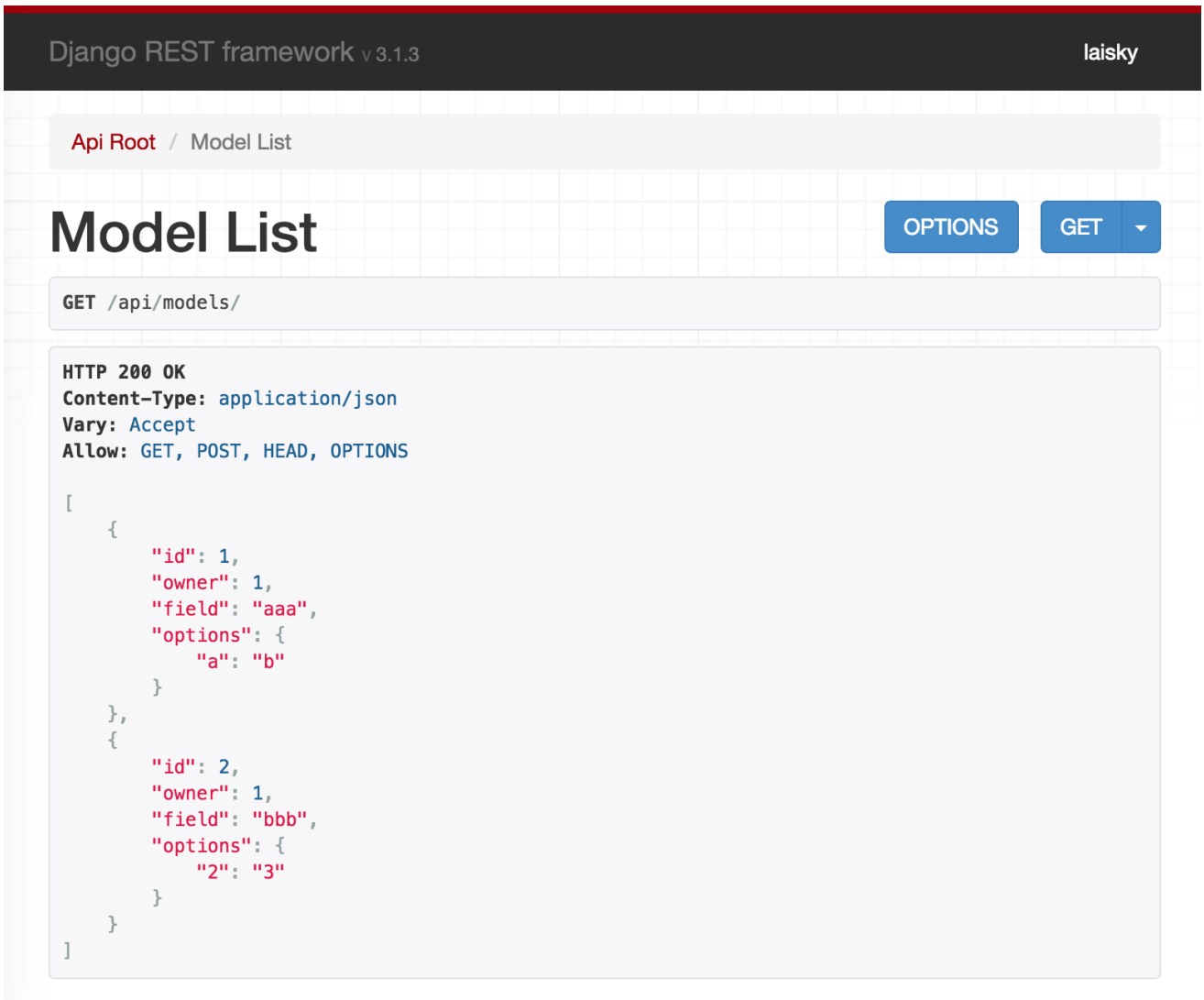
# 利用 Django REST framework 编写 RESTful API

- 自动生成符合 RESTful 规范的 API
  - 支持 OPTION、HEAD、POST、GET、PATCH、PUT、DELETE
  - 根据 `Content-Type` 来动态的返回数据类型（如 text、json）
- 生成 browserable 的交互页面（自动为 API 生成非常友好的浏览器页面）
- 非常细粒度的权限管理（可以细粒度到 field 级别）

*示意图*

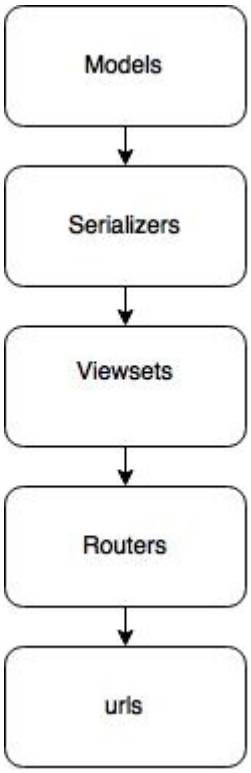



## 安装

```
$ pip install djangorestframework
$ pip install markdown
```

# 概述

Django Rest framework 的流程大概是这样的




1. 建立 Models
2. 依靠 Serialiers 将数据库取出的数据 Parse 为 API 的数据（可用于返回给客户端，也可用于浏览器显示）
3. ViewSet 是一个 views 的集合，根据客户端的请求（GET、POST等），返回 Serialiers 处理的数据
    - 权限 Premissions 也在这一步做处理
4. ViewSet 可在 Routers 进行注册，注册后会显示在 Api Root 页上
5. 在 urls 里注册 ViewSet 生成的 view，指定监听的 url

希望全面细致了解的人请移步去看官方文档，我这里就不一步步的细说了，而是分块来进行介绍

# 准备工作 & Models

让我们来写个小项目练练手

1. 先用 `manage.py startproject rest` 来生成一个项目
2. 再用 `manage.py createsuperuser` 创建用户（后面权限管理会用到）
3. 初始化数据库 `manage.py migrate`

然后当然是编写 models，为了展示 rest_framework 的强大之处，我给 models 定义了一个自定义的 field

```
# myproject/myapp/models.py
```

#! /usr/bin/env python # -*- coding: utf-8 from **future** import unicode_literals, absolute_import import cPickle as pickle

```
from django.db import models
from django.contrib.auth.models import User
```

class SerializedField(models.TextField):

```
    """序列化域
    用 pickle 来实现存储 Python 对象
    """
    __metaclass__ = models.SubfieldBase  # 必须指定该 metaclass 才能使用 to_python

    def validate(self, val):
        raise isinstance(val, basestring)

    def to_python(self, val):
        """从数据库中取出字符串，解析为 python 对象"""
        if val and isinstance(val, unicode):
            return pickle.loads(val.encode('utf-8'))

        return val

    def get_prep_value(self, val):
        """将 python object 存入数据库"""
        return pickle.dumps(val)
```

class MyModel(models.Model):

```
    created_at = models.DateTimeField(auto_now_add=True)
    # 注意这里建立了一个外键
    owner = models.ForeignKey(User, related_name='mymodels')
    field = models.CharField(max_length=100)
    options = SerializedField(max_length=1000, default={})
```

---

# Serializers

定义好了 Models，我们可以开始写 Serializers，这个相当于 Django 的 Form

```
# myproject/myapp/serializers.py

#! /usr/bin/env python
# -*- coding: utf-8
from __future__ import unicode_literals, absolute_import
import json

from django.contrib.auth.models import User
from rest_framework import serializers

from ..models import MyModel
from .fields import MyCustField
```

class MyCustField(serializers.CharField): """为 Model 中的自定义域额外写的自定义 Serializer Field"""

```
    def to_representation(self, obj):
        """将从 Model 取出的数据 parse 给 Api"""
        return obj

    def to_internal_value(self, data):
        """将客户端传来的 json 数据 parse 给 Model"""
        return json.loads(data.encode('utf-8'))
```

class UserSerializer(serializers.ModelSerializer):

```
    class Meta:
        model = User  # 定义关联的 Model
        fields = ('id', 'username', 'mymodels')  # 指定返回的 fields

    # 这句话的作用是为 MyModel 中的外键建立超链接，依赖于 urls 中的 name 参数
    # 不想要这个功能的话完全可以注释掉
    mymodels = serializers.HyperlinkedRelatedField(
        many=True, queryset=MyModel.objects.all(),
        view_name='model-detail'
    )
```

class MySerializer(serializers.ModelSerializer):

```
    options = MyCustField(
        max_length=1000, style={'base_template': 'textarea.html'},
    )

    class Meta:
        model = MyModel
        fields = ('id', 'owner', 'field', 'options')
        read_only_fields = ('owner',)  # 指定只读的 field
```

```
    def create(self, validated_data):
        """响应 POST 请求"""
        # 自动为用户提交的 model 添加 owner
        validated_data['owner'] = self.context['request'].user
        return MyModel.objects.create(**validated_data)

    def update(self, instance, validated_data):
        """响应 PUT 请求"""
        instance.field = validated_data.get('field', instance.field)
        instance.save()
        return instance
```

---

# ViewSet

定义好了 Serializers，就可以开始写 viewset 了

其实 viewset 反而是最简单的部分，rest_framework 原生提供了四种 ViewSet

- `ViewSet`
- `GenericViewSet`
  - 继承于 `GenericAPIView`
- `ModelViewSet`
  - 自身提供了六种方法
  - `list`
  - `create`
  - `retrieve`
  - `update`
  - `partial_update`
  - `destroy`
- `ReadOnlyModelViewSet`

我比较喜欢用 `ModelViewSet`，然后再用 Premissions 来管理权限

```
# myproject/myapp/views.py

#! /usr/bin/env python
# -*- coding: utf-8
from __future__ import unicode_literals, absolute_import

from django.contrib.auth.models import User
from rest_framework import permissions, viewsets, renderers
from rest_framework.decorators import (
    permission_classes, detail_route
)
from rest_framework.response import Response

from .serializers import MySerializer, UserSerializer
from .models import MyModel
```

class UserViewSet(viewsets.ModelViewSet): queryset = User.objects.all() serializer_class = UserSerializer # 指定权限，下面马上讲到 permission_classes = (permissions.IsAuthenticated,)

class ModelViewSet(viewsets.ModelViewSet): queryset = MyModel.objects.all() serializer_class = MySerializer permission_classes = (permissions.IsAuthenticatedOrReadOnly,)

```
    @detail_route(renderer_classes=[renderers.StaticHTMLRenderer])
    def plaintext(self, request, *args, **kwargs):
        """自定义 Api 方法"""
        model = self.get_object()
        return Response(repr(model))
```

我在 ModelViewSet 中自定义了方法 plaintext，rest_framework 中对于自定义的 viewset 方法提供了两种装饰器

- `list_route`
- `detail_route`

区别就是 `list_route` 的参数不包含 `pk`（对应 list），而 `detail_route` 包含`pk`（对应 retrieve）

看一段代码就懂了

```
@list_route(methods=['post', 'delete'])
def custom_handler(self, request):
    pass
```

@detail_route(methods=['get']) def custom_handler(self, request, pk=None): pass

---

# Filters

前面根据 serializers 和 viewset 我们已经可以很好的提供数据接口和展示了。但是有时候我们需要通过 url参数 来对数据进行一些排序或过滤的操作，为此，rest-framwork 提供了 filters 来满足这一需求。

## 全局filter

可以在 settings 里指定应用到全局的 filter：

```
REST_FRAMEWORK = {
    'DEFAULT_FILTER_BACKENDS': ('rest_framework.filters.DjangoFilterBackend',)
}
```

## viewset 的 filter

也可以为 viewset 分别指定 filter，方法就是在定义 viewset 的时候定义一个名为`filter_backend` 的类变量：

```
class UserListView(generics.ListAPIView):
    queryset = User.objects.all()
    serializer = UserSerializer
    filter_backends = (filters.DjangoFilterBackend,)
```

## 默认的 filter

rest-framework 提供了几个原生的 filter：

- SearchFilter

  filter_backends = (filters.SearchFilter,) search_fields = ('username', 'email') # 指定搜索的域

请求 `http://example.com/api/users?search=russell`。

- OrderingFilter

  filter_backends = (filters.OrderingFilter,) ordering_fields = ('username', 'email')

请求 `http://example.com/api/users?ordering=account,-username`。

## 自定义 filter

自定义 filter 非常简单，只需要定义 `filter_queryset(self, request, queryset, view)` 方法，并返回一个 queryset 即可。

直接贴一个我写的例子：

```
class NodenameFilter(filters.BaseFilterBackend):

    """根据 nodename 来删选
      [nodename]: NeiWang
    """

    def filter_queryset(self, request, queryset, view):
        nodename = request.QUERY_PARAMS.get('nodename')
        if nodename:
            return queryset.filter(nodename=nodename)
        else:
            return queryset
```

如果参数匹配有误，想要抛出异常的话，也可以自定义 APIError，举个例子：

```
from rest_framework.exceptions import APIException
```

class FilterError(APIException): status_code = 406 default_detail = 'Query arguments error!'

然后在 viewset 里直接抛出 `raise FilterError` 即可。

---

# Premissions

顾名思义就是权限管理，用来给 ViewSet 设置权限，使用 premissions 可以方便的设置不同级别的权限：

- 全局权限控制
- ViewSet 的权限控制
- Method 的权限
- Object 的权限

被 premission 拦截的请求会有如下的返回结果：

- 当用户已登录，但是被 premissions 限制，会返回 `HTTP 403 Forbidden`
- 当用户未登录，被 premissions 限制会返回 `HTTP 401 Unauthorized`

## 默认的权限

rest_framework 中提供了七种权限

- `AllowAny` # 无限制
- `IsAuthenticated` # 登陆用户
- `IsAdminUser` # Admin 用户
- `IsAuthenticatedOrReadOnly` # 非登录用户只读
- `DjangoModelPermissions` # 以下都是根据 Django 的 ModelPremissions
- `DjangoModelPermissionsOrAnonReadOnly`
- `DjangoObjectPermissions`

## 全局权限控制

在 settings.py 中可以设置全局默认权限

```
# settings.py

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

## ViewSet 的权限

可以设置 `permission_classes` 的类属性来给 viewset 设定权限，restframework 会检查元组内的每一个 premission，必须要全部通过才行。

```
class UserViewSet(viewsets.ReadOnlyModelViewSet):
    queryset = User.objects.all()
    serializer_class = UserSerializer
    # 设置权限，是一个元组
    permission_classes = (permissions.IsAuthenticated,)
```

## 自定义权限

Premissions 可以非常方便的定制，比如我就自己写了一个只允许 owner 编辑的权限

```python
# myproject/myapp/premissions.py

#! /usr/bin/env python
# -*- coding: utf-8
from __future__ import unicode_literals, absolute_import

from rest_framework import permissions

class IsOwnerOrReadOnly(permissions.BasePermission):

    def has_permission(self, request, view):
        """针对每一次请求的权限检查"""
        if request.method in permissions.SAFE_METHODS:
            return True

    def has_object_permission(self, request, view, obj):
        """针对数据库条目的权限检查，返回 True 表示允许"""
        # 允许访问只读方法
        if request.method in permissions.SAFE_METHODS:
            return True

        # 非安全方法需要检查用户是否是 owner
        return obj.owner == request.user
```

## urls & routers

```python
# myproject/myapp/urls.py

#! /usr/bin/env python
# -*- coding: utf-8
from __future__ import unicode_literals, absolute_import

from django.conf.urls import url, patterns, include
from rest_framework.routers import DefaultRouter

from . import views

# as_view 方法生成 view
# 可以非常方便的指定 `{Http Method: View Method}`
user_detail = views.UserViewSet.as_view({'get': 'retrieve'})
user_list = views.UserViewSet.as_view({'get': 'list', 'post': 'create'})

# plaintext 是我的自定义方法，也可以非常方便的指定
modal_plain = views.ModelViewSet.as_view({'get': 'plaintext'})
model_detail = views.ModelViewSet.as_view({'get': 'retrieve', 'post': 'create'})
model_list = views.ModelViewSet.as_view({'get': 'list', 'post': 'create'})

# router 的作用就是自动生成 Api Root 页面
router = DefaultRouter()
router.register(r'models', views.ModelViewSet)
```

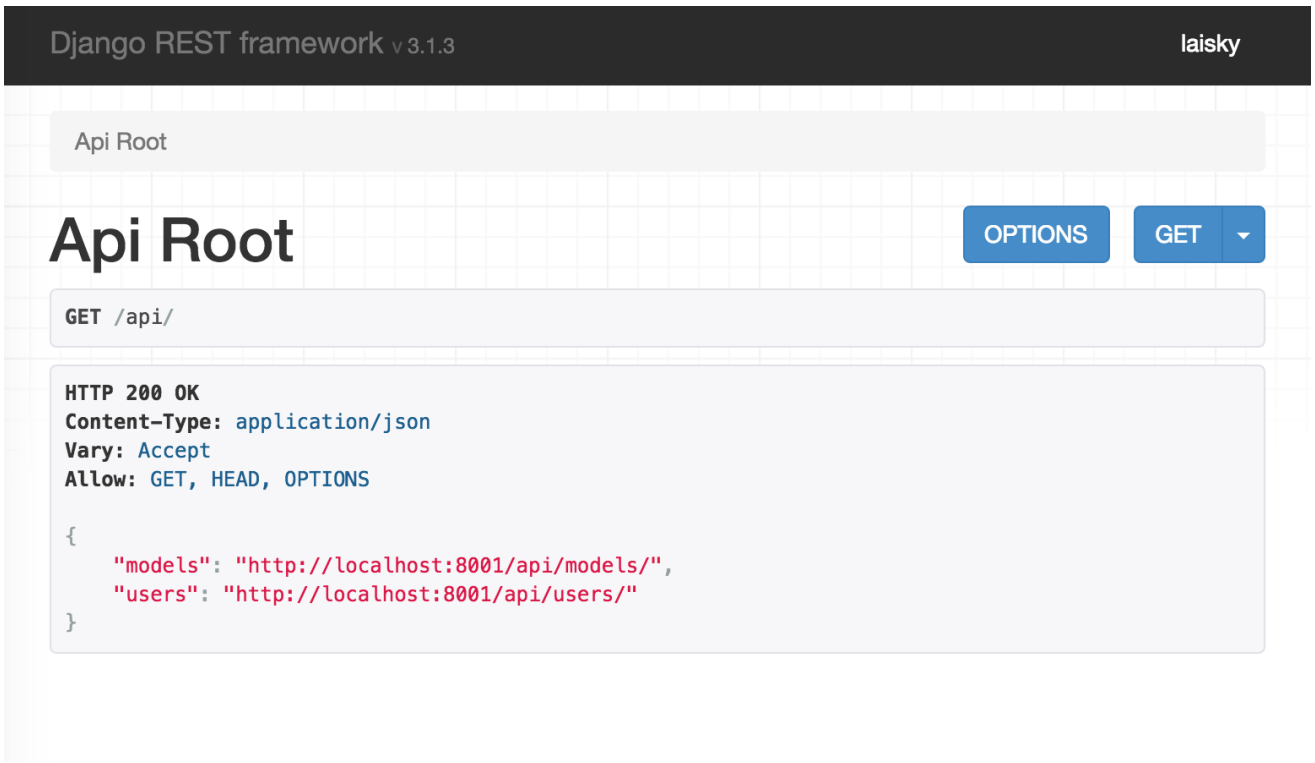
```
router.register(r'users', views.UserViewSet)

# 不要忘了把 views 注册到 urls 中
urlpatterns = patterns(
    '',
    url(r'^', include(router.urls)),  # Api Root
    url(r'^api-auth/', include('rest_framework.urls', namespace='rest_framework')),
    url(r'^models/(?P<pk>[0-9]+)/$', model_detail, name='model-detail'),
    url(r'^models/(?P<pk>[0-9]+)/plain/$', modal_plain, name='model-plain'),
    url(r'^models/$', model_list, name='model-list'),
    url(r'^users/$', user_list, name='user-list'),
    url(r'^users/(?P<pk>[0-9]+)/$', user_detail, name='user-detail'),
)
```

---

时间仓促，就介绍这些，以后有空再介绍一下在 Django 用 JWT 作为身份凭证。下面是一些效果图

- Api Root




- Users

Django REST framework v 3.1.3 laisky

# User List

OPTIONS GET

This viewset automatically provides `list` and `detail` actions.

```
GET /api/users/
```

```
HTTP 200 OK
Content-Type: application/json
Vary: Accept
Allow: GET, POST, HEAD, OPTIONS

[
    {
        "id": 1,
        "username": "laisky",
        "mymodels": [
            "http://localhost:8001/api/models/1/",
            "http://localhost:8001/api/models/2/"
        ]
    }
]
```

Raw data | HTML form

**Username**

Required. 30 characters or fewer. Letters, digits and @/./+/-/_ only.

**Mymodels**

MyModel object - http://localhost:8001/api/models/1/
MyModel object - http://localhost:8001/api/models/2/