Django Auth应用实现用户身份认证

我们知道 Django Auth 应用一般用在用户的登录注册上，用于判断当前的用户是否合法，从而可以帮助开发者快速的构建用户系统，在《Django Auth用户与用户组详述》一节我们知道了用户与用户组的概念，那么 Auth 应用又是如何是实现用户的认证的呢？当给定相关的条件或属性时候，我们可以去获取用户对象这个过程就被称为用户认证，Django Auth 应用提供了一个用来认证的方法即 authenticate 方法用来实现用户的认证行为。下面就让我们一起来认识一下吧！

1. 实现用户的认证

authenticate 方法一般接受 username 与 password 作为参数，如果通过了认证，就返回认证的实例对象，否则就会返回 None，下面我们进行一下相关的实例演示：

```
1. In [1]: from django.contrib.auth import authenticate
2. In [2]: user=authenticate(username="bookstore",password="python_django")
3. In [3]: user
4. Out[3]: <User: bookstore>
5. In [4]: user=authenticate(username="bookstore",password="python")
6. In [5]: user is None
7. Out[5]: True
```

1) 用户认证过程解析

上述代码中，我们首先引入 authenticate ，可知该方法定义在 django/contrib/auth/__init__.py 文件中，然后我们对上一节中创建的 user 用户进行可认证，输入它的两个参数值，最后验证通过返回了 user 的实例对象，最后我们更改了密码的参数值将其设置为不正确，并使用 user is None 的方法查看其返回结果是否为 None，得到的布尔值为 Ture，所以用户认证失败。这就是用户认证的的过程。

那么 authenticate 方法是如何实现的呢？我们分析一下它的源码：

```
1. def authenticate(request=None, **credentials):
2.     #__get_backends获取当前系统中定义的认证后端, 并依次迭代
3.     for backend, backend_path in _get_backends(return_tuples=True):
4.         try:
5.             inspect.getcallargs(backend.authenticate, request, **credentials)
6.         except TypeError:
```

```
            #此后端不接受这些凭据作为参数。返回继续执行循环
            continue
        try:
             #通过当前的认证后端尝试获取 User，若获取不到就会抛出异常！
            user = backend.authenticate(request, **credentials)
        except PermissionDenied:
            #抛出异常Permission
            break
         #如果没有返回，继续执行下一个认证
        if user is None:
            continue
        #添加一个属性标志，代表后端认证成功
        user.backend = backend_path
        return user
    # 所提供的凭据对所有后端、触发信号无效
    user_login_failed.send(sender=__name__, credentials=_clean_credentials(credentials), request=request)
```

2) Django获取后端认证

上述代码中，我们可以看出，authenticate 方法使用了当前系统中定义的认证后端来获取用户对象，当前系统的默认认证后端是 ModelBackend。那么 Django 是如何获取认证后端的呢？如下所示：

```
def _get_backends(return_tuples=False):
    backends = []
    #AUTHENTICATION_BACKENDS 定义了当前系统可以用的身份认证列表
    for backend_path in settings.AUTHENTICATION_BACKENDS:
        #加载后端
        backend = load_backend(backend_path)
        backends.append((backend, backend_path) if return_tuples else backend)
    #如果未定义后端列表抛出异常
    if not backends:
        raise ImproperlyConfigured(
            'No authentication backends have been defined. Does '
            'AUTHENTICATION_BACKENDS contain anything?'
        )
    return backends
```

但是我们并没有对 AUTHENTICATION_BACKENDS 进行定义，所以这里会使用户 Django 框架默认的后端认认证，它位于 django/conf/global_settings.py 文件中，如下所示：

AUTHENTICATION_BACKENDS=['django.contib.auth.backends.ModelBackend'] #当前系统默认认证后端ModelBackend

2. Auth应用获取用户模型

get_user_model 用于获取当前系统定义的“用户模型”。其源代码如下所示：

```
def get_user_model():
    """
    返回一个处于激活状态的 User
    """
    try:
        return django_apps.get_model(settings.AUTH_USER_MODEL, require_ready=False)
    except ValueError:
        raise ImproperlyConfigured("AUTH_USER_MODEL must be of the form 'app_label.model_name'")
    except LookupError:
        raise ImproperlyConfigured(
            "AUTH_USER_MODEL refers to model '%s' that has not been installed" % settings.AUTH_USER_MODEL
        )
```

get_user_model 使用实例如下所示：

```
# 使用默认User model时
>>> from django.contrib.auth import get_user_model
>>> get_user_model()
<class 'django.contrib.auth.models.User'>
# 使用自定义User model时
>>> from django.contrib.auth import get_user_model
>>> get_user_model()
<class 'xxx.models.NewUser'>
# get_user_model()实际获取的是settings.AUTH_USER_MODEL指定的User model
```

Django 允许在 settings.py 文件中定义 AUTH_USER_MODEL 覆盖默认的 auth.User，以满足特定项目的需求。所以，ModelBacakend 的 authenticate 方法首先会通过 username 尝试获取 User 对象，再去校验密码是否正确以及 is_active 的状态，最后返回 User 对象或是返回 None。在 settings.py 配置 AUTH_USER_MODEL 格式如下所示：

```
#格式： "<django_app名>.<model名>"
AUTH_USER_MODEL = "appname.NewUser"

#在models.py编写示例
from django.conf import settings
from django.db import models
class Article(models.Model):
    author = models.ForeignKey(settings.AUTH_USER_MODEL)
    title = models.CharField(max_length=100)
```

在 django/contrib/auth/__init__.py 文件中定义了用户的各种行为，例如，登录、退出、用户的认证等等，通过这些可见 Django 的强大所在，Auth 应用模块可以帮助开发者减少很多的工作量，而且 Auth 应用的源码可以给开发者提供诸多的用户系统重写思路。