

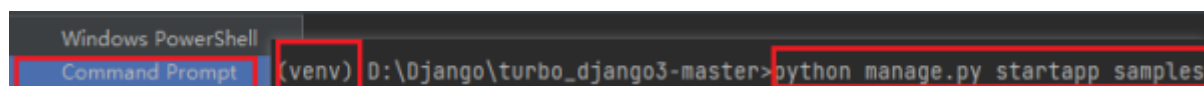
前后端开发流程

Django开发

后端的开发主要涉及到的文件有：apps、models、serializers、urls、filters、constants、views

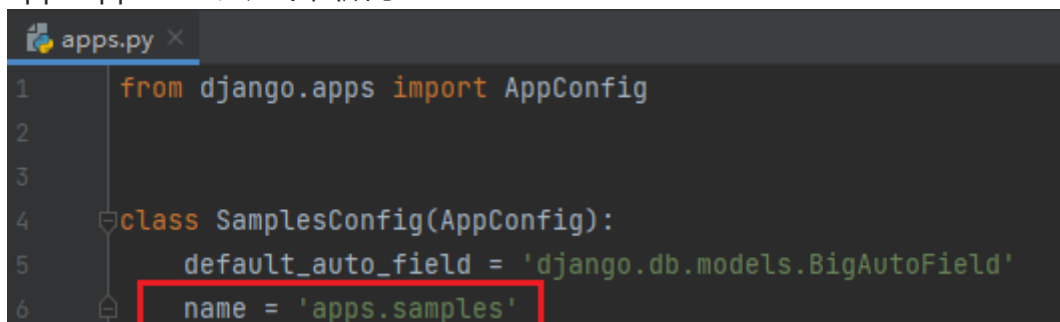
- 连接数据库。在项目的config.yaml文件下配置Mysql主数据库的连接。
- 新建app。进入Pycharm的Terminal（虚拟环境下），利用命令python [manage.py](#) [startapp](#) appname新建app。如下图所示。
新建成功后的app包含如下文件：

| | |
|------------|----------------------|
| migrations | 用于放置生成的迁移文件 |
| __init__ | Python模块初始化文件 |
| admin | 与网站后台管理相关 |
| apps | 应用程序配置文件 |
| models | 应用程序模型文件，用于与数据库交互 |
| tests | 应用程序测试文件，用于写测试代码 |
| views | 应用程序视图文件，用于接收请求、处理请求 |



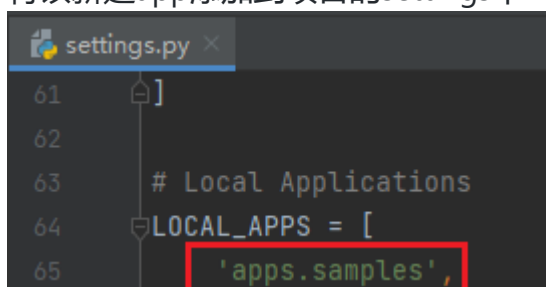
```
Windows PowerShell
Command Prompt (venv) 0:\Django\turbo_django3-master>python manage.py startapp samples
```

- 将新建的app移动到项目中的apps文件下，同时修改该新建app文件下的apps中的name为apps.appname。如下图所示



```
apps.py x
1 from django.apps import AppConfig
2
3
4 class SamplesConfig(AppConfig):
5     default_auto_field = 'django.db.models.BigAutoField'
6     name = 'apps.samples'
```

- 将该新建app添加到项目的settings中（路径：turbo>settings>LOCAL_APPS），如下图所示



```
settings.py x
61 ]
62
63 # Local Applications
64 LOCAL_APPS = [
65     'apps.samples',
```

- 添加models。可以添加多个model，有关联的model可以用外键进行关联。
 - 在Django中定义类。可以继承系统自带的models.Model，也可以继承重新封装后的APIModel（自动创建六大字段）

- 定义字段类型。模型中每一个字段都应该是某个 Field 类的实例。常用的字段类型有：

| | |
|-------------------|--|
| AutoField() | 一个自增的Integerfield(),不设置的话默认是自增id作为主键 |
| CharField() | 字符串字段，范围在254个字符之内，超出范围的话可以使用 |
| TextField() | max_length为必选项 |
| DateField() | 日期字段，使用datetime.date实例保存的日期 |
| DateTimeField() | 日期和时间字段， |
| DecimalField() | 固定精度的十进制数，max_digits：数字允许的最大位数，decimal_places：小数的最大位数。 |
| EmailField() | 字符类型且输入必须为有效的电子邮件地址 |
| IntegerField() | 整数类型 |
| ForeignKey() | 多对一关系。 |
| ManyToManyField() | 多对多关系。 |

- 定义字段选项。常用的字段选项有：

| | |
|--------------|---|
| null | 数据库字段是否可为空 |
| blank | 添加数据时是否允许为空值。与null的区别为：null是一个纯数据库级别的，而blank是表单验证级别的。 |
| choices | 作为该字段的选择 |
| default | 给字段定义默认值 |
| primary_key | 主键，对AutoField类型的字段设置主键后，就会代替来自增id列 |
| unique | 唯一性，不允许重复 |
| auto_now | 用于DateField()。每次保存或添加都会创建当前时间（可用于修改/编辑时间字段） |
| auto_now_add | 用于DateField()。每次保存或添加都会保存第一次创建的时间（可用于创建时间字段） |
| max_length | 字符串的最大长度 |

- 运行命令python [manage.py](#) [check](#) 检查model是否出错
- 运行命令python [manage.py](#) [makemigrations](#) 生成迁移文件，生成的迁移文件在migrations中
- 运行命令python [manage.py](#) [migrate](#)将model同步到数据库
- 新建serializers文件。添加serializers，在该文件中可以指定数据查询接口字段、数据新建更新字段，并在新建更新请求时对接口数据进行合法性校验并更新
 - 定义序列化器，可以继承系统的serializers.ModelSerializer，也可以继承重新封装后的APIModelSerializer（包括六大字段显示、Code Table查询、特殊字段鉴权、数据校验等功能）
 - 定义序列化器字段，常见的fields有：
 - String Field：常用的字段选项有：max_length输入的最大字符个数，min_length输入的最小字符个数，allow_blank空字符串是否为有效值，allow_null是否允许为空

| | |
|------------|--------------------|
| CharField | 字符串字段。 |
| EmailField | 验证文本输入是否为有效的电子邮件地址 |

| | |
|------------|-------------------|
| RegexField | 验证给定值是否与某个正则表达式匹配 |
|------------|-------------------|

- Numeric fields: 常用的字段选项有: max_value输入的数字不大于这个值, min_value输入的数字不小于这个值。。

| | |
|--------------|---|
| IntegerField | 整数表示 |
| FloatField | 浮点表示 |
| DecimalField | 十进制表示。max_digits数字允许的最大位数, decimal_places: 小数的最大位数。 |

- Date and time fields: 常用的字段选项有: format输出格式的字符串, input_formats可用于解析日期的输入格式的字符串列表

| | |
|---------------|--------|
| DateField | 日期表示 |
| TimeField | 时间表示 |
| DurationField | 持续时间表示 |

- Choice selection fields: 常用的字段选项有: choices有效值列表, allow_blank空字符串是否为有效值, allow_null是否允许为空

| | |
|---------------------|--|
| ChoiceField | 可以接受有限选项集中的值的字段 |
| MultipleChoiceField | 可以接受一组零个、一个或多个值的字段, 从一组有限的选选项中 中选择。 |

- Miscellaneous fields:

| | |
|-----------------------|--|
| ReadOnlyField | 只返回字段的值, 不允许修改 |
| SerializerMethodField | 只读字段。通过调用附属于序列化器类上的方法来获取其值。可用于将任何类型的数据添加到对象的序列化表示中 |
| ModelField | 可以绑定到任意模型字段的通用字段 |
| HiddenField | 不根据用户输入获取值, 而是从默认值或可调用值中获取值 |

- 根据需求做相应的数据校验, APIModelSerializer中包含的校验主要有:

| | |
|----------------------|---|
| require_fields | 必填项字段, 元组, 指定的字段在数据新建和更新的时候会进行数据 必输性校验 |
| unique_fields | 唯一性字段, 元组, 效果等同于唯一性索引。 |
| non_updatable_fields | 不可更新字段, 元组, 一般用于在特定的状态下部分数据不可修改, 配 合validate_desc使用 |
| validate_desc | 验证描述字典, 必须和non_updatable_fields配合使用 |
| validate方法 | 如果上述已经定义的字段无法实现数据校验的话, 可以重写validate方 法 (可以在重写后有选择的调用父类的validate方法) |

- 根据需求进行数据创建(create方法)和数据更新(update方法)。如果在create和update方法中有字段需要指定特殊的值；或者是需要在这两个方法中做额外的校验，可以重写该方法，并在方法中调用父类即可
- 新建filters文件。添加filters，在该文件下添加的字段，作为检索字段在前端使用，用来筛选符合检索条件的数据。

- 过滤器中常用的字段类型有：

| | |
|-----------------|--|
| CharFilter | 字符串类型 |
| BooleanFilter | 布尔类型 |
| DateTimeFilter | 日期时间类型 |
| DateFilter | 日期类型 |
| DateRangeFilter | 日期范围 |
| TimeFilter | 时间类型 |
| NumberFilter | 数值类型，对应模型中IntegerField, FloatField, DecimalField |

- 过滤器示例如下所示：

```
import django_filters
from apps.samples.models import Samples

//参数说明:field_name: 过滤字段名，一般应该对应模型中字段名；lookup_expr: 查询时所要进行的操作，和ORM中运算符一致
class SamplesFilter(django_filters.rest_framework.FilterSet): //定义序列化器
    item_no = django_filters.CharFilter(field_name='item_no', lookup_expr='contains')
    //contains: 模糊查询
    status = django_filters.CharFilter(field_name='status', lookup_expr='exact')
    //exact: 精确查询
    menu_code = django_filters.CharFilter(field_name='menu__code', lookup_expr='exact')
    valid_date = django_filters.DateFilter(field_name='valid_date', lookup_expr='gte')
    //gte: 大于等于
    invalid_date = django_filters.DateFilter(field_name='invalid_date',
        lookup_expr='lt') //lt: 小于

    class Meta:
        model = Samples //引用的模型
        fields = ['item_no', 'status', 'menu_code', 'valid_date', 'invalid_date'] //指明过滤的字段
        exclude = ['quantity'] //排除字段，不允许使用该字段进行过滤
```

- 添加views。在该文件下利用queryset检索数据，指明视图使用的序列化器、筛选器。可以根据需求重写方法，定义额外的接口方法。
 - 指明queryset，queryset属性是一个模型查询集，它指定了视图集要操作的模型实例，queryset是可迭代的
 - 返回queryset的常用的API如下所示：

| | |
|-------------------------------|---|
| <code>all()</code> | 获取所有的对象 |
| <code>filter()</code> | 过滤查询对象 |
| <code>exclude()</code> | 排除满足条件的对象 |
| <code>annotate()</code> | 使用聚合函数 |
| <code>order_by()</code> | 对查询集进行排序 |
| <code>reverse()</code> | 反向排序 |
| <code>distinct()</code> | 对查询集去重 |
| <code>values()</code> | 返回包含对象具体值的字典的QuerySet |
| <code>values_list()</code> | 与 <code>values()</code> 类似，只是返回的是元组而不是字典。 |
| <code>dates()</code> | 根据日期获取查询集 |
| <code>datetimes()</code> | 根据时间获取查询集 |
| <code>none()</code> | 创建空的查询集 |
| <code>union()</code> | 并集 |
| <code>intersection()</code> | 交集 |
| <code>difference()</code> | 差集 |
| <code>select_related()</code> | 附带查询关联对象 |
| <code>extra()</code> | 附加SQL查询 |
| <code>defer()</code> | 不加载指定字段 |
| <code>only()</code> | 只加载指定的字段 |
| <code>using()</code> | 选择数据库 |

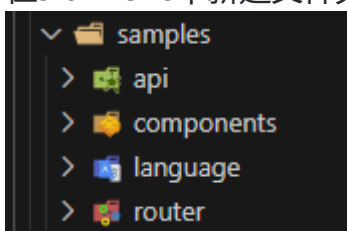
○ 视图示例如下所示：

```
class SamplesViewSet(APIModelViewSet): //定义视图
    queryset = Samples.objects.all() //检索全部数据，返回了一个包含数据库中所有对象的
    QuerySet 对象
    serializer_class = SamplesSerializer //指明视图使用的序列化器
    filter_class = SamplesFilter //指明视图使用的筛选器
```

- 新建constants文件(如果没有使用常量，该步骤可省略)，声明在该app中需要用到的常量，在使用这些常量时，需要在使用该常量的文件中引入此constants文件。
- 新建urls文件。创建路由，如果有多个画面可以创建多个路由
- 将创建的路由添加到系统文件的urlpatterns中（路径：turbo>urlpatterns>api_urlpattern）
- 运行命令python [manage.py](#) [runserver](#) 运行项目

Vue开发

- 在src>views中新建文件夹，用来盛放开发所需要的主要文件，一般涉及到的文件如下图所示。



- 新建api文件夹，在该文件夹下新建js文件，用来定义与后端进行数据交互的api方法（get、post、put、delete）。

```
import request from '@/utils/request' //导入request请求页面
export async function getSamplesListApi(params) { //定义请求获取后端数据的api方法
  var resp = await request({
    url: '/samples/list/', //后端定义的路由
    method: 'get', //接口方法
    params
  })
  return resp;
}
```

- 新建router文件夹，在该文件夹下新建js文件，创建当前开发的画面的路由、定义主菜单、子菜单。

```
import Layout from '@/layout'; //导入layout整个文件
const samplesRouter = { //声明路由
  path: '/samples', //声明当前页面的路径
  component: Layout,
  redirect: '',
  name: 'Samples',
  alwaysShow: true,
  meta: { title: '示例页面', code: 'SAMPLE-1000', icon: 'el-icon-eleme', noCache:
false }, //父菜单
  children: [ //一级子菜单
    {
      path: '',
      name: 'Samples',
      component: () => import('@/views/samples/components/SamplesList'), //导入当前菜
      单要呈现的主画面
      meta: { title: '开发示例', code: 'SAMPLE-1000', icon: 'el-icon-eleme', noCache:
false }
    }
  ]
}
```

- 将子路由放到项目的主路由下。路径为：src>router>index.js

```
import samplesRouter from '@/views/samples/router/samples.router'; //导入当前开发画面
的路由
```

- 新建components文件夹，在该文件夹下新建vue文件，用来显示前端画面
 - 新建主画面SamplesList.vue。用来呈现从后端获取的数据，同时利用vue中的父子组件完成增、删、改、查页面的呈现。

```
<template> //当前组件的DOM结构，利用vue中的表单、按钮、table等，对当前页面进行布局
和数据绑定、定义事件
```

```

</template>

<script> //主要包括两个模块
模块一：导入用到的文件
import store from '@store';
import { mapGetters } from 'vuex';
import { deleteSamplesApi, releaseSamplesApi, closeSamplesApi } from
'../api/samples.api'; //导入接口方法
import { getDropDownCodeTableListApi } from '@api/common/common' //导入接口方法
import SamplesInquire from './SamplesInquire'; //引入组件文件
import SamplesSheet from './SamplesSheet';

模块二：业务逻辑
export default {
  name: 'Samples', //name 节点为当前组件定义一个名称
  components: { //注册组件，注册需要使用的子组件
    SamplesInquire,
    SamplesSheet
  },
  data() { //vue的数据对象，可以是object也可以是function
    return { //返回vue组件渲染期间需要用到的数据
      loading: false,
      showSearch: true,
      total: 0,
      statusOptions: [],
      operateMore: false
    };
  },
  created() { //Vue实例创建之后初始化数据
    this.handleOperatePermission();
    this.handleQueryStatusOptions();
    this.getList();
  },
  computed: { //计算属性（默认使用的getter属性），根据依赖关系进行
    缓存的计算，只有在它的相关依赖发生改变时才会进行更新
    ...mapGetters(['samplesList'])
  },
  methods: { //组件中的事件处理函数
    async getList() { //获取数据
      this.loading = true;
      var resp = await store.dispatch('samples/getSamplesList',
this.queryParams); //从后端获取数据
      if (resp.success) {
        this.total = resp.data.count; //获取成功，将获取到的结果数量赋给total
        this.loading = false;
      } else {
        this.targaMessageBox(this, resp.msg); //获取失败，提示错误信息
        this.loading = false;
      }
    }
  }
}
}
</script>

```

- 新建js文件，用来封装获取后端数据的方法，路径为：src>store>modules>xxx.js。

```
import { getSamplesListApi } from '@views/samples/api/samples.api'; //引入
api接口方法
const state = {
  samplesList: []
};
const mutations = {
  SET_SAMPLES_LIST(state, data) {
    state.samplesList = data; //将data赋给List
  }
};
const actions = {
  getSamplesList({ commit }, queryParams) { //定义获取数据的方法
    return new Promise((resolve, reject) => {
      getSamplesListApi(queryParams).then(response => { //调用api接口获取数据
        const { data } = response;
        const { results } = data
        commit('SET_SAMPLES_LIST', results);
        resolve(response)
      }).catch(error => {
        reject(error);
      });
    });
  }
};
```

- 将用来盛放后台数据的list在getter.js文件中声明。

```
const getters = {
  samplesList: state => state.samples.samplesList,
}
export default getters
```

- 新建其他vue文件，在主画面中利用父子组件完成相应的增、删、改、查功能
- 新建language文件夹，在该文件夹下新建两个js文件（name.en.js、name.zh.js），用来做中英文的对照。
 - 在系统的language文件中引入两个中英文对照的js文件

```
import { samplesEn } from '../views/samples/language/samples.en';
import { samplesZh } from '../views/samples/language/samples.zh';
const locales = {
  zh: Object.assign(zh, zhLocale, samplesZh),
  en: Object.assign(en, enLocale, samplesEn)
}
```


注意事项

Django

- 新创建的app要在settings中注册定义，不然django将不会去使用这个app。
- model和表要映射好。如果改动了model层中的代码，则需要重新运行命令python [manage.py](#) makemigrations生成迁移文件，再运行python [manage.py](#) migrate将model同步到数据库
- 在views文件中要有queryset。queryset如果不指定排序字段，会默认按照id自动降序排列
- 如果业务逻辑比较复杂，不建议将业务逻辑代码放到视图层，不利于单元测试和重用代码，可以新建handlers文件用来专门处理业务逻辑
- 如果需要删除关联外键的主表数据。则需要设置model层中的外键字段的on_delete=CASCADE

Vue

- 前端页面如果有el-date-picker。则需要在该组件中加上value-format属性，否则利用日期检索数据时没有效果
- 如果需要重置表单数据。则需要正确的设置el-form-item的prop属性(为表单的model字段)
- 使用父子组件时，需要在components中注册用到的组件
- 在绑定属性，使用动态参数的时候，需要避免使用大写的字符来命名，因为浏览器会把命名强制转换为小写，其次，动态参数有一些语法约束，如空格和引号放在动态参数中是无效的
- 表单绑定时，涉及多个数据的要绑定到数组，同时需要在data中定义该数组
- vue中可以自定义指令，一般以v-xxx来命名，如：v-auth用于权限分配，v-loading用于加载

前后端交互

- 如果下拉框数据是从另一张表中获取。则需要：
 - 额外添加serializers、views、urls（路径为：apps>common）

■ 添加serializers

```
from rest_framework import serializers //引入serializers
from apps.testapp.models import Test //引入需要用到的model
class testDropdownSerializer(serializers.ModelSerializer):
    class Meta:
        model = Test //要显示在下拉框的数据的model
        fields = ('id', 'code', 'name') //要显示在下拉框中的字段
```

■ 添加views

```
from apps.common.serializers import testDropdownSerializer //引入序列化器
from apps.testapp.models import Test //引入model
from bases.viewsets import APIReadOnlyViewSet
```

```

class testDropdownViewSet(APIReadOnlyViewSet): //继承APIReadOnlyViewSet
    queryset = Test.objects.all().order_by('id') //检索全部的数据，并按照id排序
    serializer_class = testDropdownSerializer //指明视图使用的序列化器
    pagination_class = None

```

■ 添加urls

```

from django.urls import path, include
from rest_framework.routers import DefaultRouter
from apps.common.views import testDropdownViewSet //引入视图文件

common_router = DefaultRouter()
common_router.register('test', testDropdownViewSet, 'Test Dropdown List
Api') //定义显示此页面的路由

dropdown_urlpatterns = [
    path('', include(common_router.urls))
]
common_urlpatterns = [
    path('dropdown/', include(dropdown_urlpatterns))
]

```

○ 在前端定义获取该下拉框显示数据的方法

```

import request from '@utils/request';
export async function getTestDropdownViewSetApi() {
    var resp = await request({
        url: '/common/dropdown/test/',
        method: 'get'
    });
    return resp;
}

```

● 如果需要批量删除数据时。则需要

- 后端，确保Django后端继承的是已经封装完成的带有批量删除接口的APIModelViewSet
- 重写并调用父类的方法，示例如下：

```

@transaction.atomic
def perform_destroy(self, instance):
    if isinstance(instance, QuerySet):
        # 批量删除时，instance将是queryset对象
        # 如果允许批量删除，则遍历queryset进行数据校验并删除
        raise ValidationError('当前接口不允许进行批量删除操作')
    else:
        # 单个删除时，instance即模型实例
        if instance.status != OPEN:

```

```
raise ValidationError('只有开放状态下的数据可以进行删除操作')
super().perform_destroy(instance)
```

- 前端，确定正确调用后台API且传递的参数是多个主键，并且设置前端的Table表格可以多选数据。

常用的element及常用属性

| 组件 | 属性 | 作用 |
|----------------|-----------------|---|
| el-form： 表单 | rules | 表单验证规则 |
| | inline | 行内表单模式，是否一行显示 |
| | model | 表单数据对象 |
| el-form-item | prop | 表单的model字段 |
| | label | 标签文本 |
| | error | 表单验证错误信息 |
| el-row:行 | gutter | 栅格间隔，默认值为0 |
| el-col： 列 | span | 栅格占据的列数，默认值为24 |
| el-button： 按钮 | type | 表示当前按钮的类型： primary / success / warning / danger / info / text |
| | loading | 是否加载状态 |
| | disabled | 是否禁用状态 |
| | icon | 图标类名：通过el-icon-iconName直接设置即可 |
| el-input： 输入框 | value / v-model | 用来绑定值 |
| | placeholder | 输入框的占位文本 |
| | clearable | 是否可清空 |
| | disabled | 是否是禁用状态 |
| | autosize | 自适应内容高度 |
| | size | 输入框尺寸： medium / small / mini |
| el-select： 选择器 | value / v-model | 绑定值 |
| | clearable | 是否可清空 |

| 组件 | 属性 | 作用 |
|--|-----------------|---|
| | placeholder | 下拉框占位符 |
| | size | 输入框尺寸 |
| el-option: 选项 (用于: el-select 下拉框绑定, v-for遍历时需要自定值) | Label | 选项的标签 |
| | Value | 选项的值 |
| | disabled | 是否禁用该选项 |
| el-date-picker: 日期选择器 | value / v-model | 绑定值 |
| | size | 输入框尺寸 |
| | type | 显示类型 |
| | value-format | 绑定值的格式。不指定则绑定值为 Date 对象 |
| | editable | 文本框是否可输入 |
| el-table: 表格 | data | 显示的数据 |
| | border | 是否显示纵向边框 |
| el-table-column | label | 要显示的标题 |
| | prop | 显示的列的字段名 |
| | sortable | 对应列是否可排序: true, false, 'custom' (代表用户希望远程排序, 需要监听 Table 的 sort-change 事件) |
| | align | 对齐方式: left (default) /center/right |
| | width | 对应列的宽度 |
| | min-width | 对应列的最小宽度 |
| Message: 消息提示 (从顶部出现, 3 秒后自动消失) | message | 要提示的消息、文字 |
| | type | 消息类型: success/warning/info (default) /error |
| el-dropdown: 下拉菜单 | size | 菜单尺寸 |

| 组件 | 属性 | 作用 |
|-------------------------------------|----------------------|--|
| | trigger | 触发下拉的行为：hover (default) /click |
| el-dropdown-item | command | 指令 |
| | disabled | 图标类名：通过el-icon-iconName直接设置即可 |
| el-dialog：对话框（保留当前页面的状态下，弹出一个对话框） | visible | 是否显示对话框 |
| | title | Dialog标题 |
| | width | Dialog的宽度 |
| | close-on-click-modal | 是否可以通过点击 modal 关闭 Dialog |
| | destroy-on-close | 是否在关闭时销毁 Dialog 中的元素 |
| | before-close | 关闭前的回调，暂停关闭Dialog |
| Span：标签（行内元素，会在一行显示，只能添加行内元素的标签或文本） | Style | 样式 |
| el-upload：上传（通过点击或拖拽上传文件） | action（必选项） | 上传文件的地址 |
| | multiple | 是否支持多选文件 |
| | limit | 最大允许上传的文件个数 |
| | on-exceed | 文件超出最大限制个数时的钩子 |
| | on-preview | 点击文件列表中已上传的文件时的钩子 |
| | on-remove | 文件列表移除文件时的钩子 |
| | on-success | 文件上传成功时的钩子 |
| | on-error | 文件上传失败时的钩子 |
| | before-upload | 上传文件之前的钩子，若返回 false 或者返回 Promise 且被 reject，则停止上传 |

| 组件 | 属性 | 作用 |
|----------------------|---------------------|--|
| | before-remove | 删除文件之前的钩子，若返回 false 或者返回 Promise 且被 reject，则停止删除 |
| el-descriptions：描述列表 | border | 是否带有边框 |
| | column | 一行descriptions-item的数量 |
| | title | 描述列表标题 |
| | size | 列表的尺寸 |
| el-descriptions-item | label | 标签 |
| | labelClassName | 自定义标签类名 |
| | contentClassName | 自定义内容类名 |
| | contentStyle | 自定义内容样式 |
| el-pagination：分页 | total | 总条目数 |
| | page-size | 每页显示条目个数 |
| | page-count | 总页数 |
| | hide-on-single-page | 只有一页时是否隐藏 |