Django Auth装饰器验证用户身份与权限

我们知道在视图函数中，我们使实现对用户身份及权限的验证。Django为了给开发者提供方便， 还提供了便捷的装饰器来完成这类的校验。比如，@login_required 我们使用它来验证用户是否已经登录，只有登录的用户才可以访问视图函数，并获得响应，否则将重定向到登录界面。当然还有校验权限的装饰器 @permission_required，在本节我们将对这些装饰器使用方法进行逐一介绍。

1. 校验用户登录状态@login_required

为了分析这个装饰器，我们还是首先看一下 Django 的源码，它的定义如下文件中：

from django.contrib.auth.decorators import login_required

1) login_required函数参数说明

它的源码如下所示：

```
def login_required(function=None, redirect_field_name=REDIRECT_FIELD_NAME, login_url=None):

    actual_decorator = user_passes_test(
        lambda u: u.is_authenticated,
        login_url=login_url,
        redirect_field_name=redirect_field_name
    )
    if function:
        return actual_decorator(function)
    return actual_decorator
```

可以看出这个函数可以传递两个参数，我们这个函数做一下简单的分析：login_url 表示若为匿名用户访问时重定向的 URL，这里一般指定的都是登录页面的 URL 路径，默认的登录页需要在配置文件通过 LOGIN_URL 指定，然后通过使用以下方式进行调用 settings.py.LOGIN_URL；redirect_field_name 默认值为 next，作为 GET 的请求参数即参训字符串的形式，它的格式如下：

127.0.0.1:8000/login/?next=/index/add_book/1

这个参数用于登录后直接跳回到原先访问的视图。上述源码中可以看出该方法的实现核心是调用了 user_passes_test 方法。它需要传递三个参数，分析它的部分源码。如下所示：

```
def user_passes_test(test_func, login_url=None, redirect_field_name=REDIRECT_FIELD_NAME):

    def decorator(view_func):
        @wraps(view_func)
        def _wrapped_view(request, *args, **kwargs):
            #测试函数，通过后执行对应的视图函数
            if test_func(request.user):
                return view_func(request, *args, **kwargs)
            path = request.build_absolute_uri()  #返回请求完成的URL
             #获取登录页指定的URL
            resolved_login_url = resolve_url(login_url or settings.LOGIN_URL)
            # If the login url is the same scheme and net location then just
            # use the path as the "next" url.
            login_scheme, login_netloc = urlparse(resolved_login_url)[:2]
            current_scheme, current_netloc = urlparse(path)[:2]
            #如果登录页的 URL与path的协议，域都相同则执行下面代码
            if ((not login_scheme or login_scheme == current_scheme) and
                    (not login_netloc or login_netloc == current_netloc)):
                #获取视图的全路径，返回 HttpResponseRedirect
                path = request.get_full_path()
            from django.contrib.auth.views import redirect_to_login
            return redirect_to_login(
                path, resolved_login_url, redirect_field_name)
        return _wrapped_view
    return decorator
```

从 user_passes_test 的实现可以看出，它首先会判断 request.user.is_authenticated 是否会返回 True，如果成立，则会执行视图函数。否则，将重定向到登录页面。

2) login_required应用方式

它的使用方式也非常的简单，只需要在视图函数加上方稍加改动即可，如下所示：

```
from django.contrib.auth.decorators import login_required
@login_required
def search_title_views(request):
    pass
```

如果在用户未登录的情况下访问这个视图的话，那么它将会跳转到登录页，需要注意的是由于这里没有指定 login_url，因此在配置文件中的 LOGIN_URL 要设置正确。

2. 校验用户权限@permission_required

1) @permission_required的源码分析

理解了如何校验登录状态的装饰器 @login_required， 下面我们对 @permission_required 进行讲解，这样大家理解起来会更加简单。我们还是看一下它的实现源码，如下所示：

```
def permission_required(perm, login_url=None, raise_exception=False):
    """
    用于检查用户是否启用了特定权限的视图的装饰器，必要时可重定向到登录页。
    如果给定了raise_exception参数，则会引发PermissionDenied异常。
    """
    def check_perms(user):
        #如果指定权限是字符串，则将其放在元组中
        if isinstance(perm, str):
            perms = (perm,)
        else:
            perms = perm
        #校验用户是否具有指定的权限
        if user.has_perms(perms):
            return True
        # In case the 403 handler should be called raise the exception
        if raise_exception:
            raise PermissionDenied
        #最终没有通过校验返回 False
        return False
    #check_perms 即为 user_passes_test中的测试函数
    return user_passes_test(check_perms, login_url=login_url)
```

这个函数接受三个参数，它们的介绍如下：

- perm：需要校验的权限，可以是列表、元组、或者是字符串，如果是列表或者元组的话，那么用户同时拥有这些权限。
- login_url：没有指定权限的用户访问时会发生 302 重定向。
- raise_exception：默认为 False，如果设置为 True，则当没有权限的用户访问时将直接返回 403，由于权限的不足将禁止你的访问。

它的定义格式和 @login_required 是非常类似的。但是也有一点小小的区别，如下所示：

```
@permission_required("index.can_view_book")#也可校验多个权限，在方法内添加即可
def book_add_views(request):
    pass
```

我们可以这样理解，如果访问用户没有被授予 index.can_view_book 权限，就会跳转到登录页。这样不仅需要当前用户是已登庄状态，还需要用户拥有 can_view_book 的权限。

Django 的用户认证系统的到这里就介绍完了。任何一个服务型的 Web 站点都离不开用户与权限的概念，这一点我们也做了反复的强调。Django 对这两大功能提供了很好的支持。作为开发者在实际的业务开发中，应该尽量使用 Django 的用户认证系统，特别是 User 模型，它会给我们带来极大的便利。对于特定的需求，也可以考虑对其进行扩展，例如自定义权限、自定义认证后端等等。