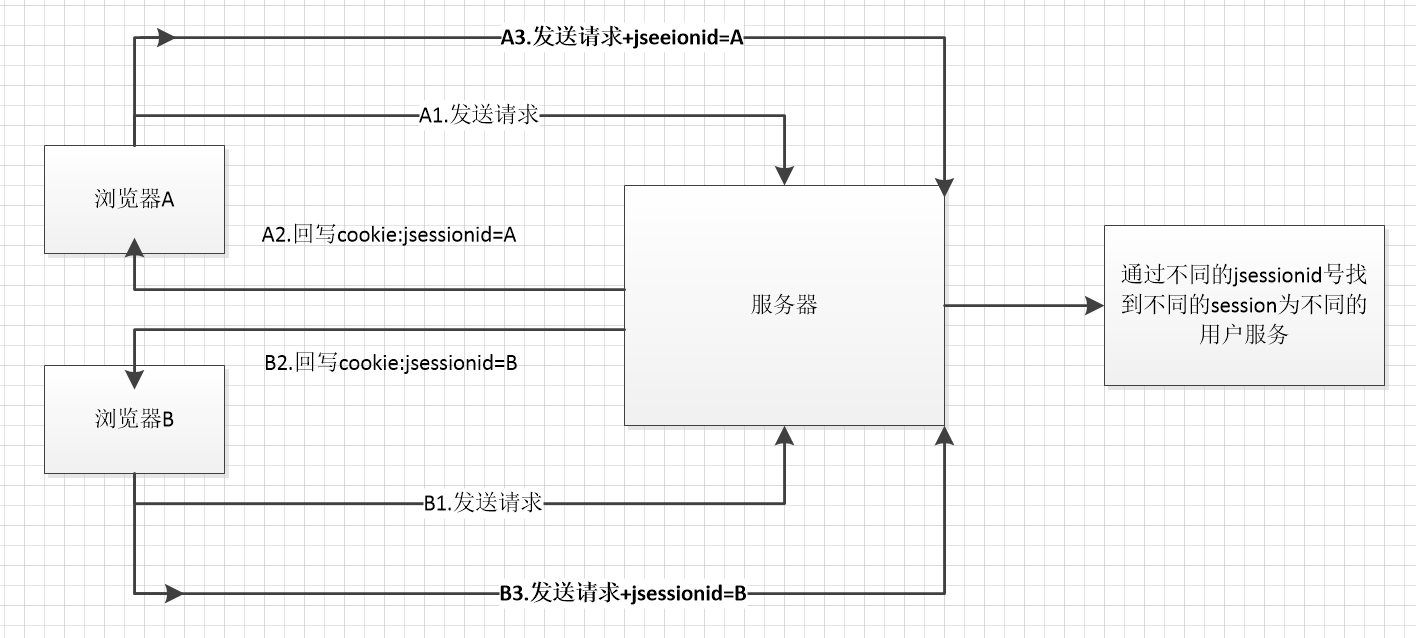
session的底层是基于cookie技术来实现的，在早器保存会话状态一直用的cookie，cookie是保存在客户端的明文键值对，但这样直接保存重要信息是不安全的，后来衍生了session，session是加密后的cookie键值，加密后的值即使被人截取，也无法获取到密码等敏感信息，避免了重要资源信息泄露。当用户打开浏览器，去访问服务器的时候，服务器会为每个用户的浏览器创建一个会话对象(session对象)，并且为每个session对象创建一个Jsessionid号。当session对象创建成功后，会以cookie的方式将这个Jsessionid号回写给浏览器，当用户再次进行访问服务器时，及带了具有Jsessionid号的cookie数据来一起访问服务器，服务器通过不同session的 Jsessionid号来找出与其相关联的session对象，通过不同的session对象来为不同的用户服务。



3.在DJango中session的使用方法

Django中默认支持Session，其内部提供了5种类型的Session供开发者使用：

**·**数据库（默认）  
　　**·**缓存  
　　**·**文件  
　　**·**缓存+数据库  
　　**·**加密cookie

（1）数据库中的Session

Django默认支持Session，并且默认是将Session数据存储在数据库中，即：django\_session 表中。

a. 配置 settings.py

SESSION\_ENGINE = 'django.contrib.sessions.backends.db' # 引擎（默认）

SESSION\_COOKIE\_NAME ＝ "sessionid" # Session的cookie保存在浏览器上时的key，即：sessionid＝随机字符串（默认）

SESSION\_COOKIE\_PATH ＝ "/" # Session的cookie保存的路径（默认）

SESSION\_COOKIE\_DOMAIN = None # Session的cookie保存的域名（默认）

SESSION\_COOKIE\_SECURE = False # 是否Https传输cookie（默认）

SESSION\_COOKIE\_HTTPONLY = True # 是否Session的cookie只支持http传输（默认）

SESSION\_COOKIE\_AGE = 1209600 # Session的cookie失效日期（2周）（默认）

SESSION\_EXPIRE\_AT\_BROWSER\_CLOSE = False # 是否关闭浏览器使得Session过期（默认）

SESSION\_SAVE\_EVERY\_REQUEST = False # 是否每次请求都保存Session，默认修改之后才保存（默认）

b. 使用

def index(request):

# 获取、设置、删除Session中数据

request.session['k1']

request.session.get('k1',None)

request.session['k1'] = 123

request.session.setdefault('k1',123) # 存在则不设置

del request.session['k1']

# 所有 键、值、键值对

request.session.keys()

request.session.values()

request.session.items()

request.session.iterkeys()

request.session.itervalues()

request.session.iteritems()

# 用户session的随机字符串

request.session.session\_key

# 将所有Session失效日期小于当前日期的数据删除

request.session.clear\_expired()

# 检查 用户session的随机字符串 在数据库中是否

request.session.exists("session\_key")

# 删除当前用户的所有Session数据

request.session.delete("session\_key")

request.session.clear()

request.session.set\_expiry(value)

\* 如果value是个整数，session会在些秒数后失效。

\* 如果value是个datatime或timedelta，session就会在这个时间后失效。

\* 如果value是0,用户关闭浏览器session就会失效。

\* 如果value是None,session会依赖全局session失效策略。

（2）缓存Session

a. 配置 settings.py

    SESSION\_ENGINE = 'django.contrib.sessions.backends.cache'  # 引擎

    SESSION\_CACHE\_ALIAS = 'default'                            # 使用的缓存别名（默认内存缓存，也可以是memcache），此处别名依赖缓存的设置

    SESSION\_COOKIE\_NAME ＝ "sessionid"                        # Session的cookie保存在浏览器上时的key，即：sessionid＝随机字符串

    SESSION\_COOKIE\_PATH ＝ "/"                                # Session的cookie保存的路径

    SESSION\_COOKIE\_DOMAIN = None                              # Session的cookie保存的域名

    SESSION\_COOKIE\_SECURE = False                             # 是否Https传输cookie

    SESSION\_COOKIE\_HTTPONLY = True                            # 是否Session的cookie只支持http传输

    SESSION\_COOKIE\_AGE = 1209600                              # Session的cookie失效日期（2周）

    SESSION\_EXPIRE\_AT\_BROWSER\_CLOSE = False                   # 是否关闭浏览器使得Session过期

    SESSION\_SAVE\_EVERY\_REQUEST = False                        # 是否每次请求都保存Session，默认修改之后才保存

b. 使用

    同上

（3）文件Session

a. 配置 settings.py

    SESSION\_ENGINE = 'django.contrib.sessions.backends.file'    # 引擎

    SESSION\_FILE\_PATH = None                                    # 缓存文件路径，如果为None，则使用tempfile模块获取一个临时地址tempfile.gettempdir()                                                            # 如：/var/folders/d3/j9tj0gz93dg06bmwxmhh6\_xm0000gn/T

    SESSION\_COOKIE\_NAME ＝ "sessionid"                          # Session的cookie保存在浏览器上时的key，即：sessionid＝随机字符串

    SESSION\_COOKIE\_PATH ＝ "/"                                  # Session的cookie保存的路径

    SESSION\_COOKIE\_DOMAIN = None                                # Session的cookie保存的域名

    SESSION\_COOKIE\_SECURE = False                               # 是否Https传输cookie

    SESSION\_COOKIE\_HTTPONLY = True                              # 是否Session的cookie只支持http传输

    SESSION\_COOKIE\_AGE = 1209600                                # Session的cookie失效日期（2周）

    SESSION\_EXPIRE\_AT\_BROWSER\_CLOSE = False                     # 是否关闭浏览器使得Session过期

    SESSION\_SAVE\_EVERY\_REQUEST = False                          # 是否每次请求都保存Session，默认修改之后才保存

b. 使用

    同上

（4）缓存+数据库Session

数据库用于做持久化，缓存用于提高效率

a. 配置 settings.py

    SESSION\_ENGINE = 'django.contrib.sessions.backends.cached\_db'        # 引擎

b. 使用

    同上

（5）加密Session

a. 配置 settings.py

    SESSION\_ENGINE = 'django.contrib.sessions.backends.signed\_cookies'   # 引擎

b. 使用

    同上

（6）Session的用户验证

def login(func):

    def wrap(request, \*args, \*\*kwargs):

        # 如果未登陆，跳转到指定页面

        if request.path == '/test/':

            return redirect('http://www.baidu.com')

        return func(request, \*args, \*\*kwargs)

    return wrap

**Session 的基本操作：**

* 以键值对的格式写 session

request.session['键']=值

* 根据键读取值

request.session.get('键',默认值)

# 或者

request.session['键']

* 清除所有 session，在存储中删除值的部分

request.session.clear()

* 清除 session 数据，在存储中删除 session 的整条数据

request.session.flush()

* 删除 session 中的指定键及值，在存储中只删除某个键及对应的值

del request.session['键']

* 设置 session 数据有效时间； 如果不设置，默认过期时间为两周

request.session.set\_expiry(value)

1. 如果过期时间的 value 是一个整数，则 session 数据 将在 value 秒没有活动后过期。
2. 如果过期时间的 value 为 None，那么会话永不过期。
3. 如果过期时间的 value 为 0，那么用户会话的 Cookie 将在用户的浏览器关闭时过期。

利用sesssion做个用户认证示例：

Views.py

from django.shortcuts import render,redirect,HttpResponse

def login(request):

if request.method=='GET':

return render(request,'login.html')

elif request.method=="POST":

user=request.POST.get('username')

pwd=request.POST.get('password')

if user=="root" and pwd=="123":

if request.POST.get('box')=="1": #checkbox被按下

request.session.set\_expiry(10) #session认证时间为10s，10s之后session认证失效

request.session['username']=user #user的值发送给session里的username

request.session['is\_login']=True #认证为真

return redirect('/index')

else:

return redirect('/login')

return render(request,'login.html')

def index(request):

if request.session.get('is\_login',None): #若session认证为真

return render(request,'index.html',{'username':request.session['username']})

else:

return HttpResponse('滚')

def logout(request): #撤销

request.session.clear() #删除session里的全部内容

return redirect('/login')

login.html

<!DOCTYPE html>

<html lang="en">

<head>

<meta charset="UTF-8">

<title>Title</title>

<script src="/static/js/jquery-1.12.4.js"></script>

<script src="/static/js/jquery.cookie.js"></script>

</head>

<body>

<form action="/login/" method="post">

<p>

<input type="text" name="username" placeholder="username" />

</p>

<!----------CSRF\_TOKEN通过form表单发送到后台进行验证---------->

{% csrf\_token %}

<p>

<input type="password" name="password" placeholder="password" />

</p>

<p>

<input type="checkbox" name="box" value="1" /> 10s超时

</p>

<p>

<input type="submit" value="提交" />

</p>

<input type="button" id="btu-1" value="Ajax-1" />

</form>

<script>

$(function () {

$.ajaxSetup({

beforeSend:function (xhr,settings) {

xhr.setRequestHeader('X-CSRFtoken',$.cookie('csrftoken'))

}

});

$('#btu-1').click(function () {

$.ajax({

url:"/login/",

type:"POST",

data:{'username':"root","password":"123"},

})

})

})()

index.html

<head>

<meta charset="UTF-8">

<title>Title</title>

</head>

<h1>欢迎登录：{{ username }},{{ request.session.username }}</h1>

<a href="http://127.0.0.1:8006/logout/"><input type="button" value="注销"/></a>

</body>

</html>

发送短信示例

# 发短信接口

def sms\_send(request):

# http://localhost:8000/duanxin/duanxin/sms\_send/?phone=18434288349

# 1 获取手机号

phone = request.GET.get('phone')

# 2 生成6位验证码

code = aliyunsms.get\_code(6, False)

# 3 缓存到Redis

#cache.set(phone,code,60) #60s有效期

#print('判断缓存中是否有:',cache.has\_key(phone))

#print('获取Redis验证码:',cache.get(phone))

#暂时用session处理

request.session['phone'] = code

request.session.set\_expiry(300) #设置5分钟后过期

print('判断缓存中是否有:', request.session.get('phone'))

print('获取session验证码:',request.session.get('phone'))

# 4 发短信

result = aliyunsms.send\_sms(phone, code)

return HttpResponse(result)

# 短信验证码校验

def sms\_check(request):

# /duanxin/sms\_check/?phone=xxx&code=xxx

# 1. 电话和手动输入的验证码

phone = request.GET.get('phone')

code = request.GET.get('code')

# 2. 获取redis中保存的code

#print('缓存中是否包含:',cache.has\_key(phone))

#print('取值:',cache.get(phone))

#cache\_code = cache.get(phone)

#获取session里的code

print('取值:', request.session.get('phone'))

cache\_code = request.session.get('phone')

# 3. 判断

if code == cache\_code:

return HttpResponse(json.dumps({'result':'OK'}))

else:

return HttpResponse(json.dumps({'result':'False'}))

## Django session验证的三种姿势

## 一.什么是session

session是保存在服务端的键值对，Django默认支持Session，并且默认是将Session数据存储在数据库中，即：django\_session 表中。

## 二.FVB中使用装饰器进行session验证

**认证装饰器：**

|  |  |
| --- | --- |
|  | # 登陆验证  def auth(func):   '''判断是否登录装饰器'''      def inner(request, \*args, \*\*kwargs):          ck = request.session.get("username")          '''如果没有登陆返回到login.html'''          if not ck:              return redirect("/login.html")          return func(request, \*args, \*\*kwargs)      return inner |

**在需要认证的函数执行前加上装饰器认证即可，实际中应用如下：**

|  |  |
| --- | --- |
|  | def login(request):      if request.method == "GET":          return render(request, "login.html")      else:          username = request.POST.get("user")          pwd = request.POST.get("pwd")          pwd = md5(pwd)          dic = {"flag":False}          obj = User.objects.filter(username=username, pwd=pwd).first()          if obj:              request.session["username"] = username              return redirect("/index.html")          else:              print(dic)              return HttpResponse(json.dumps(dic))    @auth  def index(request):      user = request.session.get("username")      business = Business.objects.all().values("name")      host\_list = Host.objects.all().values("id","host","port","business\_\_name")      username = User.objects.all().values("username")      return render(request, 'index.html', {'host\_list':host\_list,"business":business,"user":user,"username":username})    @auth  def addhost(request):      business = Business.objects.all().values("name")      if request.method == "POST":          user = request.session.get("username")          host = request.POST.get("host")          port = request.POST.get("port")          select\_business = request.POST.get("business")          business\_id = Business.objects.filter(name=select\_business).values("id")[0]          host = Host.objects.create(host=host,                                    port=port,                                    business\_id=business\_id["id"])          # host.business.add(\*business)          return render(request, "index.html")        return render(request, "index.html", {"business":business})    @auth  def up\_business(request):      if request.method == "POST":          user = request.session.get("username")          host= request.POST.get("host")          port= request.POST.get("port")          business\_name = request.POST.get("business")          username = request.POST.get("username")          print(host,port,business\_name,username)          return render(request,"保存成功") |

## 三.CBV中使用类继承的方式进行session认证

* cbv是 class based view(基于类)
* cbv基于dispatch进行反射,get获取,post提交
* 应用场景：登录认证(继承dispatch,在dispatch里做session验证)

### CBV第一种方式继承

#### 1.单继承

**扫盲：(继承的时候,一定要清楚self是哪个类实例化出来的对象,下例,self为B实例化的对象,任何属性优先从自己里面找,找不到在去父类里找)**

|  |  |
| --- | --- |
|  | class A(object):      def aaa(self):          print('from A')      def bbb(self):          self.aaa()    class B(A):      def aaa(self):          print('from B')    c = B()  c.aaa() |

**应用：**

|  |  |
| --- | --- |
|  | from django.views import View  class BaseView(View):      def dispatch(self, request, \*args, \*\*kwargs):  # 继承父类的dispatch,因为父类里有返回值,所以也要有return          if request.session.get('username'):              response = super(BaseView, self).dispatch(request, \*args, \*\*kwargs)              return response          else:              return redirect('/login.html')    class IndexView(BaseView):        def get(self, request, \*args, \*\*kwargs):          return HttpResponse(request.session['username']) |

#### 2.多继承(继承顺序从左到右)

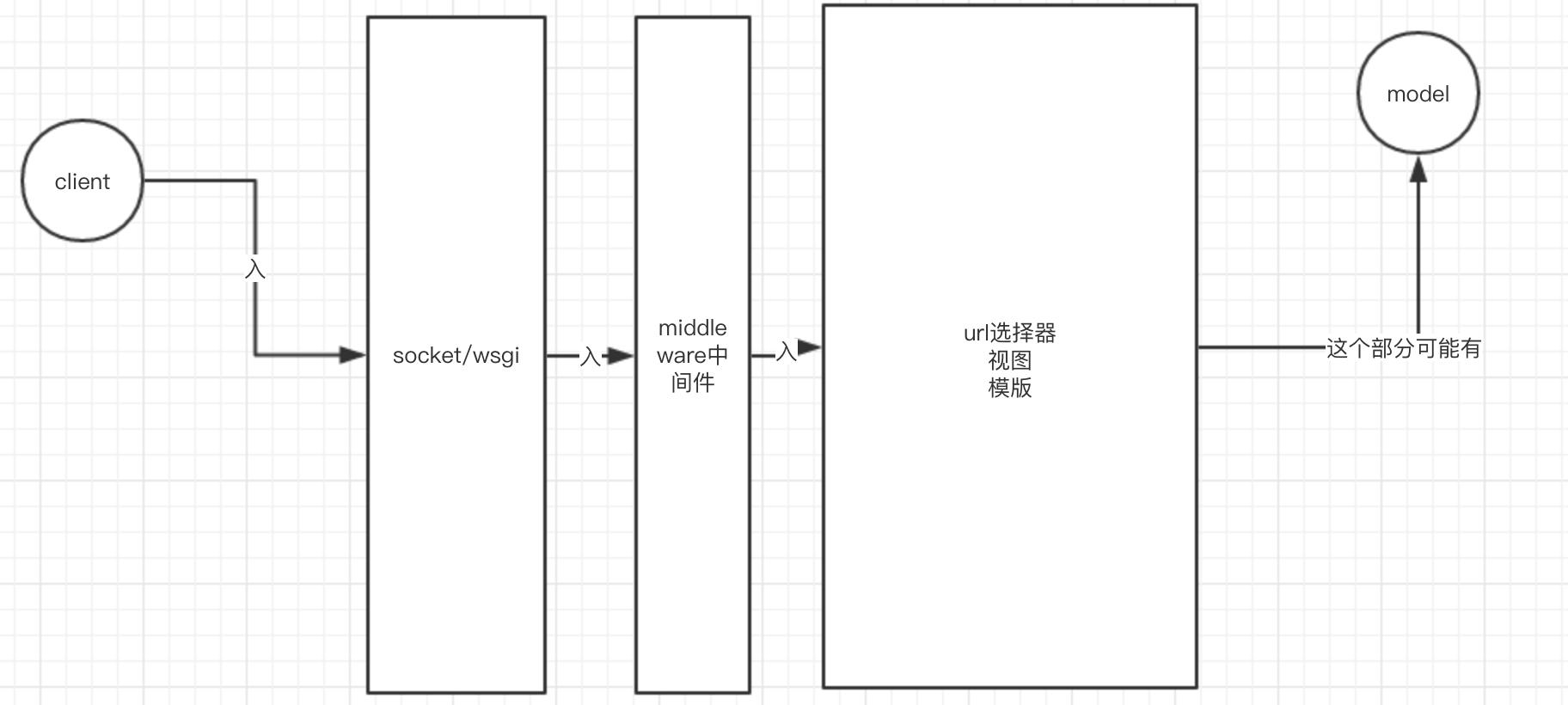
|  |  |
| --- | --- |
|  | class BaseView(object):      def dispatch(self, request, \*args, \*\*kwargs):          if request.session.get('username'):              response = super(BaseView,self).dispatch(request, \*args, \*\*kwargs)              return response          else:              return redirect('/login.html')    class IndexView(BaseView,View):#先去找BaseView,BaseView中未定义在去找View        def get(self,request,\*args,\*\*kwargs):          return HttpResponse(request.session['username']) |

### CBV第二种方式装饰器

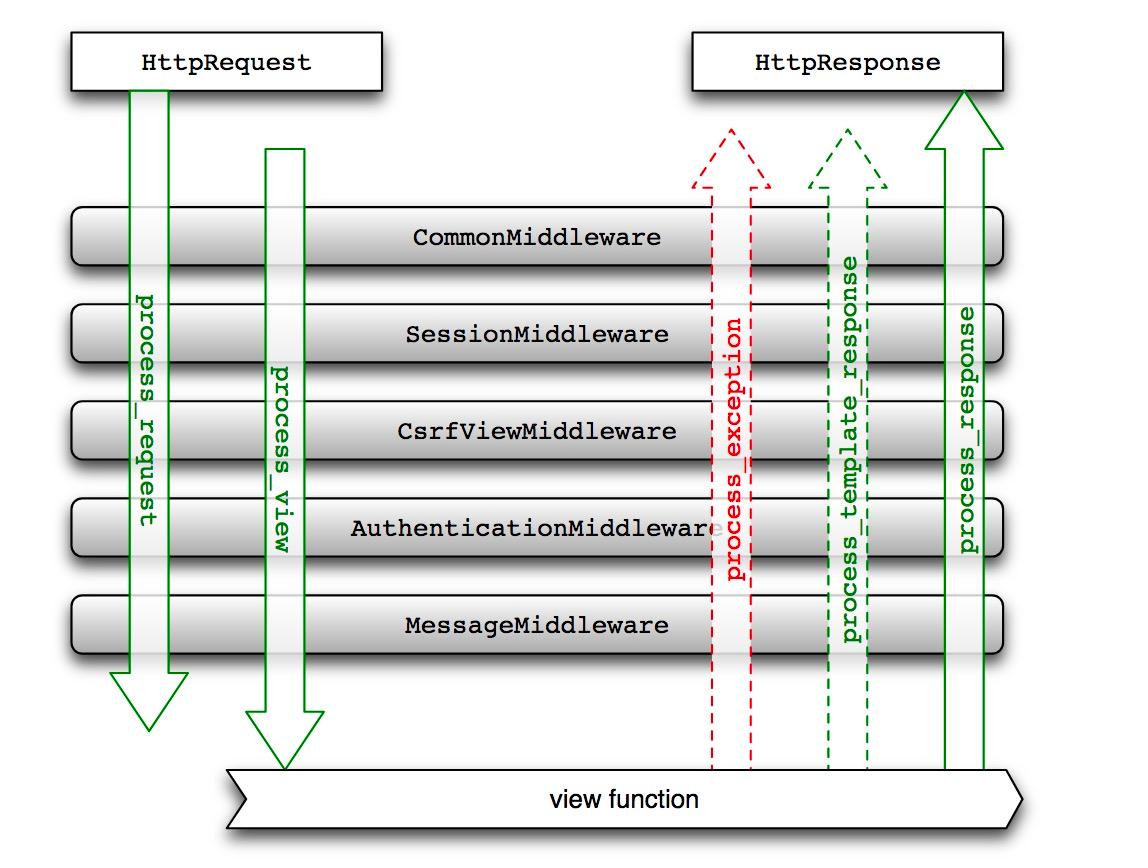
|  |  |
| --- | --- |
|  | from django.utils.decorators import method\_decorator    def auth(func): #定义装饰器      def inner(request,\*args,\*\*kwargs):          if request.session.get('username'):              obj = func(request,\*args,\*\*kwargs)              return obj          else:              return redirect('/login.html')      return inner    @method\_decorator(auth,name='get')  #放在类顶部就需要method\_decorator这个装饰器  class IndexView(View):        @method\_decorator(auth) #放在dispatch上就相当于全局都需要经过认证      def dispatch(self, request, \*args, \*\*kwargs):          if request.session.get('username'):              response = super(IndexView,self).dispatch(request, \*args, \*\*kwargs)              return response          else:              return redirect('/login.html')        @method\_decorator(auth)      def get(self,request,\*args,\*\*kwargs):          return HttpResponse(request.session['username'])        @method\_decorator(csrf\_exempt)  # 无效 csrf 放到post函数上的装饰器,是无效的,需要放到dispath上或者类上      def post(self,request,\*args,\*\*kwargs):          return HttpResponse(request.session['username']) |

## 四.中间件middleware

如下是django的生命周期



 如下为中间件的执行顺序



 - 中间件执行时机：请求到来，请求返回时  
　　- 中间件是一个类：  
　　def process\_request(self,request):  
　　　　print('m2.process\_request')

　　def process\_response(self,request, response):  
　　　　print('m2.prcess\_response')  
　　　　return response  
  
　　　　- 应用：  
　　　　- 请求日志  
　　　　- 用户登录认证

Django根目录新建md文件夹，新建Middleware.py文件

|  |  |
| --- | --- |
|  | from django.utils.deprecation import MiddlewareMixin    class M1(MiddlewareMixin):      '''先执行request,然后到url路由,url之后返回到最上方,在执行view,如果出现错误就直接到response上,执行完,到真正到视图,如果有问题就      执行exception,从下至上查找,如果找到exception就直接执行exception的return在走response返回用户      每个中间件中,4个方法不需要都写.      '''      def process\_request(self,request):            if request.path\_info == "/login.html":              return None          user\_info = request.session.get("username")          if not user\_info:              return redirect("/login.html") |

 注:新的django版本可能不存在MiddlewareMixin,需要手动写一下这个类进行继承

|  |  |
| --- | --- |
|  | class MiddlewareMixin(object):      def \_\_init\_\_(self, get\_response=None):          self.get\_response = get\_response          super(MiddlewareMixin, self).\_\_init\_\_()        def \_\_call\_\_(self, request):          response = None          if hasattr(self, 'process\_request'):              response = self.process\_request(request)          if not response:              response = self.get\_response(request)          if hasattr(self, 'process\_response'):              response = self.process\_response(request, response)          return response    class M1(MiddlewareMixin):        def process\_request(self,request):            if request.path\_info == "/login.html":              return None          user\_info = request.session.get("username")          if not user\_info:              return redirect("/login.html") |

 settings里配置：

|  |  |
| --- | --- |
|  | MIDDLEWARE = [      'django.middleware.security.SecurityMiddleware',      'django.contrib.sessions.middleware.SessionMiddleware',      'django.middleware.common.CommonMiddleware',      'django.middleware.csrf.CsrfViewMiddleware',      'django.contrib.auth.middleware.AuthenticationMiddleware',      'django.contrib.messages.middleware.MessageMiddleware',      'django.middleware.clickjacking.XFrameOptionsMiddleware',      'md.Middleware.M1',  ]    WSGI\_APPLICATION = 'BBS.wsgi.application'    SESSION\_ENGINE = 'django.contrib.sessions.backends.db'  # 引擎（默认）    SESSION\_COOKIE\_NAME = "sessionid"  # Session的cookie保存在浏览器上时的key，即：sessionid＝随机字符串（默认）  SESSION\_COOKIE\_PATH = "/"  # Session的cookie保存的路径（默认）  SESSION\_COOKIE\_DOMAIN = None  # Session的cookie保存的域名（默认）  SESSION\_COOKIE\_SECURE = False  # 是否Https传输cookie（默认）  SESSION\_COOKIE\_HTTPONLY = True  # 是否Session的cookie只支持http传输（默认）  SESSION\_COOKIE\_AGE = 1209600  # Session的cookie失效日期（2周）（默认）  SESSION\_EXPIRE\_AT\_BROWSER\_CLOSE = False  # 是否关闭浏览器使得Session过期（默认）  SESSION\_SAVE\_EVERY\_REQUEST = True  # 是否每次请求都保存Session，默认修改之后才保存（默认） |