟egon学喊麦>>]>
```

## 33.django中如何执行原生sql

```
1种方式： print(models.Book.objects.filter(publisher__name='沙河出版社').extra(where=["price>10"]))

orm操作：
ret = models.Book.objects.filter(publisher__name='沙河出版社',price__gt=10)
print(ret)
ret1 = models.Book.objects.filter(Q(publisher__name='沙河出版社')&Q(price__gt=10))
print(ret1)
执行结果：
<QuerySet [<Book: <Book object: 1 跟金老板学开车>>]>
<QuerySet [<Book: <Book object: 1 跟金老板学开车>>]>
<QuerySet [<Book: <Book object: 1 跟金老板学开车>>]>

2种方式：直接执行自定义Sql
(这种方式完全不依赖model,前两种还是要依赖于model)
from django.db import connection
cursor=connection.cursor()
#插入操作
cursor.execute('insert into app01_publisher(id,name,city) values(6,"激动出版社","甘肃")')
#更新操作
cursor.execute("update app01_book set title='跟金老板学开车' where title='开车'")
# #删除操作
cursor.execute("delete from app01_book where title='开车'")
# #查询操作
print(cursor.execute('select publish_date from app01_book '))
raw=cursor.fetchone() #返回结果行游标直读向前，读取一条
print(raw)
print(cursor.fetchall())#读取所有
```

## 34.only和defer的区别

## 35.selectrelated和prefetchrelated的区别

在数据库有外键的时候,使用select_related()和prefetch_related()可以很好的减少数据库请求的次数,从而提高性能.

##selectrelated:

1.select_related主要针一对一和多对一关系进行优化。

2.select_related使用SQL的JOIN语句进行优化，通过减少SQL查询的次数来进行优化、提高性能。

3.可以通过可变长参数指定需要select_related的字段名。也可以通过使用双下划线“__”连接字段名来实现指定的递归查询。没有指定的字段不会缓存，没有指定的深度不会缓存，如果要访问的话Django会再次进行SQL查询。

```
  #获得张三的现居省份
 zhangs = Person.objects.select_related('living__province').get(firstname=u"张",lastname=u"三")
```

4.也可以通过depth参数指定递归的深度，Django会自动缓存指定深度内所有的字段。如果要访问指定深度外的字段，
Django会再次进行SQL查询。
#select_related() 接受depth参数，depth参数可以确定select_related的深度。Django会递归遍历指定深度内的所有的OneToOneField和ForeignKey.
zhangs = Person.objects.select_related(depth = d)
d=1 相当于 select_related('hometown','living') 只在本表中查询
d=2 相当于 select_related('hometown__province','living__province') 跨表查询，有外键关系，深度加1
5.也接受无参数的调用，Django会尽可能深的递归查询所有的字段。但注意有Django递归的限制和性能的浪费。

6.Django >= 1.7，链式调用的select_related相当于使用可变长参数。Django < 1.7，链式调用会导致前边的select_related失效，只保留最后一个。

```
## django1.7以前同时指定两个外键使用
zhangs=Person.objects.select_related('hometown__province','living__province').get(firstname=u"张",lastname=u"三")
## 1.7后使用
zhangs=Person.objects.select_related('hometown__province').select_related('living__province').get(firstname=u"张",lastname=u"三")
```

#prefetchrelated
对于多对多字段（ManyToManyField）和一对多(ForeignKey)字段，可以使用prefetch_related()来进行优化.
prefetch_related()和select_related()的设计目的很相似，都是为了减少SQL查询的数量，但是实现的方式不一样。后者 是通过JOIN语句，在SQL查询内解决问题。但是对于多对多关系，使用SQL语句解决就显得有些不太明智，因为JOIN得到的表将会很长，会导致SQL 语句运行时间的增加和内存占用的增加。
prefetch_related()的解决方法是，分别查询每个表，然后用Python处理他们之间的关系。

```
zhangs = Person.objects.prefetch_related('visitation__province').filter(firstname__iexact=u'张')
```

要注意的是，在使用QuerySet的时候，一旦在链式操作中改变了数据库请求，之前用prefetch_related缓存的数据将会被忽略掉。这会导 致Django重新请求数据库来获得相应的数据，从而造成性能问题。这里提到的改变数据库请求指各种filter()、exclude()等等最终会改变 SQL代码的操作。而all()并不会改变最终的数据库请求，因此是不会导致重新请求数据库的。

None
可以通过传入一个None来清空之前的prefetch_related。就像这样：

```
prefetch_cleared_qset = qset.prefetch_related(None)
```

小结：
因为select_related()总是在单次SQL查询中解决问题，而prefetch_related()会对每个相关表进行SQL查询，因此select_related()的效率通常比后者高。
鉴于第一条，尽可能的用select_related()解决问题。只有在select_related()不能解决问题的时候再去想prefetch_related()。
你可以在一个QuerySet中同时使用select_related()和prefetch_related()，从而减少SQL查询的次数。
只有prefetch_related()之前的select_related()是有效的，之后的将会被无视掉。
https://www.cnblogs.com/tuifeideyouran/p/4232028.html

## 36.django中filter和exclude的区别

```
filter 获取满足条件的所有对象    QuerySet    对象列表
ret = models.Book.objects.filter(title="跟金老板学开车")
print(ret)
执行结果：<QuerySet [<Book: <Book object: 1 跟金老板学开车>>]>

exclude 获取不满足条件的所有对象   QuerySet    对象列表
ret = models.Book.objects.exclude(title="跟金老板学开车")
print(ret)
执行结果：<QuerySet [<Book: <Book object: 2 跟金老板学开潜艇>>, <Book: <Book
object: 3 跟老男孩学思想>>, <Book: <Book object: 4 跟egon学喊麦>>]>
```

## 37.django中values和value_list的区别

```
##values返回列表套字典：
ret = models.Book.objects.filter(title="跟金老板学开
车").values("price","publish_date")
print(ret)
for i in ret:
    print(i,type(i))
执行结果：
<QuerySet [{'price': Decimal('12.90'), 'publish_date': datetime.date(2018,
8, 3)}]>
{'price': Decimal('12.90'), 'publish_date': datetime.date(2018, 8, 3)}
<class 'dict'> 字典类型

##value_list返回列表套元祖：
ret = models.Book.objects.filter(title="跟金老板学开
车").values_list("price","publish_date")
print(ret)
for i in ret:
    print(i,type(i))
执行结果：
<QuerySet [(Decimal('12.90'), datetime.date(2018, 8, 3))]>
(Decimal('12.90'), datetime.date(2018, 8, 3)) <class 'tuple'> 元祖类型
```

## 38.如何使用django中orm批量创建数据

```
####使用：models.StudyRecord.objects.bulk_create()

def multi_init(self):
        # 拿到课程记录的ID
   course_record_ids = self.request.POST.getlist('ids')  # [1,2 ]
   for course_record_id in course_record_ids:
          # 拿到所有的学生
          course_record =
models.CourseRecord.objects.filter(pk=course_record_id).first()
```

```
            ##通过课程记录表中的ID拿到外键的对应班级，通过班级拿到多对多反向的客户状态为
studying的
            all_student =
course_record.re_class.customer_set.filter(status='studying')
              批量操作
            s_list = []
            for student in all_student:
                  ##将每一个学习记录对象放到列表中

s_list.append(models.StudyRecord(course_record_id=course_record_id,student
=student))
             models.StudyRecord.objects.bulk_create(s_list)

但是：
如果你导入数据过多，导入时出错了，或者你手动停止了，导入了一部分，还有一部分没有导入。或者
你再次运行上面的命令，你会发现数据重复了，怎么办呢？
django.db.models 中还有一个函数叫 get_or_create(),之前文章中也提到过,有就获取过
来，没有就创建，用它可以避免重复，但是速度可以会慢些，因为要先尝试获取，看看有没有
 for student in all_student:

 models.StudyRecord.objects.get_or_create(course_record_id=course_record_i
d,student=student)
```

## 39.django中Form和ModelForm的作用

```
form组件的作用如下：
    前端页面是form类的对象生成的                                  -->生成HTML标签功能
    当用户名和密码输入为空或输错之后 页面都会提示              -->用户提交校验功能
    当用户输错之后 再次输入 上次的内容还保留在input框     -->保留上次输入内容
ModelForm的作用如下：
    利用 Model 生成 Form，提高 Model 复用性
    Django Admin就是利用ModelForm的功能实现的
    modelForm的验证过程（局部钩子，全局钩子）
```

## 40.django的Form组件中，如果字段中包含choice参数，请使用两种方式实现数据源实时更新

```
1.重写构造函数
def__init__(self, *args, **kwargs):
     super().__init__(*args, **kwargs)
     self.fields["city"].widget.choices =
models.City.objects.all().values_list("id", "name") #列表套元祖形式

2.利用ModelChoiceField字段,参数为queryset对象
    authors =
form_model.ModelMultipleChoiceField(queryset=models.NNewType.objects.all()
)  //多选
```

## 41.django中的Model中的ForeignKey字段中的on_delete字段有什么作用

```
on_delete：当该表中的某条数据删除后，关联外键的操作
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关系
```

## 42.django中csrf的实现机制

```
csrf 跨站请求伪造
django中跨站请求伪造的保护机制
通过中间件中的  from  django.middleware.csrf import CsrfViewMiddleware
https://www.cnblogs.com/Bottle-cap/articles/10390893.html
```

## 43.django中如何实现websocket

```
6.https://www.cnblogs.com/sui776265233/p/10176275.html  可咨询吉冬大佬
```

## 44.基于django使用ajax发送post请求时，有哪种方法带csrf token

```
前提条件：保证访问的页面有csrftoken的cookie，目的是为了进行校验
一、模板中使用csrf-token标签
二、给视图加装饰器
from django.views.decorators.csrf import ensure_csrf_cookie      # 确保访问某个视图有csrf_cookie

方式一：
$.ajax({
  url: "/cookie_ajax/",
  type: "POST",
  data: {
    "username": "Q1mi",
    "password": 123456,
    "csrfmiddlewaretoken": $("[name = 'csrfmiddlewaretoken']").val()  // 使用jQuery取出csrfmiddlewaretoken的值，拼接到data中
  },
  success: function (data) {
    console.log(data);
  }
})
方式二：
$('#b2').click(function () {
```

```
    $.ajax({
        url: '/calc2/',
        type: 'post',
        headers:{
           'X-csrftoken': $('[name="csrfmiddlewaretoken"]').val(),    响应头
里面
        },
        data: {
            i1: $('[name="ii1"]').val(),
            i2: $('[name="ii2"]').val(),
        },
        success: function (res) {
            $('[name="ii3"]').val(res)
        }
    })
});
方式三：文件：可以在static中创建一个js文件，进行导入
function getCookie(name) {
    var cookieValue = null;
    if (document.cookie && document.cookie !== '') {
        var cookies = document.cookie.split(';');
        for (var i = 0; i < cookies.length; i++) {
            var cookie = jQuery.trim(cookies[i]);
            // Does this cookie string begin with the name we want?
            if (cookie.substring(0, name.length + 1) === (name + '=')) {
                cookieValue =
decodeURIComponent(cookie.substring(name.length + 1));
                break;
            }
        }
    }
    return cookieValue;
}

var csrftoken = getCookie('csrftoken');


function csrfSafeMethod(method) {
    // these HTTP methods do not require CSRF protection
    return (/^(GET|HEAD|OPTIONS|TRACE)$/.test(method));
}

$.ajaxSetup({
    beforeSend: function (xhr, settings) {
        if (!csrfSafeMethod(settings.type) && !this.crossDomain) {
            xhr.setRequestHeader("X-CSRFToken", csrftoken);
        }
    }
});
```

## 45.django中缓存如何设置

同14题：可咨询吉冬大佬

## 46.django中的缓存能使用redis吗？如果可以的话如何配置？

```
#1.安装 pip install django-redis
#2.在stting中配置CACHES,可以设置多个缓存,根据名字使用
        CACHES = {
            "default": {
                "BACKEND": "django_redis.cache.RedisCache",
                "LOCATION": "redis://127.0.0.1:6379",
                "OPTIONS": {
                    "CLIENT_CLASS": "django_redis.client.DefaultClient",
                    "CONNECTION_POOL_KWARGS": {"max_connections": 100}
                    # "PASSWORD": "密码",
                }
            }
        },
        #另添加缓存
        "JERD": { }
#3.根据名字去连接池中获取连接，视图中连接（手动操作redis）
from django.shortcuts import HttpResponse
from django_redis import get_redis_connection

def index(request):
    r = get_redis_connection("default")
    r.hmset("name_a", {"key_a": "value_a", "key_b": "value_b"})
    return HttpResponse("设置redis")

def order(request):
    r = get_redis_connection("default")
    val = r.hmget("name_a", ["key_a", "key_b"])
    print(val)  # [b'value_a', b'value_b']
    return HttpResponse("获取redis")
```

中间件：全站使用缓存
使用中间件，经过一系列的认证等操作，如果内容在缓存中存在，则使用
FetchFromCacheMiddleware 获取内容并返回给用户，当返回给用户之前，判断缓存中是否已经存在，如果不存在则UpdateCacheMiddleware 会将缓存保存至缓存，从而实现全站缓存。

```
MIDDLEWARE = [
    'django.middleware.cache.UpdateCacheMiddleware',  # 放在第一
    # 其他中间件...
    'django.middleware.cache.FetchFromCacheMiddleware',  # 放在最后
]

CACHE_MIDDLEWARE_ALIAS = ""             # 用于存储的缓存别名
CACHE_MIDDLEWARE_SECONDS = 600          # 每个页面应缓存的秒数
CACHE_MIDDLEWARE_KEY_PREFIX = ""        # 如果使用相同的Django安装在多个站点之间共
```

享缓存，请将其设置为站点名称或此Django实例特有的其他字符串，以防止发生密钥冲突。如果你不在乎，请使用空字符串。

## 47.django的路由系统中name的作用

就是可以给我们的URL匹配规则起个名字，一个URL匹配模式起一个名字。
这样我们以后就不需要写死URL代码了，只需要通过名字来调用当前的URL。

```
url(r'^home', views.home, name='home'),  # 给我的url匹配模式起名为 home
```

模板中使用：{% url 'home' %}

视图中使用：

```
from django.urls import reverse，redirect
redirect（reverse("index")）
```

## 48.django模板中的filter、simpletag、inclusiontag的区别

1.在app下创建一个名叫templatetags的python包
2.在包内创建py文件
3.在py文件中写：

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

##filter  参数受限
@register.filter
def add_sb(value, arg):
    return '{}_{}_sb'.format(value, arg)
```

使用：

```
{% load my_tags %}
{% if 'alex'|add_sb:'big' == 'alex_big_sb' %}
    zhen
{% endif %}
##simpletag  参数不受限
@register.simple_tag
def str_join(*args, **kwargs):
    print(args)
    print(kwargs)
    return "-".join(args) + '*'.join(kwargs.values())
{% load my_tags %}
{% str_join 'k1' 'k2' 'k3' k1='v1' k2='v2' %}
##inclusiontag 返回一个动态的代码段，多用于返回html代码片段
@register.inclusion_tag('li.html')
def show_li(num):
    return {'num': range(num)}
```

li.html中：

```
<ul>
    {% for foo in num %}
        <li> {{ foo }}</li>
    {% endfor %}
</ul>
```

使用：

```
{% load my_tags %}
{% show_li 10 %}
```

## 49.django-debug-toolbar的作用

Django Debug Toolbar是Django开发中必备利器，可以帮助开发者快速了解项目的整体信息以及每个页面包括sql信息，http相关信息。本篇将详细讲解如何django-debug-toolbar的使用。

https://blog.csdn.net/cn_1937/article/details/82715983

## 50.django中如何实现单元测试

```
from django.shortcuts import render

# Create your views here.
from django.test import TestCase     #导入Django测试包
from app01.models import Author, Publisher  #导入models中的书/出版社类

#首先创建测试类
class ModelTest(TestCase):

    #初始化：分别创建一条发布会（Event）和一条嘉宾（Guest）的数据。
    def setUp(self):
        Author.objects.create( name="小明",age=1, phone=1123345555)
        Publisher.objects.create( name="上帝出版社",city="西藏")

    #下面开始写测试用例了
    #1.通过get的方法，查询插入的作者数据，并根据年龄判断
    def test_event_models(self):
        result = Author.objects.get(name="小花")    #注意这里是小花
        self.assertEqual(result.age, 1)
        self.assertTrue(result.phone)

    #2.通过get的方法，查询插入的嘉宾数据，并根据名字判断
    def test_guest_models(self):
        result = Publisher.objects.get( name="上帝出版社")
        self.assertEqual(result.city, "西藏")
        self.assertFalse(result.pk)


执行语句：python manage.py test    执行所有test.py的文件名

Creating test database for alias 'default'...
System check identified no issues (0 silenced).
EF
======================================================================
ERROR: test_event_models (app01.test.ModelTest)
----------------------------------------------------------------------
Traceback (most recent call last):
  File "E:\学习相关\python全栈17期课程及笔记\project_django\text67天
\app01\test.py", line 19, in test_event_models
    result = Author.objects.get(name="小花")
  File "C:\Python36\lib\site-packages\django\db\models\manager.py", line
85, in manager_method
```

```
    return getattr(self.get_queryset(), name)(*args, **kwargs)
  File "C:\Python36\lib\site-packages\django\db\models\query.py", line
380, in get
    self.model._meta.object_name
app01.models.DoesNotExist: Author matching query does not exist.

======================================================================
FAIL: test_guest_models (app01.test.ModelTest)
----------------------------------------------------------------------
Traceback (most recent call last):
  File "E:\学习相关\python全栈17期课程及笔记\project_django\text67天
\app01\test.py", line 27, in test_guest_models
    self.assertFalse(result.pk)
AssertionError: 2 is not false

----------------------------------------------------------------------
Ran 2 tests in 0.097s

FAILED (failures=1, errors=1)
Destroying test database for alias 'default'...


如果成功:
Creating test database for alias 'default'...
System check identified no issues (0 silenced).

----------------------------------------------------------------------
Ran 0 tests in 0.000s

OK
Destroying test database for alias 'default'...
```

## 51.解释orm中db first 和code first的含义

```
1. db first 是现有数据库，再写代码。根据数据库的表生成类。
django里面：python manage.py  inspectdb

2. code first 是先写代码，后创建数据库。根据类创建数据库表。
djiango里面：python manage.py makemigrations  python manage.py migrate
```

## 52.django中如何根据数据库表生成model类

```
创建一个项目，修改seting文件，在setting里面设置你要连接的数据库类型和连接名称，地址之
类，和创建新项目的时候一致
DATABASES = {
    'default': {
        'ENGINE': 'django.db.backends.mysql',
        'NAME': "text67",
        "HOST":"127.0.0.1",
        "POST":3306,
```

```
        "USER":"root",
        "PASSWORD":"",
    }
}


1.运行下面代码可以自动生成models模型文件
python manage.py inspectdb

2.把模型文件导入到app中
#将模型导入创建的app中去
python manage.py inspectdb > app/models.py
```

## 53.使用orm和原生sql的优缺点

```
#1.orm的开发速度快,操作简单。使开发更加对象化
#执行速度慢。处理多表联查等复杂操作时,ORM的语法会变得复杂

#2.sql开发速度慢,执行速度快。性能强
```

## 54.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)、视图(View)和控制器(Controller)，具有耦合性低、重用性高、生命周期成本低等优点。


Django的MTV模式
        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
```

## 55.django中contenttype组件的作用

```
https://www.jianshu.com/p/f2285d77cddd
```

## 56.使用django中model filter条件过滤方法，把下面的sql语句转换为python代码

```
1.select * from company where title like "%adc%" or mecount>999;

models.Company.objects.filter(Q(title__contains="%adc%")|Q(mecount__gt=999
))
2.order by createtime desc;  #降序
    models.Company.objects.order_by("-createtime")
```

####