Django 中的用户认证

Django 自带了一个用户授权认证系统。它可以处理用户帐户、组、权限和基于 cookie 的用户会话。本篇文档将解释它是如何工作的。

目录

1 概 览

认证系统包括:

* 用户(Users)
* 权限(Permissions): 二进制 (yes/no) 的标志，用来指明用户都能做哪些事情。
* 组(Groups): 向多个用户应用标签和权限的通用方法。
* 消息(Messages): 为给定的用户排队消息的一个简单的方法。

2 安 装

认证支持作为 Django 的一个应用被绑定在 django.contrib.auth 中。安装方法如下：

1. 把 'django.contrib.auth' 放到你的 INSTALLED_APPS 设置中。
2. 运行命令 manage.py syncdb 。

注意，默认情况下，通过使用 django-admin.py startproject 来创建的工程已经在 settings.py 中的 INSTALLED_APPS 包含了 'django.contrib.auth' 。如果你的 INSTALLED_APPS 中已经包含了 'django.contrib.auth' ，你也可以再次运行 manage.py syncdb 。你可以随意运行多少次都无所谓，每一次它都仅仅安装需要的部分。

syncdb 创建必要的数据表，同时也为已经安装的 apps 创建他们需要用到的权限对象。当你第一次运行这个命令的时候，它还会提示你创建一个超级用户的帐户。

当你做完以上这些步骤之后，认证系统就安装好了。

3 用户(Users)

用户(Users) 表现为一个标准的 Django 模型，他在 django/contrib/auth/models.py 中。

3.1 API 参考

3.1.1 字 段

User 对象包含如下字段:

* username -- 必须。小于等于 30 个字符。（字母、数字和下划线）
* first_name -- 可选。小于等于 30 个字符。
* last_name -- 可选。小于等于 30 个字符。
* email -- 可选。电子邮件地址。Optional. E-mail address.
* password -- 必须。密码的 hash 值。（Django 不保存原始密码。）原始密码可以是

任意长的并且可以包含任意字符。请看下面的“密码”节。

* is_staff -- 布尔型。标识用户能否访问 admin 界面。

* is_active -- 布尔型。标识用户能否登录到 admin 界面。如果不想删除用户请把它设为 False

* is_superuser -- 布尔型。标识用户可以得到所有的权限。

* last_login -- 用户上一次登录的日期时间。默认设置为当前的日期和时间。

* date_joined -- 用户帐户创建的日期。默认设置为帐户创建时的日期和时间。

3.1.2 方 法

User 对象有 2 个多对多（many-to-many）的字段: groups 和 user_permissions 。 User 对象可以像其他 Django 对象(Django model)那样访问他们关联的对象。

```
myuser.objects.groups = [group_list]
myuser.objects.groups.add(group, group,...)
myuser.objects.groups.remove(group, group,...)
myuser.objects.groups.clear()
myuser.objects.permissions = [permission_list]
myuser.objects.permissions.add(permission, permission, ...)
myuser.objects.permissions.remove(permission, permission, ...]
myuser.objects.permissions.clear()
```

除了这些自动生成的 API 方法外， User 对象还有如下的自定义的方法:

* is_anonymous() -- 总是返回 False 。这是区别 User and AnonymousUser 对象的一个方法。通常你应该使用 is_authenticated() 而不是这个方法。

* is_authenticated() -- 总是返回 True 。这是测试用户是否被认证了。

* get_full_name() -- 返回 first_name 加 last_name, 中间用空格隔开。

* set_password(raw_password) -- 用给定的字符串设定用户密码，并且处理密码的 hash 值。不保存 User 对象。

* check_password(raw_password) -- 如果给定的用户密码是正确的，那么返回 True 。 (通过比较密码的 hash 值来实现的。)

* get_group_permissions() -- 返回从用户所在的组里面获取的权限列表。

* get_all_permissions() -- 返回用户拥有的所有的权限。包括组权限和用户权限。

* has_perm(perm) -- 当参数的格式为 "package.codename" 的时候，并且用户拥有特殊权限的时候，返回 True 。

* has_perms(perm_list) -- 同上。用户对列表中每一个参数都有特殊权限的时候。每一个参数的格式都是 "package.codename" 。

* has_module_perms(package_name) -- 当用户对给定的包(Django app label)有权限的时候返回 True 。

* get_and_delete_messages() -- 返回用户对列中的 Message 对象列表并且从对列中删除 Message 对象。

* email_user(subject, message, from_email=None) -- 向用户发送 e-mail 。如果

from_email 是 None, Django 使用 DEFAULT_FROM_EMAIL 设置。

* get_profile() -- 返回站点特定的用户档案。如果当前站点不允许查询档案的话，Django 将抛出 django.contrib.auth.models.SiteProfileNotAvailable 异常。

3.1.3 管理功能

User 模型有一个自定义的管理器，它有如下的函数:

*

create_user(username, email, password) -- 创建，保存并返回一个 User 。username, email 和 password 被设置为给定的值，并且 User 设置了 is_active=True 。

请看下面的 创建用户 中的例子。

*

make_random_password(length=10, allowed_chars='abcdefghjkmnpqrstuvwxyzABCDEFGHJKLMNPQRSTUVWXYZ23456789') 返回一个给定长度的并且是包含给定字符的随机密码。(注意: allowed_chars 的默认值不包括 "I" 或者类似的字符，这是为了避免字符分辨不清而产生的抱怨。)

3.2 基本用法

3.2.1 创建用户

创建用户最基本的方法就是使用 Django 提供的 create_user 函数:

```
>>> from django.contrib.auth.models import User
>>> user = User.objects.create_user('john', 'lennon@thebeatles.com', 'johnpassword')

# 在这里，User 对象已经可以保存到数据库中了。
# 你还可以改变它的其它属性。
>>> user.is_staff = True
>>> user.save()
```

3.2.2 更改密码

使用 set_password() 更改密码:

```
>>> from django.contrib.auth.models import User
>>> u = User.objects.get(username__exact='john')
```

```
>>> u.set_password('new password')
>>> u.save()
```

除非你很清楚的知道你在做什么，否则不要直接设置 password 属性。这将在下一节中阐释。

3.3 密码

User 的 password 属性是一个如下格式的字符串:

hashtype$salt$hash

它们是 hashtype, salt 和 hash, 用一个美元符号来分隔的。

Hashtype 是 sha1 (默认) 或 md5 -- 单向加密的方法。Salt 是一个随机字符串用来为密码的原文创建 hash。

例如:

sha1$a1976$a36cc8cbf81742a8fb52e221aaeab48ed7f58ab4

User.set_password() 和 User.check_password() 函数负责处理设置和检测这些密码。

在 Django 先前的版本中, 例如 0.90, 仅使用 MD5 而不使用 salt。为了向后兼容，这些依然被支持。他们会在第一次使用 check_password() 来成功检测用户密码的时候自动将其转换为新的格式。

3.4 匿名用户

django.contrib.auth.models.AnonymousUser 是一个类，它实现了 django.contrib.auth.models.User 接口, 有如下的不同点:

* id 总是 None.
* is_anonymous() 返回 True 而不是 False.
* is_authenticated() 返回 False 而不是 True.
* has_perm() 总是返回 False.
* set_password(), check_password(), save(), delete(), set_groups() and set_permissions() 抛出 NotImplementedError 异常.

在实际应用中，你可能并不需要 AnonymousUser 对象。但是他们被应用到网络请求(Web request)中，这将在下面的章节中讲述。

3.5 创建超级用户

在你将 'django.contrib.auth' 添加到 INSTALLED_APPS 中之后，并且第一次运行 manage.py syncdb 命令时，它将提示你创建一个超级用户。但是如果你需要在之后的操作过程中用命令行创建超级用户的话，你可以使用 create_superuser.py 。命令如下:

python /path/to/django/contrib/auth/create_superuser.py

请确认你已经将 /path/to/ 加入到 PYTHON_PATH 中。

4 Web 请求中的认证

到目前为止，本文档已经介绍了操作认证相关的对象的低级的 API。在高一级上，Django 可以将认证框架挂接到它本身系统的请求对象(request objects)中。

首先，安装 SessionMiddleware 和 AuthenticationMiddleware 中间件。把他们加入到 MIDDLEWARE_CLASSES 设置中即可。更多信息请看 session documentation 。

当你安装好这些中间件之后，你就可以在视图(view)中访问 request.user 了。request.user 将返回当前登录的用户的一个 User 对象。如果当前没有用户登录，那么 request.user 将返回一个 AnonymousUser 对象的实例。你可以通过 is_authenticated() 来判断是否有用户登录，如下：

```
if request.user.is_authenticated():
    # 认证的用户
else:
    # 匿名用户
```

4.1 如何登录一个用户

Django 在 django.contrib.auth 提供了 2 个函数: authenticate() 和 login() 。

如果通过给定的用户名和密码做认证，请使用 authenticate() 函数。他接收 2 个参数，一个是 username 一个是 password 。如果认证成功，它返回一个 User 对象。如果密码无效，它返回一个 None 。例如:

```
from django.contrib.auth import authenticate
user = authenticate(username='john', password='secret')
if user is not None:
    print "用户名、密码正确！"
else:
    print "用户名、密码错误！"
```

在视图中登录一个用户的话，使用 login() 函数。它接收 HttpRequest 对象和一个 User 对象。 login() 通过 Django 的 session 框架把用户的 ID 保存到 session 中。所以，你要确认你

已经安装了 session 中间件。

下面是合用 authenticate() 和 login() 的例子:

```
from django.contrib.auth import authenticate, login

def my_view(request):
    username = request.POST['username']
    password = request.POST['password']
    user = authenticate(username=username, password=password)
    if user is not None:
        login(request, user)
        # 转到成功页面
    else:
        # 返回错误信息
```

4.2 如何登出用户

要登出使用 django.contrib.auth.login() 登录的用户的话，可以在视图中使用 django.contrib.auth.logout() 。它接收一个 HttpRequest 参数，没有返回值。例如:

```
from django.contrib.auth import logout

def logout_view(request):
    logout(request)
    # 转到成功页面
```

请注意：如果用户没有登录的话， logout() 也不会抛出任何异常的。

4.3 限制已登录用户的访问

4.3.1 原始的方法

最简单、最原始的限制页面访问的方法是在每个页面上加入 request.user.is_authenticated() 并且把它重定向到登录页面。

```
from django.http import HttpResponseRedirect

def my_view(request):
    if not request.user.is_authenticated():
        return HttpResponseRedirect('/login/?next=%s' % request.path)
    #...
```

或者显示一条出错信息

```
def my_view(request):
    if not request.user.is_authenticated():
        return render_to_response('myapp/login_error.html')
    #...
```

4.3.2 login_required 修饰符

你可以使用 login_required 修饰符来作为一个快捷方式

```
from django.contrib.auth.decorators import login_required

def my_view(request):
    # ...
my_view = login_required(my_view)
```

下面是一个等价的例子，使用了 Python 2.4 的 decorator 样式

```
from django.contrib.auth.decorators import login_required

@login_required
def my_view(request):
    # ...
```

login_required 作下面这些事情:

* 如果用户没有登录，那么重定向到 /accounts/login/ ，传入当前的绝对 URL 路径作为 query string next 的值。例如: /accounts/login/?next=/polls/3/ 。
* 如果用户已经登录了，那么就正常执行 view 的代码。

请注意，你需要映射正确的处理登录用的视图(view)到 /accounts/login/ 。把下面的行加入到你的 URLconf 中:

```
(r'^accounts/login/$', 'django.contrib.auth.views.login'),
```

django.contrib.auth.views.login 的作用是:

* 如果通过 ``GET`` 方式调用的话, 它显示一个登录表单并通过 POST 的方式登录。

* 如果通过 ``POST`` 方式调用的话，它试图把用户登录进去。 如果登录成功，视图(view)重定向到 ``/accounts/profile/`` (目前是硬性编码的，就是写死的。)。
  如果登录失败，则继续显示登录表单。

你需要自己提供一个登录表单的模版，默认叫 registration/login.html 。这个模版需要获得 3 个模版上下文的变量:

* form: 一个 FormWrapper 对象，用来显示登录表单。更多请看``FormWrapper`` 对象的 forms documentation 。
* next: 登录成功后重定向的 URL。也可能包含一个查询字符串。
* site_name: 当前 Site 的名字。根据 SITE_ID 设置的信息获取。参考 site framework docs 。

如果你不想使用 registration/login.html 这个模版，你可以为在 URLconf 中的视图(view)传入一个 template_name 作为扩展的参数。

(r'^accounts/login/$', 'django.contrib.auth.views.login', {'template_name': 'myapp/login.html'}),

下面是一个 registration/login.html 的例子，你可以以它为基础来开始你的工作。它扩展自 base.html 并且定义了一个 content 块:

```
{% extends "base.html" %}

{% block content %}

{% if form.has_errors %}
<p>用户名和密码不匹配。请重试。</p>
{% endif %}

<form method="post" action=".">
<table>
<tr><td><label for="id_username">用户名:</label></td><td>{{ form.username }}</td></tr>
<tr><td><label for="id_password">密码:</label></td><td>{{ form.password }}</td></tr>
</table>

<input type="submit" value="登录" />
<input type="hidden" name="next" value="{{ next }}" />
</form>

{% endblock %}
```

4.4 已登录用户通过通行测试(pass test)来限制访问

为了实现基于特定的权限或其他的测试的访问限制，你应该做同上一节一样的基本事情。To limit access based on certain permissions or some other test, you'd do essentially the same thing as described in the previous section.

简单的方法是在 view 中直接运行测试 request.user 的代码。例如，这个 view 检测并确认用户已经登陆并且拥有 polls.can_vote 的权限。

```
def my_view(request):
    if not (request.user.is_authenticated() and request.user.has_perm('polls.can_vote')):
        return HttpResponse("你不能投票。")
    # ...
```

你可以使用 user_passes_test 作为快捷方式

```
from django.contrib.auth.decorators import user_passes_test

def my_view(request):
    # ...
my_view = user_passes_test(lambda u: u.has_perm('polls.can_vote'))(my_view)
```

Python 2.4 的写法

```
from django.contrib.auth.decorators import user_passes_test

@user_passes_test(lambda u: u.has_perm('polls.can_vote'))
def my_view(request):
    # ...
```

user_passes_test() 需要一个必须的参数: 一个包含 User 的可调用的对象，如果用户被允许察看这个页面的话，返回 True 。注意，如果 User 不是匿名的话， user_passes_test 并不自动监测。

user_passes_test() 需要一个可选的 login_url 参数, 它可以让你指定登录表单的 URL(默认是 /accounts/login/)。

例子，Python 2.3 写法

```
from django.contrib.auth.decorators import user_passes_test

def my_view(request):
    # ...
my_view = user_passes_test(lambda u: u.has_perm('polls.can_vote'), login_url='/login/')(my_view)
```

例子，Python 2.4 写法

```
from django.contrib.auth.decorators import user_passes_test
```

```
@user_passes_test(lambda u: u.has_perm('polls.can_vote'), login_url='/login/')
def my_view(request):
    # ...
```

4.5 限制访问 generic views

为了限制访问 generic view, 可以为 view 写一个包装器，并且在 URLconf 中指定为它。

例如

```
from django.views.generic.date_based import object_detail

@login_required
def limited_object_detail(*args, **kwargs):
    return object_detail(*args, **kwargs)
```

5 权限(Permissions)

Django 自带了一个简单的权限系统。它为向用户和用户组付权限提供了一个途径。

它被用在了 Django 的 admin 站点中，当然你也可以把它用在自己的代码中。

Django 的 admin 站点是这样应用权限的:

* 通过"add"权限来控制用户是否可以访问添加表单并添加一个指定类型的对象。
* 通过"change"权限来控制用户是否可以访问指定类型对象的列表和修改表单。
* 通过"delete"权限来控制用户是否可以删除指定类型的对象。

权限被赋予每种类型的对象，而不是对象的特定的实例。你可以说“玛丽可以修改新的故事(stories)”，但是你不能说“玛丽可以修改她创建的新的故 事”或者 “玛丽只能修改特定状态的、特定发布时间的、特定 ID 的故事等等”。这些功能目前 Django 的开发人员还在讨论之中。

5.1 默认权限

3 个基本的权限 -- 添加(add)，创建(create)和删除(delete) -- 在创建包含有 class Admin 的 Django 模型的时候都自动被创建好了。在表面现象的后面，当你运行 manage.py syncdb 的时候，这些权限被添加到了 auth_permission 数据表中。

请注意，如果你的模型里没有 class Admin 的话，当你运行 manage.py syncdb 的时候这些权限不会被创建出来。如果你初始化数据库之后还想添加这些权限，可以在模型中加入 class

Admin 然后再运行一次 manage.py syncdb 。

5.2 自定义权限

为了给指定的模型自定义权限，可以使用 权限(permissions) 的 model Meta attribute 。

这个例子创建了 3 个自定义的权限

```
class USCitizen(models.Model):
    # ...
    class Meta:
        permissions = (
            ("can_drive", "Can drive"),
            ("can_vote", "Can vote in elections"),
            ("can_drink", "Can drink alcohol"),
        )
```

接下来的事情就是运行 syncdb 来创建这些权限。

5.3 API 参考

就像用户对象，权限也是实现自 Django 的模型 django/contrib/auth/models.py.

5.3.1 字段

Permission 有如下这些字段:

* name -- 必须。小于等于 50 个字符。例如: 'Can vote' 。
* content_type -- 必须。引用自 django_content_type 数据表, 它包含了已经安装的 Django 模型的类型。
* codename -- 必须。小于等于 100 个字符。例如: 'can_vote' 。

5.3.2 方法

Permission 有着与其它 Django model 一样的数据访问方法。

6 模板中的认证数据

如果你使用 RequestContext 的话，已经登录的用户的 user 和权限对象就保存在 template context 。

技术细节

在技术上，只有当你使用 RequestContext 的时候 并且 你的 TEMPLATE_CONTEXT_PROCESSORS 设置包含 "django.core.context_processors.auth" 的时候，这个变量才是有效的。更多参见 RequestContext docs.

6.1 用户(Users)

当前登录的用户，不管是否是匿名的，存储在模版变量 {{ user }} 中。

```
{% if user.is_authenticated %}
    <p>欢迎, {{ user.username }}。谢谢您的来访。</p>
{% else %}
    <p>欢迎，请登录。</p>
{% endif %}
```

6.2 权限(Permissions)

当前登录用户的权限存储在模版变量 {{ perms }} 中。他是 django.core.context_processors.PermWrapper 的实例。

在 {{ perms }} 对象中，单个属性的查找是使用 User.has_module_perms 的。下面这个例子中，如果用户对 foo 这个 app 有任何权限的话，它就返回 True。

```
{{ perms.foo }}
```

二级属性查找是使用 User.has_perm 。下面这个例子中，如果用户有 foo.can_vote 权限的话，它就返回 True。

```
{{ perms.foo.can_vote }}
```

因此，你可以在模板中用 {% if %} 语句来判断权限

```
{% if perms.foo %}
    <p>你有操作 foo 的权限。</p>
    {% if perms.foo.can_vote %}
        <p>你可以投票。</p>
    {% endif %}
    {% if perms.foo.can_drive %}
        <p>你可以开车。</p>
    {% endif %}
{% else %}
    <p>你没有操作 foo 的权限。</p>
{% endif %}
```

7 组(Groups)

组通常用来归类用户，这样你就可以为这些组里面的用户应用权限或者贴其他的标签。一个用户可以属于任意数量的组。

组中的用户自动获得赋予组的权限。例如，如果组 Site editors 有 can_edit_home_page 的权限，那么任何加入这个组的用户都自动拥有这个权限。

组也是归类用户并给他们贴标签或扩展功能的一个方便的途径。例如，你创建一个 'Special users' 的组，你可以写代码来让他们访问网站的会员专区或者发送给他们会员专用的电子邮件。

8 消息(Messages)

消息系统是为给定用户排队消息的轻量级的途径。

一个消息关联到一个 User 对象。没有过期标志和时间戳。

Django 的 admin 站点用它来发送成功操作后的消息。例如， "poll 成功被创建。" 就是一条消息。

API 很简单

* 创建消息用
  ``user_obj.message_set.create(message='message_text')`` 。
* 检索/删除消息用 ``user_obj.get_and_delete_messages()``,
  他返回用户消息队列(如果有的话)中的一个 ``Message`` 列表，并且从队列中删除这些消息。

下面的例子中，用户创建一个播放列表之后系统把消息保存下来

```
def create_playlist(request, songs):
    # 用给定的歌曲创建一个播放列表
    # ...
    request.user.message_set.create(message="播放列表添加成功。")
    return render_to_response("playlists/create.html",
        context_instance=RequestContext(request))
```

如果你使用 RequestContext ，当前用户和他的消息保存在 template context 的模版变量 {{ messages }} 里。 下面是在模板中显示消息的例子

```
{% if messages %}
<ul>
```

```
    {% for message in messages %}
    <li>{{ message.message }}</li>
    {% endfor %}
</ul>
{% endif %}
```

注意， RequestContext 调用 get_and_delete_messages 。所以，就算你不显示他们的话，他们也会被删除的。

最后，请注意这个消息框架仅适用于在用户数据库中的用户。如果要向匿名用户发送消息的话，请使用 session framework 。

9 其他认证资源

Django 自带的认证系统对于大多数情况来说已经足够用了，但是你也可能需要使用其他的认证源。其他的用户名和密码或者认证方法等。

例如，你的公司可能已经有了 LDAP，并且已经存储了每一位雇员的用户名和密码。如果你让网络管理员和用户在 LDAP 和基于 Django 的应用上使用不同的登录帐户的话，他们会非常不高兴的。

所以，为了解决这个问题，Django 的认证系统允许你插入其他的认证源。你可以重载 Django 默认的数据库结构，或者你可以把默认的系统和其他的系统串联起来。

9.1 指定认证后端

在暗中，Django 维护一个"authentication backends"的列表用来测试认证。当某人调用 django.contrib.auth.authenticate() -- 上面提到的"如何登录一个用户" -- Django 将尝试所有的认证后端。如果第一个认证方法失败了，Django 将会继续尝试第二个，直到所有的都被尝试过。

认证后端的列表在 AUTHENTICATION_BACKENDS 设置。内容应该是包含 Python 路径的元组。

默认情况下， AUTHENTICATION_BACKENDS 设置为

('django.contrib.auth.backends.ModelBackend',)

这是检测 Django 用户数据库的基本认证方案。

按照 AUTHENTICATION_BACKENDS 的排列顺序, 如果同样的用户名和密码在第一次就匹配了，那么 Django 将停止处理后面的东西。

9.2 编写一个认证后端

一个认证后端是一个类，实现了 2 个方法: get_user(id) 和 authenticate(**credentials) 。

get_user 方法接受一个 id -- 可以是用户名，数据库 ID 或者其他的什么 -- 并且返回一个 User 对象。

authenticate 方法接受字典型认证信息的参数。大多情况下是如下样子的

```
class MyBackend:
    def authenticate(username=None, password=None):
        # 检测用户名和密码，并返回一个 User。
```

他也可以处理一个代号(token)，像这样

```
class MyBackend:
    def authenticate(token=None):
        # 检测并返回 User。
```

当 authenticate 接受的参数被验证为有效的时候，应该返回一个 User 对象；如果无效的时候，应该返回 None 。

在本文当开头提到，Django 的 admin 系统紧密地与 User 对象绑定在一起。目前，最好的处理方法就是为你每一个现存的后端(例如，你的 LDAP 目录或者你的外部 SQL 数据库等等。)数据创建一个 Django 的 User 对象。你可以预先写一个脚本来做这些事情，或者在用户第一次登录的时候在你的 authenticate 方法中做这些事情。

下面是一个例子，使用在 settings.py 文件里定义的用户名和密码并且在用户第一次登录的时候创建一个 Django 的 User 对象。

```
from django.conf import settings
from django.contrib.auth.models import User, check_password

class SettingsBackend:
    """
    Authenticate against the settings ADMIN_LOGIN and ADMIN_PASSWORD.

    Use the login name, and a hash of the password. For example:

    ADMIN_LOGIN = 'admin'
    ADMIN_PASSWORD = 'sha1$4e987$afbcf42e21bd417fb71db8c66b321e9fc33051de'
    """
    def authenticate(self, username=None, password=None):
        login_valid = (settings.ADMIN_LOGIN == username)
        pwd_valid = check_password(password, settings.ADMIN_PASSWORD)
```

```
        if login_valid and pwd_valid:
            try:
                user = User.objects.get(username=username)
            except User.DoesNotExist:
                # 创建新用户。
                # 我们可以设置任何新的密码，因为它不会被检测。
                # 在这里我们使用"get from settings.py"。
                user = User(username=username, password='get from settings.py')
                user.is_staff = True
                user.is_superuser = True
                user.save()
            return user
        return None

    def get_user(self, user_id):
        try:
            return User.objects.get(pk=user_id)
        except User.DoesNotExist:
            return None
```