Django自定义认证后端实现多种登录方式验证

通过前面内容的学习，我们对用户认证系统有了基本的认识。我们见到几乎所有的 Web 网站或者手机 App 也好，它们的最终目的都是要留住用户，提升自己网站的用户注册量，所以说用户的概念也必须深入到每一个程序开发者的心中。而权限呢？它限制了用户可以拥有哪些功能，比如说某项只有付费会员才可以观看的教程，这就用到了用户的权限设置，可以一言一概之，用户的实现业务系统中的数据隔离，而权限则限定了用户可以使用的哪些功能。

那么，Django 提供的用户认证系统如何在项目中进行应用呢？在本节我们将给大家介绍如何使用用户认证系统，实现我们业务场景中常见的多种登录方式验证。这里就用到了自定义认证后端。

1. 实现自定义认证后端

在《Django Auth应用实现用户身份认证》一节，我们讲到用户的认证需要通过 authenticate 方法实现，而该方法就是使用 Django 默认认证后端 ModeBackend 进行用户验证的，但这种验证只是简单地比对数据库中存储的用户名和密码是否匹配一致，这样就会导致在很多情况下不能满足实际的业务的需求。这个时候我们就可以自定义一个认证后端，来实现某些需求。

1) 实现认证后端思路分析

那么如何实现自定义认证后端呢？如果你没有思路，不妨先分析一下 Django 默认的认证后端是如何实现的，从源码中你也许会找到一些启发。

首先如何想要实现用户的认证必须先要获得用户对象，然后调用 authenticate 方法实现认证，所以可想而认证后端是实现了 get_user 和 authenticate 这两个方法的 Python 类。其中 authenticate 将用户身份凭据作为关键字参数，下面我就实现一个简单的认证后端。

2) 实现认证后端的过程

在 user 应用下新建 backends.py 文件，将其作为单独模块出来，定义如下代码：

```
1. from django.contrib.auth.models import User
```

```
class EmailBackend(object):
    def authenticate(self, request, **credentials):
        #获取邮箱的认证信息即邮箱账号实例
        email = credentials.get('email', credentials.get('username'))
        try:
            user = User.objects.get(email=email)
        except Exception as error:
            print(error)
        else:
            #检查用户密码
            if user.check_password(credentials["password"]):
                return user

    def get_user(self, user_id):
        try:
            return User.objects.get(pk=user_id)
        except Exception as e:
            print(e)
            return None
```

Django 模型类都有一个主键字段 (ID)，它用来维护模型对象的唯一性。Django 提供了一个 pk 字段来代表主键 ID。

我们在 authenticate 方法中，首先判断在用户名与密码不为空的情况下，尝试根据 username 获取 User 对象，然后再去比较 password 是否等同于 python_django，若相同则通过验证，所以这个密码可以实现任意用户的验证。最后如果要让自定义认证后端成功认证，还需要在配置文件 settings 中进行如下配置：

```
#自定义认证后端
AUTHENTICATION_BACKENDS=[
    'django.contrib.auth.backends.ModelBackend',
    'user.backends.EmailBackend',
]
```

这里需要大家注意一下：我们需要在 AUTHENTICATION_BACKENDS 变量中列出所有的认证后端，包含 Django 默认的以及自定义的，否则不能通过 username 和password 匹配的方式实现用户认证，比如上述代码中，我们自定义了一个通过邮箱和密码实现用户认证的后端，那么当我们在不使用用户名的情况下，还可以使用邮箱与正确的密码进行认证。

3) 验证认证**后端是否生效**

打开 Django Shell 环境 进行测试，如下所示，这里我们使用的用户 bookstore 已经在《Django Auth用户与用户组详述》一节进行了创建，不再赘述：

```
In [1]: from django.contrib.auth import authenticate
```

```
In [2]: user=authenticate(username="bookstore",password="python_django")
In [3]: user.backend
Out[3]: 'django.contrib.auth.backends.ModelBackend' #返回Django默认后端
In [4]: user.backend
Out[4]: 'django.contrib.auth.backends.ModelBackend'
In [5]: user=authenticate(username="bookstore",password="python")
In [6]: user is None
Out[6]: True
In [7]: user=authenticate(email="123@163.com",password="python_django")
In [8]: user.bachend
Out[8]: 'user.backends.EmailBackend' #返回自定义后端
```

通过上述测试，我们可以看到已经实现可自定义认证后端的应用。用户不仅可以使用用户名认证，还可以通过邮箱实现认证，从而在密码正确的情况下，可以使用两种方式实现用户的登录。实现自定义后端有很多的应用场景，比如你还可以换成手机号来进行用户认证等等。希望你们通过本节知识的学习会有所收获，下节继续！