一、权限Permissions

DRF的权限类位于permissions模块中。

连同认证和限流，权限决定是否应该接收请求或拒绝访问。

权限检查始终在视图的最开始处执行，在继续执行任何其他代码之前。 权限检查通常会使用 `request.user` 和 `request.auth` 属性中的身份认证信息来决定是否允许传入请求。

在不同类别的用户访问不同类别API的过程中，使用权限来控制访问的许可。

最简单粗暴的权限设置：允许任何经过身份验证的用户访问API，并拒绝任何未经身份验证的用户。这对应于DRF框架中的 `IsAuthenticated` 类。

稍微严格点的权限控制是对经过身份验证的用户的允许完全访问（可读可写），但对未经身份验证的用户只允许读取类型访问。这对应于DRF框架中的 `IsAuthenticatedOrReadOnly` 类。

如何确定权限

DRF框架中的权限始终被定义为一个权限类的列表，表示拥有列表中所有类型的权限。

在运行视图的主体代码之前，系统会检查列表中的每个权限。 如果任何权限检查失败，将会抛出一个 `exceptions.PermissionDenied` 或 `exceptions.NotAuthenticated` 异常，并且视图的主体代码将不会运行。

当权限检查失败时，将返回 `"403 Forbidden"` 或 `"401 Unauthorized"` 响应，具体根据以下规则：

- 请求已成功通过身份验证，但不具备访问权限。 — 返回403 Forbidden响应。
- 请求未通过身份认证，并且最高优先级的认证类未使用 `WWW-Authenticate` 标头。 — 返回403 Forbidden响应。
- 请求未通过身份认证，但是最高优先级的认证类使用了 `WWW-Authenticate` 标头。 — 返回HTTP 401未经授权的响应，并附带适当的WWW-Authenticate报头。

设置对象级别的权限

DRF框架还支持对象级别的权限设置。对象级权限用于确定是否允许用户操作特定的对象（通常是模型实例）。

当调用 `.get_object()` 方法时，由DRF框架的通用视图运行对象级权限检测。 与视图级别权限一样，如果不允许用户操作给定对象，则会抛出 `exceptions.PermissionDenied` 异常。

如果你正在编写自己的视图并希望强制执行对象级权限检测，或者你想在通用视图中重写 `get_object` 方法，那么你需要在检索对象的时候显式地调用视图上的 `.check_object_permissions(request, obj)` 方法。这将抛出 `PermissionDenied` 或 `NotAuthenticated` 异常；或者如果视图具有适当的权限，则返回。

例如：

```
def get_object(self):
    obj = get_object_or_404(self.get_queryset(), pk=self.kwargs["pk"])
    self.check_object_permissions(self.request, obj)
    return obj
```

注意：出于性能考虑，通用视图在返回对象列表时不会自动将对象级权限应用于查询集中的每个实例上面。通常，当你使用对象级权限时，你还需要适当地过滤查询集，以确保用户只能看到他们被允许查看的实例。

下面是两个方法的源码：

```
    def check_permissions(self, request):
        """
        Check if the request should be permitted.
        Raises an appropriate exception if the request is not permitted.
        """
        for permission in self.get_permissions():
            if not permission.has_permission(request, self):
                self.permission_denied(
                    request, message=getattr(permission, 'message', None)
                )

    def check_object_permissions(self, request, obj):
        """
        Check if the request should be permitted for a given object.
        Raises an appropriate exception if the request is not permitted.
        """
        for permission in self.get_permissions():
            if not permission.has_object_permission(request, self, obj):
                self.permission_denied(
                    request, message=getattr(permission, 'message', None)
                )
```

设置权限策略

默认权限策略可以使用 `DEFAULT_PERMISSION_CLASSES` 配置项进行全局设置。比如：

```
REST_FRAMEWORK = {
    'DEFAULT_PERMISSION_CLASSES': (
        'rest_framework.permissions.IsAuthenticated',
    )
}
```

如果未指定，则此设置默认为允许无限制的访问，也就是下面的配置：

```
'DEFAULT_PERMISSION_CLASSES': (
   'rest_framework.permissions.AllowAny',
)
```

你还可以为使用基于 `APIView` 的类视图在每个视图或每个视图集的上设置权限策略。

```
from rest_framework.permissions import IsAuthenticated
from rest_framework.response import Response
from rest_framework.views import APIView

class ExampleView(APIView):
    permission_classes = (IsAuthenticated,)  # 看这里！注意参数形式

    def get(self, request, format=None):
        content = {
            'status': 'request was permitted'
        }
        return Response(content)
```

或者为使用 `@api_view` 装饰器的基于函数的视图设置权限。

```python
from rest_framework.decorators import api_view, permission_classes
from rest_framework.permissions import IsAuthenticated
from rest_framework.response import Response

@api_view(['GET'])
@permission_classes((IsAuthenticated, ))  # 看这里！注意参数形式
def example_view(request, format=None):
    content = {
        'status': 'request was permitted'
    }
    return Response(content)
```

注意： 当你通过类属性或装饰器设置新的权限类时，会忽略settings中设置的默认权限列表。也就是粒度越细的权限级别越高，这也符合我们通常的逻辑思维。

DRF还提供一种权限的关系运算操作，可以实现组合类权限，也就是与或非运算。除了花哨，没啥特别的，参考下面的例子：

```python
from rest_framework.permissions import BasePermission, IsAuthenticated, SAFE_METHODS
from rest_framework.response import Response
from rest_framework.views import APIView

class ReadOnly(BasePermission):
    def has_permission(self, request, view):
        return request.method in SAFE_METHODS

class ExampleView(APIView):
    permission_classes = (IsAuthenticated|ReadOnly,) # 看这里，注意竖线表示'或者'

    def get(self, request, format=None):
        content = {
            'status': 'request was permitted'
        }
        return Response(content)
```

二、API参考

AllowAny

`AllowAny` 权限类允许不受限制的访问，并且**不管该请求是否通过身份验证或未经身份验证。**

一般来说，我们不用专门指定此权限，因为你可以通过使用空列表或元组进行权限设置来获得相同的结果，但显式地指定可以使我们地意图更明确。

IsAuthenticated

`IsAuthenticated` 权限类将拒绝任何未经身份验证的用户的访问，通过身份验证的用户则拥有所有权限。 如果你希望你的API仅供注册用户访问，则此权限比较合适。

IsAdminUser

除非 `user.is_staff` 属性的值为 `True` ，否则 `IsAdminUser` 权限类将拒绝来者访问API。 `user.is_staff` 属性是Django的auth框架中，管理员的类别，可以在Admin后台中查看和指定，也可以通过createsuperuser命令创建。如果你希望你的API只能被部分受信任的管理员访问，则此权限比较适合。

IsAuthenticatedOrReadOnly

`IsAuthenticatedOrReadOnly` 权限类允许经过身份验证的用户执行任何请求。未经授权的用户请求，只有当请求方法是“安全”的（即 `GET` ，`HEAD` 或 `OPTIONS` 之一时），才允许访问。

如果你希望你的API允许匿名用户读取，并且只允许对通过身份验证的用户可读写，则此权限比较适合。这也是大多数公开服务的选择。

DjangoModelPermissions

Django模型级别的权限。

此权限类与Django标准的 `django.contrib.auth` model权限相关。此权限只能应用于具有 `.queryset` 属性集的视图。只有在用户通过身份验证并分配了相关模型权限的情况下，才会被授予此权限。

- `POST` 请求要求用户对模型具有 `add` 权限。
- `PUT` 和 `PATCH` 请求要求用户对模型具有 `change` 权限。

- `DELETE` 请求想要求用户对模型具有 `delete` 权限。

也可以通过自定义模型权限，重写以上的默认行为。例如，你可能希望为 `GET` 请求包含一个 `view` 模型的权限。

要使用自定义模型权限，请覆盖 `DjangoModelPermissions` 并设置 `.perms_map` 属性。

如果你在重写了 `get_queryset()` 方法的视图中使用此权限，有可能这个视图上却没有 `queryset` 属性。在这种情况下，建议使用保护性的查询集来标记视图，以便确定所需的权限。比如：

```
1   queryset = User.objects.none()  # Required for DjangoModelPermissions
```

DjangoModelPermissionsOrAnonReadOnly

类似 `DjangoModelPermissions` ，但允许未经身份验证的用户对API的只读权限。

DjangoObjectPermissions

Django的模型的对象级别的权限！粒度最细！

此权限类与Django标准的对象权限框架相关联，该框架允许为模型上的每个对象设置权限。为了使用此权限类，你还需要添加支持对象级权限的权限后端，例如 **django-guardian** 。

与 `DjangoModelPermissions` 一样，此权限只能应用于具有 `.queryset` 属性或 `.get_queryset()` 方法的视图。只有在用户通过身份验证并且具有 *相关的每个对象权限* 和相关的 *模型权限* 后，才会被授予此权限。

- `POST` 请求要求用户对模型实例具有 `add` 权限。
- `PUT` 和 `PATCH` 请求要求用户对模型示例具有 `change` 权限。
- `DELETE` 请求要求用户对模型示例具有 `delete` 权限。

与 `DjangoModelPermissions` 一样，你可以通过重写 `DjangoObjectPermissions` 并设置 `.perms_map` 属性来使用自定义模型权限。

三、自定义权限

很显然DRF自带的权限类别太简单了，可能我们需要大量的自定义权限类别。

要自定义权限，需要继承 `BasePermission` 类，并实现以下方法中的一个或两个：

- `.has_permission(self, request, view)`
- `.has_object_permission(self, request, view, obj)`

如果请求被授予访问权限，方法应该返回 `True` ，否则返回 `False` 。

如果你需要测试请求是读取操作还是写入操作，则应该根据permissions模块中常量 `SAFE_METHODS` 的值检查请求方法， `SAFE_METHODS` 是包含 `'GET'` ， `'OPTIONS'` 和 `'HEAD'` 的元组。例如：

```
if request.method in permissions.SAFE_METHODS:
    # 检查只读请求的权限
else:
    # 检查写入请求的权限
```

注意: 仅当视图级的 `has_permission` 方法检查通过时，才会调用实例级的 `has_object_permission` 方法。另外，**为了运行实例级别检查，视图代码应显式地调用 `.check_object_permissions(request, obj)` 方法。如果你使用的是通用视图，那么默认会自动为你处理。**

如果权限测试失败，自定义地权限类将引发 `PermissionDenied` 异常。要更改与异常关联的错误信息，请直接在自定义地权限类中添加一个 `message` 属性。否则将使用 `PermissionDenied` 的 `default_detail` 属性。

```
from rest_framework import permissions

class CustomerAccessPermission(permissions.BasePermission):
    message = 'Adding customers not allowed.' # 自定义异常提示信息

    def has_permission(self, request, view):
        ...
```

下面是一个自定义权限类的例子，根据黑名单检查传入请求的IP地址，如果该IP在黑名单中，则拒绝请求。

```python
from rest_framework import permissions

class BlacklistPermission(permissions.BasePermission): # 继承
    """
    全局的黑名单ip检查
    """

    def has_permission(self, request, view):
        ip_addr = request.META['REMOTE_ADDR'] # 获取请求方的ip
        blacklisted = Blacklist.objects.filter(ip_addr=ip_addr).exists() # ORM过滤查询
        return not blacklisted # 返回一个布尔值
```

除了针对所有传入请求运行的全局权限设置外，还可以创建对象级权限，这些权限仅影响特定对象实例的操作行为。例如：

```python
class IsOwnerOrReadOnly(permissions.BasePermission):
    """
    Object-level permission to only allow owners of an object to edit it.
    Assumes the model instance has an `owner` attribute.
    """

    def has_object_permission(self, request, view, obj):
        # Read permissions are allowed to any request,
        # so we'll always allow GET, HEAD or OPTIONS requests.
        if request.method in permissions.SAFE_METHODS:
            return True

        # Instance must have an attribute named `owner`.
        return obj.owner == request.user
```

请注意，通用视图将检查适当的对象级权限，但如果你正在编写自己的自定义视图，则需要确保检查自己的对象级权限检查，也就是说这活得你自己干。你可以通过在获得对象实例后从视图中调用 `self.check_object_permission(request, obj)` 来执行此操作。如果任何对象级权限检查失败，此调用将引发对应的 `APIException` ，否则将简单地返回。

四、第三方模块

下面是和DRF权限功能相关的一些第三方模块

- Composed Permissions

- REST Condition
- DRY Rest Permissions
- Django Rest Framework Roles
- Django REST Framework API Key
- Django Rest Framework Role Filters
- django-guardian