HR3入参数据校验器

上节课我们完成了系统的生成用例文件的功能。但是目前json用例文件在转化成py文件过程中提示文件内容不合法。

比如我们的案例中提示的错误

```
2021-06-25 10:57:37.989 | WARNING  | httprunner.make:__make:567 - Invalid
testcase file: D:\Course\course_django\a
utotpsite_day8\testcase\荣耀王者_test.json
TestCaseFormatError: Invalid teststep validate: {}
```

这个原因是由于"validate"字段应该是 []而不是 {}，对照下课程1的文档，里面有各个字段的类型定义规范。

如果我们想在源头杜绝这个事情，就要在保存和更新数据时，校验入参，如果不合法就拒绝保存。

自己写过滤器来校验数据。

当字段或模型的规则不足以验证入参数据时，我们需要自定义规则来做数据验证。

ConfigSerializer

```python
# 配置
class ConfigSerializer(serializers.ModelSerializer):
    project = ProjectSerializer(read_only=True)

    class Meta:
        model = Config
        fields = '__all__'

    def validate(self, attrs):
        if 'variables' in attrs and not isinstance(attrs['variables'], dict):
            raise ValidationError('请传递正确的variables格式:dict')
        if 'parameters' in attrs and not isinstance(attrs['parameters'], dict):
            raise ValidationError('请传递正确的parameters格式:dict')
        if 'export' in attrs and not isinstance(attrs['export'], list):
            raise ValidationError('请传递正确的export格式:dict')
        return attrs
    ...
```

StepSerializer

```python
# 步骤模型的序列化器
class StepSerializer(serializers.ModelSerializer):
    request = RequestSerializer()
    belong_case_id = serializers.IntegerField(write_only=True, required=False)
 # 只写
```

```
    class Meta:
        model = Step
        fields = ['name', 'variables', 'request', 'extract', 'validate',
'setup_hooks', 'teardown_hooks',
                  'belong_case_id', 'sorted_no']

    def validate(self, attrs):
        if 'variables' in attrs and not isinstance(attrs['variables'], dict):
            raise ValidationError('请传递正确的variables格式:dict')
        if 'extract' in attrs and not isinstance(attrs['extract'], dict):
            raise ValidationError('请传递正确的extract格式:dict')
        if 'validate' in attrs and not isinstance(attrs['validate'], list):
            raise ValidationError('请传递正确的validate格式:list')
        if 'setup_hooks' in attrs and not isinstance(attrs['setup_hooks'],
list):
            raise ValidationError('请传递正确的setup_hooks格式:list')
        if 'teardown_hooks' in attrs and not isinstance(attrs['teardown_hooks'],
list):
            raise ValidationError('请传递正确的teardown_hooks格式:list')
        if 'request' in attrs and not isinstance(attrs['request'], dict):
            raise ValidationError('请传递正确的request格式:dict')
        return attrs
    ...
```

RequestSerializer

```
# 请求模型的序列化器
class RequestSerializer(serializers.ModelSerializer):
    method = serializers.SerializerMethodField()  # 1.自定义字段获取方法
    step_id = serializers.IntegerField(write_only=True, required=False)  # 不要求
请求入参携带此参数

    # 2.配套方法
    def get_method(self, obj):  # rest框架获取method时，会自动调用该方法
        return obj.get_method_display()  # 返回choice的displayname

    class Meta:
        model = Request  # 指定对应的模型
        fields = ['step_id', 'method', 'url', 'params', 'headers', 'json',
'data']
        # fields = '__all__'  # 显示对应模型的所有字段
        # 定义可以显示和操作的字段

    def validate(self, attrs):
        if 'params' in attrs and not isinstance(attrs['params'], dict):
            raise ValidationError('请传递正确的params格式:dict')
        if 'headers' in attrs and not isinstance(attrs['headers'], dict):
            raise ValidationError('请传递正确的headers格式:dict')
        if 'cookies' in attrs and not isinstance(attrs['cookies'], dict):
            raise ValidationError('请传递正确的cookies格式:dict')
        return attrs
```

CaseSerializer

```
# 用例
class CaseSerializer(serializers.ModelSerializer):
    config = ConfigSerializer()  # 显示指定config字段为序列化对象，REST会自动提取其值
    teststeps = StepSerializer(many=True, required=False)  # 以列表形式展示
many=True

    project_id = serializers.CharField(write_only=True)  # project_id只用于输入

    create_time = serializers.DateTimeField(format='%Y-%m-%d %H:%M:%S',
read_only=True)
    update_time = serializers.DateTimeField(format='%Y-%m-%d %H:%M:%S',
read_only=True)  # 格式化输出时间
    class Meta:
        model = Case
        fields = ['id', 'config', 'teststeps', 'desc', 'project_id',
'file_path', 'create_time', 'update_time']

    def validate(self, attrs):
        # 校验config 和 teststeps
        if 'config' in attrs and not isinstance(attrs['config'], dict):  # 如果格
式不符合要求
            raise ValidationError('请传递正确的config格式:dict')
        if 'teststeps' in attrs and not isinstance(attrs['teststeps'], list):  #
如果格式不符合要求
            raise ValidationError('请传递正确的teststeps格式:list')
        return attrs

    ...
```

用例文件输出过滤

目前生成用例文件，一些不必要的字段也显示到了文件上

```
{
    "config": {
        "project": {
            "id": 1,
            "admin": null,
            "name": "测试开发3",
            "status": 0,
            "version": "v3",
            "desc": "测试开发",
            "create_time": "2021-11-06 08:57:56",
            "update_time": "2021-11-06 08:57:56"
        },
        "name": "case003",
        "base_url": "",
        "variables": null,
        "parameters": null,
        "export": null,
        "verify": false
    },
```

如果文件中的字段不属于HR3，或者HR3对应的字段为空字符串，空列表，空字典等，则不输出到用例文件中。

因此我们可以自定义一个过滤器，过滤后的字段再输出到用例文件中（jsonfile）

模板匹配法

```
    def filter_data(self,data):
        template = {
            'config': {
                'name': str,
                'base_url': str,
                'variables': dict,
                'parameters': dict,
                'verify': bool,
                'export': list
            },
            'teststeps': [{
                'name': str,
                'variables': list,
                'extract': dict,
                'validate': list,
                'setup_hooks': list,
                'teardown_hooks': list,
                'request': {
                    'method': str,
                    'url': str,
                    'params': list,
                    'headers': dict,
                    'cookies': dict,
                    'data': dict,
                    'json': dict
                },
            }]
        }
        return self.merge_dict(template,data)
```

字典同步

```
    def merge_dict(self,left, right):
        # 覆盖左侧同类项
        for k in right:
            if k in left:
                if isinstance(left[k], dict) and isinstance(right[k], dict):
                    self.merge_dict(left[k], right[k])
                elif isinstance(left[k], list) and isinstance(right[k], list):
                    # 左右元素数量保持一致
                    item = copy.deepcopy(left[k][0])
                    left[k] = [item for a in right[k]]
                    for idx, one in enumerate(right[k]):
                        le = left[k][idx]
                        self.merge_dict(le, one)
                elif right[k]:  # 合并的条件：取交集，且right不为空(也包含空字符串，空字典和列表)
                    left[k] = right[k]
                elif not right[k]:
```

```
                    left.pop(k)
        # 去除左侧多余项
        for k in list(left.keys()):
            if k not in right:
                left.pop(k)
        return left
```

重新执行**GET**/cases/{id}/run_case/

查看生成的用例文件中，应该去除掉多余的字段

单用例执行

用例文件生成后，我们就可以采用hrun3方法来执行用例了

hr底层采用Pytest, pytest.main方法的返回值code代表不同的结果：

```
Exit code 0
All tests were collected and passed successfully
所有测试均已收集并成功通过
Exit code 1
Tests were collected and run but some of the tests failed
收集并运行了测试，但有些测试失败了
Exit code 2
Test execution was interrupted by the user
测试执行被用户中断
Exit code 3
Internal error happened while executing tests
执行测试时发生内部错误
Exit code 4
pytest command line usage error
pytest 命令行使用错误
Exit code 5
No tests were collected
未收集测试
```



```
...
from httprunner.cli import main_run

class CaseViewSet(viewsets.ModelViewSet):
    queryset = Case.objects.all()
    serializer_class = CaseSerializer

    @action(methods=['GET'],detail=True,url_path='run',url_name='run_case')
    def run_case(self,request,pk):
        case = Case.objects.get(pk=pk) #  根据id获取用例
        serializer = self.get_serializer(instance=case) # 调用序列化器
        path = serializer.to_json_file()  # 生成用例文件
        # hr3运行测试用例
        exit_code = main_run([path])  # 入参是列表=pytest参数包括用例文件路径，Pytest
命令选项
        # 只要exit_code不是0，就是执行失败了
        if exit_code != 0:
```

```
            return Response(status=status.HTTP_500_INTERNAL_SERVER_ERROR,
                            data={'error': 'failed run case', 'retcode':
exit_code})
        return Response(data={'retcode': status.HTTP_200_OK, 'msg': 'run
success'})
```

多用例执行

模型设计

多用例我们可以采用测试计划关联测试用例，然后

```
# models/task.py
from django.conf import settings
from django.db import models

from .base import CommonInfo
from .hr3 import Case
from .mgr import Environment

class Plan(CommonInfo):
    status_choice = (
        (0, '未执行'),
        (1, '执行中'),
        (2, '中断'),
        (3, '已执行'),
    )
    cases = models.ManyToManyField(Case, verbose_name='测试用例', blank=True)
    # 执行者
    executor = models.ForeignKey(settings.AUTH_USER_MODEL,
on_delete=models.DO_NOTHING, null=True, verbose_name='执行者')
    # 测试环境
    environment = models.ForeignKey(Environment, on_delete=models.SET_NULL,
null=True, verbose_name='测试环境')

    name = models.CharField('计划名称', max_length=32, unique=True)

    status = models.SmallIntegerField(choices=status_choice, default=0,
verbose_name='计划状态')

    exec_counts = models.PositiveSmallIntegerField(default=0, verbose_name='执行次
数')
```

```
# models/__init__.py
from .task import Plan  # 模块文件记得对外暴露
```

同步数据库

```
python manage.py makemigrations
python manage.py migrate
```

序列化器设计

serializers/task.py

```
class PlanSerializer(serializers.ModelSerializer):
    class Meta:
        model = Plan
        fields = '__all__'
```

```
# models/__init__.py
from .task import Plan  # 模块文件记得对外暴露
```

视图设计

```
class PlanViewSet(viewsets.ModelViewSet):
    queryset = Plan.objects.all()
    serializer_class = PlanSerializer
```

视图模块化

views/auth.py

```
"""
@author: haiwen
@date: 2021/11/9
@file: auth.py
"""
from django.contrib import auth
from rest_framework import status
from rest_framework.authentication import BasicAuthentication,
SessionAuthentication
from rest_framework.decorators import api_view, authentication_classes,
permission_classes, action
from rest_framework.permissions import IsAuthenticated
from rest_framework.response import Response

from sqtp.models import User
from sqtp.serializers import UserSerializer, RegisterSerializer, LoginSerializer

@api_view(['GET'])
@authentication_classes((BasicAuthentication,SessionAuthentication))
@permission_classes((IsAuthenticated,))
def user_list(request):
    query_set = User.objects.all()
    serializer = UserSerializer(query_set, many=True)
    return Response(serializer.data)


@api_view(['GET'])
```

```
def user_detail(request, _id):
    try:
        user = User.objects.get(pk=_id)
    except User.DoesNotExist:
        return Response(status=status.HTTP_404_NOT_FOUND)
    serializer = UserSerializer(instance=user)
    return Response(serializer.data)

#注册用户
@api_view(['POST'])
@permission_classes(())
def register(request):
    # 获取序列化器
    serializer = RegisterSerializer(data=request.data)
    if serializer.is_valid(): #根据序列器和模型字段综合检查数据是否合法
        user = serializer.register() #创建用户数据
        auth.login(request,user) # 完成用户登录状态设置
        return Response(data={'msg':'register
success','is_admin':user.is_superuser,'retcode':status.HTTP_201_CREATED},status=
status.HTTP_201_CREATED)
    return Response(data=
{'msg':'error','retcode':status.HTTP_400_BAD_REQUEST,'error':serializer.errors},
status=status.HTTP_400_BAD_REQUEST)

@api_view(['POST'])
@permission_classes(())
def login(request):
    # 获取登录信息--序列化器
    serializer = LoginSerializer(data=request.data)
    user = serializer.validate(request.data)
    if user:
        auth.login(request,user) #登录存储session信息
        return Response(data={'msg':'login
success','to':'index.html'},status=status.HTTP_302_FOUND)
    return Response(data=
{'msg':'error','retcode':status.HTTP_400_BAD_REQUEST,'error':serializer.errors},
status=status.HTTP_400_BAD_REQUEST)


@api_view(['GET'])
def logout(request):
    if request.user.is_authenticated: #如果当前用户处于登录状态
        auth.logout(request) #登出，清除session
    return Response(data={'msg':'logout
success','to':'login.html'},status=status.HTTP_302_FOUND)

#当前用户信息
@api_view(['GET'])
@permission_classes(())
def current_user(request):
    if request.user.is_authenticated: #如果当前用户处于登录状态
        # 返回当前用户信息
        serializer = UserSerializer(request.user)
        return Response(data=serializer.data)
    else:
        return Response(data={'retcode':403,'msg':'未登
录','to':'login.html'},status=403)
```

views/hr3.py

```
"""
@author: haiwen
@date: 2021/11/9
@file: hr3.py
"""
from httprunner.cli import main_run
from rest_framework import status
from rest_framework.decorators import  action
from rest_framework.response import Response

from sqtp.models import Request, Case, Step
from sqtp.serializers import RequestSerializer, CaseSerializer, StepSerializer
from drf_yasg.utils import swagger_auto_schema
from rest_framework import viewsets
from django.utils.decorators import method_decorator


# 优化3：视图集--增删改查
@method_decorator(name='list',
                  decorator=swagger_auto_schema(operation_summary='列出数据',
operation_description='列出请求数据...'))
@method_decorator(name='create',
                  decorator=swagger_auto_schema(operation_summary='增加数据',
operation_description='增加请求数据...'))
@method_decorator(name='retrieve',
                  decorator=swagger_auto_schema(operation_summary='查看详情',
operation_description='查看单个请求数据...'))
@method_decorator(name='destroy',
                  decorator=swagger_auto_schema(operation_summary='删除数据',
operation_description='删除请求数据...'))
@method_decorator(name='update',
                  decorator=swagger_auto_schema(operation_summary='更新数据',
operation_description='更新请求数据...'))
class RequestViewSet(viewsets.ModelViewSet):
    queryset = Request.objects.all()  # 数据的查询集
    serializer_class = RequestSerializer


class CaseViewSet(viewsets.ModelViewSet):
    queryset = Case.objects.all()
    serializer_class = CaseSerializer
    # 同步创建用户
    def perform_create(self, serializer):
        serializer.save(create_by=self.request.user)
    # 同步更新用户
    def perform_update(self, serializer):
        serializer.save(updated_by=self.request.user)

    @action(methods=['GET'],detail=True,url_path='run',url_name='run_case')
    # 完整的url等于/cases/<int:case_id>/run
    def run_case(self,request,pk):
        # 获取序列化器
        case = Case.objects.get(pk=pk) #根据ID获取当前用例
        serializer = self.get_serializer(instance=case)
        path = serializer.to_json_file()
```

```
        # subprocess.Popen(f'hrun {path}',shell=True)  #命令行执行法
        # HR3 API执行法
        exit_code=main_run([path])
        # 根据推出代码判断是否执行成功
        if exit_code !=0:
            return Response(data={'error':'failed run
case','retcode':exit_code},status=status.HTTP_500_INTERNAL_SERVER_ERROR)
        return Response(data={'msg':'run success','retcode':200})


class StepViewSet(viewsets.ModelViewSet):
    queryset = Step.objects.all()
    serializer_class = StepSerializer
```

views/mgr.py

```
"""
@author: haiwen
@date: 2021/11/9
@file: mgr.py
"""

# Create your views here.

from sqtp.models import  Project, Environment
from sqtp.permissions import IsOwnerOrReadOnly
from sqtp.serializers import  ProjectSerializer, \
    EnvironmentSerializer
from rest_framework import viewsets


class ProjectViewSet(viewsets.ModelViewSet):
    queryset = Project.objects.all()
    serializer_class = ProjectSerializer
    #权限
    permission_classes = (IsOwnerOrReadOnly,)


class EnvironmentViewSet(viewsets.ModelViewSet):
    queryset = Environment.objects.all()
    serializer_class = EnvironmentSerializer
    #权限
    #authentication_classes = (()) #传入空元组表示禁用全局认证
    permission_classes = (())
```

views/task.py

```
"""
@author: haiwen
@date: 2021/11/9
@file: task.py
"""
from sqtp.models import Plan
from sqtp.serializers import PlanSerializer
from rest_framework import viewsets
```

```
class PlanViewSet(viewsets.ModelViewSet):
    queryset = Plan.objects.all()
    serializer_class = PlanSerializer
    # 定义运行测试计划方法,批量运行测试用例并生成测试报告
    def run_plan(self):
        pass
```

views/**init**.py

```
"""
@author: haiwen
@date: 2021/11/9
@file: __init__.py.py
"""
from .auth import user_list,user_detail,current_user,register,login,logout
from .hr3 import CaseViewSet,RequestViewSet,StepViewSet
from .mgr import ProjectViewSet,EnvironmentViewSet
from .task import PlanViewSet
```

记得注册路由 sqtp.urls.py

```
router.register(r'plans', views.PlanViewSet)
```

视图集自定义方法回顾(补充知识)

用例运行生成的方法采用的是目前是单独设定的函数视图方法

```
#用例运行
@api_view(['GET'])
def run_case(request,_id):
    case = Case.objects.get(pk=_id) #  根据id获取用例
    serializer = CaseSerializer(instance=case) # 调用序列化器
    path = serializer.to_json_file() # 生成用例文件
    return Response({'retcode': status.HTTP_200_OK, 'msg': path})
```

这样做不能复用视图集的序列化器，而且URL不够统一，所以改查视图集中自定义方法

视图集自定义action动作

在视图集中，除了默认的动作(list() 提供一组数据,retrieve() 提供单个数据,create() 创建数据,update() 保存数据,destory() 删除数据)外，可以自定义动作

- 添加自定义动作时，需要rest_framework.decorators.action装饰器
- 在自定义动作方法上，添加@action(),接收两个参数
    - methods，该action支持的请求方式，列表传递
    - detail，action中要处理的是否是视图资源的对象（即是否通过url路径获取主键）
        - True 表示使用通过URL获取的主键对应的数据对象
        - False 表示不使用URL获取主键

```python
class CaseViewSet(viewsets.ModelViewSet):
    queryset = Case.objects.all()
    serializer_class = CaseSerializer

    @action(methods=['GET'],detail=True,url_path='run',url_name='run_case')
    def run_case(self,request,pk):
        case = Case.objects.get(pk=pk) #  根据id获取用例
        serializer = self.get_serializer(instance=case) # 调用序列化器
        path = serializer.to_json_file() # 生成用例文件
        return Response({'retcode': status.HTTP_200_OK, 'msg': path})
```