1. 所有DRF能做的，Django都能做，只不过更麻烦。你可以不使用DRF的序列化等所有功能，但你必须自己实现，你可以认为DRF的功能不够强大，提供的类比较单薄，甚至所有的东西都自定义，而不使用DRF的任何已有类，那么你还用DRF干什么呢？你实际上已经创建了一个新模块。反过来说，如果只用DRF的类，而不做任何自定义或者修改，又无法满足实际工作的需求，甚至连基本功能可能都无法实现，这也说明DRF不够强大，不够体系。
2. RESTful只是个建议的规范，不是强制的
3. 以前，解析请求和渲染结果，都是Django帮我们做了，都是黑盒子。
4. 做前后端分离的项目的后端开发，就是做API或者接口的开发。后端将url暴露给前端，前端使用url访问后端的API。
5. FBV:基于函数的视图；CBV:基于类的视图
6. 关于CSRF：

```python
# 对于普通的基于函数的视图，如果想取消CSRF限制，直接使用csrf_exempt装饰器即可

from django.views.decorators.csrf import csrf_exempt

@csrf_exempt
def someview(request):
    pass


# 对于基于类的视图，如果想取消CSRF限制，第一种方法是重写dispatch方法，添加csrf_exempt装饰器
from django.views import View
from django.utils.decorators import method_decorator


class SomeView(View):

    @method_decorator(csrf_exempt)
    def dispatch(self, request, *args, **kwargs):
        pass

    def get(self, request, *arg, **kwargs):
        pass

    pass


```

```python
# 对于基于类的视图，如果想取消CSRF限制，第二种方法是直接在类上添加csrf_exempt装饰器，注意参数
@method_decorator(csrf_exempt, name='dispatch')
class AnotherView(View):

    def get(self, request, *arg, **kwargs):
        pass

    pass


# 对于Django REST framework 它的APIView中的as_view()方法，帮我们使用了csrf_exempt()方法。

from django.views.decorators.csrf import csrf_exempt
class APIView(View):

    pass

    @classmethod
    def as_view(cls, **initkwargs):
        if isinstance(getattr(cls, 'queryset', None), models.query.QuerySet):
            def force_evaluation():
                raise RuntimeError(
                    'Do not evaluate the `.queryset` attribute directly, '
                    'as the result will be cached and reused between requests. '
                    'Use `.all()` or call `.get_queryset()` instead.'
                )
            cls.queryset._fetch_all = force_evaluation

        view = super(APIView, cls).as_view(**initkwargs)
        view.cls = cls
        view.initkwargs = initkwargs

        # Note: session based authentication is explicitly CSRF validated,
        # all other authentication is CSRF exempt.
        return csrf_exempt(view)  # 看这里！！！！！！！！！！！！！！！
```

7. 阅读DRF的源码，从APIView类的dispatch方法开始。
8. 从DRF的request对象中获取Django原生的request对象：`requst._request`
9. 从Django的ORM的查询集中获取第一个对象使用`first()`方法

10. 可以在Pycharm里直接使用Sqlite3的插件对数据库内的数据进行CRUD，不用进入admin后台。
11. 在执行 `json.dumps` 的时候，如果提供参数 `ensure_ascii=False` ，不会将中文编码成ASCII码，更方便和直观。
12. serializer中的字段的名字，必须和对应的model中字段的名字一样。但是你如果使用了source参数，那么就可以不一样了。
13. 使用Django原生FBV能力编写API接口

```
import json
from django.views.decorators.csrf import csrf_exempt
from django.http import JsonResponse

user_dict = {
    "name": 'tom',
    "sex": " male",
    "age": 18
}

@csrf_exempt
def userList(request):
    if request.method == 'GET':
        # return HttpResponse(json.dumps(user_dict),
content_type='application/json')
        return JsonResponse(user_dict)
    if request.method == 'POST':
        # print(request.POST)
        # print(request.body)
        user_form = json.loads(request.body.decode('utf-8'))
        print(user_form)
        # return JsonResponse(user_form, safe=False)
        return HttpResponse(json.dumps(user_form),
content_type='application/json')
```

14. 使用Django的类视图CBV编写API：

```
url.py:
###################
path('users/', views.UserList.as_view())


```

```
views.py
#####################
from django.views import View
from django.utils.decorators import method_decorator


@method_decorator(csrf_exempt, name='dispatch')
class UserList(View):

    def get(self,request):
        return JsonResponse(user_dict)

    def post(self,request):
        data = json.loads(request.body.decode('utf-8'))
        print(data, type(data))
        # return HttpResponse(json.dumps(data),
content_type='application/json')
        return JsonResponse(data)
```

15. Django2.0之后的urls.py文件中的url模式，是完全匹配，所以'/users/'不会短路'/users/high/'。