一、 路由器

有一些像Rails这样的Web框架提供自动生成Urls的功能。但是Django并没有。

REST framework为Django添加了这一功能，以一种简单、快速、一致的方式。

DRF的routers模块没有什么东西，主要包含下面几个类：

- BaseRouter：路由的基类
- SimpleRouter： 继承了BaseRouter，常用类之一
- DefaultRouter：继承了SimpleRouter，常用类之一
- DynamicDetailRoute
- DynamicListRoute
- RenameRouterMethods
- APIRootView

我们主要使用的其实就是SimpleRouter和DefaultRouter。

基本用法

下面是一个简单的例子，这里继承了 `SimpleRouter` 类。

```
from rest_framework import routers

router = routers.SimpleRouter()
router.register(r'users', UserViewSet,  basename='user')
router.register(r'accounts', AccountViewSet)
urlpatterns = router.urls
```

1. 导入routers模块
2. 实例化一个SimpleRouter对象router
3. 使用router的register方法注册两条路由模式
4. 将router的urls属性赋值给Django的基本路由变量urlpatterns

对于 `register()` 方法，有两个必填参数:

- `prefix` ： 用户视图集的URL路径的前缀
- `viewset` ： 对应的视图集类

另外还有一个可选，但又特别重要的参数：

- `basename` ：URL的name属性的基础部分。如果没有显式指定这个参数，那么将以视图集的 `queryset` 属性的值，自动生成。如果视图集没有 `queryset` 属性，那么你必须在register方法里设置。

上面的例子将自动生成下面的URL模式：

- `users/` Name: `'user-list'`
- `users/<int:pk>/` Name: `'user-detail'`
- `accounts/` Name: `'account-list'`
- `accounts/<int:pk>/` Name: `'account-detail'`

`basename` 参数就是用于指定上面模式中，Name值对应的字符串的短横线的前面部分，也就是字符串 `user` 或者 `account` 部分。

前面说了，通常我们不需要指定 `basename` 参数的值，但是如果你在视图集类中写了一个自定义的 `get_queryset` 方法，可能视图集就会缺少一个 `.queryset` 属性，这种情况下，如果你不指定 `basename` 参数的值，会导致下面的异常：

```
'basename' argument not specified, and could not automatically determine the name from the viewset, as it does not have a '.queryset' attribute.
```

在路由器的语法中使用 `include` 方法转发路由

其实，DRF路由器对象的urls属性，本质上是一个Django标准的URL模式对象，同样可以使用标准的Django路由语法和功能。

例如，你可以将 `router.urls` 追加到现有的urlpatterns里，也就是列表+列表的合并操作：

```
router = routers.SimpleRouter()
router.register(r'users', UserViewSet)
router.register(r'accounts', AccountViewSet)

urlpatterns = [
    path('forgot-password/', ForgotPasswordFormView.as_view()),
]

# 注意这一行!
urlpatterns += router.urls
```

或者使用Django的 `include` 函数，如下所示：

```
urlpatterns = [
    path('forgot-password/', ForgotPasswordFormView.as_view()),
    path('', include(router.urls)),  # 转发到DRF的router
]
```

也可以提供一个app的命名空间参数：

```
urlpatterns = [
    path('forgot-password/', ForgotPasswordFormView.as_view()),
    path('api/', include((router.urls, 'app_name'))),
]
```

甚至同时使用app和实例的命名空间：

```
urlpatterns = [
    path('forgot-password/', ForgotPasswordFormView.as_view()),
    path('api/', include((router.urls, 'app_name'),
namespace='instance_name')),
]
```

注意：如果使用了超链接序列化器时，使用命名空间要确保序列化器的 `view_name` 参数正确的映射到了命名空间。在上面的例子中，你就需要提供一个类似 `view_name='app_name:user-detail'` 的参数给序列化器的超链接字段，用于指向用户的详情视图。

默认情况下，自动生成的 `view_name` 属性是 `%(model_name)-detail` 这种格式的。

为额外的动作（actions）生成路由

DRF的视图集可以通过 `@action` 装饰器，为类的方法提供额外动作。路由器也会为这些动作自动生成URL模式。比如下面的例子：

```
from myapp.permissions import IsAdminOrIsSelf
from rest_framework.decorators import action

class UserViewSet(ModelViewSet):
    ...
     # 参考视图集中action的帮助
    @action(methods=['post'], detail=True, permission_classes=
[IsAdminOrIsSelf])
    def set_password(self, request, pk=None):
        ...
```

自动生成的路由模式如下：

- `users/<int:pk>/set_password/` name: `'user-set-password'`

默认情况下，生成的URL模式以方法名为基础。而name参数的值则组合了 `ViewSet.basename` 和方法名，以短横线连接，比如 `'user-set-password'` 。

如果你不想用上面的生成规则，可以自己指定 `@action` 装饰器的 `url_path` 和 `url_name` 参数，如下所示：

```
from myapp.permissions import IsAdminOrIsSelf
from rest_framework.decorators import action

class UserViewSet(ModelViewSet):
    ...
      # 注意参数名
    @action(methods=['post'], detail=True, permission_classes=
[IsAdminOrIsSelf],
            url_path='change-password', url_name='change_password')
    def set_password(self, request, pk=None):
        ...
```

然后，生成的URL模式就是这样的了：

- `users/<int:pk>/change-password/` name: `'user-change_password'`

二、API参考

SimpleRouter

SimpleRouter类包含标准的 `list`，`create`，`retrieve`，`update`，`partial_update` and `destroy` 这些操作所对应的url，以及 `@action` 装饰器带来的操作。 下表列出了对应的关系：

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

默认情况下， `SimpleRouter` 将为每一条url添加一个斜杠后缀，这也是Django的通常做法。如果你不想这么做，可以在初始化的时候提供 `trailing_slash` 参数，并设置为 `False` ：

```
router = SimpleRouter(trailing_slash=False)
```

路由器将匹配包含除斜杠和句点字符之外的任何字符的匹配值。对于更严格（或宽松）的查找模式，请在视图集上设置 `lookup_value_regex` 属性。例如，可以将查找范围限制为有效的UUID：

```
class MyModelViewSet(mixins.RetrieveModelMixin, viewsets.GenericViewSet):
    lookup_field = 'my_model_id'
    lookup_value_regex = '[0-9a-f]{32}'
```

DefaultRouter

DefaultRouter类和 `SimpleRouter` 基本类似，不同之处在于它还会自动生成默认的API根视图的url路径。另外，它还可以为 `.json` 类型的请求生成对应的url

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

同样的，也可以指定是否追加最后的那个斜杠：

```
1   router = DefaultRouter(trailing_slash=False)
```