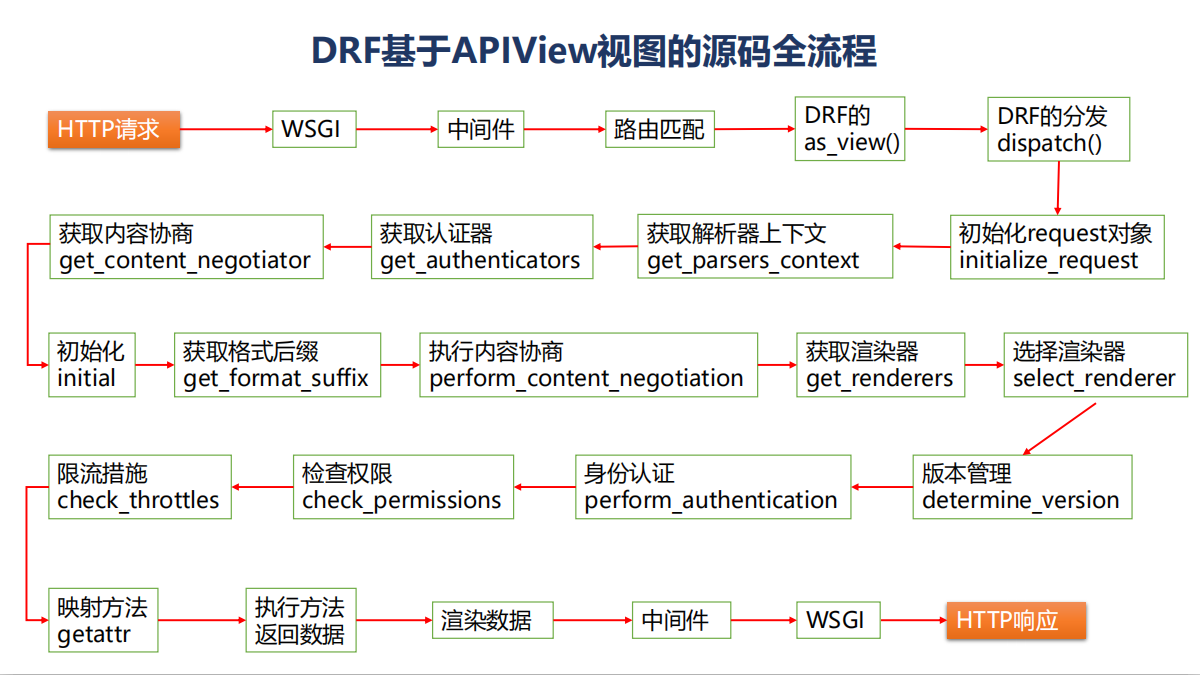


APIView



GenericAPIView

**get\_queryset(self)**

返回list视图中使用的查询集，该查询集还用作detail视图中的查找基础。

如：queryset = Post.objects.all()

**get\_object(self)**

返回用于detail视图的对象实例。默认使用 lookup\_field 参数过滤基本的查询集。 该方法可以被重写以提供更复杂的行为，例如基于多个 URL 参数的对象查找。

如：

filter\_kwargs = {self.lookup\_field: self.kwargs[lookup\_url\_kwarg]}

# 默认是{“pk”:”1”}

# 然后在根据过滤的来找，找不到就报错404

obj = get\_object\_or\_404(queryset, \*\*filter\_kwargs)

**filter\_queryset(self, queryset)**

给定一个queryset，使用任何过滤器后端进行过滤，返回一个新的queryset。

在过滤

**get\_serializer\_class(self)**

选择你想要使用的序列化类。默认返回 serializer\_class 属性的值。

可以被重写以提供动态的行为，例如对于读取和写入操作使用不同的序列化器或者为不同类 型的用户提供不同的序列化器。

get\_serializer\_class(self)本身没做啥，只是返回serializer\_class

这是一个钩子，通过这个可以指定返回自己需要的序列化类，如：

def get\_serializer\_class(self):

if self.request.user.is\_staff:

return FullAccountSerializer

return BasicAccountSerializer

**保存与删除操作中提供的钩子**:

以下方法由mixin类提供，并提供对象保存或删除行为的简单重写。

perform\_create(self, serializer) - 在保存新对象实例时 由 CreateModelMixin 调用。

perform\_update(self, serializer) - 在保存现有对象实例时 由 UpdateModelMixin 调用。

perform\_destroy(self, instance) - 在删除对象实例时由 DestroyModelMixin 调

用。

**三、Mixins**

Mixin 类用于提供视图的基本操作行为。注意mixin类提供动作方法，而不是直接定义处理程序

方法，例如 .get() 和 .post() ， 这允许更灵活的自定义。

Mixin 类可以从 rest\_framework.mixins 导入。

**ListModelMixin**

提供一个 .list(request, \*args, \*\*kwargs) 方法，返回查询结果的列表。

如果查询集被填充了数据，则返回 200 OK 响应，将查询集的序列化表示作为响应的主体。

相应数据可以任意分页。

**CreateModelMixin**

提供 .create(request, \*args, \*\*kwargs) 方法，实现创建和保存一个新model实例的功能。如果创建了一个对象，这将返回一个 201 Created 响应，将该对象的序列化表示作为响应的主体。如果序列化的表示中包含名为 url 的键，则响应的 Location 头将填充该值。

如果为创建对象提供的请求数据无效，将返回 400 Bad Request ，其中错误详细信息作为响 应的正文。

**RetrieveModelMixin**

提供一个 .retrieve(request, \*args, \*\*kwargs) 方法，返回响应中现有模型的实例。

如果可以检索对象，则返回 200 OK 响应，将该对象的序列化表示作为响应的主体。否则将

返回 404 Not Found 。

**UpdateModelMixin**

提供 .update(request, \*args, \*\*kwargs) 方法，实现更新和保存现有模型实例的功

能。 同时还提供了一个 .partial\_update(request, \*args, \*\*kwargs) 方法，这个方法

和 update 方法类似，但更新的所有字段都是可选的。这允许支持 HTTP PATCH 请求。

如果一个对象被更新，这将返回一个 200 OK 响应，并将对象的序列化表示作为响应的主 体。如果为更新对象提供的请求数据无效，将返回一个 400 Bad Request 响应，错误详细信息作为响应的正文。

**DestroyModelMixin**

提供一个 .destroy(request, \*args, \*\*kwargs) 方法，实现删除现有模型实例的功能。

如果成功删除对象，则返回 204 No Content 响应，否则返回 404 Not Found 。

DRF对mixin类的设计是让它们可以尽量的组合使用，不是一次只能继承一个mixin，可以同时继承多个mixin。

**注意：**

**1.**对于**CreateModelMixin，DestroyModelMixin，UpdateModelMixin**这三个可以对数据造成影响的，在他们对应的类都留下了一个钩子，用于对数据进行额外的操作

**2.**Create，Destroy，Update，List, Retrieve的叫法，分别对应Django的post,

delete,put,get,get方法，主要是为了区分Django原有的

class CreateAPIView(mixins.CreateModelMixin,

GenericAPIView):

def post()

# 就是调用mixins.CreateModelMixin里面的create()方法

return self.create()

------------------------------------------------------------------------------

class ListAPIView(mixins.ListModelMixin,

GenericAPIView):

def get()

# 就是调用mixins.ListModelMixin里面的list()方法

return self.list()

class RetrieveAPIView(mixins.RetrieveModelMixin,

GenericAPIView):

def get()

# 就是调用mixins.RetrieveModelMixin里面的retrieve()方法

return self.retrieve()

------------------------------------------------------------------------------ class DestroyAPIView(mixins.DestroyModelMixin,

GenericAPIView):

def delete()

# 就是调用mixins.DestroyModelMixin里面的destory()方法

return self.destory()

------------------------------------------------------------------------------ class UpdateAPIView(mixins.UpdateModelMixin,

GenericAPIView):

# 就是调用mixins.UpdateModelMixin里面的update()方法

def put(self, request, \*args, \*\*kwargs):

return self.update(request, \*args, \*\*kwargs)

# 就是调用mixins.UpdateModelMixin里面的partial\_update()方法

def patch(self, request, \*args, \*\*kwargs):

return self.partial\_update(request, \*args, \*\*kwargs)

其他组合类型同理也是一样的

ListCreateAPIView，RetrieveUpdateAPIView，RetrieveDestroyAPIView，RetrieveUpdateDestroyAPIView

**就是基于这上面的组合**

只继承GenericAPIView这个比较基础的

比如要通过get访问某条记录时就要写get()方法如：

ClassDetailPost(generics.GenericAPIView):

queryset = Post.objects.all()  
serializer\_class = PostSerializer

def get(self, request, \*args, \*\*kwargs):  
 instance = self.get\_object()  
 erializer = self.get\_serializer(instance)  
 return Response(serializer.data)

实际上就是重写RetrieveMixin里面的内容。

往上一步如果继承了RetrieveAPIView,只需要返回retrieve()

ClassDetailPost(mixins.RetrieveAPIView,generics.GenericAPIView):

queryset = Post.objects.all()  
serializer\_class = PostSerializer

def get(self, request, \*args, \*\*kwargs):  
 return self.retrieve(request, \*args, \*\*kwargs)

往上一步如果继承了RetrieveUpdateDestroyAPIView,只需写查询集和序列化类。

ClassDetailPost(mixins.RetrieveUpdateDestroyAPIView,generics.GenericAPIView):

queryset = Post.objects.all()  
serializer\_class = PostSerializer

提供读写删除单个模型实例。提供 get , put , patch 和 delete 方法