上午课程核心要点

1. 视图集

1）视图集的基本使用

```
class BookInfoViewSet(ViewSet):
    # GET /books/ -> list
    def list(self, request):
        books = BookInfo.objects.all()
        serializer = BookInfoSerializer(books, many=True)
        return Response(serializer.data)

    # GET /books/(?P<pk>\d+)/ -> retrieve
    def retrieve(self, request, pk):
        try:
            book = BookInfo.objects.get(pk=pk)
        except BookInfo.DoesNotExist:
            raise Http404

        serializer = BookInfoSerializer(book)
        return Response(serializer.data)


urlpatterns = [
    re_path(r'^books/$', views.BookInfoViewSet.as_view({
        'get': 'list'
    })),
    re_path(r'^books/(?P<pk>\d+)/$', views.BookInfoViewSet.as_view({
        'get': 'retrieve'
    }))
]
```

2）视图集的 4 个父类

```
class ViewSet(ViewSetMixin, views.APIView):
    """
    The base ViewSet class does not provide any actions by default.
    """
    pass


class GenericViewSet(ViewSetMixin, generics.GenericAPIView):
    """
    The GenericViewSet class does not provide any actions by default,
    but does include the base set of generic view behavior, such as
    the `get_object` and `get_queryset` methods.
    """
    pass


class ReadOnlyModelViewSet(mixins.RetrieveModelMixin,
                           mixins.ListModelMixin,
                           GenericViewSet):
    """
    A viewset that provides default `list()` and `retrieve()` actions.
    """
    pass
```

```python
class ModelViewSet(mixins.CreateModelMixin,
                   mixins.RetrieveModelMixin,
                   mixins.UpdateModelMixin,
                   mixins.DestroyModelMixin,
                   mixins.ListModelMixin,
                   GenericViewSet):
    """
    A viewset that provides default `create()`, `retrieve()`, `update()`,
    `partial_update()`, `destroy()` and `list()` actions.
    """
    # list
    # create
    # retrieve
    # update
    # destroy
    pass
```

3）视图集中添加额外的 API

在视图集中，除了上述常见的 5 个 API(即：action处理方法)之外，根据需求，还可能需要添加额外的 API。

问题：视图集和类视图的区别?

1. 直接继承的父类不同

   ```
   类视图：View、APIView、GenericAPIView、子类视图
   视图集：ViewSet、GenericViewSet、ReadOnlyModelViewSet、ModelViewSet
   ```

2. 处理方法的名称不同

   ```
   类视图：get、post、put、delete
   视图集：list、retrieve、create、update、destroy
   ```

3. URL 地址配置不同

   ```
   类视图：类视图.as_view()
   视图集：视图集.as_view({'请求方式': '处理方法', ...})
   ```

2. 路由 Router

作用：动态生成视图集中处理方法的 URL 配置项，不需要自己再手动进行配置

1）基本使用

```python
# 1. 创建Router对象
from rest_framework.routers import SimpleRouter, DefaultRouter
router = SimpleRouter()
# router = DefaultRouter()

# 2. 注册视图集
# router.register(prefix, viewset, basename)
router.register('books', views.BookInfoViewSet, basename='books')

# 3. 添加路由数据
urlpatterns += router.urls
```

```python
for url in router.urls:
    print(url)
```

```
drf_demo
/Users/smart/.virtualenvs/meiduo_site/bin/python /Users/smart/Desktop/code/drf_demo/manage.py runse
Watching for file changes with StatReloader
Performing system checks...

<URLPattern '^books/$' [name='books-list']>
<URLPattern '^books/(?P<pk>[^/.]+)/$' [name='books-detail']>
System check identified no issues (0 silenced).
November 19, 2020 - 02:00:54
Django version 2.2.5, using settings 'drf_demo.settings'
Starting development server at http://127.0.0.1:8000/
Quit the server with CONTROL-C.
```

2）lookup_value_regex 属性

```
/Users/smart/.virtualenvs/meiduo_site/bin/python /Users/smart/Desktop/code/drf_demo/manage.py runserv
Watching for file changes with StatReloader
Performing system checks...

<URLPattern '^books/$' [name='books-list']>
<URLPattern '^books/(?P<pk>[^/.]+)/$' [name='books-detail']>
System check identified no issues (0 silenced).
November 19, 2020 - 02:00:54
Django version 2.2.5, using settings 'drf_demo.settings'
Starting development server at http://127.0.0.1:8000/
Quit the server with CONTROL-C.
```

`[^/.]+`: 匹配除 `/` 和 `.` 之外的内容

3）action 装饰器

作用：生成视图集中额外 API 接口的地址配置项。

```python
    # GET /books/latest/ -> latest
    @action(methods=['get'], detail=False)
    def latest(self, request):
        """获取 ID 最新的一本图书的数"""
        # self.action: 字符串, 表示要调用的方法是谁
        # TODO: 具体代码不作实现
        return Response({'message': '获取 ID 最新的一本图书的数'})

    # PUT /books/(?P<pk>\d+)/read/ -> read
    @action(methods=['put'], detail=True)
    def read(self, request, pk):
        """修改指定图书的阅读量(只修改阅读量)"""
        # TODO: 具体代码不作实现
        return Response({'message': '修改指定图书的阅读量(只修改阅读量)'})
```

```
<URLPattern '^books/$' [name='books-list']>
<URLPattern '^books/latest/$' [name='books-latest']>
<URLPattern '^books/(?P<pk>\d+)/$' [name='books-detail']>
System check identified no issues (0 silenced).
```

detail=False

```
Watching for file changes with StatReloader
<URLPattern '^books/$' [name='books-list']>
<URLPattern '^books/(?P<pk>\d+)/$' [name='books-detail']>
<URLPattern '^books/(?P<pk>\d+)/latest/$' [name='books-latest']>
System check identified no issues (0 silenced).
November 19, 2020 - 02:21:33
```

detail=True

4）SimpleRouter 路由生成规则

URL Style	HTTP Method	Action	URL Name
{prefix}/	GET	list	{basename}-list
	POST	create	
{prefix}/{url_path}/	GET, or as specified by `methods` argument	`@action(detail=False)` decorated method	{basename}-{url_name}
{prefix}/{lookup}/	GET	retrieve	{basename}-detail
	PUT	update	
	PATCH	partial_update	
	DELETE	destroy	
{prefix}/{lookup}/{url_path}/	GET, or as specified by `methods` argument	`@action(detail=True)` decorated method	{basename}-{url_name}

list 和 create

lastest

retrieve、update、destroy

read

```
Watching for file changes with StatReloader
<URLPattern '^books/$' [name='books-list']>
<URLPattern '^books/latest/$' [name='books-latest']>
<URLPattern '^books/(?P<pk>\d+)/$' [name='books-detail']>
<URLPattern '^books/(?P<pk>\d+)/read/$' [name='books-read']>
```

方法	URL地址	URL name
list和create	books/	books-list
latest	books/latest/	books-latest
retrieve、update、destroy	books/(?P<pk>\d+)/	books-detail
read	books/(?P<pk>\d+)/read/	books-read

下午课程核心要点

1. 认证&权限&限流

认证设置：全局认证设置和指定视图认证设置

权限设置：全局权限设置和指定视图权限设置

限流设置：分别限流设置和统一限流设置

2. 过滤&排序&分页

分页设置：

1）DRF框架提供的分页类：**PageNumberPagination** 和 **LimitOffsetPagination**

2）全局分页类设置

```
REST_FRAMEWORK = {
    # 设置DRF框架所使用的全局分页类
    "DEFAULT_PAGINATION_CLASS": "rest_framework.pagination.PageNumberPagination",
    # 指定页容量为2
    "PAGE_SIZE": 2,
}
```

3）自定义分页类

```
# 自定义分页类
# ?page=<页码>&pagesize=<页容量>
class StandardResultPagination(PageNumberPagination):
    # 指定分页的默认页容量
    page_size = 2
    # 指定获取分页数据时，页容量参数的名称
    # 注意：下面这个属性不要写成：page_query_param
    page_size_query_param = 'pagesize'
    # 指定分页时的最大页容量
    max_page_size = 5
```

3. 异常处理

默认异常处理函数：

ython3.8 〉 site-packages 〉 rest_framework 〉 settings.py drf_demo Git:

settings.py

```
    # Exception handling
    # DRF框架的默认异常处理函数配置项
    'EXCEPTION_HANDLER': 'rest_framework.views.exception_handler',
    'NON_FIELD_ERRORS_KEY': 'non_field_errors',
```

自定义异常处理：

需求：补充数据库 DatabaseError 异常处理功能。

views.py utils.py

```
from rest_framework.views import exception_handler as drf_exception_handler
from rest_framework.response import Response
from rest_framework import status

from django.db import DatabaseError


# 自定义DRF框架的异常处理函数
def exception_handler(exc, context):
    # ① 先调用 DRF 框架默认的异常处理函数
    response = drf_exception_handler(exc, context)

    # ② 如果默认的异常处理函数不能处理，自己补充想要处理的异常
    if response is None:
        # 补充自己想要处理的异常，比如：DatabaseError
        if isinstance(exc, DatabaseError):
            response = Response({'message': '数据库操作有误!'},
                                status=status.HTTP_500_INTERNAL_SERVER_ERROR)

    return response
```

```
131
132 REST_FRAMEWORK = {
133
134     # 指定 DRF 框架的默认异常处理函数
135     'EXCEPTION_HANDLER': 'booktest.utils.exception_handler'
136 }
```