除了前面已经让人头疼的视图体系外，REST框架还给我们提供了一个更加抽象的ViewSets视图集。ViewSets提供一套自动的urlconf路由，让开发人员可以将更多的精力用于对API的状态和交互，而不必操心路由路径的编写工作。

`ViewSet`类与`View`类几乎相同，不同之处在于它们提供诸如`read`或`update`之类的操作，而不是`get`或`put`等方法处理程序。这是DRF在设计的时候，为了防止和原生的Django语法之间的冲突。

`ViewSet`通常使用`Router`类来处理URL conf。

一、使用ViewSets重构视图

我们准备把目前的视图重构成视图集。

首先让我们将`UserList`和`UserDetail`视图重构为一个`UserViewSet`。我们可以删除这两个视图，并用一个类替换它们：

```
from rest_framework import viewsets

class UserViewSet(viewsets.ReadOnlyModelViewSet):
    """
    这个视图集会自动提供`list`和 `detail`操作
    """
    queryset = User.objects.all()
    serializer_class = UserSerializer
```

可以看到，一个视图集同时提供了原来两个类视图的功能。

这里，我们使用`ReadOnlyModelViewSet`类来自动提供默认的“只读”操作。我们仍然像使用常规视图那样设置`queryset`和`serializer_class`属性，但我们不再需要向两个不同的类提供相同的信息。

接下来，我们将替换`SnippetList`，`SnippetDetail`和`SnippetHighlight`视图类。我们可以删除三个视图，并再次用一个类替换它们。

```
from rest_framework.decorators import action
from rest_framework.response import Response

class SnippetViewSet(viewsets.ModelViewSet):
    """
    This viewset automatically provides `list`, `create`, `retrieve`,
```

```
    `update` and `destroy` actions.

    Additionally we also provide an extra `highlight` action.
    """
    queryset = Snippet.objects.all()
    serializer_class = SnippetSerializer
    permission_classes = (permissions.IsAuthenticatedOrReadOnly,
                          IsOwnerOrReadOnly,)

    @action(detail=True, renderer_classes=[renderers.StaticHTMLRenderer])
    def highlight(self, request, *args, **kwargs):
        snippet = self.get_object()
        return Response(snippet.highlighted)

    def perform_create(self, serializer):
        serializer.save(owner=self.request.user)
```

这次我们使用了 `ModelViewSet` 类来获取完整的默认读写操作。

请注意，我们还使用 `@detail_route` 装饰器创建一个名为 `highlight` 的自定义操作。这个装饰器可用于添加不符合标准 `create` / `update` / `delete` 样式的任何自定义路径。

默认情况下，使用 `@detail_route` 装饰器的自定义操作将响应 `GET` 请求。如果我们想要一个响应 `POST` 请求的动作，我们可以使用 `methods` 参数。

默认情况下，自定义操作的URL取决于方法名称本身。如果要更改URL的构造方式，可以为装饰器设置url_path关键字参数。

> 以上的内容对于新手来说，太难理解了。不看后面的API参考，你根本无法明白它的意思。放在快速入门教程里，真的好吗？

至此，完整的，包括被注释掉的，用于对比的，先前的视图，整个veiws.py的代码如下（调整了import语句的位置）：

```
from snippets.models import Snippet
from snippets.serializers import SnippetSerializer
from rest_framework import generics
from django.contrib.auth.models import User
from snippets.serializers import UserSerializer
from rest_framework import permissions
from snippets.permissions import IsOwnerOrReadOnly


from rest_framework.decorators import api_view
```

```python
from rest_framework.response import Response
from rest_framework.reverse import reverse
from rest_framework import renderers

from rest_framework import viewsets
from rest_framework.decorators import action
from rest_framework.response import Response


@api_view(['GET'])
def api_root(request, format=None):
    return Response({
        'users': reverse('user-list', request=request, format=format),
        'snippets': reverse('snippet-list', request=request, format=format)
    })


# class SnippetHighlight(generics.GenericAPIView):
#     queryset = Snippet.objects.all()
#     renderer_classes = (renderers.StaticHTMLRenderer,)
#
#     def get(self, request, *args, **kwargs):
#         snippet = self.get_object()
#         return Response(snippet.highlighted)
#
#
# class SnippetList(generics.ListCreateAPIView):
#     queryset = Snippet.objects.all()
#     serializer_class = SnippetSerializer
#     permission_classes = (permissions.IsAuthenticatedOrReadOnly,)
#
#     def perform_create(self, serializer):
#         serializer.save(owner=self.request.user)
#
#
# class SnippetDetail(generics.RetrieveUpdateDestroyAPIView):
#     queryset = Snippet.objects.all()
#     serializer_class = SnippetSerializer
#     permission_classes = (permissions.IsAuthenticatedOrReadOnly,
#                           IsOwnerOrReadOnly,)


class SnippetViewSet(viewsets.ModelViewSet):
    """
```

```
    This viewset automatically provides `list`, `create`, `retrieve`,
    `update` and `destroy` actions.

    Additionally we also provide an extra `highlight` action.
    """
    queryset = Snippet.objects.all()
    serializer_class = SnippetSerializer
    permission_classes = (permissions.IsAuthenticatedOrReadOnly,
                          IsOwnerOrReadOnly,)

    @action(detail=True, renderer_classes=[renderers.StaticHTMLRenderer])
    def highlight(self, request, *args, **kwargs):
        snippet = self.get_object()
        return Response(snippet.highlighted)

    def perform_create(self, serializer):
        serializer.save(owner=self.request.user)


# class UserList(generics.ListAPIView):
#     queryset = User.objects.all()
#     serializer_class = UserSerializer
#
#
# class UserDetail(generics.RetrieveAPIView):
#     queryset = User.objects.all()
#     serializer_class = UserSerializer


class UserViewSet(viewsets.ReadOnlyModelViewSet):
    """
    This viewset automatically provides `list` and `detail` actions.
    """
    queryset = User.objects.all()
    serializer_class = UserSerializer

```

二、显式地将ViewSets绑定到URL路由上

既然视图类发生了改变，那么我们的路由也必须针对性的调整。 在`urls.py`文件中，我们将`ViewSet`类绑定到一组具体视图中。

```
from snippets.views import SnippetViewSet, UserViewSet, api_root
from rest_framework import renderers

snippet_list = SnippetViewSet.as_view({
    'get': 'list',
    'post': 'create'
})
snippet_detail = SnippetViewSet.as_view({
    'get': 'retrieve',
    'put': 'update',
    'patch': 'partial_update',
    'delete': 'destroy'
})
snippet_highlight = SnippetViewSet.as_view({
    'get': 'highlight'
}, renderer_classes=[renderers.StaticHTMLRenderer])
user_list = UserViewSet.as_view({
    'get': 'list'
})
user_detail = UserViewSet.as_view({
    'get': 'retrieve'
})
```

上面的不理解也没关系，因为我们马上要删掉它。如果看不懂，等你以后再来看也行。

现在我们可以像通常一样在URL conf中注册视图。

```
urlpatterns = format_suffix_patterns([
    path('', api_root),
    path('snippets/', snippet_list, name='snippet-list'),
    path('snippets/<int:pk>/', snippet_detail, name='snippet-detail'),
    path('snippets/<int:pk>/highlight/', snippet_highlight, name='snippet-highlight'),
    path('users/', user_list, name='user-list'),
    path('users/<int:pk>/', user_detail, name='user-detail')
])
```

三、使用DRF提供的路由器Router

DRF为 `ViewSet` 视图集，设计了专门的路由器类Router。 `Router` 类专门为ViewSet提供全自动的urlconf。我们需要做的就是使用路由器注册相应的视图集，然后让它执行其余操作。

上面的内容全删了，重写 `urls.py` 文件。

```
from django.urls import path, include
from rest_framework.routers import DefaultRouter
from snippets import views

# Create a router and register our viewsets with it.
router = DefaultRouter()
router.register(r'snippets', views.SnippetViewSet)
router.register(r'users', views.UserViewSet)

# The API URLs are now determined automatically by the router.
urlpatterns = [
    path('', include(router.urls)),
]
```

使用路由器注册viewsets类似于提供urlpattern。我们包含两个参数 - 视图的URL前缀和视图本身。

`DefaultRouter` 类也会自动为我们创建API根视图，因此可以从views.py中删除 `api_root` 方法。

完整的snippets/urls.py内容如下（带先前版本的对比，被注释了）：

```
# from django.urls import path
# from snippets import views
#
# urlpatterns = [
#     path('snippets/', views.snippet_list),
#     path('snippets/<int:pk>/', views.snippet_detail),
# ]


# from django.urls import path
# from rest_framework.urlpatterns import format_suffix_patterns
# from snippets import views

# urlpatterns = [
#     path('snippets/', views.SnippetList.as_view()),
#     path('snippets/<int:pk>/', views.SnippetDetail.as_view()),
#     path('users/', views.UserList.as_view()),
```

```
#     path('users/<int:pk>/', views.UserDetail.as_view()),
#     path('', views.api_root),
#     path('snippets/<int:pk>/highlight/',
views.SnippetHighlight.as_view()),
# ]


# 略微调整了一下顺序
# urlpatterns = [
#     path('', views.api_root),
#     path('snippets/', views.SnippetList.as_view(), name='snippet-list'),
#     path('snippets/<int:pk>/', views.SnippetDetail.as_view(),
name='snippet-detail'),
#     path('snippets/<int:pk>/highlight/',
views.SnippetHighlight.as_view(), name='snippet-highlight'),
#     path('users/', views.UserList.as_view(), name='user-list'),
#     path('users/<int:pk>/', views.UserDetail.as_view(), name='user-
detail')
# ]
#
# urlpatterns = format_suffix_patterns(urlpatterns)

from django.urls import path, include
from rest_framework.routers import DefaultRouter
from snippets import views

# Create a router and register our viewsets with it.
router = DefaultRouter()
router.register(r'snippets', views.SnippetViewSet)
router.register(r'users', views.UserViewSet)

# The API URLs are now determined automatically by the router.
urlpatterns = [
    path('', include(router.urls)),
]
```

四、三种视图编写方法之间的权衡

使用视图集可以最大限度地减少代码量，让你能够专注于API提供的交互和表示，而不是URLconf的细节。但这并不是说它就是最佳最好最优的解决方案，事实上，抽象得越多，可定制性就越低，适用的场景就越少。

三种视图的构建方法：

- 基于函数的视图 @api_view
- 基于类的视图 APIView GenericView ListModelView
- 基于视图集的视图 ViewSet

没有绝对的好坏，使用哪种取决于你的需求。