1、Django 请求的生命周期

    路由系统 -> 试图函数(获取模板+数据=》渲染) -> 字符串返回给用户

2、路由系统

```
/index/                    ->  函数或类.as_view()
/detail/(\d+)              ->  函数(参数) 或 类.as_view()（参数）
/detail/(?P<nid>\d+)       ->  函数(参数) 或 类.as_view()（参数）
/detail/                   ->  include("app01.urls")
/detail/    name='a1'      ->  include("app01.urls")
                               - 视图中：reverse
                               - 模板中：{% url "a1" %}
```

3、视图

FBV：函数

```
def index(request,*args,**kwargs):
    ..
```

CBV：类

```
class Home(views.View):

    def get(self,reqeust,*args,**kwargs):
        ..
```

获取用户请求中的数据：

```
request.POST.get
request.GET.get
reqeust.FILES.get()

# checkbox,
........getlist()

request.path_info



文件对象 = reqeust.FILES.get()
文件对象.name
文件对象.size
文件对象.chunks()

# <form 特殊的设置></form>
```

给用户返回数据：

```
render(request, "模板的文件的路径", {'k1': [1,2,3,4],"k2": {'name': '张扬','age': 73}})
redirect("URL")
HttpResponse(字符串)
```

4、模板语言

```
render(request, "模板的文件的路径", {'obj': 1234, 'k1': [1,2,3,4],"k2": {'name': '张扬','age': 73}})

<html>

<body>
    <h1> {{ obj }} </h1>
    <h1> {{ k1.3 }} </h1>
    <h1> {{ k2.name }} </h1>
    {% for i in k1 %}
        <p> {{ i }} </p>
    {% endfor %}

    {% for row in k2.keys %}
        {{ row }}
    {% endfor %}

    {% for row in k2.values %}
        {{ row }}
    {% endfor %}

    {% for k,v in k2.items %}
        {{ k }} - {{v}}
    {% endfor %}

</body>
</html>
```

5、ORM

a. 创建类和字段

```
class User(models.Model):
    age = models.IntergerFiled()
    name = models.CharField(max_length=10)#字符长度

Python manage.py makemigrations
python manage.py migrate

# settings.py  注册 APP
```

b. 操作

增

```
models.User.objects.create(name='qianxiaohu',age=18)
dic = {'name': 'xx', 'age': 19}
models.User.objects.create(**dic)

obj = models.User(name='qianxiaohu',age=18)
obj.save()
```

删

```
models.User.objects.filter(id=1).delete()
```

改

```
        models.User.objects.filter(id__gt=1).update(name='alex',age=84)
        dic = {'name': 'xx', 'age': 19}
        models.User.objects.filter(id__gt=1).update(**dic)
查
        models.User.objects.filter(id=1,name='root')
        models.User.objects.filter(id__gt=1,name='root')
        models.User.objects.filter(id__lt=1)
        models.User.objects.filter(id__gte=1)
        models.User.objects.filter(id__lte=1)

        models.User.objects.filter(id=1,name='root')
        dic = {'name': 'xx', 'age__gt': 19}
        models.User.objects.filter(**dic)

        v1 = models.Business.objects.all()
        # QuerySet ,内部元素都是对象

        # QuerySet ,内部元素都是字典
        v2 = models.Business.objects.all().values('id','caption')
        # QuerySet ,内部元素都是元组
        v3 = models.Business.objects.all().values_list('id','caption')

        #  获取到的一个对象，如果不存在就报错
        models.Business.objects.get(id=1)
        对象或者 None = models.Business.objects.filter(id=1).first()


        外键:
                v = models.Host.objects.filter(nid__gt=0)
                v[0].b.caption    ---->    通过.进行跨表
```

```
外键:
        class UserType(models.Model):
                caption = models.CharField(max_length=32)
            id    caption
        # 1，普通用户
        # 2，VIP 用户
        # 3，  游客

        class User(models.Model):
                age = models.IntergerFiled()
                name = models.CharField(max_length=10)#字符长度
                # user_type_id = models.IntergerFiled() #  约束，
                user_type = models.ForeignKey("UserType",to_field='id') #  约束，
```

```
       name age    user_type_id
    #  张扬    18         3
    #  张 A 扬  18         2
    #  张 B 扬  18         2
```

position:fixed absolute relative

Ajax

```
    $.ajax({
        url: '/host',
        type: "POST",
        data: {'k1': 123,'k2': "root"},
        success: function(data){
            // data 是服务器端返回的字符串
            var obj = JSON.parse(data);
        }
    })
```

建议：永远让服务器端返回一个字典

return HttpResponse(json.dumps(字典))

多对多:

创建多对多:

方式一：自定义关系表

```
        class Host(models.Model):
            nid = models.AutoField(primary_key=True)
            hostname = models.CharField(max_length=32,db_index=True)
            ip = models.GenericIPAddressField(protocol="ipv4",db_index=True)
            port = models.IntegerField()
            b = models.ForeignKey(to="Business", to_field='id')
        # 10
        class Application(models.Model):
            name = models.CharField(max_length=32)
        # 2

        class HostToApp(models.Model):
            hobj = models.ForeignKey(to='Host',to_field='nid')
            aobj = models.ForeignKey(to='Application',to_field='id')

        # HostToApp.objects.create(hobj_id=1,aobj_id=2)
```

方式二：自动创建关系表

```
class Host(models.Model):
    nid = models.AutoField(primary_key=True)
    hostname = models.CharField(max_length=32,db_index=True)
    ip = models.GenericIPAddressField(protocol="ipv4",db_index=True)
    port = models.IntegerField()
    b = models.ForeignKey(to="Business", to_field='id')
# 10
class Application(models.Model):
    name = models.CharField(max_length=32)
    r = models.ManyToManyField("Host")
```

无法直接对第三张表进行操作

```
obj = Application.objects.get(id=1)
obj.name

# 第三张表操作
obj.r.add(1)
obj.r.add(2)
obj.r.add(2,3,4)
obj.r.add(*[1,2,3,4])

obj.r.remove(1)
obj.r.remove(2,4)
obj.r.remove(*[1,2,3])

obj.r.clear()

obj.r.set([3,5,7])

# 所有相关的主机对象"列表" QuerySet
obj.r.all()
```