报告展示功能开发

上节课我们完成了报告的模型、序列化器和视图的创建

接下来，我们要集成报告的生成和报告的展示。

首先，报告的生成应该是在运行测试计划后就开始产生测试报告。

计划执行时生成报告

采用uuid生成不同的报告路径。

```
# view/task.py
class PlanViewSet(viewsets.ModelViewSet):
    queryset = Plan.objects.all()
    serializer_class = PlanSerializer
    ...
    # 执行测试计划--/api/plans/<int:id>/run
    @action(methods=['GET'], detail=True, url_path='run', url_name='run_plan')
    def run(self, request, pk):
        plan = Plan.objects.get(pk=pk)  # 根据id获取计划
        plan.status = 1  # 更新计划状态为执行中
        plan.save()
        # 执行关联的用例 先生成文件再去执行
        case_list = []  # 用例文件路径
        for case in plan.cases.all():
            serializer = CaseSerializer(instance=case)  # 调用序列化器
            path = serializer.to_json_file()  # 生成用例文件
            case_list.append(path)
        #报告路径采取uuid随机值
        allure_path = f'report/{uuid.uuid4()}'
        # hr3批量运行用例,
        exit_code = main_run([*case_list, f'--alluredir={allure_path}'])  #
pytest的命令，批量执行用例列表中的用例，再生成测试报告
        # 生成allure报告文件,--报告文件存储到静态文件目录下
        subprocess.Popen(f'allure generate {allure_path} -o
dist/{allure_path}',shell=True)
        # 保存报告内容到数据库

 Report.objects.create(plan=plan,path=f'{allure_path}/index.html',trigger=reques
t.user)
        # 更新计划状态
        plan.status = 3  # 更新状态：执行完成
        plan.exec_counts += 1  # 执行次数加1
        plan.save()
        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'})
```

保存报告详情

如果我们对报告的结果要求精益求精，希望能把hr3的log也保存到数据库中，保存到详情字段

首先要获取当前hr3产生的log内容。

```
# sqtp/utils.py

import os

...
# 执行前清空logs中现有的内容
def setup_logs_dir(log_path):
    empty_dir_files(log_path, 'log')

def collect_log(path):
    content_list = []
    for fi in os.listdir(path):
        with open(f'{path}/{fi}') as f:
            content_list.append(f.read())
    return '\n'.join(content_list)
```

视图部分增加测试报告保存时储存详情页：

```
    @action(methods=['GET'], detail=True, url_path='run', url_name='run_plan')
    def run(self, request, pk):
        setup_logs_dir('logs')
        ...
        # 获取报告详情
        detail = collect_log('logs')
        # 保存报告内容到数据库
Report.objects.create(plan=plan,path=f'{allure_path}/index.html',trigger=request
.user,detail=detail)
```

用例上传

目前我们已经将用例的增删改查功能集成到了平台，录制用例需要我们手动来填写用例数据。如果我们想要更有效率的录制用例，就需要用到上传功能了，我们可以批量上传用例文件，然后系统自动录入。

文件上传

在实现这个功能之前，我们先熟悉下REST框架自带的文件上传功能。

```

from rest_framework.parsers import FileUploadParser

class FileUploadView(APIView):
    parser_classes = [FileUploadParser, ]

    def put(self, request, filename, format=None):
        # 接收文件
        file_obj = request.data['file']
        if not os.path.exists('upload'):
            os.makedirs('upload')
        with open(f'upload/{filename}', 'wb') as f:
            for chunk in file_obj.chunks():
                f.write(chunk)
        # 检查文件内容
        try:
            content = loader.load_test_file(f'upload/{filename}')
            valida_case = compat.ensure_testcase_v3(content)
        except Exception as e:
            raise serializers.ValidationError(f'错误的hr3用例格式: {repr(e)}')
        # 内容导入到数据库
        serializer = CaseSerializer(data=valida_case)
        if serializer.is_valid():
            serializer.save()
        return Response({'retcode': 204, 'msg': f'{filename} uploaded'},
status=204)
```

对外暴露

```
# sqtp/views/__init__.py
from .hr3 import FileUploadView
```

```
# 注册路由
path('upload/<str:filename>/', views.FileUploadView.as_view())
```

尝试上传文件，发现文件内容错误，打开查看文件内容，发现加入了HTTP上传出现的分割符。

去除上传出现的分隔符

```
class FileUploadView(APIView):
    parser_classes = [FileUploadParser, ]
    def put(self, request, filename, format=None):
        # 接收文件
        file_obj = request.data['file']
        if not os.path.exists('upload'):
            os.makedirs('upload')
        with open(f'upload/{filename}', 'wb') as f:
            for chunk in file_obj.chunks():
                f.write(chunk)
        # 去除前三行和最后1行
        with open(f'upload/{filename}',) as f:
            lines = f.readlines()[3:][:-1]
        with open(f'upload/{filename}','w') as f:
            for line in lines:
                f.write(line)
        # 检查文件内容
        try:
            content = loader.load_test_file(f'upload/{filename}')
            valida_case = compat.ensure_testcase_v3(content)
        except Exception as e:
            raise serializers.ValidationError(f'错误的hr3用例格式: {repr(e)}')
        # 内容导入到数据库
        valida_case['project_id']= 1 #增加默认project
        serializer = CaseSerializer(data=valida_case)
        if serializer.is_valid():
            serializer.save()
        else:
            raise serializers.ValidationError(serializer.errors)
        return Response({'retcode': 204, 'msg': f'{filename} uploaded'},
status=204)
```

路由

```
path('upload/<str:filename>/', views.FileUploadView.as_view())
```

修改bug

文件上传后没有保存数据，发现是入参校验不通过。修改序列化器，改变自定义字段，曾加teststeps传参。

```
class CaseSerializer(serializers.ModelSerializer):
    project_id = serializers.CharField(write_only=True,required=False)  # 只做为入
参,非必填
    ...

    def create(self, validated_data):
        '''
        validated_data: 校验后的入参--字典形式
        '''
        # 创建config
        config_kws = validated_data.pop('config')  # 取出config参数
        project = Project.objects.get(pk=validated_data.pop('project_id'))
        config = Config.objects.create(project=project, **config_kws)  # 关联
project

        steps_kws=[]
        if 'teststeps' in validated_data:
            steps_kws = validated_data.pop('teststeps')

        # 创建用例
        file_path = f'{project.name}_{config.name}.json'  # 项目名+用例名.json
        case = Case.objects.create(config=config, file_path=file_path,
**validated_data)

        # 创建步骤
        if steps_kws:
            for step_kw in steps_kws:
                step_kw['belong_case_id']=case.id
                serializer = StepSerializer(data=step_kw)
                if serializer.is_valid(raise_exception=True):
                    serializer.save()

        return case
```

数据分页

知识点：全局分页

```
# rest框架配置
#rest框架配置
REST_FRAMEWORK={
    # 默认的渲染器
    'DEFAULT_RENDERER_CLASSES': (
        # 'rest_framework.renderers.JSONRenderer',
        # 'rest_framework.renderers.BrowsableAPIRenderer',
        'utils.renderers.MyRenderer',  #注册自定义渲染器
    ),
    #异常处理模块
    'EXCEPTION_HANDLER':'utils.exception.my_exception_handler',
    #全局认证模块--如果需要生效需要和权限模块一起配置才会生效
    'DEFAULT_AUTHENTICATION_CLASSES':(
        'rest_framework.authentication.BasicAuthentication',
        'rest_framework.authentication.SessionAuthentication',
    ),
    #全局权限模块
    'DEFAULT_PERMISSION_CLASSES':(
        'rest_framework.permissions.IsAuthenticated',
    ),

    #分页--全局配置
    'DEFAULT_PAGINATION_CLASS':'rest_framework.pagination.PageNumberPagination',
    'PAGE_SIZE': 10,
}
```

默认的全局分页返回的字段和前端需求不符，需要自定义返回字段

自定义分页

第一步: 在app目录下新建pagination.py, 添加如下代码:

```
"""
@author: haiwen
@date: 2021/7/1
@file: pagination.py
"""

from rest_framework.pagination import PageNumberPagination
from rest_framework.response import Response

class MyPageNumberPagination(PageNumberPagination):
    page_size = 5                           # default page size
    page_size_query_param = 'page_size'  # ?page_size=5&page_index=1
    page_query_param = 'page_index'      # ?page_size=5&page_index=1

    # 覆盖父类返回的数据格式
    def get_paginated_response(self, data):
        resp_data={
            'retlist' : data,
            'total': self.page.paginator.count   # 总数据量
        }
        return Response(resp_data)
```

重构渲染器，使其能返回正确的结果

```
"""
@author: haiwen
@date: 2021/6/10
@file: renderers.py
"""

# 通用返回过滤器
from rest_framework.renderers import JSONRenderer
# 自定义渲染器
class MyRenderer(JSONRenderer):
    #重构render方法
    def render(self, data, accepted_media_type=None, renderer_context=None):
        status_code = renderer_context['response'].status_code
        # 正常返回 status_code 以2开头
        if str(status_code).startswith('2'):
            #处理自定义分页
            res={'msg':'success','retcode':status_code}
            if not isinstance(data,list):
                if 'retlist' not in data:
                    res.update({'retlist':[data]})
                else:
                    res.update(data)
            else:
                res.update({'retlist': data})
            return super().render(res,accepted_media_type,renderer_context)
        else: # 异常情况
            return super().render(data,accepted_media_type,renderer_context)
```

全局分页修改为自定分页器

```
#rest框架配置
REST_FRAMEWORK={
    ...
    #分页--全局配置
    'DEFAULT_PAGINATION_CLASS':'sqtp.pagination.MyPageNumberPagination',
    'PAGE_SIZE': 10,
}
```

或者：局部视图使用自定义分页器

在基于类的视图中，你可以使用pagination_class这个属性使用自定义的分页类，如下所示：

```
from sqtp.pagination import MyPageNumberPagination

class ReportViewSet(viewsets.ReadOnlyModelViewSet):
    queryset = Report.objects.all()
    serializer_class = ReportSerializer
    pagination_class = MyPageNumberPagination
```

或者全局模式下使用自定义的分页类：

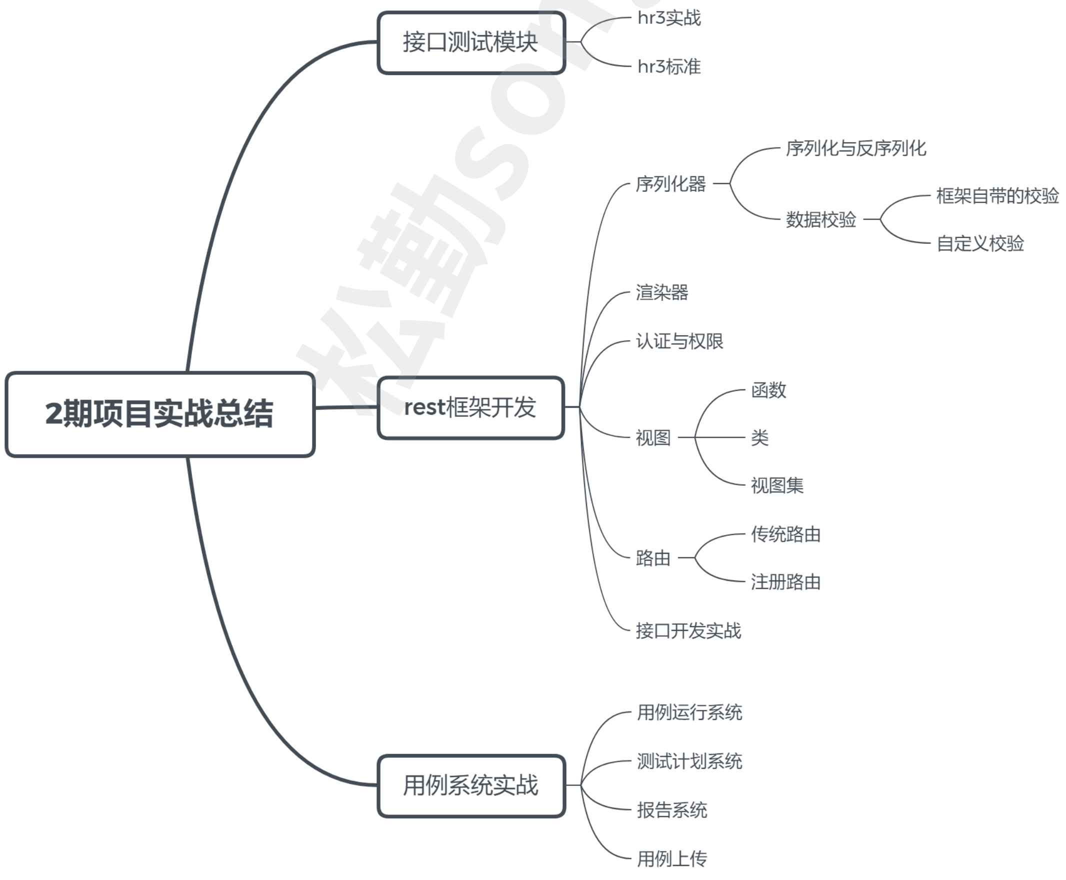
```
# rest框架配置
REST_FRAMEWORK = {
...
    # 分页全局配置
    'DEFAULT_PAGINATION_CLASS': 'sqtp.pagination.MyPageNumberPagination',
...
}
```

附录

升级报告只提供读取功能，可以直接继承ReadOnlyModelViewSet

```
class ReportViewSet(viewsets.ReadOnlyModelViewSet):
    queryset = Report.objects.all()
    serializer_class = ReportSerializer
```

实战小结



2期项目实战总结
接口测试模块
hr3实战
hr3标准
rest框架开发
序列化器
序列化与反序列化
数据校验
框架自带的校验
自定义校验
渲染器
认证与权限
视图
函数
类
视图集
路由
传统路由
注册路由
接口开发实战
用例系统实战
用例运行系统
测试计划系统
报告系统
用例上传