1、两个模块的位置

from django.shortcuts import render, render\_to\_response

2、作用

两者均是用来

render\_to\_response() 一次性地载入某个模板文件，渲染它，然后将此作为 HttpResponse返回展示模板页面的

可传入相应的参数，以字典的方式

3、参数区别

最明显的一个，render的第一个参数是request,后面的参数则和render\_to\_response相同

例如：

render\_to\_response('weibologin/login.html', {'current\_time': datetime.now(),'error':error1})

render(request, 'search.html', {'current\_time': datetime.now()})

4、功能区别

由于传入参数的不同，造成了一个最直接的问题：

能否在模板中使用request的属性，例如session等。

所以，如果你要在模板中使用request的属性，请务必使用render。否则模板渲染时是得不到相应的request变量值的

5、什么是CSRF？

CSRF（Cross-site request forgery），中文名称：跨站请求伪造，也被称为：one click attack/session riding，缩写为：CSRF/XSRF。

原理如下：



从上图可以看出，要完成一次CSRF攻击，受害者必须依次完成两个步骤 ：

1.登录受信任网站A，并在本地生成Cookie 。

2.在不退出A的情况下，访问危险网站B。

ps：注意并不是你退出了A后登陆B就会没事，因为你不能保证你关闭浏览器了后，你本地的Cookie立刻过期，你上次的会话已经结束

CSRF预防机制

CSRF的防御可以从服务端和客户端两方面着手，防御效果是从服务端着手效果比较好，现在一般的CSRF防御也都在服务端进行。

token防御的整体思路是：

第一步：后端随机产生一个token，把这个token保存在SESSION状态中；同时，后端把这个token交给前端页面；

第二步：下次前端需要发起请求（比如发帖）的时候把这个token加入到请求数据或者头信息中，一起传给后端；

第三步：后端校验前端请求带过来的token和SESSION里的token是否一致；

Django下的CSRF预防机制

django 第一次响应来自某个客户端的请求时，会在服务器端随机生成一个 token，把这个 token 放在 cookie 里。然后每次 POST （GET请求则无）请求都会带上这个 token，这样就能避免被 CSRF 攻击。

在 templete 中, 为每个 POST form 增加一个 {% csrf\_token %} tag. 如下:

在返回的 HTTP 响应的 cookie 里，django 会为你添加一个 csrftoken 字段，其值为一个自动生成的 token

在所有的 POST 表单模板中，加一个{% csrf\_token %} 标签，它的功能其实是给form增加一个隐藏的input标签，如下

<input type="hidden" name="csrfmiddlewaretoken" value="{{ csrf\_token }}">，而这个csrf\_token = cookie.csrftoken，在渲染模板时context中有context['csrf\_token'] = request.COOKIES['csrftoken']

在通过表单发送POST到服务器时，表单中包含了上面隐藏了crsrmiddlewaretoken这个input项，服务端收到后，django 会验证这个请求的 cookie 里的 csrftoken 字段的值和提交的表单里的 csrfmiddlewaretoken 字段的值是否一样。如果一样，则表明这是一个合法的请求，否则，这个请求可能是来自于别人的 csrf 攻击，返回 403 Forbidden.

在通过 ajax 发送POST请求到服务器时，要求增加一个x-csrftoken header，其值为 cookie 里的 csrftoken 的值，服务湍收到后，django会验证这个请求的cookie里的csrftoken字段与ajax post消息头中的x-csrftoken header是否相同，如果相同，则表明是一个合法的请求

体实现方法

django为用户实现防止跨站请求伪造的功能，通过中间件 django.middleware.csrf.CsrfViewMiddleware 来完成。而对于django中设置防跨站请求伪造功能有分为全局和局部。

全局：

　　中间件 django.middleware.csrf.CsrfViewMiddleware

局部：

@csrf\_protect，为当前函数强制设置防跨站请求伪造功能，即便settings中没有设置全局中间件。

@csrf\_exempt，取消当前函数防跨站请求伪造功能，即便settings中设置了全局中间件。

注：from django.views.decorators.csrf import csrf\_exempt,csrf\_protect

1、原理

　　在客户端页面上添加csrftoken, 服务器端进行验证，服务器端验证的工作通过'django.middleware.csrf.CsrfViewMiddleware'这个中间层来完成。在django当中防御csrf攻击的方式有两种：

　　 1.在表单当中附加csrftoken

　　 2.通过request请求中添加X-CSRFToken请求头。

注意:Django默认对所有的POST请求都进行csrftoken验证，若验证失败则403错误侍候。

故

return render\_to\_response('Login.html',data,context\_instance=RequestContext(request))

#在渲染模块时，使用 RequestContext。RequestContext 会处理 csrf\_token 这个 tag, 从而自动为表单添加一个名为 csrfmiddlewaretoken 的 input

或者

return render(request, 'xxx.html', data) #使用render则会自动生成，不用ReqestContext

当使用

return render\_to\_response('Login.html',data,context\_instance=RequestContext(request))　在新版的Django中还会出现以下错误：

render\_to\_response() got an unexpected keyword argument 'context\_instance'

错误原因是：Django更新了

查看源码：进入原函数

def render\_to\_response(template\_name, context=None, content\_type=None, status=None, using=None):

"""

Return a HttpResponse whose content is filled with the result of calling

django.template.loader.render\_to\_string() with the passed arguments.

"""

warnings.warn(

'render\_to\_response() is deprecated in favor render(). It has the '

'same signature except that it also requires a request.',

RemovedInDjango30Warning, stacklevel=2,

)

content = loader.render\_to\_string(template\_name, context, using=using)

return HttpResponse(content, content\_type, status)

原函数里面根本没有了这个参数

再看官方文档：

<http://djx.readthedocs.io/en/stable/ref/templates/api.html?highlight=RequestContext#django.template.RequestContext>

render\_to\_response()

render\_to\_response(template\_name, context=None, content\_type=None, status=None, using=None)[source]

This function preceded the introduction of render() and workssimilarly except that it doesn’t make the request available in theresponse. It’s not recommended and is likely to be deprecated in the future.

看到了吧，根本没有这个参数，而且最后一句说的很明白，这个垃圾函数人家不想伺候了。嗯推荐大家用render。不过也不要灰心，render和这个函数基本上是完全一样的，只是参数顺序不同而已。

In addition to these, RequestContext always enables'django.template.context\_processors.csrf'. This is a security relatedcontext processor required by the admin and other contrib apps, and, in caseof accidental misconfiguration, it is deliberately hardcoded in and cannot beturned off in the context\_processors option

这个什么什么.csrf 看上去为我们封装了一个csrf对象

看了这么长的文档，最后可实现的方法为：

return render\_to\_response('XX.html',context=csrf(request))

#或者

return render(request,'xx.html',context=csrf(request))