请求传参

利用HTTP请求报文传递参数有3种方式

1. 查询字符串，GET方法，参数在URL中，`http://127.0.0.1:8000/emp/?x=123&x=abc&y=789`
2. 表单，前端网页中填写表单，一般使用POST方法，参数在body中

```
POST /xxx/yyy?id=5&name=magedu HTTP/1.1
HOST: 127.0.0.1:9999
content-length: 26
content-type: application/x-www-form-urlencoded

age=5&weight=80&height=170
```

   也可以POST、PUT提交Json格式数据

3. URL本身就是数据的表达，`http://127.0.0.1:8080/python/2010/u101`

APIView

在Django中，View是视图类基类，路由配置中需要把类通过as_view()伪装成视图函数，请求通过路由进入到这个视图函数中，内部为每一个请求实例化一个视图类实例，并根据request.method找到对应的handler。而这个基本流程已经被View类固定在其内部，我们只需要定义get、post等方法即可，简化了编程。

使用DRF，基于Django，也要使用View类。请求参数是Json格式，先要序列化它，然后验证，验证合格可以入库。响应的数据应该序列化成Json格式。你会发现这也是固定的套路，是否也能够简化呢?

参考 https://www.django-rest-framework.org/api-guide/views/

APIView

- as_view()调用基类View的，但是使用了csrf_exempt(view)来排除CSRF保护
- 重新定义了Request类来替代Django的，虽是重写，但是依然有联系
- 重新定义了Response类来增强替代Django的
- 异常类都是基于APIException类的
- 对请求进行认证和授权

APIView没有提供增删改查的handler方法，也就是说和View一样，需要自己定义get、post、put、delete方法。

GET请求测试

GET请求：http://127.0.0.1:8000/emp/?x=123&x=abc&y=789

```python
from rest_framework.views import APIView, Request, Response

class TestIndex(APIView):
    def get(self, request:Request):
        print('~' * 30)
        print(request.method)       # HttpRequest的属性
        print(request.GET)          # HttpRequest的属性
        print(request.query_params) # Request的属性，使用小写的
        print(request.content_type)
        print('~' * 30)
        return Response({})
```

```
GET
<QueryDict: {'x': ['123', 'abc'], 'y': ['789']}>
<QueryDict: {'x': ['123', 'abc'], 'y': ['789']}>
text/plain
```

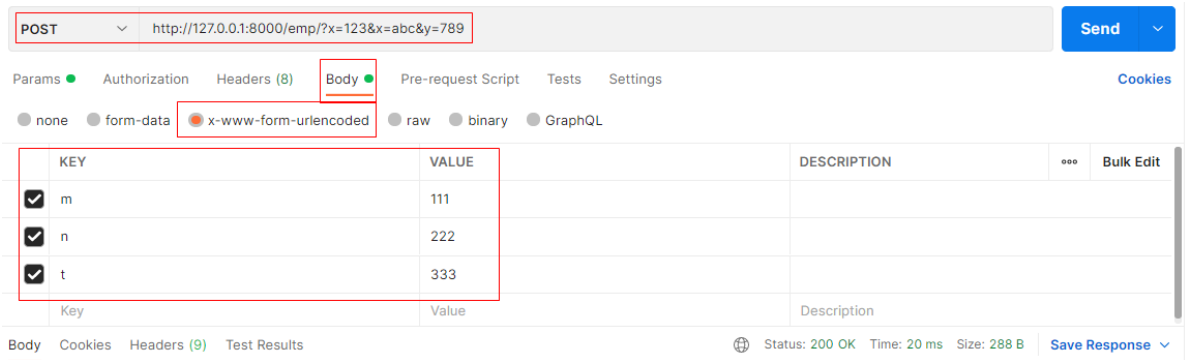
QueryDict本质就是字典，对于同一个参数有多值情况使用了列表。

POST请求测试

POST请求：http://127.0.0.1:8000/emp/?x=123&x=abc&y=789

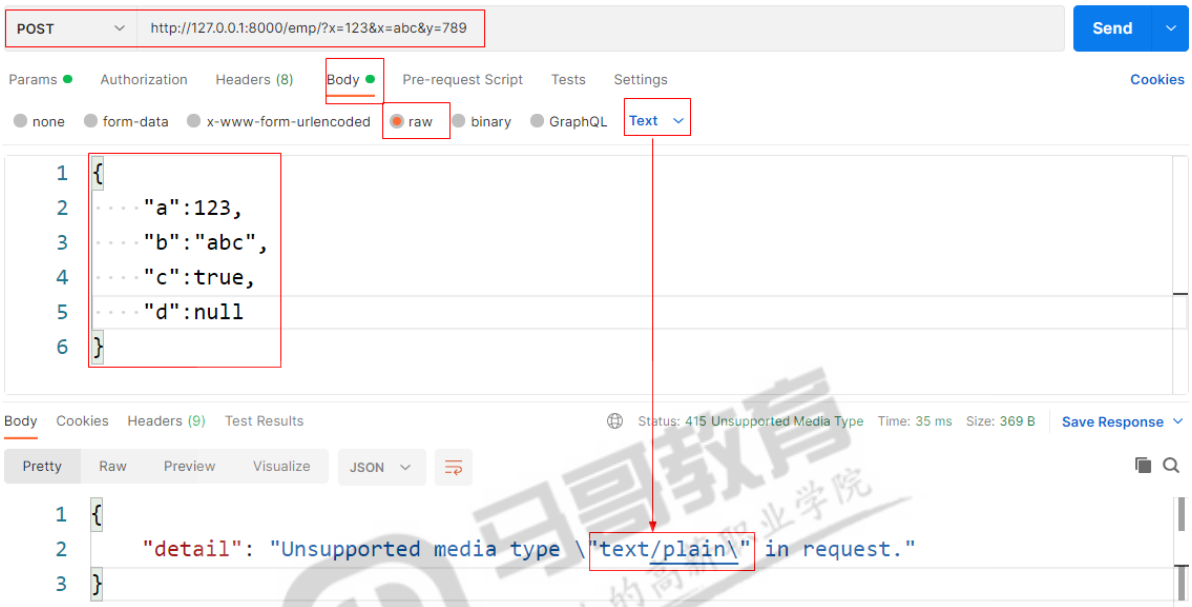
一、采用表单提交方式

```python
from rest_framework.views import APIView, Request, Response

class TestIndex(APIView):
    def post(self, request:Request):
        print('~' * 30)
        print(request.method)       # HttpRequest的属性
        print(request.GET)          # HttpRequest的属性
        print(request.query_params) # Request的属性，使用小写的
        print(request.content_type)
        print(request.POST)         # HttpRequest的属性
        print(request.data)         # Request的属性，使用小写的
        print('~' * 30)
        return Response({})
```



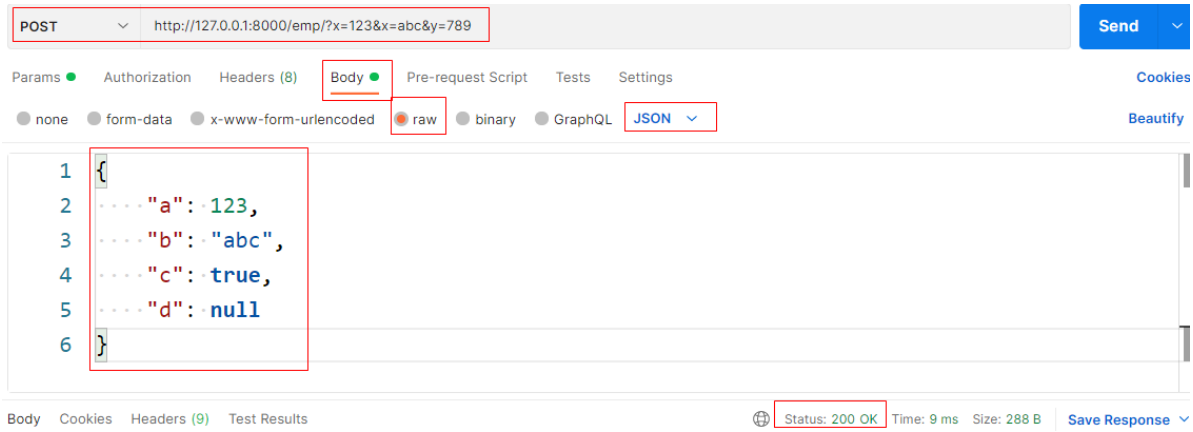
```
b'm=111&n=222&t=333' 这是request报文的body的原始数据

POST
<QueryDict: {'x': ['123', 'abc'], 'y': ['789']}>
<QueryDict: {'x': ['123', 'abc'], 'y': ['789']}>
application/x-www-form-urlencoded 表单提交的方法
<QueryDict: {'m': ['111'], 'n': ['222'], 't': ['333']}>
<QueryDict: {'m': ['111'], 'n': ['222'], 't': ['333']}>
```

二、Json数据




```
b'{\r\n    "a":123,\r\n    "b":"abc",\r\n    "c":true,\r\n    "d":null\r\n}'
text/plain

Unsupported Media Type: /emp/
```

说明上例解析失败，原因就是类型选择了Text，改为Json



```
b'{\r\n    "a": 123,\r\n    "b": "abc",\r\n    "c": true,\r\n    "d": null\r\n}'

POST
<QueryDict: {'x': ['123', 'abc'], 'y': ['789']}>
<QueryDict: {'x': ['123', 'abc'], 'y': ['789']}>
application/json
<QueryDict: {}>
{'a': 123, 'b': 'abc', 'c': True, 'd': None}
```

请求总结

- GET请求，查询字符串使用**query_params**提取
- POST请求，访问**data**属性
  - 支持了POST、PUT、PATCH方法（增、改）
  - 统一将请求处理放到data属性中
  - 如果是Json数据，帮我们序列化

响应

DRF对Django的响应类也做了增强，使用更加简单方便。

`Response(data=None, status=None, template_name=None, headers=None, content_type=None)`

- 兼具了模板渲染的能力
- data将要序列化的数据，例如字典
- staus状态码，默认200
- headers响应报文头，字典
- content_type响应内容类型，有时候需要手动设置

```python
from rest_framework.views import APIView, Request, Response

class TestIndex(APIView):
    def get(self, request:Request):
        print(request.query_params) # Request的属性，使用小写的
        return Response({})

    def post(self, request:Request):
        print(request.query_params) # Request的属性，使用小写的
        print(request.data)         # Request的属性，使用小写的
        return Response({
            'host':'python', 'domain':'magedu.com'
        }, status=201, headers={'X-Server':'Magedu'})
```

应用

需要构建2个类

- 列表页，返回列表和新增功能
- 详情页，基于**主键**的查看详情、修改、删除

路由

```python
from django.urls import path
from .views import EmpsView, EmpView

urlpatterns = [
    path('', EmpsView.as_view()), # /emp/
    path('<int:pk>/', EmpView.as_view()), # /emp/10021/
]
```

列表页

```python
from rest_framework.views import APIView, Request, Response

class EmpsView(APIView):
    """
    实现列表页get、新增post
    http://127.0.0.1:8000/emp/
    """
    def get(self, request):
        pass

    def post(self, request):
        pass
```

详情页

```python
class EmpView(APIView):
    """
    实现详情页get、修改put、删除delete
    http://127.0.0.1:8000/emp/10021
    """
    def get(self, request, pk:int):
        pass

    def put(self, request, pk:int):
        pass

    def delete(self, request, pk:int):
        pass

```

异常处理

参考 https://www.django-rest-framework.org/api-guide/exceptions/

测试http://127.0.0.1:8000/emp/100，返回500服务器内部错误。

按平常做法，就应该出什么问题，返回什么出错信息。但这样做会有安全风险，且在浏览器端的普通用户根本不管什么错误，就认为网站出问题了。所以，返回更加友好的出错页面是有必要的，例如更换404页面。

我们的项目采用前后端分离，返回出错页面不合适，需要返回json格式的出错信息，有必要拦截所有的异常做处理。

异常全局配置

自定义全局异常处理器，参考 rest_framework.views.exception_handler 实现

```
REST_FRAMEWORK = {
    'EXCEPTION_HANDLER': 'utils.exceptions.global_exception_handler'
}
```

全局异常处理

exception_handler会转化Django的Http404、PermissionDenied为DRF的基于APIException的类。

项目根目录下新建包utils，其中新建模块exceptions.py，内容如下

```
from django.http import Http404
from django.core.exceptions import PermissionDenied
from rest_framework.views import set_rollback
from rest_framework import exceptions
from rest_framework.views import Response, exception_handler

class MagBaseException(exceptions.APIException):
    """基类定义基本的异常"""
    code = 10000 # code为0表示正常，非0都是错误
    message = '非法请求' # 错误描述

    @classmethod
    def get_message(cls):
        return {'code': cls.code, 'message': cls.message}

# 内部异常暴露细节
exc_map = {

}


def global_exception_handler(exc, context):
    """
    全局异常处理

    照抄rest_framework.views.exception_handler，略作修改
    不管什么异常这里统一处理。根据不同类型显示不同的
    为了前端解析方便，这里响应的状态码采用默认的200
    异常对应处理后返回对应的错误码和错误描述
    异常找不到对应就返回缺省
    """
    if isinstance(exc, Http404):
        exc = exceptions.NotFound()
    elif isinstance(exc, PermissionDenied):
        exc = exceptions.PermissionDenied()

    print('异常', '=' * 30)
    print(type(exc), exc.__dict__)
    print('=' * 30)

    if isinstance(exc, exceptions.APIException):
        headers = {}
```

```
        if getattr(exc, 'auth_header', None):
            headers['WWW-Authenticate'] = exc.auth_header
        if getattr(exc, 'wait', None):
            headers['Retry-After'] = '%d' % exc.wait

        if isinstance(exc.detail, (list, dict)):
            data = exc.detail
        else:
            data = {'detail': exc.detail}

        set_rollback()

        errmsg = exc_map.get(exc.__class__.__name__,
MagBaseException).get_message()
        return Response(errmsg, status=200) # 状态恒为200
        #return Response(data, status=exc.status_code, headers=headers)

    return None
```

抛出异常后，拦截它们，这里做统一的处理

列表页和新增实现

```
from rest_framework.views import APIView, Request, Response
from .models import Employee
from .serializers import EmpSerializer

class EmpsView(APIView):
    """
    实现列表页get、新增post
    http://127.0.0.1:8000/emp/
    """
    def get(self, request):
        emps = Employee.objects.all()
        return Response(EmpSerializer(emps, many=True).data)

    def post(self, request):
        serializer = EmpSerializer(data=request.data)
        serializer.is_valid(True)
        serializer.save()
        return Response(serializer.data)
```

post测试数据

```
{
    "emp_no": 10023,
    "birth_date": "2000-06-01",
    "first_name": "sam",
    "last_name": "lee",
    "gender": 1,
    "hire_date": "2020-08-24"
}
```

详情页、修改和删除实现

```python
from rest_framework.views import APIView, Request, Response
from .models import Employee
from .serializers import EmpSerializer

class EmpView(APIView):
    """
    实现详情页get、修改put、删除delete
    http://127.0.0.1:8000/emp/10021
    """
    def get(self, request, pk:int):
        obj = Employee.objects.get(pk=pk)
        return Response(EmpSerializer(obj).data)


    def put(self, request, pk:int):
        # 必须先查后改
        obj = Employee.objects.get(pk=pk)
        serializer = EmpSerializer(obj, data=request.data)
        serializer.is_valid(True)
        serializer.save()
        return Response(serializer.data, 201)


    def delete(self, request, pk:int):
        Employee.objects.get(pk=pk).delete()
        return Response(status=204)

```

PUT测试用Json

```json
{
    "emp_no": 10023,
    "birth_date": "2000-06-01",
    "first_name": "sam",
    "last_name": "lee",
    "gender": 2,
    "hire_date": "2019-08-24"
}
```

完整参考代码

employee/views.py

```python
from rest_framework.views import APIView, Request, Response
from .models import Employee
from .serializers import EmpSerializer


class EmpsView(APIView):
    """
    实现列表页get、新增post
    http://127.0.0.1:8000/emp/

```

```
    """
    def get(self, request):
        emps = Employee.objects.all()
        return Response(EmpSerializer(emps, many=True).data)

    def post(self, request):
        serializer = EmpSerializer(data=request.data)
        serializer.is_valid(True)
        serializer.save()
        return Response(serializer.data)

class EmpView(APIView):
    """
    实现详情页get、修改put、删除delete
    http://127.0.0.1:8000/emp/10021
    """
    def get(self, request, pk:int):
        obj = Employee.objects.get(pk=pk)
        return Response(EmpSerializer(obj).data)

    def put(self, request, pk:int):
        # 必须先查后改
        obj = Employee.objects.get(pk=pk)
        serializer = EmpSerializer(obj, data=request.data)
        serializer.is_valid(True)
        serializer.save()
        return Response(serializer.data, 201)

    def delete(self, request, pk:int):
        Employee.objects.get(pk=pk).delete()
        return Response(status=204)
```

employee/models.py

```
from django.db import models

class Employee(models.Model):
    class Gender(models.IntegerChoices): # 枚举类型，限定取值范围
        MAN = 1, '男'
        FEMALE = 2, '女'
    class Meta:
        db_table = 'employees'
        verbose_name = '员工'
    emp_no = models.IntegerField(primary_key=True, verbose_name='工号')
    birth_date = models.DateField(verbose_name='生日')
    first_name = models.CharField(max_length=14, verbose_name='名')
    last_name = models.CharField(max_length=16, verbose_name='姓')
    gender = models.SmallIntegerField(verbose_name='性别',
choices=Gender.choices)
    hire_date = models.DateField()

class Salary(models.Model):
    class Meta:
        db_table = "salaries"
    #id = models.AutoField(primary_key=True) # 额外增加的主键，Django不支持联合主
键
    emp_no = models.ForeignKey(Employee, on_delete=models.CASCADE,
```

```
                                db_column='emp_no', related_name='salaries')
    from_date = models.DateField()
    salary = models.IntegerField(verbose_name='工资')
    to_date = models.DateField()
```

employee/serializers.py

```python
from rest_framework import serializers
from .models import Employee, Salary

class SalarySerializer(serializers.ModelSerializer):
    class Meta:
        model = Salary
        fields = '__all__'

class EmpSerializer(serializers.ModelSerializer):
    class Meta:
        model = Employee
        fields = '__all__'
```