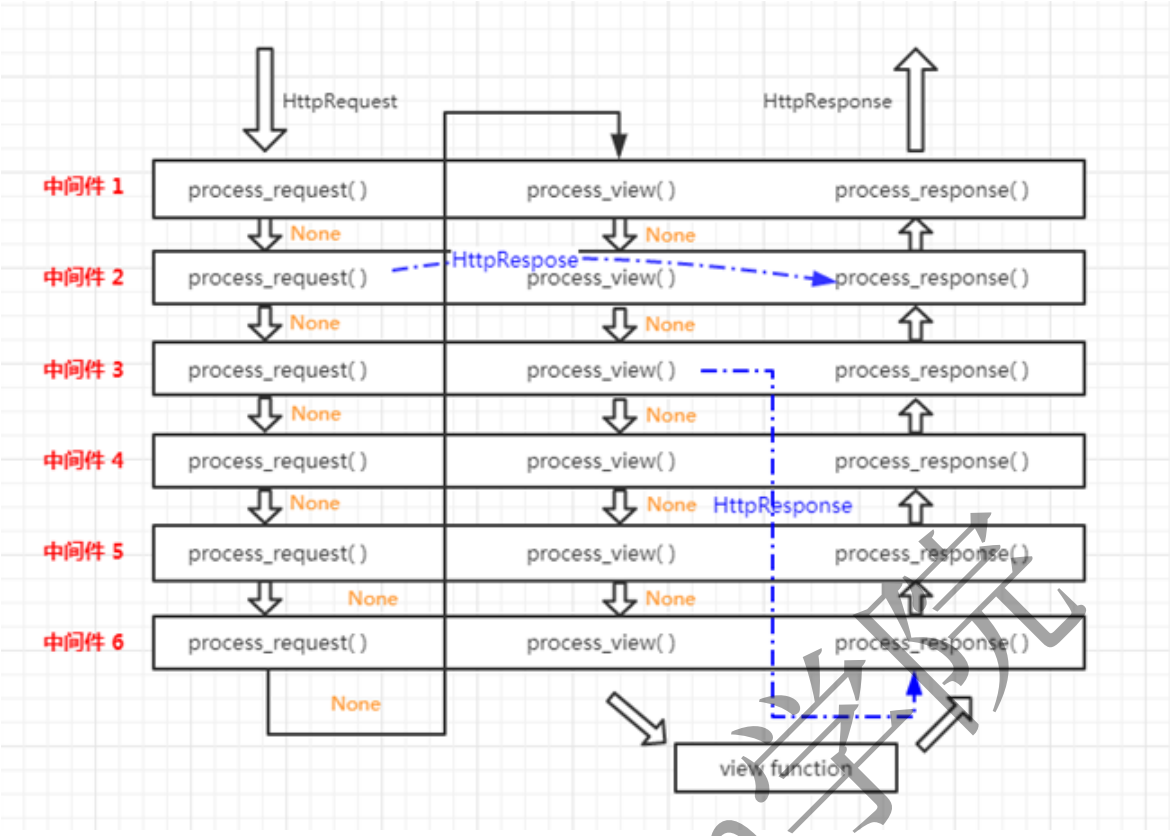
一、中间件

中间件其实就是一个类，是介于request与response处理之间的一道处理过程（类似装饰器），相对比较轻量级，每个中间件都会负责一个功能，例如，AuthenticationMiddleware,与sessions处理相关，中间件，在请求到来和结束后，django会根据自己的规则在合适的时机执行中间件中相应的方法并且在全局上改变django的输入与输出。因为改变的是全局，所以需要谨慎使用，用不好会影响到性能。

1. 中间件执行过程

   请求到达中间件之后，先按照正序执行每个注册中间件的process_reques方法，process_request方法返回的值是None，就依次执行，如果返回的值是HttpResponse对象，不再执行后面的process_request方法，而是执行当前对应中间件的process_response方法，将HttpResponse对象返回给浏览器。也就是说：如果MIDDLEWARE中注册了6个中间件，执行过程中，第3个中间件返回了一个HttpResponse对象，那么第4,5,6中间件的process_request和process_response方法都不执行，顺序执行3,2,1中间件的process_response方法。

   process_request方法都执行完后，匹配路由，找到要执行的视图函数，先不执行视图函数，先执行中间件中的process_view方法，process_view方法返回None，继续按顺序执行，所有process_view方法执行完后执行视图函数。假如中间件3 的process_view方法返回了HttpResponse对象，则4,5,6的process_view以及视图函数都不执行，直接从最后一个中间件，也就是中间件6的process_response方法开始倒序执行。



2. Django中的中间件方法
   - process_request方法
     - 在执行路由前被调用，每个请求上都会调用，不主动进行返回或返回HttpResponse对象
     - 说明

```
process_request(self, request)
参数：request，是一个HttpRequest请求对象
返回值：返回None会继续调用下一个中间件的process_request方法，返回HttpResponse，则执行自己process_response
```

   - process_view方法
     - 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
     - 说明

```
process_view(self,request,view_func,view_args,view_kwargs)
参数:
    request：HttpRequest对象
    view_func：是一个即将调用的视图函数，不是字符串函数名
    view_args：传递给视图函数的位置参数
    view_kwargs：传递给视图函数的关键字参数
返回值：如果返回None，会继续执行处理此请求，然后调用下一个中间件的process_view，直至执行视图函数；如果返回HttpResponse,则直接执行最后一个中间件的process_response
```

- process_template_response方法
  - 在视图刚好执行完后进行调用，只要视图返回一个render方法返回的对象，就会调用process_template_response，不主动进行返回或返回HttpResponse对象

```
process_template_response(self, request, response)
参数：request  HttpRequest对象
      response 是一个由Django view或者中间件返回的TemplateResponse 对象
返回值： 必须返回一个render方法执行后的response对象，它可以修改view中返回的 response.template_name 和 response.context_data，或者为view返回的模板增加一个商标等等。你不需要明确的渲染响应，当所有的template响应中间件处理完成后会被自动渲染。
```

- process_response
  - 所有响应返回浏览器之前调用，每个请求都会调用，返回HttpResponse对象

```
process_response(self,request,response)
参数：request HttpRequest对象
     response  HttpResponse对象
返回值：必须是HttpResponse对象
```

- process_exception
  - 当视图抛出异常时调用，返回None或返回HttpResponse对象

```
process_exception(self,request,exception)
参数：request:  HttpRequest 对象
     exception:view函数中raise的Exception对象，当view 函数raise一个exception的时候调用process_exception
```

3. 可实现功能
    - 统计
    - 黑名单
    - 白名单
    - 界面友好化（捕获异常）
4. 实现

```
#App下middleware.py
from django.utils.deprecation import MiddlewareMixin
class MyMiddle1(MiddlewareMixin):

    def process_request(self,request):
        pass
    def process_view(self, request, callback, callback_args, callback_kwargs):
        i =1
        pass
    def process_exception(self, request, exception):
        pass

    def process_response(self, request, response):
        return response
```

    - 启用中间件

在settings中进行配置，MIDDLEWARE中添加：模块名.middleware.类名

```
MIDDLEWARE = [
   .......
    'App.middleware.MyMiddle1',
]
```

二、缓存

缓存是一类可以更快的读取数据的介质统称，也指其它可以加快数据读取的存储方式。

在Django中，当用户请求到达视图后，视图会先从数据库提取数据放到模板中进行动态渲染，渲染后的结果就是用户看到的网页。如果用户每次请求都从数据库提取数据并渲染，将极大降低性能，不仅服务器压力大，而且客户端也无法即时获得响应。如果能将渲染后的结果放到速度更快的缓存中，每次有请求过来，先检查缓存中是否有对应的资源，如果有，直接从缓存中取出来返回响应，节省取数据和渲染的时间，不仅能大大提高系统性能，还能提高用户体验。

缓存使用场景：缓存主要适用于对页面实时性要求不高的页面。存放在缓存的数据，通常是频繁访问的，而不会经常修改的数据。

缓存方式：数据库、文件、内存、redis等

1 缓存配置

- 数据库缓存

```
# settings.py
CACHES = {
    'default':{
        'BACKEND':'django.core.cache.backends.db.DatabaseCache',
        'LOCATION':'my_cache_table',
    }
}
#生成缓存表
python manage.py createcachetable
```

- 文件缓存

```
# CACHES = {
#  'default': {
#   'BACKEND':
'django.core.cache.backends.filebased.FileBasedCache', #指定缓存使
用的引擎
#   'LOCATION': '/var/tmp/django_cache',         #指定缓存的路径
#   'TIMEOUT':300,                #缓存超时时间(默认为300秒,None表示永
不过期)
#   'OPTIONS':{
#    'MAX_ENTRIES': 300,             # 最大缓存记录的数量（默认300）
#    'CULL_FREQUENCY': 3,            # 缓存到达最大个数之后，剔除缓存
个数的比例，即：1/CULL_FREQUENCY（默认3）
#   }
#  }
# }
```

- redis缓存

  需要安装：pip3 install django-redis

```
CACHES = {
    'default':{
        'BACKEND':'django_redis.cache.RedisCache',#指定缓存类型
redis缓存
        'LOCATION':'redis://:123@127.0.0.1:6379/1', #缓存地址，@前
面是reids的密码，如果没有则去掉
       # 'LOCATION':'redis://127.0.0.1:6379/1', # 没密码
    }
}
```

2 缓存使用

- 视图函数

```
from django.views.decorators.cache import cache_page
@cache_page(60 * 0.5)  #缓存过期时间
def dbcache(request):
    return render(request,'index.html',context={'content':77665})
```

- 模板局部缓存

```
{% load cache %}
<!DOCTYPE html>
<html lang="en">
<head>
    <meta charset="UTF-8">
    <title>Title</title>
</head>
<body>
{% cache 30 'content' %}
{{ content }}
{% endcache %}
</body>
</html>
```

- 全站缓存

```
MIDDLEWARE = [
    'django.middleware.cache.UpdateCacheMiddleware',
    .....
    'django.middleware.cache.FetchFromCacheMiddleware',
]
CACHE_MIDDLEWARE_SECONDS = 20  #设置超时时间 20秒
```

- 手动设置缓存
  - 设置缓存：cache.set（key,value,timeout)
  - 获取缓存：cache.get(key)

```
def myCache(req):
    # 从缓存获取页面
    mycache = cache.get('mycache')
    if mycache:
        print('走缓存了')
        html = mycache
    else:
        print('没走缓存')
        tem = loader.get_template('mycache.html')
        html = tem.render({'con':'缓存测试的模板'})
        cache.set('mycache',html,60)
    return HttpResponse(html)
```