准备工作-模型与序列化器更新

记得更新前端文件，将本次云盘代码中的dist文件复制到项目中，原有的dist文件删除

修改Config模型

去掉name字段的唯一约束

```
name = models.CharField('名称',max_length=128,)
```

修改Request模型

为了和hr3的字段名称保持统一，更改reqeust针对step的反向查询名由`testrequest`为`request`

```
# models/hr3.py
class Request(CommonInfo):
    ...
    step = models.OneToOneField(Step, on_delete=models.CASCADE, null=True,
related_name='request')
    ...
```

修改Step模型

除此以外，步骤需要1个`sorted_no`字段用于步骤的顺序排列

现在模型添加排序字段, 并且增加联合约束和修改排序字段

```
#models/hr3.py

class Step(models.Model):
    # 同个模型中，两个字段关联同1个模型，必须指定related_name,且名字不能相同
    # 属于哪个用例
    ...
    sorted_no = models.PositiveSmallIntegerField('步骤顺序', default=1)

    class Meta:
        verbose_name = '测试步骤表'
        ordering = ['sorted_no','id']  # 先根据sorted_no排序，再根据id排序
        unique_together = ['belong_case', 'sorted_no']  # 同1个用例的步骤顺序应该是不
一样的
```

同步数据库

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

更新Case序列化器

接着上节课继续开发，前端页面完成了详情页的迭代，现在可以展示用例详情页了，为了配合前端页面展示用例详情，我们需要把用例的ID,创建时间和更新时间进行展示。

fields加上`'create_time', 'update_time'`

```
# serializers/hr3.py

class CaseSerializer(serializers.ModelSerializer):
    ...
    class Meta:
        model = Case
        fields = ['id','config', 'teststeps', 'desc', 'project_id', 'file_path',
'create_time', 'update_time']
```

更新Step序列化器

1Step序列化器增加排序字段，更新belong_case为belong_case_id，

2构造步骤代码部分 参数`testrequest`改成`request`

3更新fields字段，只展示需要展示的信息

```
# serializers/hr3.py

class StepSerializer(serializers.ModelSerializer):
    testrequest = RequestSerializer()
    belong_case_id = serializers.IntegerField(write_only=True, required=False)
 # 只写

    def create(self, validated_data):
        # 构造请求
        req_kws = validated_data.pop('request')
        req_serializer = RequestSerializer(data=req_kws)
        if req_serializer.is_valid():
            req_obj = req_serializer.save()
        else:
            raise ValidationError(req_serializer.errors)
        # 构造步骤
        step_obj = Step.objects.create(request=req_obj, **validated_data)
        return step_obj

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

Case编辑功能补充

修改common部分无法更新的问题

原case更新功能只能更新config部分内容，common部分无法更新，经过查看是 instance设置后没有调用save()同步到数据库

```
class CaseSerializer(serializers.ModelSerializer):
    ...
    # 修改用例
    def update(self, instance, validated_data):
        ...
     # 利用python反射自动赋值
        for k, v in validated_data.items():
            # 注意validated_data不要包含instance数据对象没有的字段参数
            setattr(instance, k, v)
        instance.save() # 记得调用save()
        return instance
```

重新测试更新功能，common部分更新OK

完成Step创建关联本用例

这里记得更新当天的前端文件，否则无法生成步骤序号，以及所属用例ID

修改用例部分，线删除关联的步骤，再重新创建

```
# 修改用例
    def update(self, instance, validated_data):
        '''
        instance 当前被修改的数据对象
        validated_data 校验后的入参--字典形式
        '''
        ...

        # teststeps更新
        # 删除当前用例下的所有step
        steps = instance.teststeps.all()
        for step in steps:
            step.delete()
        teststeps = validated_data.pop('teststeps')
        for step in teststeps:
            # 取出步骤关联的用例ID
            step['belong_case'] = self.instance.id
            ss = StepSerializer(data=step)
            if ss.is_valid():
                ss.save()
            else:
                raise ValidationError(ss.errors)

        ...
```

完成Request数据关联Step

发现Request数据创建了，但是并没有成功关联上Step，更改步骤序列化器的创建方法

```
# serializers/hr3.py

class StepSerializer(serializers.ModelSerializer):
    request = RequestSerializer()
    belong_case_id = serializers.IntegerField(required=False)

    def create(self, validated_data):
        req_kws = validated_data.pop('request')
        # 构造步骤
        step_obj = Step.objects.create(**validated_data)
        # 构造请求
        req_kws['step_id']=step_obj.id
        req_serializer = RequestSerializer(data=req_kws)
        if req_serializer.is_valid():
            req_obj = req_serializer.save()
        else:
            raise ValidationError(req_serializer.errors)
        return step_obj
```

同时，修改请求序列化器

```
# 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']
       #请求数据显示的内容较多，剔除不必要的字段
```

数据与用户关联

我们目前创建的数据并没有和用户产生关联，用户并不是作为序列化参数传入序列化器的，而是通过请求的user属性获取的（user不在传过来的数据中，而是通过request.user获得）

我们处理的方式是在我们的代码片段视图中重写一个`.perform_create()`方法，这样我们可以修改实例保存的方法，并处理传入请求或请求URL中隐含的任何信息。

在视图类中，添加这个方法：

```
def perform_create(self, serializer):
    serializer.save(creator=self.request.user)
```

同理，更新功能，可以增加以下函数

```
def perform_update(self, serializer):
    serializer.save(updater= self.request.user)
```

creator表示创建者字段，updater表示更新者字段，二者都要出现在数据模型中

同时记得序列化器fileds里需要增加creator和updater对应的字段,否则不生效。

以CaseSerializer为例,增加对应的字段

```
class CaseSerializer(serializers.ModelSerializer):
    config = ConfigSerializer()  # config字段就对应其序列化器返回的内容
    teststeps = StepSerializer(required=False,many=True) # read_only=True为只读参数，required=False 表示非必填，就不会校验入参 many=True展示为列表形式
    project_id = serializers.CharField(write_only=True) # 只做为入参
    create_by = UserSerializer(write_only=True,required=False)
    updated_by = UserSerializer(write_only=True,required=False)

    class Meta:
        model = Case
        fields = ['config','teststeps','project_id','desc','id','file_path','create_time','update_time','create_by','updated_by'] #序列化器定义的字段必须再此展示

    # 覆盖父类新增反方法
    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
        # 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)
        return case
```


新增项目没有管理员的问题

配合前端，入参增加admin_id字段，只写模式

```
class ProjectSerializer(serializers.ModelSerializer):
    admin_id = serializers.IntegerField(write_only=True)
    admin = UserSerializer()
    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 = Project
        fields = ['id', 'admin_id', 'admin', 'name', 'desc', 'status',
'version', 'create_time', 'update_time']
```

用例文件生成

用例数据我们采用序列化器中保存好的json数据

```
 # 用例
class CaseSerializer(serializers.ModelSerializer):
     # 生成json文件
    def to_json_file(self,path=None):
        if path is None:
            path = self.instance.file_path
        if not path.endswith('.json'):
            path = path + '.json'
        path = f'testcase/{path}'
        # 生成json文件
        content = JSONRenderer().render(self.data)
        with open(path,'wb',) as f:
            f.write(content)
        return path
```

该方法目前只会忠实的把数据库中的用例数据以json格式输出到文件上，如果json格式不符合hr3规范，也是不能生成用例的。后面我们会把用例数据的准入规范做起来。