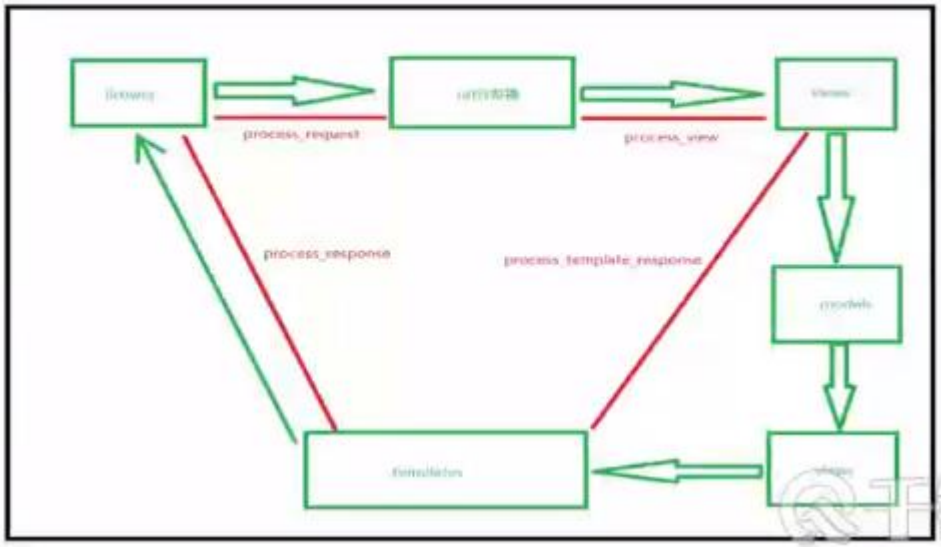
中间件

中间件：是一个轻量级的，底层的插件，可以介入Django的请求和相应过程（面向切面编程）

中间件的本质就是一个python类

面向切面编程（Aspect Oriented Programming）简称AOP。AOP的主要实现目的是针对业务处理过程中的切面进行提取，它所面对的是处理过程中的某个步骤或阶段，以获得逻辑过程中各部分之间低耦合的隔离效果。

中间件的可切入点



切入函数

__init__:没有参数，服务器响应第一个请求的时候自动调用，用户确定是否启用该中间件

process_request(self,request):在执行视图前被调用，每个请求上都会调用，不主动进行返回或返回HttpResponse对象

process_view(self,request,view_func,view_args,view_kwargs)：调用视图之前执行，每个请求都会调用，不主动进行返回或返回HttpResponse对象

process_template_response(self,request,response):在视图刚好执行完后进行调用，每个请求都会调用，不主动进行返回或返回HttpResponse对象

process_response(self,request,response):所有响应返回浏览器之前调用，每个请求都会调用，不主动进行返回或返回HttpResponse对象

自定义中间件

process_exception(self,request,exception):当视图抛出异常时调用，不主动进行返回或返回HttpResponse对象

自定义中间件流程

1. 在工程目录下创建middleware目录
2. 目录中创建一个python文件
3. 在python文件中导入中间件的基类

```
from django.utils.deprecation import MiddlewareMixin
```

4. 在类中根据功能需求，创建切入需求类，重写切入点方法

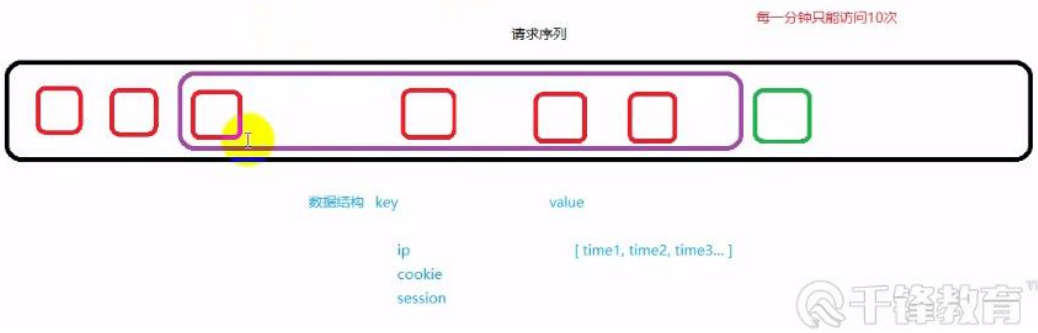
```
class LearnAOP(MiddlewareMixin):
    def process_request(self,request):
        print('request的路径',request.GET.path)
```

5. 启用中间件，在settings中进行配置，MIDDLEWARE中添加

```
middleware.文件名.类名
```

AOP中间件

- 实现统计功能
  - 统计IP
  - 统计浏览器
- 实现权重控制
  - 黑名单
  - 白名单
- 实现反爬
  - 反爬虫
    - 十秒之内只能搜索一次
  - 实现频率控制
- 界面友好化
- 应用交互友好化



中间件

- 调用顺序
  - 中间件注册的时候是一个列表
  - 如果我们没有在切点处直接进行返回，中间件会依次执行
  - 如果我们直接进行了返回，后续中间件就不再执行了
- 切点
  - process_request
  - process_view
  - process_template_response
  - process_response
  - process_exeption
- 切面

分页

django提供了分页的工具，存在于django.core中

Paginator　　：数据分页工具

Page　　：具体的某一页面

Paginator:

对象创建:　　Paginator(数据集，每一页数据数)

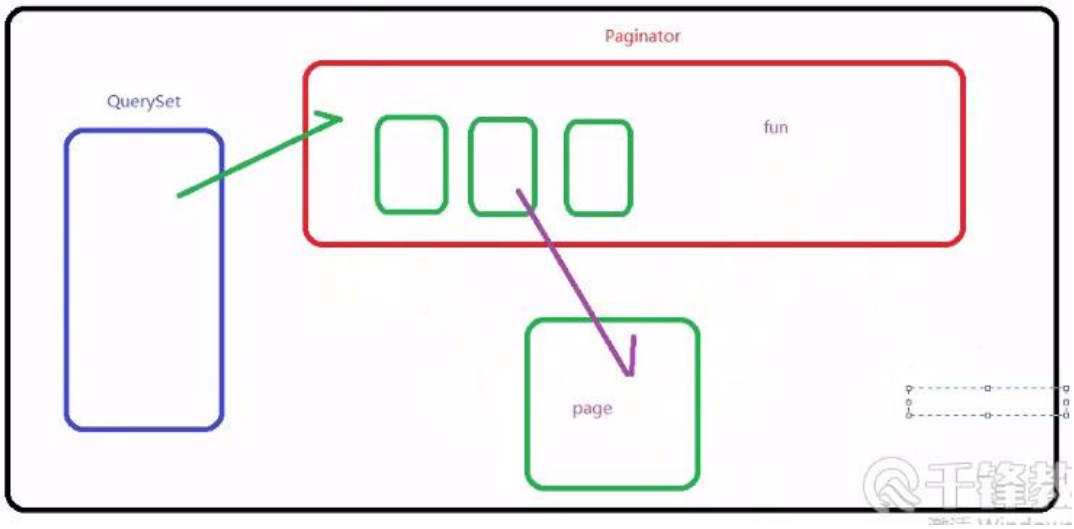
属性:

count对象总数

num_pages：页面总数

page_range: 页码列表，从1开始

方法:　　page(整数): 获得一个page对象



分页

常见错误:

InvalidPage：page()传递无效页码
PageNotAnInteger：page()传递的不是整数
Empty：page()传递的值有效，但是没有数据

Page:

对象获得，通过Paginator的page()方法获得

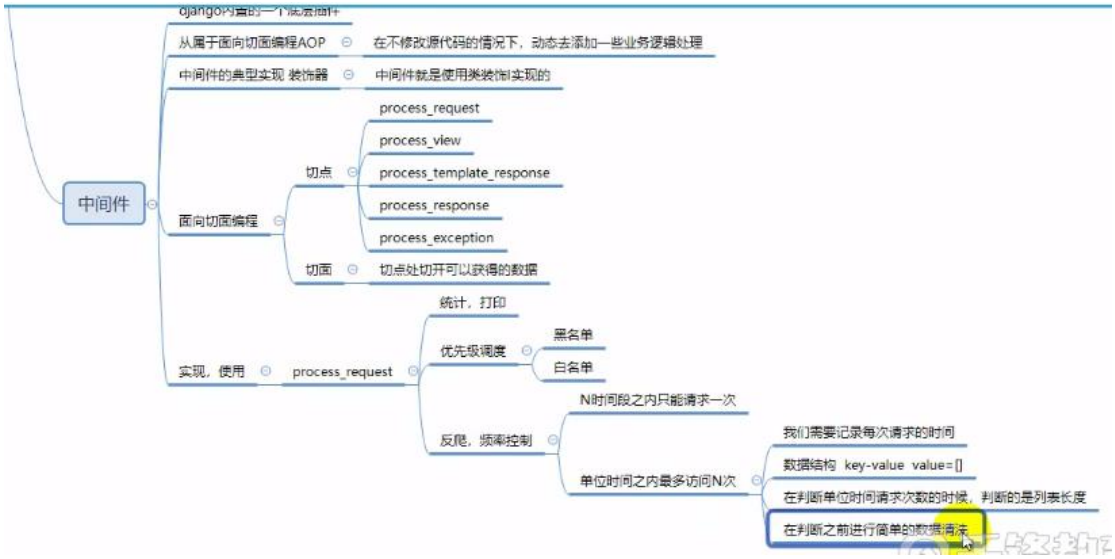
属性:

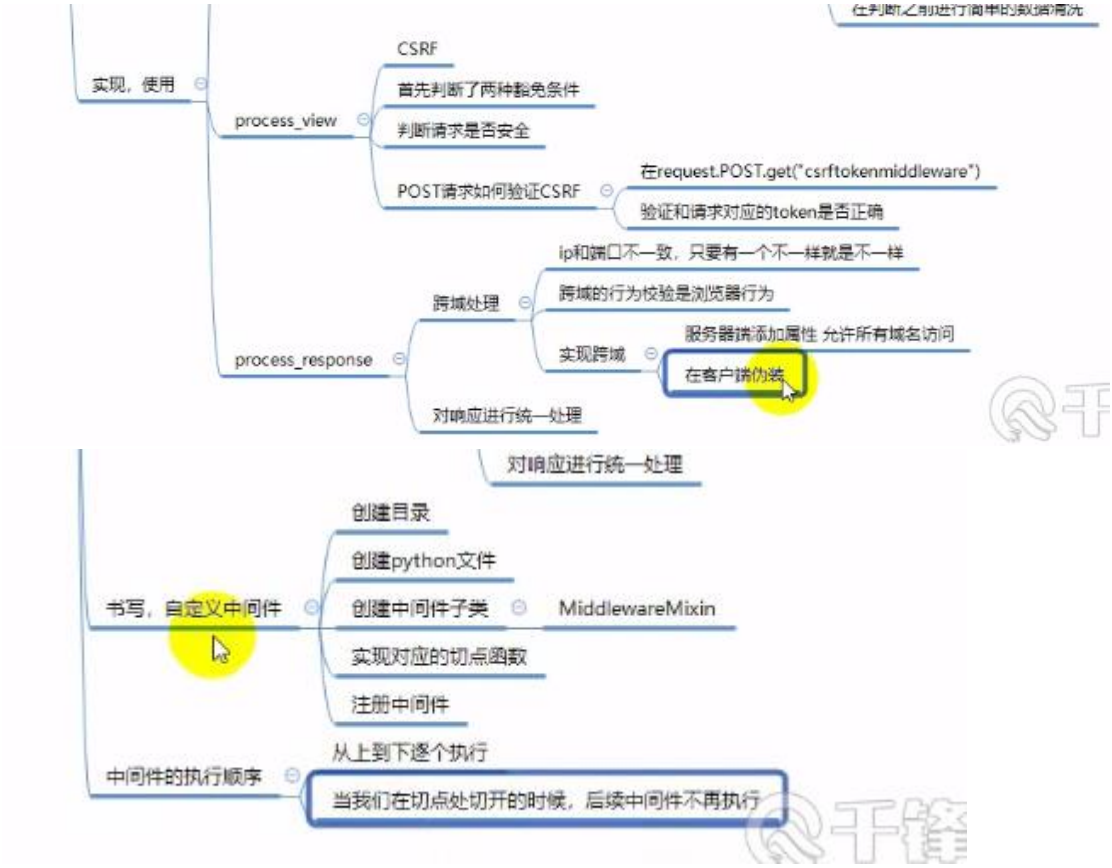
object_list：当前页面上所有的数据对象
number：当前页的页码值
paginator：当前page关联的Paginator对象

分页

方法：

has_next()　:判断是否有下一页
has_previous():判断是否有上一页
has_other_pages():判断是否有上一页或下一页
next_page_number():返回下一页的页码
previous_page_number():返回上一页的页码
len()：返回当前页的数据的个数




在判断之前进行简单的数据清洗
实现，使用
process_view
CSRF
首先判断了两种豁免条件
判断请求是否安全
POST请求如何验证CSRF
在request.POST.get("csrftokenmiddleware")
验证和请求对应的token是否正确
process_response
跨域处理
ip和端口不一致，只要有一个不一样就是不一样
跨域的行为校验是浏览器行为
实现跨域
服务器端添加属性 允许所有域名访问
在客户端伪装
对响应进行统一处理
对响应进行统一处理

书写，自定义中间件
创建目录
创建python文件
创建中间件子类
MiddlewareMixin
实现对应的切点函数
注册中间件
中间件的执行顺序
从上到下逐个执行
当我们在切点处切开的时候，后续中间件不再执行