

Python

Python

 落霞孤鹜 · 9 篇内容 · ...

Vue3+TypeScript+Django Rest Framework 搭建个人博客（六）：踩坑汇总

 落霞孤鹜，求知若渴

1 人赞同了该文章



博客网站已经部署完成，在过程中我们解决了很多问题，曾子曰：吾日三省，这一篇就是总结我们遇到的问题，俗称：踩坑

大家好，我是 落霞孤鹜，经过几个星期的努力，我们已经把开发的博客部署上线了，这一章我们主要讲讲过程中解决的比较麻烦的问题和可以作为经验的地方。

一、软件版本

▲ 赞同 1 ▼ ● 添加评论 ↗ 分享 ★ 收藏 📁 举报

收起 ^

专栏
Python

新的解决方案，而后端更注重的是业务逻辑的完备度和系统的稳定性，因此在技术选型的时候，尽可能选择稳定版本。

1.1 Python 技术栈选型

- Python

目前 Python 的版本已经到 3.10 版本，但我没有选择最高的版本，而是选择了 3.7 版本，这里面有几个考量：

1. Django 版本对 Python 版本的支持程度
2. Django 生态中对 Python 和 Django 版本的支持程度，比如 Django Rest Framework、Django-Filter、Django-MPTT 等。
3. Django

在选型中，基于核心框架进行选择，Django 的版本我没有选择最新的 3.0，而是选择了 2.2 版本中最新版本。在这样的选择下，我这边在使用第三方包时，就不用太担心第三方包对 Python 和 Django 版本的兼容问题。

可能有人会疑惑，为什么没有选择 Python 2，我的观点是，毕竟 Python 3 才是后面的发展方向。

1.2 Vue 技术栈选型

- Vue

目前 Vue 也存在两个版本，我们这里选择了 Vue 3，虽然 Vue 2 更成熟，但考虑到前端是一个发展非常快的领域，因此尽可能选择跟随时代潮流的软件和工具。

- 构建工具

构建工具选择了 Vite，确实热编译的速度太快了，我们的博客网站由于代码量较少，几乎感觉不到有延迟，修改完立马生效。

- TypeScript

TypeScript 的选择也是前端领域里面的一种趋势，它更适合多人协作的中大型项目，由于



我们使用了 Less 作为 CSS 的预编译器，从而支持更灵活的 CSS 定义和管理。

- UI 组件 Element-Plus

是国内比较成熟的UI组件库，是Element-UI 对 Vue 3 版本的支持版本。

二、后端踩坑记

2.1 Django 跨域问题

我们的博客网站使用的是 Session 认证机制，而 Django 默认会验证跨域 cookie 和 header，因此我们需要做 3 件事情

2.1.1 去掉中间件

```
MIDDLEWARE = [  
    'django.middleware.security.SecurityMiddleware',  
    'django.contrib.sessions.middleware.SessionMiddleware',  
    'django.middleware.common.CommonMiddleware',  
    # 'django.middleware.csrf.CsrfViewMiddleware',  
    'django.contrib.auth.middleware.AuthenticationMiddleware',  
    'django.contrib.messages.middleware.MessageMiddleware',  
    'django.middleware.clickjacking.XFrameOptionsMiddleware',  
]
```

2.1.2 编写不验证 CSRF header 中间件

```
from django.utils.deprecation import MiddlewareMixin  
  
class DisableCSRF(MiddlewareMixin):  
    def process_request(self, request):  
        setattr(request, '_dont_enforce_csrf_checks', True)
```

2.1.3 加入自定义的中间件



```
'django.contrib.sessions.middleware.SessionMiddleware',
'django.middleware.common.CommonMiddleware',
# 'django.middleware.csrf.CsrfViewMiddleware',
'django.contrib.auth.middleware.AuthenticationMiddleware',
'django.contrib.messages.middleware.MessageMiddleware',
'project.middleware.request.DisableCSRF',
]
```

2.2 扩充 Django 用户表

在 Django 中默认的用户表，属性一般不能完全满足业务需求，因此会考虑扩充表属性，这里需要做几个点：

2.2.1 继承 AbstractUser

```
class User(AbstractUser, AbstractBaseModel):
    avatar = models.CharField('头像', max_length=1000, blank=True)
    nickname = models.CharField('昵称', null=True, blank=True, max_length=200)

    class Meta(AbstractUser.Meta):
        db_table = 'blog_user'
        swappable = 'AUTH_USER_MODEL'
```

2.2.2 修改认证用户表名称

在 `settings.py` 中增加一条配置

```
AUTH_USER_MODEL = 'common.User'
```

2.2.3 自定义登录和登出接口

这里通过调用 Django 自带的方法 `authenticate` 和 `login` 完成账号认证和登录，这里面实现了 session 管理的机制

登录代码：

```
password = request.data.get('password', '')

user = authenticate(username=username, password=password)
if user is not None and user.is_active:
    login(request, user)
    serializer = UserSerializer(user)
    return Response(serializer.data, status=200)
else:
    ret = {'detail': 'Username or password is wrong'}
    return Response(ret, status=403)
```

通过 django 自带的方法 auth_logout 实现登出，这里面实现了 session 失效机制。代码如下：

```
def get(self, request, *args, **kwargs):
    auth_logout(request)
    return Response({'detail': 'logout successful !'})
```

2.3 Filter 外键查询

在列表查询中，一般都会提供各种入参条件来完成查询，这里我们使用的是 Django-Filter 第三方包实现的。具体教程见：[django-filter — django-filter 2.4.0 documentation](#)

如果希望能通过外键ID作为过滤条件，比如通过分类ID查询文章列表，同时也希望在返回的列表中能展示分类名称，此时就需要做如下操作。

在 serializer 中，对外键字段不做什么调整，采用默认方式，对返回结果中的分类信息，通过 SerializerMethodField 定一个只读字段实现。

如下代码，catalog_info 通过方法定义，而原始的 catalog 字段，不做任何定义，默认即可，然后在 field 中列出。这样就可以在查询条件中，通过分类ID来过滤文章列表。

```
class ArticleListSerializer(serializers.ModelSerializer):
    tags_info = serializers.SerializerMethodField(read_only=True)
    catalog_info = serializers.SerializerMethodField(read_only=True)
    status = serializers.SerializerMethodField(read_only=True)
```

```
extra_kwargs = {
    'tags': {'write_only': True},
    'catalog': {'write_only': True},
    'views': {'read_only': True},
    'comments': {'read_only': True},
    'words': {'read_only': True},
    'likes': {'read_only': True},
    'created_at': {'read_only': True},
    'modified_at': {'read_only': True},
}
```

2.4 路由定义

通过 Rest Framework 中的 `routers.DefaultRouter()` 定义路由后，如果在 `project/urls.py` 中通过 `include` 方式引入，则需要在 app 中的 `urls.py` 中定义 app 的 `blog/urls.py` 的代码：

```
from django.urls import include, path
from rest_framework import routers

from blog import views

router = routers.DefaultRouter()
router.register('article', views.ArticleViewSet)

app_name = 'blog'

urlpatterns = [
    path('', include(router.urls)),
]
```

`project/urls.py` 的代码

```
path('', include('blog.urls', namespace='blog')),
```

所以在 `project/urls.py` 中，所有的接口前都增加 `api` 前缀。

```
path('api/', include('blog.urls', namespace='blog')),
path('api/', include('common.urls', namespace='common')),
```

2.6 Windows 和 类 Unix 系统路径兼容

在我们开发的时候，我们使用的是 windows 系统，其路径表达方式是 \，而部署的时候是放在类 unix 系统中，其路径表达方式是 /，为了兼容两种表达方式，Python 默认使用 / 方式，因此在上传文件的时候，统一成一种方式，将 \ 转换成 /。

```
def get_upload_file_path(upload_name):
    # Generate date based path to put uploaded file.
    date_path = datetime.now().strftime('%Y/%m/%d')

    # Complete upload path (upload_path + date_path).
    upload_path = os.path.join(settings.UPLOAD_URL, date_path)
    full_path = os.path.join(settings.BASE_DIR, upload_path)
    make_sure_path_exist(full_path)
    file_name = slugify_filename(upload_name)
    return os.path.join(full_path, file_name).replace('\\', '/'), os.path.join('/
```

2.7 表中自动添加创建时间和修改时间

在业务表中，我们一般都会填写时间戳字段，用来记录创建时间和最后一次修改时间，如果每一个表都要单独定义和维护，是一件非常麻烦的事情，我们通过定义抽象类实现。

Django 的 ORM 提供了两个非常有用的字段类属性 `auto_now_add` 和 `auto_now`：

- `auto_now_add` 表示新增时自动写入当前时间
- `auto_now` 表示修改时自动写入当前时间

```
class AbstractBaseModel(models.Model):
    creator = models.IntegerField('creator', null=True)
```

```
class Meta:  
    abstract = True
```

然后所有的模型类继承这个抽象类，就可以自动完成创建时间和修改时间的维护。

2.8 自动记录创建人和修改人

在模型中我们自动完成了创建时间和修改时间的维护，那么创建人和修改人可以通过定义公共的 `ViewSet` 类方法完成。

通过重写 `perform_update` 和 `perform_create` 方法，然后所有业务类通过混入（Python 可以多继承）继承的方式，实现自动维护这两个字段的信息。

```
class BaseViewSetMixin(object):  
    def perform_update(self, serializer):  
        user = self.fill_user(serializer, 'update')  
        return serializer.save(**user)  
  
    def perform_create(self, serializer):  
        user = self.fill_user(serializer, 'create')  
        return serializer.save(**user)  
  
    @staticmethod  
    def fill_user(serializer, mode):  
        """  
        before save, fill user info into para from session  
        :param serializer: Model's serializer  
        :param mode: create or update  
        :return: None  
        """  
        request = serializer.context['request']  
  
        user_id = request.user.id  
        ret = {'modifier': user_id}  
  
        if mode == 'create':  
            ret['creator'] = user_id
```

pass

```
class ArticleViewSet(BaseViewSetMixin,
                     mixins.CreateModelMixin,
                     mixins.RetrieveModelMixin,
                     mixins.UpdateModelMixin,
                     mixins.DestroyModelMixin,
                     GenericViewSet):
    queryset = Article.objects.all()
    serializer_class = ArticleSerializer

    def perform_create(self, serializer):
        extra_infos = self.fill_user(serializer, 'create')
        extra_infos['author'] = self.request.user
        serializer.save(**extra_infos)

    def filter_queryset(self, queryset):
        queryset = super(ArticleViewSet, self).filter_queryset(queryset)
        if self.is_reader():
            queryset = queryset.exclude(status=Constant.ARTICLE_STATUS_DRAFT).exclude(
                status=Constant.ARTICLE_STATUS_DELETED)
        return queryset

    def perform_destroy(self, instance: Article):
        instance.status = Constant.ARTICLE_STATUS_DELETED
        instance.save()

    def retrieve(self, request, *args, **kwargs):
        instance: Article = self.get_object()
        serializer = self.get_serializer(instance)
        if self.is_reader():
            instance.views += 1
            instance.save()
        return Response(serializer.data)
```

三、前端踩坑记

 赞同 1 添加评论 分享 收藏 举报收起 



这个文件的配置点不多，但却非常重要，包括后端接口代理地址，BASE URL 等。

1. 代理地址配置

有几个点需要注意：

proxy 下出现的 key 是我们在调试时，前端页面访问后端接口的时候，通过 src/api/index.ts 中配置的 URL 前缀，增加了 api 前缀，然后通过 rewrite 规则，决定对该前缀是否做替换，这里自动增加 api 前缀的原因是因为 Vite 自己本身也是一个静态资源服务器，为了区分请求是通过 vite 代理的静态资源还是后端接口而做的处理。

vite.config.ts 核心代码如下：

```
server: {
    host: "localhost",
    port: 3000,
    proxy: {
        '/api': {
            target: 'http://localhost:8000/',
            changeOrigin: true,
            ws: false,
            rewrite: (pathStr) => pathStr.replace('/api', '/api'),
            timeout: 5000,
        },
        '/upload': {
            target: 'http://localhost:8000/',
            changeOrigin: true,
            ws: false,
            rewrite: (pathStr) => pathStr.replace('/api', ''),
            timeout: 5000,
        },
    },
}
```

src/api/index.ts 核心代码：

```
const request = axios.create({
    baseURL: import.meta.env.MODE !== 'production' ? '/api' : '',
})
```

▲ 赞同 1

▼

● 添加评论

↗ 分享

★ 收藏

⚑ 举报

收起 ^



在配置服务器地址的时候，Vite 提供的是 `base` 属性。这里有两种配置方式 `base: '/'` 和 `base: './'`，我们配置的是第一种，这里取决于我们使用的路由模式

- 如果使用的是 History 模式，则使用第一种
- 如果使用的是 Hash 模式，则使用第二种

`base: '/',`

1. Vue 插件配置

需要 Vite 支持 Vue，通过引入Vue插件方式实现，核心代码：

```
plugins: [  
  vue(),  
,
```

3.1.2 package.json

如果要使用 Vite，需要在这个文件里面配置 `scripts`。然后才能在命令行中使用 `yarn dev` 和 `yarn build` 命令。核心代码如下：

```
"scripts": {  
  "dev": "vite",  
  "build": "vue-tsc --noEmit && vite build",  
  "serve": "vite preview"  
,
```

3.1.3 vite-evn.d.ts

文件位置： `src/vite-evn.d.ts`，核心代码：

```
/// <reference types="vite/client" />
```

3.2 TypeScript 配置

3.2.1 tsconfig.json

核心代码：

```
{  
  "compilerOptions": {  
    "target": "esnext",  
    "module": "esnext",  
    "moduleResolution": "node",  
    "strict": true,  
    "jsx": "preserve",  
    "sourceMap": true,  
    "resolveJsonModule": true,  
    "esModuleInterop": true,  
    "lib": ["esnext", "dom"]  
  "include": ["src/**/*.ts", "src/**/*.d.ts", "src/**/*.tsx", "src/**/*.vue"]  
}
```

3.2.2 package.json

在 scripts 部分的 build 命令中，主要是在编译前先基于 TypeScript 语法进行检测，如果有语法不正确的地方，则编译不通过，用这种方式可以保证在运行之前发现尽可能多的问题。

```
"scripts": {  
  "dev": "vite",  
  "build": "vue-tsc --noEmit && vite build",  
  "serve": "vite preview"  

```

3.3 懒加载

3.3.1 图片懒加载

我们实现的方式：

1、就是创建一个自定义属性`data-src`存放真正需要显示的图片路径，而`img`自带的`src`放一张大小为通用的图片路径。2、当页面滚动直至此图片出现在可视区域时，用取到该图片的`data-src`的值并赋值各`src`，然后将`data-has-lazy-src`设置为`true`，表示已经加载过图片。

TypeScript 核心代码：

```
const viewHeight = window.innerHeight || document.documentElement.clientHeight;
const lazyload = throttle(() => {
    const imgs = document.querySelectorAll("#list .item img");
    let num = 0;
    for (let i = num; i < imgs.length; i++) {
        let distance = viewHeight - imgs[i].getBoundingClientRect().top;
        let imgItem: any = imgs[i];
        if (distance >= 100) {
            let hasLaySrc = imgItem.getAttribute("data-has-lazy-src");
            if (hasLaySrc === "false") {
                imgItem.src = imgItem.getAttribute("data-src");
                imgItem.setAttribute("data-has-lazy-src", "true");
            }
            num = i + 1;
        }
    }
}, 1000);

onMounted(() => {
    window.onscroll = () => {
        if (getScrollTop() + getWindowHeight() > getDocumentHeight() - 100) {
            if (state.isLoading === false && state.isLoadEnd === false) {
                console.info("222");
                handleSearch();
            }
        }
    };
    document.addEventListener("scroll", lazyload);
    handleSearch();
});
```

自定义。

```

    import("../views/client/ArticleDetail.vue")
},
```

这里的组件不是通过文件头部的 `import` 方式导入，而是在具体需要的地方通过匿名函数导入，从而在需要展示该页面的时候开始加载，从而第一次网络请求的流量。

3.4 组件通信

在单页应用中，组件通信是一个非常重要的功能，也很复杂，有各种解决方案。我这里主要介绍一下在博客开发中用到的几种方案。

3.4.1 父子通信

在博客的开发过程中，有很多地方都出现了父组件的数据传递给子组件中，也是单页开发中最常见的场景，实现方案有两种：

▲ 赞同 1 ▾ ● 添加评论 ↗ 分享 ★ 收藏 📣 举报

收起 ^

```
<UserDetail
  :user-id="state.userId"
  :visible="state.showDialog"
  @close="state.showDialog = false"
/>
```

- **slot传值**

通过在子组件中标记slot和名称，占位，在父组件中引用子组件时，在子组件的 <></> 中，通过 `<template v-slot:slot_name>` 的方式定义 slot 内容，此时需要传递到子组件 slot 的数据，可以直接在父组件中直接赋值。比如下面代码中的 loading 属性等

```
<template v-slot:footer>
  <div class="dialog-footer">
    <el-button
      v-if="isLogin"
      :loading="state.btnLoading"
      type="primary"
      @click="handleOk"
    >
      登录
    </el-button>
    <el-button
      v-if="isRegister"
      :loading="state.btnLoading"
      type="primary"
      @click="handleOk"
    >注册
    </el-button>
  </div>
</template>
```

3.4.2 子父通信

- **回调函数**

子组件向父组件通信，有两种，一种是通过父组件提供回调函数，在属性中传入函数，在子

专栏
Python

- **\$emit 事件触发**

父组件中，通过 `@close` 定义一个事件，并传入一个定义好的函数 `handleCloseDrawer`

```
<EditArticle
  :article-id="state.articleId"
  :visible="state.showDrawer"
  @close="handleCloseDrawer"
/>
```

子组件中，定义 `emits`

```
emits: ["close"],
```

在需要触发事件的时候，调用 `context.emit('close', param)` 或者 `this.$emit('close', param)`，这里的 `param` 是需要传递的数据。

```
const saveArticle = async () => {
  try {
    state.loading = true
    if (state.catalogs.length) {
      state.article.catalog = state.catalogs[state.catalogs.length - 1]
    }
    if (props.articleId) {
      await remoteSaveArticle('put', state.article)
    } else {
      await remoteSaveArticle('post', state.article)
    }
    state.loading = false
    context.emit('close', true)
  } catch (e) {
    state.loading = false
  }
}
```

3.4.3 跨级及兄弟组件通信

1. store 定义

```
export const store = createStore<State>({
  state() {
    return {
      navIndex: '1',
    }
  },
  mutations: {
    setNavIndex(state: object | any, navIndex: string) {
      state.navIndex = navIndex
    },
  },
})
```

1. 需要获取数据的组件中通过 computed 获取

```
computed: {
  navIndex() {
    const store = useStore(StateKey);
    return store.state.navIndex;
  },
}
```

1. 其他组件通过 mutations 中定义的方法改变 store 中的值

```
store.commit(SET_NAV_INDEX, "-1");
```

一旦这个值发生变化，通过 computed 定义的属性，就会同步发生变化，从而实现组件间的通信，在博客网站中，我们的用户信息也是这么使用的。

1. 需要获取数据的组件也可以通过 watch 监控

```
watch: {
  "$store.state.articleParams": {
    handler(val: any, oldVal: any) {
      this.state.params.tags = val.tags;
    }
  }
}
```

[▲ 赞同 1](#)

[添加评论](#)
[分享](#)
[收藏](#)
[举报](#)
[收起 ^](#)

```
  },
  },
},
```

这种是当 state 发生变化时，可以获取到变换前和变化后的值，通过 handler 做更多的处理，完成该组件处理的逻辑。博客网站中我们在文章列表中点击标签筛选文章列表，就是通过这种方式实现的

watch 中可以监控的对象有很多，可以参考 Vue 官网介绍：[计算属性和侦听器 | Vue.js \(vuejs.org\)](#)

```
watch: {
  '$props.visible': {
    handler(val, oldVal) {
      if (val != oldVal) {
        this.state.visible = val
      }
    }
  },
  '$route: {
    handler(val: any, oldVal: any) {
      this.routeChange(val, oldVal);
    },
    immediate: true,
  },
  '$route.path': {
    handler(val, oldVal) {
      if (val !== oldVal && [/admin/tag"].includes(val)) this.handleSearch();
    },
    deep: true,
  },
},
```

3.5 css 样式穿透

```
//抽屉//去掉element-ui的drawer标题选中状态
:deep(:focus){
  outline: 0;
}
```

这里需要注意几点，一般我们在定义一个组件的样式时，会设置 `<style lang="less" scoped>`，这样会导致该样式穿透无效果，因此还需要做如下调整

1. 在需要调整组件样式的地方，将组件包裹在一个 `div` 中
2. 给这个 `div` 定义一个 `class`
3. 在 `style` 中通过 `global` 穿透

```
.parent-div{
  :global(.el-card__body){
    margin: 0;
    padding: 0;
  }
}
```

也可以设置 `<style lang="less">`，去掉 `scoped`，然后使用 `:deep` 方式穿透。我这里的语法是基于 vue 3 和 less，如果使用其他的 css 编译器，请自行搜索验证。

3.6 刷新 404 问题

这个问题我们在部署篇已经介绍，这里贴一下 Nginx 的配置，关键的是 `if` 判断。

```
location / {
  root ~/blog/frontend/dist;
  index index.html index.htm;
  if (!-e $request_filename) {
    rewrite ^/(.*) /index.html last;
    break;
}
```

Vue3+TypeScript+Django Rest Framework 搭建个人博客（五）：网站部署



落霞孤鹜，求知若渴

4 人赞同了该文章



博客网站开发完成后，最后一棒是将博客网站部署到公网服务器上，提供域名访问，能通过搜索引擎搜索到的博客网站才是真正的博客网站。

大家好，我是 落霞孤鹜， 经过几个星期的努力， 我们已经完成了博客的开发。这一章我们开始开始部署博客网站， 通过公网服务器和域名访问我们的博客网站。

一、准备工作

为了能实现我们的网站能通过域名访问，我们需要具备几个条件：

1. 购买云服务器，这里可以自行选择购买阿里、腾讯、华为、百度，各个平台都有自己特定

▲ 赞同 1 ▼ ● 添加评论 ↗ 分享 ★ 收藏 📁 举报

收起 ^

4. 购买域名证书，只有具备了证书后，通过域名访问才会不被浏览器提示存在安全问题。
5. 对网站进行备案，目前备案需要进行两次，一次是在工信部的平台进行 ICP 备案，通过这个备案后，还需要完成公安联网备案。

二、网站部署

在完成了准备工作后，正式开始我们的网站部署。

2.1 Python

由于我们的后端代码是 Python 编写的，所以在服务器上部署的时候，需要安装 Python 运行环境。

这里的安装方式，我采用官网的教程，版本是3.7。教程地址：docs.python.org/zh-cn/3...

安装完成后在命令行中运行如下命令验证是否安装成功

```
python -V  
pip -V
```

2.2 代码发布

2.2.1 前端代码

2.2.1.1 编译

由于我们采用的是 Vue 和 TypeScript 编写的前端，因此在部署之前，需要编译成原生 Javascript 代码。

在前端代码根目录下，执行如下命令

```
yarn build
```

执行完成后，会在代码路径下生成 dist 文件夹，里面是编译后的代码，文件结构如下图



在服务器中，Home 目录下，创建一个文件夹 blog，在blog下创建文件夹 frontend。

```
cd ~  
mkdir blog  
cd blog  
mkdir frontend
```

然后将本地 dist 文件夹下的前端文件通过FTP工具上传到 ~/blog/frontend 路径下，前端代码发布完成。

2.2.2 后端代码

由于 Python 是脚本语言，所以代码本身不需要经过编译，直接将源代码上传到服务器对应路径即可。

2.2.2.1 依赖安装

Python 的运行服务器，我们采用 uwsgi，因此需要先在服务器上安装依赖，执行如下命令

```
pip install uwsgi
```

安装完成后，通过命令验证是否安装成功

```
uwsgi --version
```

2.2.2.2 编写 uwsgi 配置文件

在后端代码项目路径下新建文件 uwsgi.ini，我们通过socket的方式连接后端接口，配置信息如下：

```
[uwsgi]  
socket = 127.0.0.1:8000  
# 可以理解为此文件的绝对路径  
chdir = ~/blog/backend/  
# wsgi与chdir的相对路径
```

专栏
Python

```
pidfile = ~/blog/backend/uwsgi.pid
master = True
```

在后端代码项目路径下新建文件 uwsgi.param , 文件内容如下

```
uwsgi_param QUERY_STRING      $query_string;
uwsgi_param REQUEST_METHOD    $request_method;
uwsgi_param CONTENT_TYPE      $content_type;
uwsgi_param CONTENT_LENGTH    $content_length;

uwsgi_param REQUEST_URI       $request_uri;
uwsgi_param PATH_INFO          $document_uri;
uwsgi_param DOCUMENT_ROOT     $document_root;
uwsgi_param SERVER_PROTOCOL   $server_protocol;
uwsgi_param REQUEST_SCHEME    $scheme;
uwsgi_param HTTPS              $https if_not_empty;

uwsgi_param REMOTE_ADDR        $remote_addr;
uwsgi_param REMOTE_PORT        $remote_port;
uwsgi_param SERVER_PORT         $server_port;
uwsgi_param SERVER_NAME         $server_name;
```

2.2.2.3 上传代码

在服务器的 ~/blog 文件夹下，创建 backend 文件夹

```
cd ~/blog
mkdir backend
```

通过 FTP 工具将后端代码上传到 ~/blog/backend 文件夹下，包括刚刚创建的 uwsgi.ini 文件然后在backend文件夹下，创建 logs 文件夹，用来记录日志

```
mkdir logs
```

至此，后端代码已经部署到位。

▲ 赞同 1

▼

● 添加评论

↗ 分享

★ 收藏

☞ 举报

收起 ^

专栏
Python

```
uwsgi --ini ~/blog/backend/uwsgi.ini
```

测试可以通过日志文件查看启动后的服务器日志

```
tail -f ~/blog/backend/logs/uwsgi.log
```

停止命令如下：

```
uwsgi --stop ~/blog/backend/uwsgi.pid
```

后期如果后端代码更新，则需要先停止 uwsgi 服务后，再启动代码的更新。

2.3 Nginx

在当前的网站部署中，Nginx作为反向代理服务器部署网站是主流操作，也已经是成熟的网站部署方案。

Nginx在网站部署中，一般会有以下几个作用：

1. **反向代理**，通过代理内网服务器，从而让外网的客户端访问到内网服务器提供的能力和数据。
2. **负载均衡**，分担后端服务器的压力
3. **HTTP服务器**，Nginx本身也是一个静态资源的服务器，且在处理静态资源的请求和响应，比老牌的 Web 服务器更优秀，性能更好。通过 Nginx 实现动态和静态的分离部署。

2.3.1 安装

网上有很多教程，我这里提供一个参考教程 [Linux下nginx的安装 - 沽酒尽 - 博客园 \(cnblogs.com\)](#)

2.3.2 证书文件

在前面的准备工作中，有提到域名证书，此时将域名证书提供商平台上提供的证书文件下载下来，解压后，将 Nginx 的证书上传到服务器的 Nginx 安装路径中的 conf 文件下。

▲ 赞同 1 ▾

● 添加评论

↗ 分享

★ 收藏

⚑ 举报

收起 ^

2.3.3.1 静态文件代理

这里有两部分静态文件，一部分是前端代码文件，一部分是通过上传接口获得的媒体文件

```
location / {
    root ~/blog/frontend/dist;
    index index.html index.htm;
    if (!-e $request_filename) {
        rewrite ^/(.*) /index.html last;
        break;
    }
}
location /upload/ {
    root ~/blog/backend/;
    expires 24h;
}
```

2.3.3.2 后端接口代理

在 location 里面有很关键的两行配置 uwsgi_pass blog 和 include
~/blog/backend/uwsgi.param，明确的是通过 uwsgi 的方式代理 Python 的接口

```
# server
upstream blog{
    server 127.0.0.1:8000;
}

location /api/ {
    uwsgi_pass blog;
    include ~/blog/backend/uwsgi.param;
    proxy_set_header X-Forwarded-For $proxy_add_x_forwarded_for;
    proxy_set_header Host $host;
    proxy_set_header accept-encoding 'gzip, deflate';
    proxy_set_header content-type 'application/json';
    proxy_set_header X-Real-IP $remote_addr;
    proxy_set_header authorization $http_authorization;
    proxy_set_header accept '*/*';
    proxy_set_header x-bce-date $http_x_bce_date;
}
```

知乎

专栏
Python

配置 Gzip 压缩的原因是在前后端分离的场景下，网站首屏需要加载的内容很多，通过缩小网络传输过程中的文件大小，从而加快首屏渲染时的静态文件加载。

```
# Gzip
    gzip on;
    gzip_min_length 1k;
    gzip_comp_level 3;
    gzip_types text/plain application/javascript application/x-javascript text/css;
    gzip_vary on;
    gzip_disable "MSIE [1-6]\.";
    gzip_buffers 32 4k;
    gzip_http_version 1.0;
```

2.3.3.3 域名配置

```
server {
    listen 80;
    server_name www.longair.cn;
    rewrite ^(.*)$ https://{$server_name}$1 permanent;
}

server {
    listen      443 ssl;
    server_name www.longair.cn;
    root html;
    index index.html index.htm;
}
```

2.3.3.3 Https 证书文件配置

```
ssl_certificate      /etc/nginx/longair.cn_nginx/server.crt;
ssl_certificate_key  /etc/nginx/longair.cn_nginx/server.key;
ssl_session_timeout  5m;
ssl_ciphers ECDHE-RSA-AES128-GCM-SHA256:ECDHE:ECDSA:RSA:AES:HIGHS:!NULL:!aNULL:!MD5:!ADH;
ssl_protocols TLSv1 TLSv1.1 TLSv1.2;
ssl_prefer_server_ciphers on;
```

[▲ 赞同 1](#)[添加评论](#)[分享](#)[收藏](#)[举报](#)[收起 ^](#)

在访问 80 端口时，重定向到 433 端口中。

```
server {
    listen 80;
    server_name www.longair.cn;
    rewrite ^(.*)$ https://{$server_name}$1 permanent;
}
```

2.3.3 博客文章详情跳转404问题配置

由于我们使用 Vue 框架编写的前端代码，路由模式采用的是 `history`，这种模式的路由可读性强，但文章详情页面的路由如果通过新开页面的方式，会出现404的问题，因此我们需要对这种情况做处理，配置原理是，访问的路径如果不是文件结尾时，则自动重定向到首页，进入到 `index.html` 后就可以通过路由变动进入到单页应用的路由，从而正确渲染详情页面。Vue-router 官网文档：[HTML5 History 模式 | Vue Router \(vuejs.org\)](#)

```
location / {
    root ~/blog/frontend/dist;
    index index.html index.htm;
    if (!-e $request_filename) {
        rewrite ^/(.*) /index.html last;
        break;
    }
}
```

2.3.4 完整配置

完整配置如下：

```
user root;
worker_processes 1;

events {
    worker_connections 1024;
}

http {
```

```
client_max_body_size 200M;

# Gzip
gzip on;
gzip_min_length 1k;
gzip_comp_level 3;
gzip_types text/plain application/javascript application/x-javascript text/css;
gzip_vary on;
gzip_disable "MSIE [1-6]\.";
gzip_buffers 32 4k;
gzip_http_version 1.0;

# server
upstream blog{
    server 127.0.0.1:8000;
}

# blog http
server {
    listen 80;
    server_name www.longair.cn;
    rewrite ^(.*)$ https://{$server_name}$1 permanent;
}

server {
    listen *:80;
    listen *:443 ssl;
    listen [::]:80;
    listen [::]:443 ssl;
    server_name longair.cn;
    ssl_certificate      /etc/nginx/longair.cn_nginx/server.crt;
    ssl_certificate_key  /etc/nginx/longair.cn_nginx/server.key;
    return 301 https://www.longair.cn$request_uri;
}

# blog https
server {
    listen      443 ssl;
    server_name www.longair.cn;
    root html;
```

[▲ 赞同 1](#)[▼](#)[添加评论](#)[分享](#)[收藏](#)[举报](#)[收起 ^](#)

知乎

专栏
Python

```
ssl_session_timeout 5m;
ssl_ciphers ECDHE-RSA-AES128-GCM-SHA256:ECDHE:ECDSA:HIGH:!NULL:!aNULL:
ssl_protocols TLSv1 TLSv1.1 TLSv1.2;
ssl_prefer_server_ciphers on;

location / {
    root ~/blog/frontend/dist;
    index index.html index.htm;
    if (!-e $request_filename) {
        rewrite ^/(.*) /index.html last;
        break;
    }
}
location /upload/ {
    root /blog/backend/;
    expires 24h;
}
location /api/ {
    uwsgi_pass blog;
    include ~/blog/backend/uwsgi.param;
    proxy_set_header X-Forwarded-For $proxy_add_x_forwarded_for;
    proxy_set_header Host $host;
    proxy_set_header accept-encoding 'gzip, deflate';
    proxy_set_header content-type 'application/json';
    proxy_set_header X-Real-IP $remote_addr;
    proxy_set_header authorization $http_authorization;
    proxy_set_header accept '*/*';
    proxy_set_header x-bce-date $http_x_bce_date;
}

error_page 500 502 503 504 /50x.html;
location = /50x.html {
    root html;
}
location = /50x.html {
    root ~/blog/frontend/dist;
}
}

}
```

▲ 赞同 1



● 添加评论

↗ 分享

★ 收藏

☞ 举报

收起 ^

3.1 Nginx的维护

3.1.1 启动命令

```
/usr/local/nginx/sbin/nginx -s start
```

3.1.2 停止命令

```
/usr/local/nginx/sbin/nginx -s stop
```

3.3.3 重启命令

```
/etc/nginx/sbin/nginx -s reload
```

3.3.4 日志查看命令

```
tail -f /etc/nginx/logs/access.log  
tail -f /etc/nginx/logs/error.log
```

3.2 UWSGI的维护

3.2.1 启动命令

```
uwsgi --ini ~/blog/backend/uwsgi.ini
```

3.2.2 停止命令

```
uwsgi --stop ~/blog/backend/uwsgi.pid
```

3.3.3 日志查看命令

四、关于数据库

在开发阶段，我们采用的是非常方便的sqlite数据库，但是如果当网站部署到服务器上以后，再采用sqlite就会出现很多问题，比如数据安全问题，数据更新问题等，因此我们在博客部署上线时，需要切换成 Mysql 数据库。

4.1 Mysql 安装

我使用的5.7版本，安装操作，请依据自己的操作系统参考官网教程安装。[MySQL :: MySQL 5.7 Reference Manual :: 2 Installing and Upgrading MySQL](#)

4.2 数据库搭建

在服务器上，通过命令行完成数据库的搭建。

4.2.1 root登录

```
mysql -uroot -p
```

然后输入密码，回车，登录进入数据库。

4.2.2 创建数据库

1. 创建数据库

```
create database blog character set utf8mb4;
```

1. 创建用户

```
create user 'blog'@'%' identified by '12345678';
```

1. 用户赋权

```
grant all privileges on blog.* to 'blog'@'%';
```

4.3.1 生产和开发环境隔离

在开发阶段，通过 sqlite 数据库，调试方便，访问速度快，而生产环境上，使用 Mysql 会更安全。所以我们希望开发和生产环境使用不同的数据库配置。

因此我们通过在系统中的环境变量来判断是生产环境还是测试环境，此种方式代码可以保持一套，部署时不用做任何调整。

同时需要调整允许访问的 host 信息。

4.3.2 数据库密码保护

由于我们的代码可能会通过代码仓库管理，如果是 Github 等仓库，可以会泄露自己的数据库密码，因此我们这里通过单独的配置文件来记录数据库账号信息，然后在Python中通过读取数据库账号配置文件来获取信息，从而实现数据库密码保护。

4.3.3 配置

4.3.3.1 project/settings.py

```
ALLOWED_HOSTS = ['www.longair.cn', '127.0.0.1', 'localhost', ]  
  
ENV_PROFILE = os.getenv("PYTHON_ENV")  
  
if ENV_PROFILE == "production":  
    import configparser  
  
    cf = configparser.ConfigParser()  
    cf.read('/blog/db.cnf') # 这个文件的内容在下一步会创建，至于路径是哪里都可以，只要[]  
    DATABASES = {  
        'default': {  
            'ENGINE': 'django.db.backends.mysql',  
            'NAME': cf.get('db', 'database'),  
            'USER': cf.get('db', 'user'), # 从配置文件中获取数据库用户名  
            'PASSWORD': cf.get('db', 'password'), # 从配置文件中获取数据库密码  
            'HOST': 'localhost',  
            'PORT': '3306'  
    }
```



```

DATABASES = {
    'default': {
        'ENGINE': 'django.db.backends.sqlite3',
        'NAME': os.path.join(BASE_DIR, 'data/db.sqlite3'),
    }
}
DEBUG = True

```

4.3.3.2 配置环境变量

通过修改环境变量管理文件，设置环境变量，输入命令 `vi ~/.bashrc`，在里面添加一行：

`export PYTHON_ENV=production`，然后按 `esc`，输入：`:wq` 保存

最后输入 `source ~/.bashrc`

4.3.3.3 增加数据库账号配置文件

在 `~/blog` 下新增文件 `vi db.cnf`，然后增加如下内容：

```

[db]
database = blog
user = blog
password = 12345678
default-character-set = utf8

```

数据库、账号、密码和上面的数据库信息保持一致。

恭喜你，至此，博客网站部署完成，可以通过域名来访问你的博客网站了。

下一篇我将总结一下在整个博客开发过程中遇到的问题和解决方案。

发布于 2021-08-25 21:06

▲ 赞同 4 ▾ ● 1 条评论 ↗ 分享 ★ 收藏 📌 举报

收起 ^

Vue3+TypeScript+Django Rest Framework 搭建个人博客（四）：博客页面

▲ 赞同 1 ▾ ● 添加评论 ↗ 分享 ★ 收藏 📌 举报

收起 ^

个页面可以查找文章，浏览文章详情，评论，点赞等。

大家好，我是 落霞孤鹜，上一篇我们已经博客的管理后台功能，这一章我们开始搭建博客的前台，实现对博客网站文章的查看，浏览，评论，点赞等功能。我同样按照一个完整的功能，从需求分析到代码编写来阐述如何实现。

一、需求分析

作为一个完整的博客网站，前台是内容呈现的核心部分，大部分的博客搭建文章着重介绍的也是这一部分。这里我们从实际需要出发，整理了如下需求要点：

1. **首页**：主要展示整个博客网站的文章，一般按照发布时间倒序呈现，展示标题，摘要，浏览量，点赞量，评论量，留言量等内容，提供标签筛选
2. **文章详情**：主要用来展示文章的详情，涵盖文章的所有细节，同时提供文章的章节目录导航、点赞、评论功能。
3. **文章分类**：通过分类呈现文章列表，方便用户通过类型快速查找感兴趣的文章。
4. **归档**：按照年份倒序呈现博客网站的文章列表。
5. **关于**：一般介绍博客的博主情况和博客网站的主题信息。

以上功能也算是一套简单的个人博客网站的核心功能框架。

二、后端接口部分

后端接口部分在上一篇的管理后端中，已经全部实现，这里就不再重复介绍。

三、前端界面部分

前端按照需求，我们从首页，文章详情，文章分类，归档，关于五个部分呈现。这一部分的页面全部放在 `src/views/client` 文件中。

首页的功能，实际上是一个文章列表展示，而分类页面是增加了一个分类树的文章列表展示，因此在设计页面的时候，将文章展示列表作为一个组件，这样分类展示可以通过列表和分类两个组件拼装而成。

关于页面可以类比为一篇介绍博客和博主的文章详情页面，因此，可以将文章详情展示作为

3.1.1 Type 层

在处理管理后台时，已经在 `src/types/index.ts` 文件中定义好了文章相关的 `interface Article, ArticleArray, ArticleParams`。

3.1.2 API 层

在处理管理后台时，已经在 `src/api/service.ts` 文件中定义好了方法 `getArticleList`。

3.1.3 Component 层

依据上面的分析，我们需要将文章列表封装成一个组件，因此在 `src/components` 下新增文件 `ArticleList.vue`，通过点击文章，在一个新的页面中查看文章详情，代码如下：

```
<template>
  <ul id="list" class="articles-list">
    <transition-group name="el-fade-in">
      <li v-for="article in articleList" :key="article.id" class="item">
        <a :href="href + article.id" target="_blank">
           data-has-lazy-src="false" src="/src/assets/cover.jpg"/>
        </a>
        <div class="content">
          <a :href="href + article.id" target="_blank">
            <h4 class="title">{{ article.title }}</h4>
            <p class="abstract">{{ article.excerpt }}</p>
          </a>
          <div class="meta">
            <span>查看 {{ article.views }}</span>
            <span>评论 {{ article.comments }}</span>
            <span>赞 {{ article.likes }}</span>
            <router-link v-for="tag in article.tags_info" :key="tag.id"
              :to="`/articles?tags=${tag.id}&catalog=`">
              <el-tag size="mini">{{ tag.name }}</el-tag>
            </router-link>
            <span v-if="article.created_at" class="time">{{ formatTime(article.creat<br/>ed_at) }}</span>
          </div>
        </div>
      </li>
    </transition-group>
  </ul>
</template>
```

知乎

专栏
Python

```
<script lang="ts">
import {timestampToTime} from "../utils";
import {defineComponent, PropType} from "vue";
import {Article} from "../types";

export default defineComponent({
  name: 'ArticleList',
  props: {
    articleList: {
      type: Array as PropType<Array<Article>>,
      default: []
    }
  },
  setup() {
    const formatTime = (value: string | Date): string => {
      return timestampToTime(value, true);
    };
    const href: string = '/article/?id='
    return {
      formatTime,
      href,
    }
  }
})
</script>
<style lang="less" scoped>
.articles-list {
  margin: 0;
  padding: 0;
  list-style: none;

  .title {
    color: #333;
    margin: 7px 0 4px;
    display: inherit;
    font-size: 18px;
    font-weight: 700;
    line-height: 1.5;
  }

  .item > div {

```

知乎

专栏
Python

```
position: absolute;
top: 50%;
margin-top: -50px;
right: 0;
width: 125px;
height: 100px;
border-radius: 4px;
img {
    width: 100%;
    height: 100%;
    border: 1px solid #f0f0f0;
}
}

li {
    line-height: 20px;
    position: relative;
    // width: 100%;
    padding: 15px 0px;
    padding-right: 150px;
    border-bottom: 1px solid #f0f0f0;
    word-wrap: break-word;
    cursor: pointer;

    &:hover {
        .title {
            color: #000;
        }
    }
}

.abstract {
    min-height: 30px;
    margin: 0 0 8px;
    font-size: 13px;
    line-height: 24px;
    color: #555;
}

.meta {
    padding-right: 0 !important;
    font-size: 12px;
```

▲ 赞同 1



● 添加评论

↗ 分享

★ 收藏

⚑ 举报

收起 ^

知乎

专栏
Python

```
margin-right: 10px;
color: #b4b4b4;

&::hover {
    transition: 0.1s ease-in;
    -webkit-transition: 0.1s ease-in;
    -moz-transition: 0.1s ease-in;
    -o-transition: 0.1s ease-in;
    -ms-transition: 0.1s ease-in;
}

span {
    margin-right: 10px;
    color: #666;
}
}

}

}

</style>
```

处理页面在请求后端时的加载状态组件和结束加载状态后的组件。

在 `src/components` 下新增文件 `Loading.vue`，代码如下：

```
<template>
<div class="loading">

</div>
</template>
<script lang="ts">
import {defineComponent} from "vue";

export default defineComponent({
    name: "CustomLoading",
});
```

```
padding: 30px;
}
</style>
```

在 `src/components` 下新增文件 `EndLoading.vue`，代码如下：

```
<template>
  <div class="load-end"> ----- 我也是有底线的啦 -----
</template>
<script lang="ts">
import { defineComponent } from "vue";

export default defineComponent({
  name: "EndLoading",
});
</script>
<style scoped>
.load-end {
  text-align: center;
  padding: 30px;
  color: #969696;
  font-size: 14px;
}
</style>
```

3.1.4 Utils层

在文章列表中，我们为了更好的体验，对图片展示提供了限流处理和无极滚动加载。工具方法有：`getDocumentHeight`，`getQueryStringByName`，`getScrollTop`，`getWindowHeight`，`throttle`，在 `src/utils/index.ts` 增加代码：

```
// fn是我们需要包装的事件回调，delay是时间间隔的阈值
export function throttle(fn: Function, delay: number) {
  let last = 0,
    timer: any = null;
  return function (this: any) {
    let context = (this as any);
```



```
clearTimeout(timer);
timer = setTimeout(function () {
    last = now;
    fn.apply(context, args);
}, delay);
} else {
    last = now;
    fn.apply(context, args);
}
};

//根据 QueryString 参数名称获取值
export function getQueryStringByName(name: string) {
let result = window.location.search.match(
    new RegExp("[?&]" + name + "=([^\&]+)", "i")
);
if (result == null || result.length < 1) {
    return "";
}
return result[1];
}

//获取页面顶部被卷起来的高度
export function getScrollTop() {
return Math.max(
    //chrome
    document.body.scrollTop,
    //firefox/IE
    document.documentElement.scrollTop
);
}

//获取页面文档的总高度
export function getDocumentHeight() {
//现代浏览器（IE9+和其他浏览器）和IE8的document.body.scrollHeight和document.docum
return Math.max(
    document.body.scrollHeight,
    document.documentElement.scrollHeight
);
}
```

▲ 赞同 1▼添加评论分享收藏举报收起 ^



```
? document.documentElement.clientHeight  
: document.body.clientHeight;  
}
```

3.1.5 View 层

修改 `src/views/client/Home.vue` 文件，编写如下代码：

```
<template>
<div class="left clearfix">
  <h3 v-if="state.params.tags" class="left-title">
    {{ state.tag_name }} 相关的文章:
  </h3>
  <ArticleList :article-list="state.articlesList" />
  <Loading v-if="state.isLoading"></Loading>
  <EndLoading v-if="state.isLoadEnd"></EndLoading>
</div>
</template>

<script lang="ts">
import { defineComponent, nextTick, onMounted, reactive } from "vue";
import {
  getDocumentHeight,
  getQueryStringByName,
  getScrollTop,
  getWindowHeight,
  throttle,
} from "../../utils";
import EndLoading from "../../components/EndLoading.vue";
import Loading from "../../components>Loading.vue";
import { Article, ArticleArray, ArticleParams } from "../../types";
import { getArticleList } from "../../api/service";
import ArticleList from "../../components/ArticleList.vue";

const viewHeight = window.innerHeight || document.documentElement.clientHeight;
const lazyload = throttle(() => {
  const imgs = document.querySelectorAll("#list .item img");
  let num = 0;
  for (let i = 0; i < imgs.length; i++) {
    if (imgs[i].naturalWidth < 0) {
      imgs[i].naturalWidth = imgs[i].width;
      imgs[i].width = 0;
    }
    if (imgs[i].width === 0) {
      num++;
      if (num === imgs.length) {
        EndLoading();
      }
    }
  }
}, 100);
onMounted(() => {
  lazyload();
  window.addEventListener("scroll", lazyload);
  document.addEventListener("resize", lazyload);
});
</script>
```

知乎

 专栏
Python

```
let hasLaySrc = imgItem.getAttribute("data-has-lazy-src");
if (hasLaySrc === "false") {
    imgItem.src = imgItem.getAttribute("data-src");
    imgItem.setAttribute("data-has-lazy-src", "true");
}
num = i + 1;
}
},
}, 1000);

export default defineComponent({
name: "Home",
components: {
ArticleList,
EndLoading,
Loading,
},
watch: {
"$store.state.articleParams": {
handler(val: any, oldVal: any) {
    this.state.params.tags = val.tags;
    this.state.params.catalog = val.catalog;
    this.state.articlesList = [];
    this.state.params.page = 1;
    this.handleSearch();
},
},
},
setup() {
const state = reactive({
isLoadEnd: false,
isLoading: false,
articlesList: [] as Array<Article>,
total: 0,
tag_name: decodeURI(getQueryStringByName("tag_name")),
params: {
title: undefined,
tags: undefined,
catalog: undefined,
page: 1,
page_size: 10,

```

▲ 赞同 1▼添加评论分享收藏举报收起 ^

专栏
Python

```
state.isLoading = true;

try {
    const data: ArticleArray = await getArticleList(state.params);
    state.isLoading = false;
    state.articlesList = [...state.articlesList, ...data.results];
    state.total = data.count;
    state.params.page++;
    await nextTick(() => {
        lazyload();
    });
    if (
        data.results.length === 0 ||
        state.total === state.articlesList.length
    ) {
        state.isLoadEnd = true;
        document.removeEventListener("scroll", () => {});
        window.onscroll = null;
    }
} catch (e) {
    console.error(e);
    state.isLoading = false;
}
};

onMounted(() => {
    window.onscroll = () => {
        if (getScrollTop() + getWindowHeight() > getDocumentHeight() - 100) {
            if (state.isLoadEnd === false && state.isLoading === false) {
                console.info("222");
                handleSearch();
            }
        }
    };
    document.addEventListener("scroll", lazyload);
    handleSearch();
});

return {
    state,
    handleSearch,
```

[▲ 赞同 1](#)[添加评论](#)[分享](#)[收藏](#)[举报](#)[收起 ^](#)

```
<style lang="less">
a {
  text-decoration: none;
}
</style>
```

3.1.5 Router 层

由于主页的路由已经在 `src/route/index.ts` 文件中配置，这里增加Articles的路由。

```
{
  path: "/articles",
  name: "Articles",
  component: () =>
    import("../views/admin/Home.vue")
},
```

3.1.6 Less

在 `src` 下新增文件夹 `less`，新增 `index.less` 代码如下：

```
body {
  padding: 10px;
  margin: 0;
}

a {
  text-decoration: none;
}

.clearfix:before,
.clearfix:after {
  display: table;
  content: '';
}
```



```
.right {
    width: 350px;
}

.left {
    flex: 1;
    padding-right: 20px !important;
}

.clearfix:after {
    clear: both;
}

.f1 {
    float: left;
}

.fr {
    float: right;
}

h1,
h2,
h3,
h4,
h5,
h6 {
    margin-top: 1em;
}

strong {
    font-weight: bold;
}

p > code:not([class]) {
    padding: 2px 4px;
    font-size: 90%;
    color: #c7254e;
    background-color: #f9f2f4;
```

▲ 赞同 1添加评论分享收藏举报收起 ^

知乎

专栏
Python

```
max-width: 100%;  
}  
  
.container {  
    width: 1200px;  
    margin: 0 auto;  
}  
  
.article-detail {  
    img {  
        /* 图片居中 */  
        display: flex;  
        max-width: 100%;  
        margin: 0 auto;  
    }  
  
    table {  
        text-align: center;  
        border: 1px solid #eee;  
        margin-bottom: 1.5em;  
    }  
  
    th,  
    td {  
        // text-align: center;  
        padding: 0.5em;  
    }  
  
    tr:nth-child(2n) {  
        background: #f7f7f7;  
    }  
}  
  
.article-detail {  
    font-size: 16px;  
    line-height: 30px;  
}  
  
.anchor-fix {  
    position: relative !important;  
    top: -80px !important;  
}
```

▲ 赞同 1



● 添加评论

↗ 分享

★ 收藏

☞ 举报

收起 ^

```
.article-detail .desc ul,  
.article-detail .desc ol {  
    color: #333333;  
    margin: 1.5em 0 0 25px;  
}  
  
.article-detail .desc h1,  
.article-detail .desc h2 {  
    border-bottom: 1px solid #eee;  
    padding-bottom: 10px;  
}  
  
.article-detail .desc a {  
    color: #009a61;  
}  
  
.article-detail blockquote {  
    margin: 0 0 1em;  
    background-color: rgb(220, 230, 240);  
    padding: 1em 0 0.5em 0.5em;  
    border-left: 6px solid rgb(181, 204, 226);  
}
```

在 src/App.vue 的 style 部分，增加导入 index.less 的代码

```
@import url("./less/index.less");
```

3.2 文章详情

文章详情通过路径中的 query 参数传递文章id的方式区别不同的文章，这样的好处是方便文章可以通过url实现分享，比如想发表在公众号中，原文链接就可以直接用该 URL

3.2.1 Type 层

文章详情中涉及到文章，分类，标签，点赞、评论，结合之前已经定义的内容，在 src/types/index.ts 文件中增加代码如下：



```
    user: number,  
}
```

3.2.2 API 层

在 `src/api/service.ts` 编写如下代码：

```
export function postLikeArticle(data: Like) {  
    return request({  
        url: '/like/',  
        method: 'post',  
        data,  
    })  
}  
  
export function getArticleComments(articleId: number) {  
    return request({  
        url: '/comment/',  
        method: 'get',  
        params: {  
            article: articleId,  
        },  
    }) as unknown as ResponseData  
}  
  
export function addComment(data: CommentPara) {  
    return request({  
        url: '/comment/',  
        method: 'post',  
        data  
    })  
}
```

3.2.3 Component 层

这里需要用到评论列表、评论组件。

在 `src/components` 下新增文件 `Comment.vue`，负责新增评论，代码如下：

▲ 赞同 1 ▼ ● 添加评论 ↗ 分享 ★ 收藏 📌 举报

收起 ^

知乎

专栏
Python

```
<el-input  
    v-model="state.content"  
    placeholder="文明社会，理性评论"  
    type="textarea"  
/>  
<el-button  
    :loading="state.btnLoading"  
    style="margin-top: 15px"  
    type="primary"  
    @click="handleOk"  
>  
    发送  
</el-button>  
</div>  
  
<el-dialog  
    v-else  
    v-model="state.showDialog"  
    title="评论"  
    width="60%"  
    @close="cancel"  
>  
<el-form>  
    <el-form-item>  
        <el-input  
            v-model="state.content"  
            autocomplete="off"  
            placeholder="文明社会，理性评论"  
            type="textarea"  
        />  
    </el-form-item>  
</el-form>  
<template v-slot:footer>  
    <div class="dialog-footer">  
        <el-button type="default" @click="cancel">取消</el-button>  
        <el-button type="primary" @click="handleOk">确定</el-button>  
    </div>  
</template>  
</el-dialog>  
</template>
```

▲ 赞同 1



● 添加评论

↗ 分享

★ 收藏

⚑ 举报

收起 ^



```
import { addComment } from "../api/service";
import { useStore } from "vuex";
import { StateKey } from "../store";
import { CommentPara } from "../types";

export default defineComponent({
  name: "Comment",
  props: {
    forArticle: {
      type: Boolean,
      require: true,
    },
    showDialog: {
      type: Boolean,
      default: false,
    },
    article_id: {
      type: Number,
      require: true,
    },
    reply: {
      type: Number,
      default: undefined,
    },
  },
  emits: ["ok", "cancel"],
  setup(props, context) {
    const state = reactive({
      showDialog: props.showDialog,
      btnLoading: false,
      content: "",
      cacheTime: 0, // 缓存时间
      times: 0, // 留言次数
    });
    const store = useStore(StateKey);
    const cancel = (): boolean => {
      context.emit("cancel", false);
      return false;
    };

    const handleOk = async (): Promise<void> => {
```

知乎

专栏
Python

```
});  
return;  
}  
  
if (state.times > 2) {  
  ElMessage({  
    message: "您今天评论的次数已经用完，明天再来评论吧！",  
    type: "warning",  
  });  
  return;  
}  
  
let now = new Date();  
let nowTime = now.getTime();  
if (nowTime - state.cacheTime < 4000) {  
  ElMessage({  
    message: "您评论太过频繁，1分钟后再来评论吧！",  
    type: "warning",  
  });  
  return;  
}  
  
if (!state.content) {  
  ElMessage({  
    message: "评论内容不能为空",  
    type: "error",  
  });  
  return;  
}  
  
let user_id: number;  
if (store.state.user.id > 0) {  
  user_id = store.state.user.id;  
} else {  
  ElMessage({  
    message: "登录才能评论，请先登录！",  
    type: "warning",  
  });  
  return;  
}  
state.btnLoading = true;
```

知乎

专栏
Python

```
        reply: props.reply,
        content: state.content,
    } as CommentPara);
state.btnLoading = false;
state.times++;

state.cacheTime = nowTime;
state.content = "";
context.emit("ok", false);
ElMessage({
    message: "评论成功",
    type: "success",
});
} catch (e) {
    ElMessage({
        message: "评论失败, 请重试哦",
        type: "success",
    });
    state.btnLoading = false;
}
};

watch(props, (val, oldVal) => {
    state.showDialog = val.showDialog;
});

return {
    state,
    cancel,
    handleOk,
};
},
});
</script>
<style scoped>
.dialog-footer {
    text-align: right;
}
</style>
```

在 `src/components` 下新增文件 `commentList.vue`，负责展示文章的评论列表。代码如下：

▲ 赞同 1 ▾ 添加评论 分享 收藏 举报

收起 ^

知乎

专栏
Python

```
<div class="top-title">
    <span>{{ numbers }} 条评论</span>
</div>
<div v-for="(item, i) in state.comments" :key="item.id" class="item">
    <div class="item-header">
        <div class="author">
            <div class="avatar">
                
                
            </div>
        </div>
        <div class="info">
            <div class="name">
                {{ item.user_info.name
                }}{{ item.user_info.role === "Admin" ? "(作者)" : "" }}
            </div>
            <div class="time">{{ formatTime(item.created_at) }}</div>
        </div>
    </div>
    <div class="comment-detail">{{ item.content }}</div>
    <div class="item-comment">
        <div
            class="message"
            @click="showCommentModal(item.id, item.user_info.id)"
        >
            <el-button size="small">回复</el-button>
        </div>
    </div>
    <div v-for="e in item.comment_replies" :key="e._id" class="item-other">
        <div class="item-header">
            <div class="author">
                <div class="avatar">
                    
```

▲ 赞同 1



● 添加评论

↗ 分享

★ 收藏

⚑ 举报

收起 ^



```
<div class="info">
    <div class="name">
        {{ e.user_info.name }}
        {{ e.user_info.role === "Admin" ? "(作者)" : "" }}
    </div>
    <div class="time">
        {{ formatTime(e.created_at) }}
    </div>
</div>
<div class="comment-detail">
    {{ e.content }}
</div>
</div>
<Comment
    :article_id="article_id"
    :forArticle="false"
    :reply="state.comment_id"
    :show-dialog="state.visible"
    @cancel="handleCancel"
    @ok="handleOk"
/>
</div>
</template>

<script lang="ts">
import { ElMessage } from "element-plus";
import {
    defineAsyncComponent,
    defineComponent,
    onMounted,
    reactive,
} from "vue";
import { timestampToTime } from "../utils";
import { CommentInfo } from "../types";
import { getArticleComments } from "../api/service";

export default defineComponent({
    name: "CommentList",
    components: {
```

▲ 赞同 1▼添加评论分享收藏举报收起 ^

知乎

专栏
Python

```
type: Number,  
default: 0,  
,  
article_id: {  
type: Number,  
default: undefined,  
,  
},  
  
setup(props, context) {  
const state = reactive({  
visible: false,  
comment_id: 0,  
comments: [] as Array<CommentInfo>,  
reply: 0,  
});  
  
const formatTime = (value: string | Date): string => {  
return timestampToTime(value, true);  
};  
  
const handleCancel = (): void => {  
state.visible = false;  
};  
  
const getCommentList = async () => {  
try {  
const response = await getArticleComments(props.article_id);  
state.comments = response.results as unknown as Array<CommentInfo>;  
} catch (e) {  
console.error(e);  
}  
};  
  
const handleOk = async (): Promise<void> => {  
state.visible = false;  
await getCommentList();  
};  
  
// 添加评论  
const showCommentModal = (
```

▲ 赞同 1

▼

● 添加评论

↗ 分享

★ 收藏

⚑ 举报

收起 ^



```
if (!window.sessionStorage.userInfo) {
    ElMessage({
        message: "登录才能点赞, 请先登录!",
        type: "warning",
    });
    return false;
}
// 添加三级评论
if (secondUser) {
    state.comment_id = commentId;
} else {
    // 添加二级评论
    state.comment_id = commentId;
}
state.visible = true;
};

onMounted(() => {
    getCommentList();
});

return {
    state,
    showCommentModal,
    handleOk,
    handleCancel,
    formatTime,
};
},
});
};

</script>
<style lang="less" scoped>
.comment-list {
    text-align: center;
}

.comment-list {
    position: relative;
    text-align: left;
    padding-top: 30px;
    margin-top: 30px;
    border-top: 1px solid #eee;
```

▲ 赞同 1▼添加评论分享收藏举报收起 ^

}

```
.el-icon-circle-plus {  
    font-size: 40px;  
}  
}  
  
.clearfix {  
    clear: both;  
}  
  
.comment-list {  
    margin-top: 30px;  
  
.top-title {  
    padding-bottom: 20px;  
    font-size: 17px;  
    font-weight: 700;  
    border-bottom: 1px solid #f0f0f0;  
}  
  
.item {  
    padding: 20px 0 30px;  
    border-bottom: 1px solid #f0f0f0;  
  
.item-header {  
    position: relative;  
    padding-left: 45px;  
    padding-bottom: 10px;  
  
.author {  
    position: absolute;  
    left: 0;  
    display: inline-block;  
  
.avatar {  
    display: inline-block;  
    margin-right: 5px;  
    width: 40px;  
    height: 40px;  
    vertical-align: middle;
```

知乎

专栏
Python

```
border-radius: 50%;  
}  
}  
}  
  
.info {  
display: inline-block;  
  
.name {  
font-size: 15px;  
color: #333;  
}  
  
.time {  
font-size: 12px;  
color: #969696;  
}  
}  
}  
  
.comment-detail {  
min-height: 40px;  
}  
  
.item-comment {  
.like {  
margin-right: 20px;  
}  
}  
}  
}  
  
.item-other {  
margin: 20px;  
padding: 10px;  
border-left: 2px solid #f0f0f0;  
  
.item-header {  
position: relative;  
padding-left: 45px;  
padding-bottom: 10px;
```

▲ 赞同 1



● 添加评论

↗ 分享

★ 收藏

⚑ 举报

收起 ^

知乎

专栏
Python

```
display: inline-block;

.avatar {
    display: inline-block;
    margin-right: 5px;
    width: 38px;
    height: 38px;
    vertical-align: middle;

    img {
        width: 100%;
        height: 100%;
        border-radius: 50%;
    }
}

.info {
    display: inline-block;

    .name {
        font-size: 15px;
        color: #333;
    }

    .time {
        font-size: 12px;
        color: #969696;
    }
}

.comment-detail {
    min-height: 40px;
    border-bottom: 1px dashed #f0f0f0;
}

.message {
    padding: 10px;
}
```

▲ 赞同 1



● 添加评论

↗ 分享

★ 收藏

⚑ 举报

收起 ^

知乎

专栏
Python

由于我们编写的文章正文是用markdown的方式记录的，而在博客网站上需要展示的是HTML，因此我们需要在展示之前将markdown转换成html，同时能展示文章的章节目录，所以先安装依赖

```
yarn add marked@2.0.3
```

在 src/utils 下新增文件 markdown.ts，编写代码如下：

```
import highlight from 'highlight.js'  
// @ts-ignore  
import marked from 'marked'  
  
const tocObj = {  
    add: function (text: any, level: any) {  
        let anchor = `#toc${level}${++this.index}`;  
        this.toc.push({anchor: anchor, level: level, text: text});  
        return anchor;  
    },  
  
    toHTML: function () {  
        let levelStack: any = [];  
        let result = "";  
        const addStartUL = () => {  
            result += '<ul class="anchor-ul" id="anchor-fix">';  
        };  
        const addEndUL = () => {  
            result += "</ul>\n";  
        };  
        const addLI = (anchor: any, text: any) => {  
            result +=  
                '<li><a class="toc-link" href="#" + anchor + '">' + text + "</a></li>";  
        };  
  
        this.toc.forEach(function (item: any) {  
            let levelIndex = levelStack.indexOf(item.level);  
            // 没有找到相应level的ul标签，则将li放入新增的ul中  
            if (levelIndex === -1) {  
                levelStack.unshift(item.level);  
                addStartUL();  
            }  
            addLI(item.anchor, item.text);  
        });  
    },  
};
```

▲ 赞同 1 ▼

● 添加评论

↗ 分享

★ 收藏

⚑ 举报

收起 ^



```
        } // 找到了相应level的ul标签，但是不在栈顶位置，需要将之前的所有level出栈并上
        else {
            while (levelIndex--) {
                levelStack.shift();
                addEndUL();
            }
            addLI(item.anchor, item.text);
        }
    });
    // 如果栈中还有level，全部出栈打上闭合标签
    while (levelStack.length) {
        levelStack.shift();
        addEndUL();
    }
    // 清理先前数据供下次使用
    this.toc = [];
    this.index = 0;
    return result;
},
toc: [] as any,
index: 0
};

class MarkUtils {
    private readonly rendererMD: any;

    constructor() {
        this.rendererMD = new marked.Renderer() as any;
        this.rendererMD.heading = function (text: any, level: any, raw: any) {
            let anchor = tocObj.add(text, level);
            return `<a id=${anchor} class="anchor-fix"></a><h${level}>${text}</h$`;
        };
        this.rendererMD.table = function (header: any, body: any) {
            return '<table class="table" border="0" cellspacing="0" cellpadding="0" style="width: 100%; border-collapse: collapse; border: none; margin-bottom: 10px;">';
        }
        highlight.configure({useBR: true});
        marked.setOptions({
            renderer: this.rendererMD,
            headerIds: false,
            gfm: true,
            // tables: true,
        });
    }
}
```

知乎

专栏
Python

```
smartypants: false,  
highlight: function (code: any) {  
    return highlight.highlightAuto(code).value;  
}  
});  
}  
  
async marked(data: any) {  
    if (data) {  
        let content = await marked(data);  
        let toc = tocObj.toHTML();  
        return {content: content, toc: toc};  
    } else {  
        return null;  
    }  
}  
}  
  
const markdown: any = new MarkUtils();  
  
export default markdown;
```

3.2.5 View 层

在 src/views/client 下新增文件 ArticleDetail.vue 文件，编写如下代码：

```
<template>  
  <div style="width: 100%">  
    <div class="article clearfix">  
      <div v-show="!state.isLoading" :style="{'width': '75%'}" class="article-left">  
        <div class="header">  
          <h1 class="title">{{ state.detail.title }}</h1>  
          <div class="author">  
            <div class="avatar">  
                
            </div>  
            <div class="info">  
              <span class="name">
```

▲ 赞同 1



● 添加评论



分享



收藏



举报

收起 ^



```


    {{ state.detail.created_at ? formatTime(state.detail.created_at) }}

字数 {{ state.detail.words }}
阅读 {{ state.detail.views }}
评论 {{ state.detail.comments }}
喜欢 {{ state.detail.likes }}

```

</div>

</div>

<div class="tags" title="标签">

```

    <el-tag v-for="tag in state.detail.tags_info" :key="tag.id" class="tag-item">
        {{ tag.name }}
    </el-tag>

```

</div>

</div>

</div>

<div class="content">

```

    <div id="content" class="article-detail" v-html="state.detail.html"></div>

```

<div class="heart">

```

    <el-button :loading="state.isLoading" icon="heart" size="large" type="danger">
        点赞
    </el-button>

```

</div>

```

<Comment :article_id="state.params" :for-article="true" :show-dialog="false" />
<CommentList :article_id="state.params" :numbers="state.detail.comments" />

```

</div>

<div class="article-right fr anchor" style="width: 23%">

```

    v-html="state.detail.toc"></div>

```

<Loading v-if="state.isLoading"/>

</div>

</div>

</template>

```

<script lang="ts">
import {defineComponent, onMounted, reactive} from "vue";
import {ElMessage} from "element-plus";
import {useRoute} from "vue-router";
import {timestampToTime} from "../../utils";
import markdown from "../../utils/markdown";

```



专栏

Python

```
import {StateKey} from "../../store";
import {useStore} from "vuex";
import Comment from "../../components/Comment.vue";

declare let document: Document | any;

export default defineComponent({
  name: "ArticleDetail",
  components: {
    Comment,
    Loading,
    CommentList,
  },
  setup() {
    const store = useStore(StateKey)

    const state = reactive({
      btnLoading: false,
      isLoadEnd: false,
      isLoading: false,
      params: 0,
      content: "",
      detail: {
        id: 0,
        title: "",
        excerpt: "",
        cover: "",
        markdown: "",
        created_at: "",
        modified_at: "",
        tags_info: [] as Array<Tag>,
        catalog_info: {} as Catalog,
        views: 0,
        comments: 0,
        words: 100,
        likes: 0,
        author: '落霞孤鹜',
      } as Article,
      cacheTime: 0, // 缓存时间
      times: 0, // 评论次数
    })
  }
})
```

▲ 赞同 1



● 添加评论



★ 收藏



举报

收起 ^



```
return timestampToTime(value, true);
};

const handleSearch = async (): Promise<void> => {
  state.isLoading = true;
  try {
    const data: any = await getArticleDetail(state.params);
    state.isLoading = false;

    state.detail = data;
    const article = markdown.marked(data.markdown);
    article.then((res: any) => {
      state.detail.html = res.content;
      state.detail.toc = res.toc;
    });
    document.title = data.title;
    document.querySelector("#keywords").setAttribute("content", data.keyword)
    document.querySelector("#description")
      .setAttribute("content", data.excerpt);

  } catch (e) {
    state.isLoading = false;
  }
};

const likeArticle = async (): Promise<void> => {
  if (!state.detail.id) {
    ElMessage({
      message: "该文章不存在!",
      type: "warning",
    });
    return;
  }

  if (state.likeTimes > 0) {
    ElMessage({
      message: "您已经点过赞了! 悠着点吧!",
      type: "warning",
    });
  }
};
```

[赞同 1](#)[▼](#)[添加评论](#)[分享](#)[收藏](#)[举报](#)[收起 ^](#)

知乎

专栏
Python

```
if (user_id === 0) {
  ElMessage({
    message: "登录才能点赞，请先登录！",
    type: "warning",
  });
  return;
}

let params: Like = {
  article: state.detail.id,
  user: user_id,
};
try {
  await postLikeArticle(params);

  state.isLoading = false;

  state.likeTimes++;
  ++state.detail.likes;
  ElMessage({
    message: "操作成功",
    type: "success",
  });
} catch (e) {
  state.isLoading = false;
}
};

const route = useRoute()
if (route.path === '/about') {
  state.params = 1
} else {
  state.params = Number(route.query.id)
}

onMounted(() => {
  handleSearch();
});

return {
  state,
```

▲ 赞同 1

▼

添加评论

分享

收藏

举报

收起 ^

知乎

专栏

Python

```
},
beforeUnmount(): void {
    document.title = "落霞孤鹜的博客网站";
    document
        .getElementById("keywords")
        .setAttribute("content", "落霞孤鹜 的博客网站");
    document
        .getElementById("description")
        .setAttribute(
            "content",
            "分享人工智能相关的产品和技术。"
        );
},
});
</script>
<style lang="less" scoped>
.anchor {
    display: block;
    position: sticky;
    top: 213px;
    margin-top: 213px;
    border-left: 1px solid #eee;
    min-height: 48px;

    .anchor-ul {
        position: relative;
        top: 0;
        max-width: 250px;
        border: none;
        -moz-box-shadow: 0 0 0 #fff;
        -webkit-box-shadow: 0 0 0 #fff;
        box-shadow: 0 0 0 #fff;
    }

    li.active {
        color: #009a61;
    }
}

a {
    color: #333;
}
}
```

赞同 1添加评论分享收藏举报收起 ^

```
.header {  
    border-bottom: #eeeeee 1px solid;  
  
.title {  
    margin: 0;  
    text-align: center;  
    font-size: 34px;  
    font-weight: bold;  
}  
  
.author {  
    position: relative;  
    margin: 30px 0 40px;  
    padding-left: 50px;  
  
.avatar {  
    position: absolute;  
    left: 0;  
    top: 0;  
    width: 48px;  
    height: 48px;  
    vertical-align: middle;  
    display: inline-block;  
  
    img {  
        width: 100%;  
        height: 100%;  
        border-radius: 50%;  
    }  
}  
  
.info {  
    float: left;  
    vertical-align: middle;  
    // display: inline-block;  
    margin-left: 8px;  
  
    a {  
        color: #333;  
    }  
}
```

▲ 赞同 1● 添加评论↗ 分享★ 收藏⚑ 举报收起 ^



```
    font-size: 16px;
    vertical-align: middle;
}

.meta {
    margin-top: 5px;
    font-size: 12px;
    color: #969696;

    span {
        padding-right: 5px;
    }
}

.tags {
    float: right;
    padding-top: 15px;
    // padding-right: 20px;
    .tag {
        // padding: 0 10px;
        margin-left: 5px;
        border-right: 2px solid #eee;
    }
}
}

.content {
    min-height: 300px;
}
}

.heart {
    height: 60px;
    text-align: center;
    margin: 50px;
}

.loader {
    color: rgb(226, 44, 44);
    text-align: center;
```

```
.clearfix {  
    clear: both;  
}  
  
.anchor-fix1 {  
    display: block;  
    height: 60px; /*same height as header*/  
    margin-top: -60px; /*same height as header*/  
    visibility: hidden;  
}  
  
</style>
```



3.2.6 Router 层

定义 route 来完成路由跳转。在 src/route/index.ts 文件中新增代码：

```
{  
    path: "/article/",  
    name: "ArticleDetail",  
    component: () =>  
        import("../views/client/ArticleDetail.vue")  
},
```

3.3 分类

3.3.1 Store 层

为了实现文章列表组件的复用，我们在 Store 保存当前的文章的检索条件，调整后的 src/store/index.ts

```
import { InjectionKey } from 'vue'  
import { createStore, Store } from 'vuex'  
import { Nav, User, ArticleParams } from "../types";  
  
export interface State {
```

▲ 赞同 1



● 添加评论

↗ 分享

★ 收藏

☞ 举报

收起 ^

```
export const StateKey: InjectionKey<Store<State>> = Symbol();  
  
export const SET_USER = 'setUser';  
export const CLEAR_USER = 'clearUser'  
export const SET_NAV_INDEX = 'setNavIndex'  
export const SET_ARTICLE_PARAMS = 'setArticleParams'  
  
  
export const initDefaultUserInfo = (): User => {  
    let user: User = {  
        id: 0,  
        username: "",  
        avatar: "",  
        email: '',  
        nickname: '',  
        is_superuser: false,  
    }  
    if (window.sessionStorage.userInfo) {  
        user = JSON.parse(window.sessionStorage.userInfo);  
    }  
    return user  
}  
  
export const initDefaultArticleParams = (): ArticleParams => {  
    let params: ArticleParams = {  
        title: undefined,  
        status: 'Published',  
        tags: undefined,  
        catalog: undefined,  
        page: 1,  
        page_size: 10,  
    }  
    if (window.sessionStorage.articleParams) {  
        params = JSON.parse(window.sessionStorage.articleParams);  
    }  
    return params  
}  
  
export const store = createStore<State>({  
    state() {
```

▲ 赞同 1▼添加评论分享收藏举报收起 ^



```
navs: [
  {
    index: "1",
    path: "/",
    name: "主页",
  },
  {
    index: "2",
    path: "/catalog",
    name: "分类",
  },
  {
    index: "3",
    path: "/archive",
    name: "归档",
  },
  {
    index: "4",
    path: "/about",
    name: "关于",
  },
],
},
},
mutations: {
  setUser(state: object | any, userInfo: object | any) {
    for (const prop in userInfo) {
      state[prop] = userInfo[prop];
    }
  },
  clearUser(state: object | any) {
    state.user = initDefaultUserInfo();
  },
  setNavIndex(state: object | any, navIndex: string) {
    state.navIndex = navIndex
  },
  setArticleParams(state: object | any, params: object) {
    state.articleParams = {...state.articleParams, ...params}
  }
}
```

[▲ 赞同 1](#)[▼](#)[添加评论](#)[分享](#)[收藏](#)[举报](#)[收起 ^](#)



通过表格查看评论，在 `src/views/client` 下新增文件 `Catalog.vue` 文件，编写如下代码：

```
<template>
  <div class="catalog">
    <div :style="{ 'min-height': height + 'px' }" class="catalog-tree">
      <el-tree
        :current-node-key="state.currentNodeKey"
        :data="state.catalogs"
        :props="defaultProps"
        node-key="id"
        @node-click="handleNodeClick"
      ></el-tree>
    </div>
    <div class="article-list">
      <Home />
    </div>
  </div>
</template>

<script lang="ts">
import { defineComponent, onMounted, reactive } from "vue";
import { Article, ArticleParams, Catalog } from "../../types";
import { getCatalogTree } from "../../api/service";
import Home from "./Home.vue";
import { SET_ARTICLE_PARAMS, StateKey } from "../../store";
import { useStore } from "vuex";

export default defineComponent({
  name: "Catalog",
  components: { Home },

  setup() {
    const state = reactive({
      catalogs: [] as Array<Catalog>,
      articleParams: { catalog: 1 } as ArticleParams,
      articleList: [] as Array<Article>,
      currentNodeKey: 1,
    });
  }
);
```

▲ 赞同 1

▼

● 添加评论

↗ 分享

★ 收藏

⚑ 举报

收起 ^

专栏
Python

```
const defaultProps = {
  children: "children",
  label: "name",
};

const store = useStore(StateKey);

onMounted(() => {
  getCatalogs();
  state.currentNodeKey = store.state.articleParams.catalog || 1;
});

let height = window.innerHeight || document.documentElement.clientHeight;
height = height - 200;
const handleNodeClick = (catalog: Catalog) => {
  store.commit(SET_ARTICLE_PARAMS, { catalog: catalog.id });
};

return {
  state,
  defaultProps,
  height,
  handleNodeClick,
};
},
});
</script>

<style lang="less" scoped>
.catalog {
  display: flex;
}

.catalog-tree {
  width: 200px;
  border-right: 1px solid #eeeeee;
  margin-right: 24px;
  padding-top: 24px;
  margin-top: -12px;
  color: #2c3e50;
}

```

[▲ 赞同 1](#)[添加评论](#)[分享](#)[收藏](#)[举报](#)[收起 ^](#)

3.3.5 Router 层

定义 route 来完成路由跳转。在 src/route/index.ts 文件中新增代码：

```
{  
    path: '/catalog',  
    name: 'Catalog',  
    component: () =>  
        import("../views/client/Catalog.vue")  
,
```

3.4 归档

3.4.1 Type 层

在 src/types/index.ts 文件中增加代码如下：

```
export interface PageInfo {  
    page: number,  
    page_size: number  
}  
  
export interface ArticleArchiveList {  
    year: number,  
    list: Array<Article> | any  
}
```

3.4.2 API 层

列表查询，在 src/api/service.ts 编写如下代码：

```
export function getArchiveList(params: PageInfo) {  
    return request({  
        url: '/archive/',  
        method: 'get',
```

3.4.3 Store 层

在 `src/store/index.ts` 中增加如下代码：

```
export const SET_NAV_INDEX_BY_ROUTE = 'setNavIndexByRoute'
```

在 `src/store/index.ts` 中的 `store` 中增加如下代码：

```
actions: {
    setNavIndexByRoute({commit, state}, route: string) {
        const index = state.navs.findIndex(r => r.path === route)
        if (state.navIndex === state.navs[index].index)
            return
        if (index > -1) {
            commit(SET_NAV_INDEX, state.navs[index].index)
        }
    }
}
```

3.4.4 View 层

在 `src/views/client` 下新增文件 `Archive.vue` 文件，编写如下代码：

```
<template>
<div class="archive left">
<el-timeline>
    <el-timeline-item v-for="(l, i) in state.articlesList" :key="l.year" hide-t:
        <h3 class="year">{{ l.year }}</h3>
        <el-timeline-item
            v-for="(item, index) in l.list"
            :key="item.id"
            hide-timestamp
            placement="top"
        >
            <router-link :to="`/article/?id=${item.id}`" target="_blank">
                <h3 class="title">{{ item.title }}</h3>
            </router-link>
        </el-timeline-item>
    </el-timeline-item>
</el-timeline>
</div>
```

▲ 赞同 1

▼

● 添加评论

↗ 分享

★ 收藏

⚑ 举报

收起 ^

知乎

专栏
Python

```
</el-timeline>
</div>
</template>
<script lang="ts">
import {defineComponent, onMounted, reactive} from "vue";
import {timestampToTime} from "../../utils";
import {ArticleArchiveList, PageInfo} from "../../types";
import {getArchiveList} from "../../api/service";
import {useStore} from "vuex";
import {SET_NAV_INDEX_BY_ROUTE, StateKey} from "../../store";

export default defineComponent({
  name: "Archive",
  setup() {
    const state = reactive({
      isLoadEnd: false,
      isLoading: false,
      articlesList: [] as Array<ArticleArchiveList>,
      total: 0,
      params: {
        page: 1,
        page_size: 10,
      } as PageInfo
    });

    const formatTime = (value: string | Date): string => {
      return timestampToTime(value, true);
    }

    const handleSearch = async (): Promise<void> => {
      state.isLoading = true;
      const params: PageInfo = state.params
      try {
        const data: any = await getArchiveList(params)
        state.isLoading = false;
        state.articlesList = [...state.articlesList, ...data.results];
        state.total = data.count;
        state.params.page++;
        if (state.total === state.articlesList.length) {
          state.isLoadEnd = true;
        }
      } catch (error) {
        console.error(error);
      }
    }
  }
})
```

▲ 赞同 1

▼

● 添加评论

↗ 分享

★ 收藏

⚑ 举报

收起 ^

```
}

onMounted(() => {
  const store = useStore(StateKey)
  store.dispatch(SET_NAV_INDEX_BY_ROUTE, '/archive')
  handleSearch();
})

return {
  state,
  formatTime,
  handleSearch
};
},
})
;
</script>

<style lang="less" scoped>
.archive {
  padding: 40px 0;

  .year {
    font-size: 30px;
    font-weight: bold;
    color: #000;
    margin-top: 0;
  }

  a {
    text-decoration: none;
  }

  .title {
    color: #333;

    &:hover {
      color: #1890ff;
    }
  }
}
```

▲ 赞同 1▼添加评论分享收藏举报收起 ^

3.4.5 Router 层

定义 route 来完成路由跳转。在 src/route/index.ts 文件中新增代码：

```
{  
    path: "/archive/",  
    name: "Archive",  
    component: () =>  
        import("../views/client/Archive.vue")  
},
```

3.5 关于

3.5.1 改造 index.html

```
<!DOCTYPE html>  
<html lang="en">  
<head>  
    <meta charset="UTF-8"/>  
    <link href="/favicon.ico" rel="icon"/>  
    <meta id="viewport" content="width=device-width, initial-scale=1.0, maximum-scale=1.0, user-scalable=no" name="viewport"/>  
    <title>微谈小智</title>  
  
    <meta id="referrer" content="always" name="referrer"/>  
    <meta content="7XGPmF2RtW" name="baidu-site-verification"/>  
    <meta id="keywords" content="落霞孤鹜 的博客网站, 技术性产品经理" name="keywords"/>  
    <meta id="description" content="落霞孤鹜 的博客网站。落霞孤鹜, 目前是一名AI产品经理" name="description"/>  
</head>  
<body>  
    <div id="app"></div>  
    <script src="/src/main.ts" type="module"></script>  
</body>  
</html>
```

3.5.2 View 层

▲ 赞同 1 ▼ 添加评论 分享 收藏 举报

收起 ^

```
const route = useRoute()
if (route.path === '/about') {
  state.params = 1
} else {
  state.params = Number(route.query.id)
}
```

3.5.2 Router 层

定义 route 来完成路由跳转。在 src/route/index.ts 文件中新增代码：

```
{
  path: '/about',
  name: 'About',
  component: () =>
    import("../views/client/ArticleDetail.vue")
},
```

至此，博客针对用户的页面全部开发完成。

四、界面效果

4.1 首页



The screenshot shows the Zhihu homepage with a navigation bar at the top. The 'About' section is highlighted. It contains a bio: '工作这么多年，一直想把自己的所思所想所做用一种有意义的方式记录下来，本人是一个遇事比较纠结的人，在犹豫良久之后，一直没有付诸行动。' Below the bio is a timestamp: '查看 0 评论 0 赞 0 2021-08-23 22:23:14'. To the right is a user profile picture. At the bottom right, it says '知乎 @落霞孤鹜'.

4.2 分类

▲ 赞同 1 ▼ ● 添加评论 ↗ 分享 ★ 收藏 📢 举报 收起 ^



全部分类

Python

关于我

工作这么多年，一直想把自己的所思所想所用一种有意义的方式记录下来，本人是一个遇事比较纠结的人，在犹豫良久之后，一直没有付诸行动。

查看 0 评论 0 赞 0 2021-08-23 22:23:14



-----我也是有底线的啦-----

知乎 @落霞孤鹜

4.3 归档

• 2021

- 关于我

2021-08-23 22:23:14

知乎 @落霞孤鹜

4.4 文章详情和关于

▲ 赞同 1 ▼ ● 添加评论 ↗ 分享 ★ 收藏 🗑 举报 收起 ^

知乎

专栏
Python

天子我



2021-08-23 22:23:14 字数 0 阅读 0 评论 0 喜欢 0

1. 前言

工作这么多年，一直想把自己的所思所想所做用一种有意义的方式记录下来，本人是一个遇事比较纠结的人，在犹豫良久之后，一直没有付诸行动。

前年，和我一位朋友兼导师聊天，他说工作这么多年（马上快20年），感觉自己没留下什么值得纪念的东西，他决定在他退休之前，把深耕多年的领域业务知识，写成了一份文档，聊以纪念自己半生付出。

这件事情给了我很大的触动，我虽然没有这么丰富的领域知识沉淀，但是过程中多多少少有一些自己的思考和积累，决定再次提笔，把它们记录下来，算是对我自己这11年工作的一个交代。

最开始写博客，是在大学的时候。我是一个感性的人，喜欢在QQ空间（07、08年比较流行），写一些看来有一些幼稚的文字。在大学，我参加了学校的一个维修电脑的社团，也是在那个时候，我写下了人生第一篇技术文章 [电脑故障拾锦](#)，紧接着，我又写下了大学校园网如何共享的文章 [关于校园网共享问题的解决方法](#)。

毕业后，一直忙于工作，鲜有提笔的时间和契机。最后一次是在17年，记录的是“[Axure RP使用基础教程](#)”，是目前阅读量最大的一遍文章了。

- 1. 前言
- 2. 关于工作
- 3. 关于博客内容

2. 关于工作

在我11年的工作中，个人的方向调整了几次。

- 最开始：做2G端交付型项目的项目经理，带领一个小团队在客户现场支撑项目上线和售后维护
- 后面转到需求分析（那个时候B端还没有明确的产品经理概念）
- 然后转做售前解决方案，整理售前方案，编写项目标书，客户现场方案讲解等
- 最后转到AI领域做AI产品经理，负责做NLP领域中的智能问答机器人产品。
- 目前在负责中台体系下的智能客服产品体系建设

工作岗位转换了4次，所以对整个软件开发的生命周期都有一定的认识，自己也在工作过程中自学Java、Python、React、Vue，保证自己对软件开发和产品设计有更深刻的理解。

3. 关于博客内容

基于我的工作经历，我的博客内容涵盖的内容主要有产品设计（AI方向、2B、中台产品）、软件开发（Java、Python、Vue），感觉挖的坑有点大，那就慢慢填吧。

知乎 @落霞孤鹜

五、项目代码

项目的代码按照章节的代码进行的提交，能从提交记录中看到每一个模块是如何添加进去的。

个人博客地址：[微谈小智 \(longair.cn\)](http://www.longair.cn)

前端代码地址：[gitee.com/Zhou_Jimmy/bl...](https://gitee.com/Zhou_Jimmy/)

▲ 赞同 1 ▼

● 添加评论

↗ 分享

★ 收藏

⚑ 举报

收起 ^

▲ 赞同 1 ▼

2 条评论

分享

收藏

举报

收起 ^

Vue3+TypeScript+Django Rest Framework 搭建个人博客（三）：博客管理后台（3）



落霞孤鹜，求知若渴

1 人赞同了该文章

一个完整的网站都是有前台和管理后台组成的，前台用来给真正的用户浏览和使用，后台用来给管理员管理网站内容，配置各种功能和数据等。博客的管理后台就是用来承载创建博客，发布博客，查看留言，管理博客用户这些功能的子系统。

大家好，我是 落霞孤鹜，上一篇我们已经实现了管理后台的前端部分页面，这一章我们继续搭建博客的管理后台的前端，实现对博客网站的管理功能。

一、前端开发

1.4 分类和文章管理

文章和分类是关系比较密切的两个业务对象，因此这里把分类管理的功能和文章管理的功能放在同一个页面处理。

1.4.1 Type 层

在 `src/types/index.ts` 文件中增加代码如下：

```
export interface Catalog {
    id: number,
    name: string,
    parent: number,
    parents: Array<number>,
    children: Array<Catalog>
}
```

```
export interface Article {
```

▲ 赞同 1 ▼

添加评论

分享

收藏

举报

收起 ^



```
excerpt: string,
markdown: string,
html: string,
create_at: string,
views: number,
likes: number,
comments: number,
words: number,
tags: Array<number> | any,
tags_info: Array<Tag> | any
catalog: number,
catalog_info: Catalog,
created_at: string,
modified_at: string,
author: string,
status?: string,
}

export interface ArticleArray {
  count: number,
  results: Array<Article> | any
}

export interface ArticleParams {
  title: string | any,
  status: string | any,
  tags: Array<number> | any,
  catalog: number | any,
  page: number,
  page_size: number,
}
```

1.4.2 API 层

这里要编写标签管理相关的接口，列表查询、新增、修改、删除。在 `src/api/service.ts` 编写如下代码：

```
export function getCatalogTree() {
  return request({
```

▲ 赞同 1

▼

● 添加评论

↗ 分享

★ 收藏

⚑ 举报

收起 ^

专栏
Python

```
export function saveCatalog(method: string, data: Catalog) {
    let url = '/catalog/'
    if(['put', 'patch'].includes(method)) {
        url += data.id + '/'
    }
    // @ts-ignore
    return request({
        url,
        method,
        data,
    }) as unknown as ResponseData
}

export function deleteCatalog(catalogId: number) {

    return request({
        url: '/catalog/' + catalogId + '/',
        method: 'delete',
    }) as unknown as ResponseData
}

export function getArticleList(params: ArticleParams) {
    return request({
        url: '/list/',
        method: 'get',
        params
    }) as unknown as ArticleArray
}

export function remoteDeleteArticle(articleId: number) {
    return request({
        url: '/article/' + articleId + '/',
        method: 'delete',
    }) as unknown as ResponseData
}

export function getArticleDetail(articleId: number) {
    return request({
```

▲ 赞同 1



● 添加评论

↗ 分享

★ 收藏

⚑ 举报

收起 ^



```

export function remoteSaveArticle(method: string, data: Article) {
  let url = '/article/'
  if (['put', 'patch'].includes(method)) {
    url += data.id + '/'
  }
  // @ts-ignore
  return request({
    url,
    method,
    data,
  }) as unknown as Article
}

export function remotePublishArticle(articleId: number) {

  // @ts-ignore
  return request({
    url: '/publish/' + articleId + '/',
    method: 'patch',
  }) as unknown as Article
}

export function remoteOfflineArticle(articleId: number) {
  return request({
    url: '/offline/' + articleId + '/',
    method: 'patch',
  }) as unknown as Article
}

```

1.4.3 Component 层

提供一个管理分类的抽屉组件，因此在 `src/components` 下创建文件 `CatalogTree.vue`，编写代码如下：

```

<template>
<el-drawer
  v-model="state.visible"
  :before-close="handleClose"
  >
  <div>
    <ul>
      <li><el-tree></el-tree></li>
    </ul>
  </div>
</el-drawer>

```

知乎

专栏
Python

```
>
<div class="drawer-content">
  <el-tree
    :data="state.catalogs"
    :expand-on-click-node="false"
    :props="defaultProps"
    default-expand-all
    node-key="id">
    <template #default="{ node, data }">
      <span class="custom-tree-node">
        <span>{{ node.label }}</span>
        <span>
          <el-dropdown trigger="click">
            <span class="el-dropdown-link">
              <i class="el-icon-more"/>
            </span>
            <template #dropdown>
              <el-dropdown-menu>
                <el-dropdown-item icon="el-icon-edit">
                  <a class="more-button" @click.prevent="showEditDialog(data)">
                    修改
                  </a>
                </el-dropdown-item>
                <el-dropdown-item icon="el-icon-circle-plus">
                  <a class="more-button" @click.prevent="showAddDialog(data)">
                    新增
                  </a>
                </el-dropdown-item>
                <el-dropdown-item icon="el-icon-delete-solid">
                  <el-popconfirm :title="'确定删除【' + data.name + '】？'" cancelBut
                    icon="el-icon-info" iconColor="red" @confirm="'
                    <template #reference>
                      <a class="more-button">
                        删除
                      </a>
                    </template>
                  </el-popconfirm>
                </el-dropdown-item>
              </el-dropdown-menu>
            </template>
          </el-dropdown>
        </span>
      </span>
    </template>
  </el-tree>
</div>
```

▲ 赞同 1

▼

● 添加评论

↗ 分享

★ 收藏

⚑ 举报

收起 ^

知乎

专栏
Python

```
</div>
</el-drawer>
<el-dialog v-model="state.showDialog" :title="state.dialogTitle">
  <el-form class="form" label-suffix=":" label-width="120px" size="medium">
    <el-form-item label="目录名称">
      <el-input v-model="state.catalog.name" autocomplete="off"></el-input>
    </el-form-item>
  </el-form>
  <template #footer>
    <span class="dialog-footer">
      <el-button @click="state.showDialog=false">取消</el-button>
      <el-button :loading="state.loading" type="primary" @click="saveCatalog">
    </span>
  </template>
</el-dialog>
</template>

<script lang="ts">
import {defineComponent, reactive} from "vue";
import {Catalog} from "../types";
import {deleteCatalog, getCatalogTree, saveCatalog} from "../api/service";
import {ElMessage} from "element-plus";

export default defineComponent({
  name: "CatalogTree",
  props: {
    visible: {
      type: Boolean,
      require: true,
    }
  },
  watch: {
    '$props.visible': {
      handler(val, oldVal) {
        if (val != oldVal) {
          this.state.visible = val
        }
      }
    }
  },
  emits: ['close'],
}
```

▲ 赞同 1

▼

● 添加评论

↗ 分享

★ 收藏

⚑ 举报

收起 ^

知乎

专栏
Python

```
showDialog: false,
catalog: {} as Catalog,
dialogTitle: '',
loading: false,
})

const handleSearch = async () => {
  state.catalogs = await getCatalogTree();
}

const defaultProps = {
  children: 'children',
  label: 'name',
}

return {
  state,
  handleSearch,
  defaultProps
}
},
methods: {
  handleClose() {
    this.$emit('close')
  },
  showAddDialog(data: Catalog) {
    this.state.showDialog = true
    // @ts-ignore
    this.state.catalog.id = undefined
    // @ts-ignore
    this.state.catalog.name = undefined
    this.state.catalog.parent = data.id
    this.state.dialogTitle = '新增目录'
  },
  showEditDialog(data: Catalog) {
    this.state.showDialog = true
    this.state.catalog = data
    this.state.dialogTitle = '修改目录'
  },
  async saveCatalog() {
    try {
      this.state.loading = true
      const method = this.state.catalog.id ? 'patch' : 'post'
```

▲ 赞同 1

▼

● 添加评论

↗ 分享

★ 收藏

⚑ 举报

收起 ^

知乎

专栏
Python

```
        message: '保存成功',
        type: 'success'
    })
    await this.handleSearch()
} catch (e) {
    console.error(e)
    ElMessage({
        message: '保存失败',
        type: 'error'
    })
    this.state.loading = false
}
},
async remove(data: Catalog) {
    await deleteCatalog(data.id)
    ElMessage({
        message: '删除成功',
        type: 'success'
    })
    await this.handleSearch()
}
}

})
</script>

<style lang="less" scoped>
.drawer-content {
    padding: 12px 0 0 24px;
    border-top: #eeeeee 1px solid;
    overflow: auto;
}

.custom-tree-node {
    flex: 1;
    display: flex;
    align-items: center;
    justify-content: space-between;
    font-size: 14px;
    padding-right: 32px;
}

```

▲ 赞同 1

▼

● 添加评论

↗ 分享

★ 收藏

⚑ 举报

收起 ^

```
</style>
```

由于文章管理的界面需要有Markdown编辑器，因此安装markdown编辑器的依赖

```
yarn add @kangc/v-md-editor@2.3.5
yarn add highlight.js@10.7.2
```

在 main.ts 中增加编辑器的 js、css 和插件

```
import { createApp } from 'vue'
import App from './App.vue'
import router from "./router";
import { StateKey, store } from "./store";
import 'element-plus/lib/theme-chalk/index.css';
import 'element-plus/lib/theme-chalk/base.css';

// @ts-ignore
import VMdEditor from '@kangc/v-md-editor';
import '@kangc/v-md-editor/lib/style/base-editor.css';
// @ts-ignore
import githubTheme from '@kangc/v-md-editor/lib/theme/github.js';
import '@kangc/v-md-editor/lib/theme/style/github.css';

// highlightjs
import hljs from 'highlight.js';

VMdEditor.use(githubTheme, {
  Hljs: hljs,
});
import {
  ElAffix,
  ElButton,
  ElCard,
  ElCascader,
  ElCol,
  ElDescriptions,
  ElDescriptionsItem,
```

▲ 赞同 1

▼

● 添加评论

↗ 分享

★ 收藏

⚑ 举报

收起 ^

知乎

专栏
Python

```
ElDropdownMenu,  
ElForm,  
ElFormItem,  
ElIcon,  
ElInput,  
ElLoading,  
ElMenu,  
ElMenuItem,  
ElMessage,  
ElMessageBox,  
ElOption,  
ElPagination,  
ElPopconfirm,  
ElProgress,  
ElRow,  
ElSelect,  
ElTable,  
ElTableColumn,  
ElTag,  
ElTimeline,  
ElTimelineItem,  
ElTooltip,  
ElTree,  
ElUpload,  
} from 'element-plus';  
  
const app = createApp(App)
```

```
const components = [  
  ElAffix,  
  ElButton,  
  ElCard,  
  ElCascader,  
  ElCol,  
  ElDescriptions,  
  ElDescriptionsItem,  
  ElDialog,  
  ElDrawer,  
  ElDropdown,  
  ElDropdownItem,
```

▲ 赞同 1



● 添加评论

↗ 分享

★ 收藏

⚑ 举报

收起 ^



```
ElInput,
ElLoading,
ElMenu,
ElMenuItem,
ElMessage,
ElMessageBox,
ElOption,
ElPagination,
ElPopconfirm,
ElProgress,
ElRow,
ElSelect,
ElTable,
ElTableColumn,
ElTag,
ElTimeline,
ElTimelineItem,
ElTooltip,
ElTree,
ElUpload,
]

const plugins = [
  ElLoading,
  ElMessage,
  ElMessageBox,
]

components.forEach(component => {
  app.component(component.name, component)
})

plugins.forEach(plugin => {
  app.use(plugin)
})

app.use(router).use(store, StateKey).use(VMdEditor).mount('#app')
```

提供一个编辑文章的抽屉组件，因此在 `src/components` 下创建文件 `EditArticle.vue`，编写代码如下：

▲ 赞同 1 ▾ 添加评论 分享 收藏 举报

收起 ^

知乎

专栏
Python

```
v-model="state.visible"
:before-close="handleClose"
:title="articleId?'修改文章':'新增文章'"
direction="rtl"
size="800px"
@opened="handleSearch"
>
<div class="article-form" style="overflow-y: auto">
<el-form label-suffix=":" label-width="120px">
  <el-form-item label="标题">
    <el-input ref="articleTitle" v-model="state.article.title"></el-input>
  </el-form-item>
  <el-form-item label="所属分类">
    <el-cascader v-model="state.catalogs" :options="state.catalogTree"
      :props="{ checkStrictly: true, value:'id',label:'name',expandTrigger: 'click' }"
      clearable
      size="medium"
      style="width: 100%"/>
  </el-form-item>
  <el-form-item label="标签">
    <el-select v-model="state.article.tags" clearable multiple placeholder="请选择标签"
      style="width: 100%">
      <el-option v-for="s in state.tags" :label="s.name" :value="s.id" :key="s.id"></el-option>
    </el-select>
  </el-form-item>
  <el-form-item label="摘要">
    <el-input v-model="state.article.excerpt" :rows="5" type="textarea"></el-input>
  </el-form-item>
  <el-form-item label="正文">
    <v-md-editor v-model="state.article.markdown" height="600px"></v-md-editor>
  </el-form-item>
  <el-form-item label="封面">
    <el-upload
      :before-upload="beforeAvatarUpload"
      :headers="csrfToken"
      :on-success="handleAvatarSuccess"
      :show-file-list="false"
      action="/api/upload/"
      class="avatar-uploader">
    </el-upload>
    
  </el-form-item>
</el-form>
</div>
```

▲ 赞同 1



● 添加评论

↗ 分享

★ 收藏

☞ 举报

收起 ↗

知乎

专栏
Python

```
</el-form>
</div>
<div class="demo-drawer__footer">
    <el-button @click="handleClose">取消</el-button>
    <el-button :loading="state.loading" type="primary" @click="saveArticle">保存
</div>
</el-drawer>

</template>

<script lang="ts">
import {defineComponent, reactive} from "vue";
import {getArticleDetail, getCatalogTree, getTagList, remoteSaveArticle} from "...",
import {Article, Catalog, Tag, TagList} from "../types";
import {getCookie} from "../utils";

export default defineComponent({
    name: "EditArticle",
    props: {
        articleId: {
            type: Number,
            require: true,
            default: undefined,
        },
        visible: {
            type: Boolean,
            require: true,
        }
    },
    watch: {
        '$props.visible': {
            handler(val: Boolean, oldVal: Boolean) {
                if (val !== oldVal) {
                    this.state.visible = val
                }
            }
        }
    },
    emits: ["close",],
    setup(props, context) {
        const state = reactive({
```

▲ 赞同 1



● 添加评论

↗ 分享

★ 收藏

⚑ 举报

收起 ^

专栏
Python

```
tags: [] as Array<Tag>,
catalogs: [] as Array<number>
})

const saveArticle = async () => {
try {
  state.loading = true
  if (state.catalogs.length) {
    state.article.catalog = state.catalogs[state.catalogs.length - 1]
  }
  if (props.articleId) {
    await remoteSaveArticle('put', state.article)
  } else {
    await remoteSaveArticle('post', state.article)
  }
  state.loading = false
  context.emit('close', true)
} catch (e) {
  state.loading = false
}
}

const csrfToken = {'X-CSRFToken': getCookie('csrftoken')}
return {
  state, saveArticle, csrfToken
}
},
methods: {
  async handleSearch() {
    this.$refs.articleTitle.focus()
    if (this.$props.articleId) {
      this.state.article = await getArticleDetail(this.$props.articleId)
      this.state.article.tags = this.state.article.tags_info.map((tag: Tag) => {
        this.state.catalogs = this.state.article.catalog_info.parents
      } else {
        this.state.article = {} as Article
      }
      this.state.catalogTree = await getCatalogTree()

      if (!this.state.tags.length) {

```

▲ 赞同 1

▼

● 添加评论

↗ 分享

★ 收藏

☞ 举报

收起 ^

知乎

专栏
Python

```
handleClose(done: any) {
    this.$confirm('确认关闭抽屉?', '提示', {
        confirmButtonText: '关闭',
        cancelButtonText: '取消',
        type: 'warning'
    })
        .then(_: any): void => {
            this.$emit("close", false)
            this.state.article = {} as Article
            done();
        })
        .catch(_: any): void => {
            console.error(_)
        });
},
handleAvatarSuccess(res: any, file: File) {
    this.state.article.cover = res.url
},
beforeAvatarUpload(file: File) {
    const isImage = ['image/jpeg', 'image/png', 'image/gif', 'image/jpg'].includes(file.type);
    const isLt2M = file.size / 1024 / 1024 < 2;

    if (!isImage) {
        this.$message.error('上传图片只能是 JPG 格式!');
    }
    if (!isLt2M) {
        this.$message.error('上传图片大小不能超过 2MB!');
    }
    return isImage && isLt2M;
}
}
)
</script>

<style lang="less">
.article-form {
    padding: 24px;
    overflow-y: auto;
    border-top: 1px solid #e8e8e8;
    height: calc(100% - 100px);
}
```

▲ 赞同 1

▼

添加评论

分享

收藏

举报

收起 ^

知乎

专栏
Python

```
margin-bottom: 50px ;  
height: 100% !important;  
}  
  
.el-drawer__header{  
margin-bottom: 16px;  
}  
.demo-drawer__footer {  
width: 100%;  
position: absolute;  
bottom: 0;  
left: 0;  
border-top: 1px solid #e8e8e8;  
padding: 10px 16px;  
text-align: right;  
background-color: white;  
}  
  
//抽屉//去掉element-ui的drawer标题选中状态  
  
:deep(:focus){  
outline: 0;  
}  
  
.avatar-uploader {  
background-color: #fbfdff;  
border: 1px dashed #c0ccda;  
border-radius: 6px;  
box-sizing: border-box;  
width: 125px;  
height: 100px;  
cursor: pointer;  
line-height: 100px;  
text-align: center;  
font-size: 20px;  
}  
  
</style>
```



▲ 赞同 1



● 添加评论

↗ 分享

★ 收藏

⚑ 举报

收起 ^

```
<template>
  <div>
    <div>
      <el-form :inline="true" class="demo-form-inline">
        <el-form-item label="标题">
          <el-input ref="title" v-model="state.params.title" placeholder="文章标题">
        </el-form-item>
        <el-form-item label="状态">
          <el-select v-model="state.params.status" placeholder="状态">
            <el-option label="已发布" value="Published"/>
            <el-option label="草稿" value="Draft"/>
          </el-select>
        </el-form-item>
        <el-form-item>
          <el-button :loading="state.isLoading" type="primary" @click="handleSearch">搜索</el-button>
        </el-form-item>
      </el-form>
    </div>
    <div class="button-container">
      <el-button :loading="state.isLoading" type="primary" @click="showAddDrawer">添加文章</el-button>
      <el-button circle icon="el-icon-s-unfold" @click="state.showCatalogTree=true">目录</el-button>
    </div>
    <div>
      <el-table ref="articleTable" :data="state.articleList" :header-cell-style="{'text-align': 'center'}>
        <el-table-column type="selection" width="55"/>
        <el-table-column label="ID" prop="id" width="80"/>
        <el-table-column label="标题" prop="title" width="200"/>
        <el-table-column label="状态" prop="status" width="100"/>
        <el-table-column label="分类" prop="catalog_info.name"/>
        <el-table-column :formatter="datetimeFormatter" label="修改时间" prop="modified_at"/>
        <el-table-column fixed="right" label="操作" width="120">
          <template #default="scope">
            <el-popconfirm cancelButtonText='取消' confirmButtonText='删除' icon="el-icon-delete" title="确定删除该文章吗？" @confirm="deleteArticle(scope.row)">
              <template #reference>
                <el-button size="small" type="text">删除</el-button>
              </template>
            </el-popconfirm>
          </template>
        </el-table-column>
      </el-table>
    </div>
  </div>

```

知乎

专栏
Python

编辑

```
</el-button>
<el-button v-if="scope.row.status==='草稿'" size="small" type="text"
           @click.prevent="publishArticle(scope.$index, scope.row)">
```

发布

```
</el-button>
<el-button v-else size="small" type="text"
           @click.prevent="offlineArticle(scope.$index, scope.row)">
```

下线

```
</el-button>
</template>
```

```
</el-table-column>
```

```
</el-table>
```

```
</div>
```

```
<div class="pagination">
```

```
  <el-pagination :page-size="10" :total="state.total" background
                 layout="prev, pager, next"></el-pagination>
```

```
</div>
```

```
</div>
```

```
<EditArticle
```

```
  :article-id="state.articleId"
```

```
  :visible="state.showDrawer"
```

```
  @close="handleCloseDrawer"
```

```
/>
```

```
<CatalogTree
```

```
  :visible="state.showCatalogTree"
```

```
  @close="state.showCatalogTree=false"
```

```
/>
```

```
</template>
```

```
<script lang="ts">
```

```
import {defineComponent, reactive} from "vue";
```

```
import {Article, ArticleArray, ArticleParams} from "../../types";
```

```
import {getArticleList, remoteDeleteArticle, remoteOfflineArticle, remotePublishAi}
```

```
import {timestampToTime} from "../../utils";
```

```
import {ElMessage} from "element-plus";
```

```
import EditArticle from "../../components/EditArticle.vue";
```

```
import CatalogTree from "../../components/CatalogTree.vue";
```

```
export default defineComponent({
```

▲ 赞同 1

▼

添加评论

分享

收藏

举报

收起 ^

知乎

专栏
Python

```
articleList: [] as Array<Article>,
params: {
    title: undefined,
    status: undefined,
    tags: undefined,
    catalog: undefined,
    page: 1,
    page_size: 10,
} as ArticleParams,
isLoading: false,
total: 0,
showDrawer: false,
articleId: 0,
showCatalogTree: false,
});

const handleSearch = async (): Promise<void> => {
    state.isLoading = true;
    try {
        const data: ArticleArray = await getArticleList(state.params);
        state.isLoading = false;
        state.articleList = data.results;
        state.total = data.count
    } catch (e) {
        console.error(e)
        state.isLoading = false;
    }
};

const publishArticle = async (index: number, row: Article) => {
    try {
        await remotePublishArticle(row.id)
        ElMessage({
            message: "发布成功！",
            type: "success",
        });
        await handleSearch()
    } catch (e) {
        console.error(e)
    }
}
```

▲ 赞同 1

▼

添加评论

分享

收藏

举报

收起 ^

知乎

专栏
Python

```
ElMessage({
    message: "下线成功!",
    type: "success",
});
await handleSearch()
} catch (e) {
    console.error(e)
}
}

const deleteArticle = async (index: number, row: Article) => {
    await remoteDeleteArticle(row.id);
    ElMessage({
        message: "删除成功!",
        type: "success",
    });
    await handleSearch()
}

const datetimeFormatter = (row: Article, column: number, cellValue: string, index: number) => {
    return timestampToTime(cellValue, true);
}

handleSearch()

const handleCloseDrawer = (isOk: boolean) => {
    state.showDrawer = false
    if (isOk) {
        handleSearch()
    }
}
return {
    state,
    handleSearch,
    datetimeFormatter,
    deleteArticle,
    handleCloseDrawer,
    publishArticle,
    offlineArticle
}
},
```

▲ 赞同 1

▼

添加评论

分享

收藏

举报

收起 ^



```
showEditDrawer(index: number, row: Article) {
    this.$refs.articleTable.setCurrentRow(row)
    this.state.showDrawer = true;
    this.state.articleId = row.id
},
showAddDrawer() {
    this.state.showDrawer = true;
    this.state.articleId = 0;
}
})
</script>

<style scoped>
.pagination {
    text-align: right;
    margin-top: 12px;
}
</style>
```

1.4.5 Router 层

定义 route 来完成路由跳转。在 src/route/index.ts 文件中新增代码：

```
import { createRouter, createWebHistory, RouteRecordRaw } from "vue-router";
import Home from "../views/client/Home.vue";

const routes: Array<RouteRecordRaw> = [
{
    path: "/",
    name: "Home",
    component: Home,
    meta: {}
},
{
    path: "/login/",
    name: "Login",
    component: () =>
```



```
name: 'Admin',
component: () => import("../views/admin/Admin.vue"),
children: [
  {
    path: '/admin/',
    name: 'Dashboard',
    component: () => import("../views/admin/Dashboard.vue"),
  },
  {
    path: '/admin/dashboard',
    name: 'AdminDashboard',
    component: () => import("../views/admin/Dashboard.vue"),
  },
  {
    path: '/admin/user',
    name: 'UserManagement',
    component: () => import("../views/admin/User.vue"),
  },
  {
    path: '/admin/tag',
    name: 'Tag',
    component: () => import("../views/admin/Tag.vue"),
  },
  {
    path: '/admin/article',
    name: 'ArticleManagement',
    component: () => import("../views/admin/Article.vue"),
  },
],
},
],
]

const router = createRouter({
  history: createWebHistory(import.meta.env.BASE_URL),
  routes,
});

export default router;
```

```
'/upload': {  
    target: 'http://localhost:8000/',  
    changeOrigin: true,  
    ws: false,  
    rewrite: (pathStr) => pathStr.replace('/api', ''),  
    timeout: 5000,  
},
```

1.5 评论管理

15.1 Type 层

在 `src/types/index.ts` 文件中增加代码如下：

```
export interface CommentInfo {  
    id: number,  
    user: number,  
    user_info: User | any,  
    article: number,  
    article_info: Article | any,  
    created_at: string,  
    reply: number | any,  
    content: string,  
    comment_replies: CommentInfo | any,  
}  
  
export interface CommentPara {  
    user: number,  
    article: number,  
    reply: number | any,  
    content: string,  
    page: number,  
    page_size: number  
}
```

1.5.2 API 层

▲ 赞同 1 ▼ 添加评论 分享 收藏 举报

收起 ^



```
url: '/comment/',
method: 'get',
params,
}) as unknown as ResponseData
}
```

1.5.3 Component 层

由于评论无需要做修改删除等操作，只有查看评论详情，因此复用文章详情页面。

1.5.4 View 层

通过表格查看评论，在 `src/views/admin` 下新增文件 `Comment.vue` 文件，编写如下代码：

```
<template>
<div>
<div>

<el-form :inline="true" :model="state.params" class="demo-form-inline">
  <el-form-item label="账号">
    <el-select v-model="state.params.user" filterable placeholder="请选择">
      <el-option
        v-for="item in state.userList"
        :key="item.id"
        :label="item.nickname || item.username"
        :value="item.id">
        </el-option>
      </el-select>
    </el-form-item>
    <el-form-item label="内容">
      <el-input v-model="state.params.content" placeholder="评论内容"/>
    </el-form-item>
    <el-form-item>
      <el-button :loading="state.loading" type="primary" @click="handleSearch">
      </el-form-item>
    </el-form>
  </div>
  <div>
    <el-table ref="articleTable" :data="state.commentList" :header-cell-style="`background-color: #f0f0f0;`">
      <thead>
        <tr>
          <th>评论 ID</th>
          <th>用户名</th>
          <th>评论内容</th>
          <th>操作</th>
        </tr>
      </thead>
      <tbody>
        <tr v-for="comment in state.commentList" :key="comment.id">
          <td>{{comment.id}}</td>
          <td>{{comment.user.username}}</td>
          <td>{{comment.content}}</td>
          <td>
            <el-button type="text" @click="handleEdit(comment)">编辑</el-button>
            <el-button type="text" @click="handleDelete(comment)">删除</el-button>
          </td>
        </tr>
      </tbody>
    </el-table>
  </div>
</div>
```

知乎

专栏
Python

```
<el-table-column label="评论内容" prop="content" width="200"/>
<el-table-column label="文章" prop="article_info.title"/>
<el-table-column label="回复评论" prop="reply.id" width="200"/>
<el-table-column :formatter="datetimeFormatter" label="评论时间" prop="create_time"/>
<el-table-column label="操作">
  <template #default="scope">
    <el-button size="small" type="text"
      @click.prevent="showDetail(scope.row)">
      详情
    </el-button>
  </template>
</el-table-column>
</el-table>
</div>
<div class="pagination">
  <el-pagination :page-size="10" :total="state.total" background
    layout="prev, pager, next"></el-pagination>
</div>
</div>
</template>

<script lang="ts">
import {defineComponent, reactive} from "vue";
import {Article, CommentInfo, CommentPara, ResponseData, User} from "../../types";
import {ElMessage} from "element-plus";
import {timestampToTime} from "../../utils";
import {getCommentList, getUserList, saveUser} from "../../api/service";
import UserDetail from "../../components/UserDetail.vue";

export default defineComponent({
  name: "Comment",
  components: {UserDetail},
  setup: function () {
    const state = reactive({
      commentList: [] as Array<CommentInfo>,
      params: {
        user: undefined,
        article: undefined,
        reply: undefined,
        content: '',
        page: 1,
      }
    })
  }
})
```

▲ 赞同 1

▼

● 添加评论

↗ 分享

★ 收藏

⚑ 举报

收起 ^



```
loading: false,
});

const handleSearch = async (): Promise<void> => {
  state.loading = true;
  try {
    const data: ResponseData = await getCommentList(state.params);
    state.loading = false;
    state.commentList = data.results;
    state.total = data.count
  } catch (e) {
    console.error(e)
    state.loading = false;
  }
};

const getUsers = async (): Promise<void> => {
  try {
    const data: ResponseData = await getUserList({});
    state.userList = data.results;
  } catch (e) {
    console.error(e)
  }
};

const datetimeFormatter = (row: Article, column: number, cellValue: string, index: number) => {
  return timestampToTime(cellValue, true);
}

handleSearch()
getUsers()
return {
  state,
  handleSearch,
  datetimeFormatter,
}
,
methods: {
  showDetail(row: CommentInfo) {
    const {href} = this.$router.resolve({
      path: '/comment',
      query: {id: row.id}
    });
    this.$router.push(href);
  }
}
```

```
        });
        window.open(href, "_blank");
    },
}
})
</script>

<style scoped>
.pagination {
    text-align: right;
    margin-top: 12px;
}
</style>
```

1.5.5 Router 层

定义 route 来完成路由跳转。在 src/route/index.ts 文件中新增代码：

```
import { createRouter, createWebHistory, RouteRecordRaw } from "vue-router";
import Home from "../views/client/Home.vue";

const routes: Array<RouteRecordRaw> = [
{
    path: "/",
    name: "Home",
    component: Home,
    meta: {}
},
{
    path: "/login/",
    name: "Login",
    component: () =>
        import(/* webpackChunkName: "login" */ "../views/admin/Login.vue")
},
{
    path: '/admin',
    name: 'Admin',
    component: () => import(/* webpackChunkName: "admin" */ "../views/admin/Admin.vue")
}
```

▲ 赞同 1



● 添加评论



分享



收藏



举报

收起 ^

知乎

专栏
Python

```
component: () => import("../views/admin/Dashboard.vue"),
},
{
  path: '/admin/dashboard',
  name: 'AdminDashboard',
  component: () => import("../views/admin/Dashboard.vue"),
},
{
  path: '/admin/user',
  name: 'UserManagement',
  component: () => import("../views/admin/User.vue"),
},
{
  path: '/admin/tag',
  name: 'Tag',
  component: () => import("../views/admin/Tag.vue"),
},
{
  path: '/admin/article',
  name: 'ArticleManagement',
  component: () => import("../views/admin/Article.vue"),
},
{
  path: '/admin/comment',
  name: 'CommentManagement',
  component: () => import("../views/admin/Comment.vue"),
},
],
},
]

const router = createRouter({
  history: createWebHistory(import.meta.env.BASE_URL),
  routes,
});

export default router;
```

▲ 赞同 1 ▼

● 添加评论

↗ 分享

★ 收藏

⚑ 举报

收起 ^

在 `src/types/index.ts` 文件中增加代码如下：

```
export interface NumberInfo {
    views: number,
    likes: number,
    comments: number,
    messages: number
}
```

1.6.2 API 层

这里要编写标签管理相关的接口，列表查询、新增、修改、删除。在 `src/api/service.ts` 编写如下代码：

```
export function getTopArticleList() {
    return request({
        url: '/top/',
        method: 'get',
    }) as unknown as ResponseData
}

export function getNumbers() {
    return request({
        url: '/number/',
        method: 'get',
    }) as unknown as NumberInfo
}
```

1.6.3 Component 层

无需提供额外的组件。

1.6.4 View 层

通过图标和指标卡的形式展示网站的整体情况，修改 `src/views/admin/Dashboard.vue`，编写如下代码：

知乎

专栏
Python

```
<div class="title">今日博客访问情况</div>
<el-row :gutter="24" class="numbers">
  <el-col :span="6" class="el-col-6">
    <el-card>
      <div class="number-card">
        <div>
          <i class="el-icon-user number-icon"></i>
        </div>
        <div class="number-right">
          <div class="number-num">{{ state.numbers.views }}</div>
          <div>用户访问量</div>
        </div>
      </div>
    </el-card>
  </el-col>
  <el-col :span="6" class="el-col-6">
    <el-card>
      <div class="number-card">
        <div>
          <i class="el-icon-thumb number-icon" style="background: #64d572;"></i>
        </div>
        <div class="number-right">
          <div class="number-num">{{ state.numbers.likes }}</div>
          <div>点赞量</div>
        </div>
      </div>
    </el-card>
  </el-col>
  <el-col :span="6" class="el-col-6">
    <el-card>
      <div class="number-card">
        <div>
          <i class="el-icon-chat-line-square number-icon" style="background: #e6a23c;"></i>
        </div>
        <div class="number-right">
          <div class="number-num">{{ state.numbers.comments }}</div>
          <div>评论量</div>
        </div>
      </div>
    </el-card>
  </el-col>
</el-row>
```

▲ 赞同 1

▼

● 添加评论

↗ 分享

★ 收藏

⚑ 举报

收起 ^

知乎

专栏
Python

```
<div>
    <i class="el-icon-message number-icon" style="background-color: #42f4b1; color: white; font-size: 16px; width: 24px; height: 24px; border-radius: 50%; display: flex; align-items: center; justify-content: center; margin-right: 10px;"></i>
    <div class="number-right">
        <div class="number-num">{{ state.numbers.messages }}</div>
        <div>留言量</div>
    </div>
</div>
</el-card>
</el-col>
</el-row>
<div class="top-articles">
    <el-card>
        <template #header>
            文章访问量TOP10
        </template>
        <div class="article-list">
            <div v-for="( article,index ) in state.articleList" class="article" @click="handleArticleClick( article )">
                <span style="font-size: 14px">{{ index + 1 + '.' + article.title }}</span>
                <span style="color: #999999; font-size: 14px">{{ article.views }} / {{ article.likes }}</span>
            </div>
        </div>
    </el-card>
</div>
</div>
</template>

<script lang="ts">
import { defineComponent, reactive } from "vue";
import { Article } from "../../types";
import { getNumbers, getTopArticleList } from "../../api/service";

export default defineComponent({
    name: "Dashboard",
    setup() {
        const state = reactive({
            numbers: {
                views: 0,
                likes: 0,
                comments: 0,
            }
        })
    }
})
```

▲ 赞同 1

▼

● 添加评论

↗ 分享

★ 收藏

⚑ 举报

收起 ^

知乎

专栏
Python

```
return {  
    state,  
}  
,  
  
async mounted() {  
    this.state.articleList = (await getTopArticleList()).results  
    this.state.numbers = await getNumbers()  
},  
  
methods: {  
    viewArticle(id: number) {  
        const {href} = this.$router.resolve({  
            path: '/article/',  
            query: {  
                id  
            }  
        });  
        window.open(href, "_blank");  
    }  
}  
}  
</script>  
  
<style lang="less" scoped>  
.numbers {  
    width: 100%;  
}  
  
.title {  
    color: #999;  
    margin: 12px 0;  
    padding-left: 8px;  
    font-size: 14px;  
}  
  
:deep(.el-card__body){  
    margin: 0;  
    padding: 0;  
}
```

▲ 赞同 1



● 添加评论

↗ 分享

★ 收藏

⚑ 举报

收起 ^

知乎

专栏
Python

```
display: -webkit-box;
display: -ms-flexbox;
display: flex;
flex: 1;
-webkit-box-align: center;
-ms-flex-align: center;
align-items: center;
height: 80px;
border: 1px solid #ebeef5;
background-color: #fff;
border-radius: 4px;
overflow: hidden;
}

.number-right {
-webkit-box-flex: 1;
-ms-flex: 1;
flex: 1;
text-align: center;
font-size: 14px;
color: #999;
}

.number-num {
font-size: 30px;
font-weight: 700;
color: #2d8cf0;
text-align: center;
}

.number-icon {
font-size: 50px;
width: 80px;
height: 80px;
text-align: center;
line-height: 80px;
color: #fff;
background: #2d8cf0;
}
```

▲ 赞同 1



● 添加评论

↗ 分享

★ 收藏

⚑ 举报

收起 ^



```
.article-list {  
    padding: 20px;  
}  
  
.article {  
    cursor: pointer;  
    display: flex;  
    flex: 1;  
    justify-content: space-between;  
    padding: 12px 24px 12px 12px;  
    border-top: #eeeeee 1px solid;  
}  
  
.article:first-child {  
    border-top: none;  
    padding-top: 0;  
}  
  
.article:last-child {  
    padding-bottom: 0;  
}  
  
.dashboard-list {  
    display: flex;  
    flex: 1;  
    justify-content: space-evenly;  
    padding: 24px;  
    margin-right: 24px;;  
}  
  
.percentage-value {  
    display: block;  
    margin-top: 10px;  
    font-size: 28px;  
}  
  
.percentage-label {  
    display: block;  
    margin-top: 10px;  
    font-size: 12px;  
}
```

▲ 赞同 1● 添加评论↗ 分享★ 收藏⚑ 举报收起 ^

管理后台已经开发完成，因此需要在路由中做好权限控制，当访问admin路径的时候，需要判断用户是否登录，且用户是否是管理员，因此在 `src/router/index.ts` 中增加如下代码：

```
router.beforeEach((to, from, next) => {
  if (/^\/admin/.test(to.path)
    && (!store.state.user.id ||
        store.state.user.role !== 'Admin')) {
    next('/login')
    return
  }
  next()
})
```

在 `src/views/admin/Login.vue` 中第143行后增加一行代码：

```
is_superuser: data.is_superuser
```

至此管理后台的前端开发完成

二、前端效果

2.1 前端管理后台页面效果



专栏
Python

2.2 前端代码结构

[▲ 赞同 1](#)[● 添加评论](#)[↗ 分享](#)[★ 收藏](#)[⚑ 举报](#)[收起 ^](#)

知乎

专栏
Python

```
> .vscode
> node_modules
> public
< src
  < api
    TS index.ts
    TS service.ts
  > assets
  < components
    ▼ CatalogTree.vue
    ▼ EditArticle.vue
    ▼ HelloWorld.vue
    ▼ Nav.vue
    ▼ RegisterAndLogin.vue
    ▼ TagEditDialog.vue
    ▼ UserDetail.vue
  < less
    {} index.less
  < router
    TS index.ts
  < store
    TS index.ts
  < types
    TS index.ts
  < utils
    TS index.ts
  < views
    < admin
      ▼ Admin.vue
      ▼ Article.vue
      ▼ Comment.vue
      ▼ Dashboard.vue
      ▼ Login.vue
      ▼ Tag.vue
      ▼ User.vue
    < client
      ▼ Home.vue
      ▼ App.vue
    TS main.ts
    TS shims-vue.d.ts
    TS vite-env.d.ts
  ◆ .gitignore
  ▷ index.html
```

▲ 赞同 1



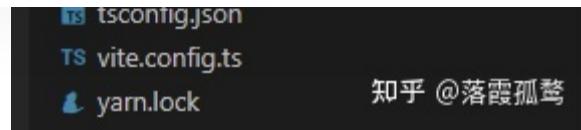
● 添加评论

↗ 分享

★ 收藏

☞ 举报

收起 ^



下一篇我们编写博客网站给用户使用的页面。

发布于 2021-08-24 09:23

▲ 赞同 1 ▾ 添加评论 分享 收藏 举报

收起 ^

Vue3+TypeScript+Django Rest Framework 搭建个人博客（三）：博客管理后台（2）



落霞孤鹜，求知若渴

1 人赞同了该文章

一个完整的网站都是有前台和管理后台组成的，前台用来给真正的用户浏览和使用，后台用来给管理员管理网站内容，配置各种功能和数据等。博客的管理后台就是用来承载创建博客，发布博客，查看留言，管理博客用户这些功能的子系统。

大家好，我是 落霞孤鹜，上一篇我们已经实现了管理后台的后端接口部分，这一章我们开始搭建博客的管理后台的前端，实现对博客网站的管理功能。

一、前端界面开发

一个管理后台的功能，一般都需要从最基础的业务对象的管理开始，在我们的博客网站上，业务对象间的依赖依次是用户、标签、分类、文章、评论、点赞、留言、首页统计。

基于这个依赖关系，我们的后台管理功能也按照这样的逻辑顺序进行构建。然后在构建每一个业务对象的管理页面时，按照 Type 、 API 、 Component 、 View 、 Route 顺序进行组织和代码编写。

在 `src/views` 下创建两个文件夹 `admin` 和 `client`，并把上一个章节中创建的 `Login.vue` 移动到 `admin` 文件夹，把 `Home.vue` 文件移动到 `client` 下。

1.1 菜单管理

▲ 赞同 1 ▾ 添加评论 分享 收藏 举报

收起 ^

```
<template>
  <div class="body">
    <div class="menu">
      <el-menu :default-active="state.activePath" :router="true">
        <el-menu-item index="AdminDashboard" route="/admin/dashboard"><i class="el-icon-s-home"></i> 首页</el-menu-item>
        <el-menu-item index="ArticleManagement" route="/admin/article"><i class="el-icon-newspaper"></i> 文章管理</el-menu-item>
        <el-menu-item index="TagManagement" route="/admin/tag"><i class="el-icon-s-tools"></i> 标签管理</el-menu-item>
        <el-menu-item index="CommentManagement" route="/admin/comment"><i class="el-icon-speech-bubble"></i> 评论管理</el-menu-item>
        <el-menu-item index="UserManagement" route="/admin/user"><i class="el-icon-user"></i> 用户管理</el-menu-item>
      </el-menu>
    </div>
    <div class="view">
      <router-view/>
    </div>
  </div>
</template>

<script>
import {defineComponent, reactive} from "vue";
import {useRoute} from "vue-router";

export default defineComponent({
  name: "Admin",
  setup() {
    const state = reactive({
      activePath: '',
    });
    const route = useRoute()
    if (route.name === 'Dashboard') {
      state.activePath = 'AdminDashboard'
    } else {
      state.activePath = route.name;
    }
    return {
      state,
    }
  }
})
```

```
<style lang="less" scoped>
.body {
  width: 100%;
  height: 100%;
  box-sizing: border-box;
  display: flex;
}

.user {
  font-size: 20px;
}

.menu {
  width: 200px;
}

.view {
  width: calc(100% - 200px);
  padding: 24px;
}

.el-menu {
  height: 100%;
}

</style>
```

3.1.2 Dashboard.vue

为了接下来的开发能很好的开展，我们先处理管理后台的默认页面 Dashboard，在 src/views/admin 下创建文件 Dashboard.vue，编写代码：

```
<template>
  <h3>Dashboard</h3>
</template>

<script lang="ts">
```

专栏
Python

</script>

3.1.3 添加路由

在 src/router/index.ts 下调整代码如下：

```
import {createRouter, createWebHistory, RouteRecordRaw} from "vue-router";
import Home from "../views/client/Home.vue";

const routes: Array<RouteRecordRaw> = [
  {
    path: "/",
    name: "Home",
    component: Home,
    meta: {}
  },
  {
    path: "/login/",
    name: "Login",
    component: () =>
      import("../views/admin/Login.vue")
  },
  {
    path: '/admin',
    name: 'Admin',
    component: () => import("../views/admin/Admin.vue"),
    children: [
      {
        path: '/admin/',
        name: 'Dashboard',
        component: () => import("../views/admin/Dashboard.vue"),
      },
      {
        path: '/admin/dashboard',
        name: 'AdminDashboard',
        component: () => import("../views/admin/Dashboard.vue"),
      },
    ]
  },
]
```

▲ 赞同 1

▼

● 添加评论

↗ 分享

★ 收藏

⚑ 举报

收起 ^



```
routes,  
});  
  
export default router;
```

1.2 用户管理

1.2.1 Type 层

在我们处理登录和注册的时候，已经完成了用户的类型定义，也即 `User` 的interface定义，这里增加所有返回结果的定义，用于管理接口返回的数据结构。在 `src/types/index.ts` 文件中代码如下：

```
export interface User {  
    id: number,  
    username: string,  
    email: string,  
    avatar: string | any,  
    nickname: string | any,  
    is_active?: any,  
    is_superuser?: boolean,  
    created_at?: string,  
}  
  
export interface ResponseData {  
    count: number;  
    results?: any;  
    detail?: string;  
}
```

1.2.2 API 层

这里要编写用户管理相关的接口，列表查询、启用、禁用、详情查看。在 `src/api/service.ts` 编写如下代码：



```

return request({
    url: '/user/' + userId + '/',
    method: 'get',
}) as unknown as User
}

export function saveUser(method: string, data: User) {
// @ts-ignore
return request({
    url: '/user/' + data.id + '/',
    method,
    data,
}) as unknown as ResponseData
}

```

1.2.3 Component 层

在查看用户详情时，我们需要一个抽屉，展示用户的详细信息，因此在 `src/components` 下创建文件 `UserDetail.vue`，编写代码如下：

```

<template>
<el-drawer
  v-model="state.visible"
  :before-close="handleClose"
  direction="rtl"
  size="500px"
  title="用户详情"
  @opened="handleSearch"
>
<el-descriptions :column="1" border class="detail" >
  <el-descriptions-item label="用户名">{{ state.user.username }}</el-descripti
  <el-descriptions-item label="角色">{{ state.user.role }}</el-descriptions-it
  <el-descriptions-item label="状态">{{ state.user.is_active }}</el-descriptio
  <el-descriptions-item label="邮箱">{{ state.user.email }}</el-descriptions-i
  <el-descriptions-item label="创建时间">{{ state.user.created_at }}</el-descr
  <el-descriptions-item label="最后登录时间">{{ state.user.last_login }}</el-de
</el-descriptions>
</el-drawer>
</template>

```

[赞同 1](#)

[添加评论](#)
[分享](#)
[收藏](#)
[举报](#)
[收起 ^](#)



```
import {getUserDetail} from "../api/service";

export default defineComponent({
  name: "UserDetail",
  props: {
    visible: {
      type: Boolean,
      require: true,
    },
    userId: {
      type: Number,
      require: true,
    },
    loading: {
      type: Boolean,
      require: true,
    }
  },
  emits: ["close"],
  watch: {
    '$props.visible': {
      async handler(val: Boolean, oldVal: Boolean) {
        if (val !== oldVal) {
          this.state.visible = val
        }
      }
    }
  },
  setup(props) {
    const state = reactive({
      visible: props.visible as Boolean,
      user: {} as User,
    });
    return {
      state,
    }
  },
  methods: {
    handleClose(isOk: Boolean) {
      this.$emit("close", {
        isOk
      })
    }
  }
})
```

赞同 1添加评论分享收藏举报收起 ^



```

async handleSearch() {
    this.state.user = await getUserDetail(this.$props.userId)
}
})
</script>

<style scoped>
.detail {
    padding: 24px;
    margin-top: -12px;
    border-top: #eeeeee 1px solid;
}
</style>

```

1.2.4 View 层

在用户管理中，我们通过一个表格，分页展示所有的用户信息，并通过表格的操作列，提供查看详情、启用、禁用功能。

在 `src/utils/index.ts` 下增加方法 `timestampToTime`

```
export function timestampToTime(timestamp: Date | any, dayMinSecFlag: boolean) {
```

在 `src/views/admin` 下新增文件 `User.vue`，引用 `UserDetail` 组件，具体代码如下：

```

<template>
<div>
<div>
    <el-form :inline="true" :model="state.params" class="demo-form-inline">
        <el-form-item label="名称">
            <el-input v-model="state.params.name" placeholder="账号"/>
        </el-form-item>
        <el-form-item label="状态">
            <el-select v-model="state.params.is_active" placeholder="请选择">
                <el-option :value="1" label="生效"/>

```

▲ 赞同 1

▼

● 添加评论

↗ 分享

★ 收藏

⚑ 举报

收起 ^

知乎

 专栏
Python

```
<el-button :loading="state.isLoading" type="primary" @click="handleSearch">
</el-form-item>
</el-form>
</div>
<div>
<el-table ref="userTable" :data="state.userList" :header-cell-style="{background-color: #f0f0f0; text-align: center; font-weight: bold; color: black; border-bottom: 1px solid black; width: 100%}" :row-class-name="rowClass">
<el-table-column type="selection" width="55"/>
<el-table-column label="ID" prop="id" width="80"/>
<el-table-column label="账号" prop="username" width="200"/>
<el-table-column label="昵称" prop="nickname" width="200"/>
<el-table-column label="状态" prop="is_active"/>
<el-table-column :formatter="datetimeFormatter" label="注册时间" prop="createTime" width="180"/>
<el-table-column label="操作">
<template #default="scope">
<el-popconfirm v-if="scope.row.is_active" cancelButtonText='取消' confirmButtonText="确定禁用该用户吗？" @confirm="disableUser(scope.$index, scope.row)" style="text-align: center; margin-top: 10px;" >
<template #reference>
<el-button size="small" type="text">
禁用
</el-button>
</template>
</el-popconfirm>
<el-button v-if="!scope.row.is_active" size="small" type="text" style="margin-left: 10px; margin-top: 10px;" @click.native.prevent="enableUser(scope.$index, scope.row)" >
启用
</el-button>
<el-button size="small" type="text" style="margin-left: 10px; margin-top: 10px;" @click.native.prevent="showUserDetail(scope.row)" >
详情
</el-button>
</template>
</el-table-column>
</el-table>
</div>
<div class="pagination">
<el-pagination :page-size="10" :total="state.total" background layout="prev, pager, next"></el-pagination>
</div>
</div>
```

[▲ 赞同 1](#)[添加评论](#)[分享](#)[收藏](#)[举报](#)[收起 ^](#)

知乎

专栏
Python

```
/>  
</template>  
  
<script lang="ts">  
import {defineComponent, reactive} from "vue";  
import {ResponseData, User} from "../../types";  
import {ElMessage} from "element-plus";  
import {timestampToTime} from "../../utils";  
import {getUserList, saveUser} from "../../api/service";  
import UserDetail from "../../components/UserDetail.vue";  
  
export default defineComponent({  
  name: "User",  
  components: {UserDetail},  
  setup: function () {  
    const state = reactive({  
      userList: [] as Array<User>,  
      params: {  
        name: '',  
        role: 'Reader',  
        is_active: undefined,  
        page: 1,  
        page_size: 10,  
      },  
      isLoading: false,  
      total: 0,  
      showDialog: false,  
      userId: 0,  
      saveLoading: false,  
    });  
  
    const handleSearch = async (): Promise<void> => {  
      state.isLoading = true;  
      try {  
        const data: ResponseData = await getUserList(state.params);  
        state.isLoading = false;  
        state.userList = data.results;  
        state.total = data.count  
      } catch (e) {  
        console.error(e)  
        state.isLoading = false;  
      }  
    };  
  },  
  methods: {  
    handleSearch()  
  }  
});
```

知乎

专栏
Python

```
await saveUser('patch', {id: row.id, is_active: false} as User);
ElMessage({
  message: "禁用成功!",
  type: "success",
});
await handleSearch()
}

const enableUser = async (index: number, row: User) => {
  await saveUser('patch', {id: row.id, is_active: true} as User);
  ElMessage({
    message: "启用成功!",
    type: "success",
  });
  await handleSearch()
}

const datetimeFormatter = (row: User, column: number, cellValue: string, index: number) => {
  return timestampToTime(cellValue, true);
}

handleSearch()
return {
  state,
  handleSearch,
  datetimeFormatter,
  disableUser,
  enableUser,
}
},
methods: {
  showUserDetail(row: User) {
    this.state.userId = row.id
    this.state.showDialog = true;
  },
}
})
</script>

<style scoped>
.pagination {
```

▲ 赞同 1

▼

● 添加评论

↗ 分享

★ 收藏

⚑ 举报

收起 ^

1.2.5 Router 层

有了一个新的页面，我们需要定义 route 来完成路由跳转。在 src/route/index.ts 文件中编写如下代码：

```
import {createRouter, createWebHistory, RouteRecordRaw} from "vue-router";
import Home from "../views/client/Home.vue";

const routes: Array<RouteRecordRaw> = [
  {
    path: "/",
    name: "Home",
    component: Home,
    meta: {}
  },
  {
    path: "/login/",
    name: "Login",
    component: () =>
      import("../views/admin/Login.vue")
  },
  {
    path: '/admin',
    name: 'Admin',
    component: () => import("../views/admin/Admin.vue"),
    children: [
      {
        path: '/admin/',
        name: 'Dashboard',
        component: () => import("../views/admin/Dashboard.vue"),
      },
      {
        path: '/admin/dashboard',
        name: 'AdminDashboard',
        component: () => import("../views/admin/Dashboard.vue"),
      },
      {
        path: '/admin/setting',
        name: 'AdminSetting',
        component: () => import("../views/admin/Setting.vue"),
      }
    ]
  }
]
```

▲ 赞同 1

▼

● 添加评论

↗ 分享

★ 收藏

⚑ 举报

收起 ^

```
]
},
]

const router = createRouter({
  history: createWebHistory(import.meta.env.BASE_URL),
  routes,
});

export default router;
```

1.3 标签管理

主要是为了方便灵活的给文章标记类型，所以才有标签管理，标签的属性很简单，就是一个名称。

1.3.1 Type 层

在 `src/types/index.ts` 文件中增加代码如下：

```
export interface Tag {
  id: number,
  name: string,
  created_at: string,
  modified_at: string,
}

export interface TagList {
  count: number,
  results: Array<Tag> | any
}
```

1.3.2 API 层

这里要编写标签管理相关的接口，列表查询、新增、修改、删除。在 `src/api/service.ts` 编写如下代码：

```
url: '/tag/',
method: 'get',
params,
}) as unknown as TagList
}

export function saveTag(method: string, data: Tag) {
let url = '/tag/'
if(['put', 'patch'].includes(method)) {
    url += data.id + '/'
}
// @ts-ignore
return request({
    url,
    method,
    data,
}) as unknown as ResponseData
}

export function addTag(data: Tag) {
return request({
    url: '/tag/',
    method: 'post',
    data,
}) as unknown as ResponseData
}

export function deleteTag(id: number) {
return request({
    url: '/tag/' + id + '/',
    method: 'delete',
}) as unknown as ResponseData
}
```

1.3.3 Component 层

提供一个新增和修改标签的弹框组件，因此在 `src/components` 下创建文件 `TagEditDialog.vue`，编写代码如下：

▲ 赞同 1



● 添加评论

↗ 分享

★ 收藏

⚑ 举报

收起 ^

知乎

专栏
Python

```
<el-input v-model="state.name" autocomplete="off" size=""></el-input>
</el-form-item>
</el-form>
<template #footer>
<span class="dialog-footer">
  <el-button @click="handleClose(false)">取 消</el-button>
  <el-button :loading="loading" type="primary" @click="handleClose(true)">确 定</el-button>
</span>
</template>
</el-dialog>
</template>

<script lang="ts">
import {defineComponent, PropType, reactive} from "vue";
import {Tag} from "../types";

export default defineComponent({
  name: "TagEditDialog",
  props: {
    visible: {
      type: Boolean,
      require: true,
    },
    tag: {
      type: Object as PropType<Tag>,
      require: true,
    },
    loading: {
      type: Boolean,
      require: true,
    }
  },
  emits: ["close"],
  watch: {
    '$props.visible': {
      handler(val: Boolean, oldVal: Boolean) {
        if (val !== oldVal) {
          this.state.visible = val
        }
        if (val) {
          this.state.name = this.$props.tag.name
        }
      }
    }
  }
})
```

[赞同 1](#)[添加评论](#)[分享](#)[收藏](#)[举报](#)[收起 ^](#)

```
},
setup(props) {
  const state = reactive({
    visible: props.visible as Boolean,
    // @ts-ignore
    name: '',
    // @ts-ignore
    title: ''
  });

  return {
    state,
  }
},
methods: {
  handleClose(isOk: Boolean) {
    this.$emit("close", {
      obj: {
        // @ts-ignore
        id: this.$props.tag.id,
        name: this.state.name
      },
      isOk,
    })
  }
}
)
</script>

<style scoped>
.form{
  padding-right: 24px;
}

</style>
```

1.3.4 View 层

通过表格管理标签，实现对标签的新增，修改，删除和列表查看，在 `src/views/admin` 下新

知乎

专栏
Python

```
<div>
  <el-form :inline="true" :model="state.params" class="demo-form-inline">
    <el-form-item label="名称">
      <el-input v-model="state.params.name" placeholder="名称" />
    </el-form-item>
    <el-form-item>
      <el-button
        :loading="state.isLoading"
        type="primary"
        @click="handleSearch"
      >查询</el-button>
    </el-form-item>
  </el-form>
</div>
<div class="button-container">
  <el-button
    :loading="state.isLoading"
    type="primary"
    @click="showAddDialog"
  ><i class="el-icon-plus" /> 新增</el-button>
</div>
<div>
  <el-table
    ref="tagTable"
    :data="state.tagList"
    :header-cell-style="{ background: '#eef1f6', color: '#606266' }"
    stripe>
    <el-table-column type="selection" width="55" />
    <el-table-column label="ID" prop="id" width="80" />
    <el-table-column label="名称" prop="name" width="200" />
    <el-table-column
      :formatter="datetimeFormatter"
      label="修改时间"
      prop="modified_at"
    />
    <el-table-column fixed="right" label="操作" width="120">
      <template #default="scope">
        <el-nonconfirm>
```

▲ 赞同 1

▼

● 添加评论

↗ 分享

★ 收藏

⚑ 举报

收起 ^

知乎

专栏
Python

```
iconColor="red"
title="确定删除系列吗？"
@confirm="deleteObject(scope.$index, scope.row)"
>
<template #reference>
    <el-button size="small" type="text"> 删除 </el-button>
</template>
</el-popconfirm>
<el-button
    size="small"
    type="text"
    @click.prevent="showEditDialog(scope.$index, scope.row)"
>
    编辑
</el-button>
</template>
</el-table-column>
</el-table>
</div>
<div class="pagination">
    <el-pagination
        :page-size="10"
        :total="state.total"
        background
        layout="prev, pager, next"
    ></el-pagination>
</div>
</div>
<TagEditDialog
    :loading="state.saveLoading"
    :tag="state.tag"
    :visible="state.showDialog"
    @close="handleCloseDialog"
/>
</template>

<script lang="ts">
import { defineComponent, reactive } from "vue";
import { ResponseData, Tag } from "../../types";
import { addTag, deleteTag, getTagList, saveTag } from "../../../api/service";
import { timestampToTime } from "../../../utils";

```

专栏
Python

```
export default defineComponent({
  name: "Tag",
  components: { TagEditDialog },
  watch: {
    "$route.path": {
      handler(val, oldVal) {
        if (val !== oldVal && [/admin/tag"].includes(val)) this.handleSearch();
      },
      deep: true,
    },
  },
  setup: function () {
    const route = useRoute();
    const state = reactive({
      tagList: [] as Array<Tag>,
      params: {
        name: undefined,
        page: 1,
        page_size: 10,
      },
      isLoading: false,
      total: 0,
      showDialog: false,
      tag: {
        id: 0,
        name: "",
      } as Tag,
      saveLoading: false,
    });
    const handleSearch = async (): Promise<void> => {
      state.isLoading = true;
      try {
        const data: ResponseData = await getTagList(state.params);
        state.isLoading = false;
        state.tagList = data.results;
        state.total = data.count;
      } catch (e) {
        console.error(e);
        state.isLoading = false;
      }
    }
  }
});
```

▲ 赞同 1



● 添加评论

↗ 分享

★ 收藏

⚑ 举报

收起 ^

知乎

专栏
Python

```
ElMessage({  
    message: "删除成功!",  
    type: "success",  
});  
  
await handleSearch();  
};  
  
const datetimeFormatter = (  
    row: Tag,  
    column: number,  
    cellValue: string,  
    index: number  
) => {  
    return timestampToTime(cellValue, true);  
};  
  
handleSearch();  
return {  
    state,  
    handleSearch,  
    datetimeFormatter,  
    deleteObject,  
};  
},  
methods: {  
    showEditDialog(index: number, row: Tag) {  
        this.state.tag = row;  
        this.state.showDialog = true;  
    },  
  
    showAddDialog() {  
        this.state.tag = {} as Tag;  
        this.state.showDialog = true;  
    },  
  
    async handleCloseDialog(params: any) {  
        if (!params.isOk) {  
            this.state.showDialog = false;  
            return;  
        }  
        this.state.saveLoading = true;  
    },  
};
```

▲ 赞同 1

▼

● 添加评论

↗ 分享

★ 收藏

⚑ 举报

收起 ^



```

        this.state.saveLoading = false;
        await this.handleSearch();
    } catch (e) {
        console.error(e);
        this.state.saveLoading = false;
    }
},
},
});
</script>

<style scoped>
.pagination {
    text-align: right;
    margin-top: 12px;
}
</style>

```

1.3.5 Router 层

定义 route 来完成路由跳转。在 src/route/index.ts 文件中新增代码：

```

import {createRouter, createWebHistory, RouteRecordRaw} from "vue-router";
import Home from "../views/client/Home.vue";

const routes: Array<RouteRecordRaw> = [
{
    path: "/",
    name: "Home",
    component: Home,
    meta: {}
},
{
    path: "/login/",
    name: "Login",
    component: () =>
        import("../views/admin/Login.vue")
},
{
    path: '/admin'
}
]

```

▲ 赞同 1



● 添加评论

↗ 分享

★ 收藏

☞ 举报

收起 ^

```
{  
    path: '/admin/',  
    name: 'Dashboard',  
    component: () => import("../views/admin/Dashboard.vue"),  
},  
{  
    path: '/admin/dashboard',  
    name: 'AdminDashboard',  
    component: () => import("../views/admin/Dashboard.vue"),  
},  
{  
    path: '/admin/user',  
    name: 'UserManagement',  
    component: () => import("../views/admin/User.vue"),  
},  
{  
    path: '/admin/tag',  
    name: 'Tag',  
    component: () => import("../views/admin/Tag.vue"),  
},  
]  
,  
]  
  
const router = createRouter({  
    history: createWebHistory(import.meta.env.BASE_URL),  
    routes,  
});  
  
export default router;
```

发布于 2021-08-24 09:21

▲ 赞同 1 ▼ 添加评论 分享 收藏 举报

收起 ^

Vue3+TypeScript+Django Rest Framework 搭建个人博客（三）：博客管理后台（1）

▲ 赞同 1 ▼ 添加评论 分享 收藏 举报

收起 ^

用来给管理员管理网站内容，配置各种功能和数据等。博客的管理后台就是用来承载创建博客，发布博客，查看留言，管理博客用户这些功能的子系统。

大家好，我是 落霞孤鹜，上一篇我们已经实现了用户注册，登录，登出的功能，这一章我们开始搭建博客的管理后台，实现对博客网站的管理功能。我会同样按照一个完整的功能，从需求分析到代码编写来阐述如何实现。

一、需求分析

作为一个完整的博客系统，管理后台是内容管理核心部分，在Python和PHP的世界里面，有很多做内容管理的库和开源项目，功能也是丰富多彩。这里我们从实际需要出发，整理了如下需求要点：

1. **Dashboard**: 主要展示整个博客网站的访问情况，包括浏览量，点赞量，评论量，留言量等内容。
2. **分类管理**: 主要用来组织文章的分类，通过分类帮助用户更好的浏览整个博客网站。
3. **标签管理**: 主要用来管理文章的标签，标注文章的类型，帮助用户更好的识别文档的类型。
4. **文章管理**: 主要用来完成文章的新增，修改，发布，删除等，考虑到文章发布的方便，需要支持 Markdown 语法。
5. **评论管理**: 主要用来查看文章的评论信息，如果存在敏感内容，可以通过后台进行删除。
6. **用户管理**: 主要用来管理博客网站注册的用户信息，可以禁用用户等。

以上功能也算是一套2B端产品的核心功能框架。

二、后端接口开发

后端承担业务逻辑处理和数据持久化的责任，基于需求分析中涉及的业务对象，我们需要先进行模型设计，映射到 Django 中，就是先建立 Model。

2.1 Model 层代码实现

2.1.1 物理模型说明

基于需求分析，通过对业务模型到物理模型的转换，这里主要有一下物理模型：

▲ 赞同 1 ▾ ● 添加评论 ↗ 分享 ★ 收藏 📣 举报

收起 ^

2. **标签表**: 存储文章的标签, 与文章表的关系是多对多, 即: 一个标签可以属于多个文章, 一篇文章可以管理多个标签
3. **文章表**: 存储文章信息, 需要记录文章的标题, 摘要, 正文, 封面, 浏览量, 评论量, 点赞量等
4. **评论表**: 存储文章的评论信息, 与文章表是多对一关系, 即一篇评论只能关联一篇文章, 但是一篇文章可以对应多篇评论
5. **点赞表**: 记录用户点赞信息, 与文章表是多对一关系, 即一则点赞对应一篇文章, 一篇文章对应多则点赞。
6. **用户表**: 记录用户信息, 包含博客的查看者, 也包含网站的管理员

2.1.2 代码实现

2.1.2.1 安装依赖

在分类表的设计中, 我们经常采用的是邻接表的方式, 通过一个 `parent_id` 自关联自己, 实现父级和子级的关联, 形成树形结构。这种设计在新增和修改的时候, 非常方便, 只需要一次查询即可完成, 但是在父查子, 子查父, 删除等操作时却需要较多的IO损耗。

而实际中, 查询要比修改多, 因此这里我们采用一种新的数据结构 MPTT, 预排序遍历树, 一种更高效的查询和管理树形数据的数据结构。因此需要安装依赖

```
pip install django-mptt==0.12.0
```

然后在 `requirements.txt` 中增加依赖信息

```
django-mptt==0.12.0
```

2.1.2.2 管理常量

后端在处理各类业务时, 会遇到各类枚举类型, 比如用户的身份, 性别, 文章状态等等, 在代码的世界里面, 尽量不要用 `Magic number`, 而是通过常量的方式进行管理。

在 `common` 下新增文件 `constants.py`, 编写代码如下:

```
class Constant(object):
```

```

        )
ARTICLE_STATUS_DELETED = 'Deleted'
ARTICLE_STATUS_PUBLISHED = 'Published'
ARTICLE_STATUS_DRAFT = 'Draft'

GENDERS = (
    ('Male', '男'),
    ('Female', '女'),
    ('Unknown', '未知'),
)
GENDERS_UNKNOWN = 'Unknown'

```

2.1.2.3 Model部分

这里需要说明几个点：

1. 在各个表的定义中，通过内部类 `Meta` 可以定义模型类的元信息，比如表名 `db_table`，排序方式 `ordering`，`-` 表示倒序。
2. MPTT 模型有一个单独的内部类 `MPTTMeta`，可以定义 `parent` 和排序字段 `order_insertion_by`。
3. 对于多对多的关系，Django 提供了 `ManyToMany` 的字段类型，这种会自动生成一张中间表用来记录两个表的多对多数据。
4. 外键关联在定义的时候，需要指定唯一的 `related_name`，以方便表与表的联合检索。
5. 外键管理中 `on_delete`，需要依据实际情况来确定是级联删除还是不做处理。

在 `blog/models.py` 下编写如下代码：

```

import mptt.models
from django.db import models

from common.constants import Constant
from common.models import AbstractBaseModel, User

class Tag(AbstractBaseModel):
    name = models.CharField('标签名称', max_length=50, unique=True, null=False, blank=False)

    class Meta:
        db_table = 'blog_tag'

```

```
class Catalog(mptt.models.MPTTModel, AbstractBaseModel):  
    name = models.CharField('分类名称', max_length=50, unique=True, null=False, blank=False)  
    parent = mptt.models.TreeForeignKey('self', on_delete=models.CASCADE, null=True, blank=True,  
                                         related_name='children')  
  
    class Meta:  
        db_table = 'blog_catalog'  
  
    class MPTTMeta:  
        order_insertion_by = ['name']  
  
    def __str__(self):  
        return self.name  
  
  
class Article(AbstractBaseModel):  
    title = models.CharField('文章标题', max_length=100, unique=True, null=False, blank=False)  
    cover = models.TextField('封面', max_length=1000, null=False, blank=False)  
    excerpt = models.CharField('摘要', max_length=200, blank=True)  
    keyword = models.CharField('关键词', max_length=200, blank=True)  
    markdown = models.TextField('正文', max_length=100000, null=False, blank=False)  
    status = models.CharField('文章状态', max_length=30, choices=Constant.ARTICLE_STATUS.choices,  
                             default=Constant.ARTICLE_STATUS_DRAFT)  
    catalog = models.ForeignKey(Catalog, verbose_name='所属分类', null=False, blank=False,  
                               on_delete=models.DO_NOTHING, related_name='cls_articles')  
    tags = models.ManyToManyField(Tag, verbose_name='文章标签', blank=True, related_name='articles')  
  
    author = models.ForeignKey(User, verbose_name='作者', on_delete=models.DO_NOTHING)  
    views = models.PositiveIntegerField('浏览量', default=0, editable=False)  
    comments = models.PositiveIntegerField('评论数量', default=0, editable=False)  
    likes = models.PositiveIntegerField('点赞量', default=0, editable=False)  
    words = models.PositiveIntegerField('字数', default=0, editable=False)  
  
    class Meta:  
        db_table = 'blog_article'  
        ordering = ["-created_at"]  
  
    def __str__(self):  
        return self.title
```

```
class Meta:
    db_table = 'blog_like'

class Comment(AbstractBaseModel):
    article = models.ForeignKey(Article, verbose_name='评论文章', on_delete=models.CASCADE,
                                related_name='article_comments')
    user = models.ForeignKey(User, verbose_name='评论者', on_delete=models.DO_NOTHING)
    reply = models.ForeignKey('self', verbose_name='评论回复', on_delete=models.CASCADE,
                             null=True, blank=True)
    content = models.TextField('评论', max_length=10000, null=False, blank=False)

    class Meta:
        db_table = 'blog_comment'

class Message(AbstractBaseModel):
    email = models.EmailField('邮箱', max_length=100, null=False, blank=False)
    content = models.TextField('内容', max_length=10000, null=False, blank=False)
    phone = models.CharField('手机', max_length=20, null=True, blank=True)
    name = models.CharField('姓名', max_length=30, null=True, blank=True)

    class Meta:
        db_table = 'blog_message'
```

2.2 Serializer 层代码实现

2.2.1 整理说明

在使用 Rest Framework 框架的时候，定义Serializer是使用这个框架最核心的内容，有几个点需要处理：

1. 对一个模型，哪一些字段需要在API中作为入参，哪一些字段作为出参（通过 `fields` 定义）
 2. 对于接口，哪一些字段只读，哪一些字段可写
 3. 对于外键字段，如何序列化和反序列化，可以具体指定每一个字段的序列化方式
 4. 如何增加orm中没有出现的字段

2.2.2 代码实现

对文章部分的定义，考虑到文章是整个博客的核心，所以对其序列化的方案，这里实现了三个版本 ArticleListSerializer、ArticleSerializer、ArticleChangeStatusSerializer：

- ArticleListSerializer：用来对应列表查询，完成界面上的展示，可以更好的隔离读和写的权限
- ArticleSerializer：用来完成新增，修改，删除，详情查看，通过集成 ArticleListSerializer 实现
- ArticleChangeStatusSerializer：用来完成上线，下线操作，这两个接口只需要有限字段的入参和出参

博客 App serializer 部分代码编写在 blog/serializers.py 文件中，具体代码如下：

```
from rest_framework import serializers

from blog.models import Catalog, Tag, Article, Like, Message, Comment


class CatalogSerializer(serializers.ModelSerializer):
    class Meta:
        model = Catalog
        fields = ['id', 'name', 'parent']


class TagSerializer(serializers.ModelSerializer):
    class Meta:
        model = Tag
        fields = ['id', 'name', 'created_at', 'modified_at']
        extra_kwargs = {
            'created_at': {'read_only': True},
            'modified_at': {'read_only': True},
        }


class ArticleListSerializer(serializers.ModelSerializer):
    tags_info = serializers.SerializerMethodField(read_only=True)
```

▲ 赞同 1 ▾ 添加评论 分享 收藏 举报

收起 ^



```
class Meta:
    model = Article
    fields = ['id', 'title', 'excerpt', 'cover', 'created_at', 'modified_at',
              'tags_info', 'catalog', 'catalog_info', 'views', 'comments', 'words',
              'likes']
    extra_kwargs = {
        'tags': {'write_only': True},
        'catalog': {'write_only': True},
        'views': {'read_only': True},
        'comments': {'read_only': True},
        'words': {'read_only': True},
        'likes': {'read_only': True},
        'created_at': {'read_only': True},
        'modified_at': {'read_only': True},
    }

@staticmethod
def get_tags_info(obj: Article) -> list:
    if not obj.title:
        article = Article.objects.get(id=obj.id)
        tags = article.tags.all()
    else:
        tags = obj.tags.all()
    return [{ 'id': tag.id, 'name': tag.name} for tag in tags]

@staticmethod
def get_catalog_info(obj: Article) -> dict:
    if not obj.catalog:
        book = Article.objects.get(id=obj.id)
        catalog = book.catalog
    else:
        catalog = obj.catalog
    return {
        'id': catalog.id,
        'name': catalog.name,
        'parents': [c.id for c in catalog.get_ancestors(include_self=True)]
    }

@staticmethod
def get_status(obj: Article) -> list:
    return obj.get_status_display()
```



```
catalog_info = serializers.SerializerMethodField(read_only=True)
```

```
class Meta(ArticleListSerializer.Meta):
    fields = ['markdown', 'keyword']
    fields.extend(ArticleListSerializer.Meta.fields)
```

```
class ArticleChangeStatusSerializer(serializers.ModelSerializer):
```

```
    class Meta:
        model = Article
        fields = ['id', 'status', ]
        extra_kwargs = {
            'status': {'read_only': True},
        }
```

```
class LikeSerializer(serializers.ModelSerializer):
```

```
    user_info = serializers.SerializerMethodField(read_only=True)
    article_info = serializers.SerializerMethodField(read_only=True)
```

```
    class Meta:
```

```
        model = Like
        fields = ['user', 'user_info', 'article', 'article_info', 'created_at']
        extra_kwargs = {
            'created_at': {'read_only': True},
        }
```

```
@staticmethod
```

```
def get_user_info(obj: Like) -> dict:
    if not obj.user:
        return {}
    else:
        user = obj.user
    return {'id': user.id, 'name': user.nickname or user.username, 'avatar': l
```

```
@staticmethod
```

```
def get_article_info(obj: Like) -> dict:
    if not obj.article:
        return {}
    else:
        article = obj.article
```

知乎

专栏
Python

```
user_info = serializers.SerializerMethodField(read_only=True)
article_info = serializers.SerializerMethodField(read_only=True)
comment_replies = serializers.SerializerMethodField(read_only=True)

class Meta:
    model = Comment
    fields = ['id', 'user', 'user_info', 'article', 'article_info', 'created_at', 'comment_replies']
    extra_kwargs = {
        'created_at': {'read_only': True},
    }

@staticmethod
def get_user_info(obj: Comment) -> dict:
    if not obj.user:
        return {}
    else:
        user = obj.user
    return {'id': user.id, 'name': user.nickname or user.username, 'avatar': user.avatar}

@staticmethod
def get_article_info(obj: Comment) -> dict:
    if not obj.article:
        return {}
    else:
        article = obj.article
    return {'id': article.id, 'title': article.title}

@staticmethod
def get_comment_replies(obj: Comment):
    if not obj.comment_reply:
        return []
    else:
        replies = obj.comment_reply.all()
    return [
        {
            'id': reply.id,
            'content': reply.content,
            'user_info': {
                'id': reply.user.id,
                'name': reply.user.nickname or reply.user.username,
                'avatar': reply.user.avatar,
            }
        }
    ]
```

▲ 赞同 1 ▼

● 添加评论

↗ 分享

★ 收藏

⚑ 举报

收起 ^

```
class MessageSerializer(serializers.ModelSerializer):
    class Meta:
        model = Message
        fields = ['email', 'phone', 'name', 'content', 'created_at']
        extra_kwargs = {
            'created_at': {'read_only': True},
        }
```

2.3 工具方法

为了更好的复用代码逻辑，我们一般会抽象一些工具方法，主要是时间处理方法和上传相关的路径处理，在 common/utils.py 中编写如下代码：

```
import os
import random
import string
import time
from datetime import datetime

from django.conf import settings
from django.template.defaultfilters import slugify


def get_upload_file_path(upload_name):
    # Generate date based path to put uploaded file.
    date_path = datetime.now().strftime('%Y/%m/%d')

    # Complete upload path (upload_path + date_path).
    upload_path = os.path.join(settings.UPLOAD_URL, date_path)
    full_path = os.path.join(settings.BASE_DIR, upload_path)
    make_sure_path_exist(full_path)
    file_name = slugify_filename(upload_name)
    return os.path.join(full_path, file_name).replace('\\', '/'), os.path.join('/'

def slugify_filename(filename):
```

▲ 赞同 1

● 添加评论

↗ 分享

★ 收藏

☞ 举报

收起 ^

```
def get_slugified_name(filename):
    slugified = slugify(filename)
    return slugified or get_random_string()

def get_random_string():
    return ''.join(random.sample(string.ascii_lowercase * 6, 6))

def make_sure_path_exist(path):
    if os.path.exists(path):
        return
    os.makedirs(path, exist_ok=True)

def format_time(dt: datetime, fmt: str = ''):
    fmt_str = fmt or '%Y-%m-%d %H:%M:%S'
    return dt.strftime(fmt_str)

def get_year(dt: datetime) -> int:
    return dt.year

def get_now() -> str:
    return format_time(datetime.now())

def format_time_from_str(date_time_str: str, fmt: str = ''):
    fmt_str = fmt or '%Y-%m-%d %H:%M:%S'
    return datetime.strptime(date_time_str, fmt_str)

def transform_time_to_str(t: int):
    return time.strftime("%Y-%m-%d %H:%M:%S", time.localtime(t))
```

2.4 ViewSet 层代码实现

 赞同 1 添加评论 分享 收藏 举报

收起 ^



为了更好的实现在列表查询时的搜索条件识别和校验，我们安装一个新的库： django-filter

```
pip install django-filter==2.4.0
```

在 requirements.txt 中增加依赖信息

```
django-filter==2.4.0
```

2.4.2 通用 ViewSet 定义

在处理接口层定义的时候，我们需要考虑接口的访问权限，分页，查询过滤条件，新增和修改时的操作人，用户角色判断等，这些处理是在每一个接口中都需要处理的，因此我们这里将这些逻辑统一抽象到一个基础类中完成，然后通过Python的多继承完成子类的能力扩充。

1. 定义了 `BaseError`，用于在出现各类业务校验不通过时抛出异常。
2. 定义了 `BasePagination`，用于列表查询接口的分页。
3. 定义了 `BaseViewSetMixin` 类，作为常规ViewSet的基类，将分页、过滤条件、权限、操作者填充、用户身份判断等。
4. 定义了 `ConstantViewSet` 类，用于将后端使用的常量提供给前端，用做前端需要判断枚举值之用。
5. 定义了 `ImageUploadViewSet` 类，用于在新增文章时上传封面之用。

在 `common/views.py` 中增加如下代码：

```
import logging

import django.conf
from django.contrib.auth import authenticate, login, logout as auth_logout
from django.contrib.auth.hashers import make_password
from django.contrib.auth.models import AnonymousUser
from django.core.mail import send_mail
from django.db.models import QuerySet
from django_filters.rest_framework import DjangoFilterBackend
from rest_framework import viewsets, permissions, status
from rest_framework.exceptions import ValidationError
from rest_framework.generics import GenericAPIView
from rest_framework.pagination import PageNumberPagination
```

知乎

专栏
Python

```
from common.models import User
from common.serializers import UserSerializer, UserLoginSerializer, UserPasswordSe
from common.utils import get_upload_file_path


def get_random_password():
    import random
    import string
    return ''.join(random.sample(string.ascii_letters + string.digits + string.pur


class BaseError(ValidationError):
    def __init__(self, detail=None, code=None):
        super(BaseError, self).__init__(detail= {'detail': detail})


class BasePagination(PageNumberPagination):
    """
    customer pagination
    """

    # default page size
    page_size = 10
    # page size param in page size
    page_size_query_param = 'page_size'
    # page param in api
    page_query_param = 'page'
    # max page size
    max_page_size = 100


class BaseViewSetMixin(object):
    pagination_class = BasePagination
    filter_backends = [DjangoFilterBackend]
    permission_classes = [permissions.IsAuthenticatedOrReadOnly]

    def __init__(self, **kwargs):
        super(BaseViewSetMixin, self).__init__(**kwargs)
        self.filterset_fields = []
        self.init_filter_field()

    def init_filter_field(self):
```

知乎

专栏
Python

```
"""
serializer = self.get_serializer_class()
if not hasattr(serializer, 'Meta'):
    return
meta = serializer.Meta

if not hasattr(meta, 'model'):
    return
model = meta.model

if not hasattr(meta, 'fields'):
    ser_fields = []
else:
    ser_fields = meta.fields

for field in ser_fields:
    if not hasattr(model, field):
        continue
    self.filterset_fields.append(field)

def perform_update(self, serializer):
    user = self.fill_user(serializer, 'update')
    return serializer.save(**user)

def perform_create(self, serializer):
    user = self.fill_user(serializer, 'create')
    return serializer.save(**user)

@staticmethod
def fill_user(serializer, mode):
    """
    before save, fill user info into para from session
    :param serializer: Model's serializer
    :param mode: create or update
    :return: None
    """

    request = serializer.context['request']

    user_id = request.user.id
    ret = {'modifier': user_id}
```

▲ 赞同 1

▼

● 添加评论

↗ 分享

★ 收藏

⚑ 举报

收起 ^



```
def get_pk(self):
    if hasattr(self, 'kwargs'):
        return self.kwargs.get('pk')

def is_reader(self):
    return isinstance(self.request.user, AnonymousUser) or not self.request.u:

class BaseModelViewSet(BaseViewSetMixin, viewsets.ModelViewSet):
    pass

class UserViewSet(viewsets.ModelViewSet):
    queryset = User.objects.all().order_by('username')
    serializer_class = UserSerializer
    permission_classes = [permissions.AllowAny]

class UserLoginViewSet(GenericAPIView):
    permission_classes = [permissions.AllowAny]
    serializer_class = UserLoginSerializer
    queryset = User.objects.all()

    def post(self, request, *args, **kwargs):
        username = request.data.get('username', '')
        password = request.data.get('password', '')

        user = authenticate(username=username, password=password)
        if user is not None and user.is_active:
            login(request, user)
            serializer = UserSerializer(user)
            return Response(serializer.data, status=200)
        else:
            ret = {'detail': 'Username or password is wrong'}
            return Response(ret, status=403)

class UserLogoutViewSet(GenericAPIView):
    permission_classes = [permissions.IsAuthenticated]
    serializer_class = UserLoginSerializer
```

专栏
Python

```
class PasswordUpdateViewSet(GenericAPIView):  
    permission_classes = [permissions.IsAuthenticated]  
    serializer_class = UserPasswordSerializer  
    queryset = User.objects.all()  
  
    def post(self, request, *args, **kwargs):  
        user_id = request.user.id  
        password = request.data.get('password', '')  
        new_password = request.data.get('new_password', '')  
        user = User.objects.get(id=user_id)  
        if not user.check_password(password):  
            ret = {'detail': 'old password is wrong !'}  
            return Response(ret, status=403)  
  
        user.set_password(new_password)  
        user.save()  
        return Response({  
            'detail': 'password changed successful !'  
        })  
  
    def put(self, request, *args, **kwargs):  
        """  
        Parameter: username->user's username who forget old password  
        """  
        username = request.data.get('username', '')  
        users = User.objects.filter(username=username)  
        user: User = users[0] if users else None  
  
        if user is not None and user.is_active:  
            password = get_random_password()  
  
            try:  
                send_mail(subject="New password for Blog site",  
                          message="Hi: Your new password is: \n{}".format(password),  
                          from_email=django.conf.settings.EMAIL_HOST_USER,  
                          recipient_list=[user.email],  
                          fail_silently=False)  
                user.password = make_password(password)  
                user.save()  
            return Response({
```

▲ 赞同 1



● 添加评论

↗ 分享

★ 收藏

⚑ 举报

收起 ^

专栏
Python

```
        return Response({
            'detail': 'Send New email failed, Please check your email addi
        })
    else:
        ret = {'detail': 'User does not exist(Account is incorrect !)'}
        return Response(ret, status=403)

class ConstantViewSet(GenericAPIView):
    permission_classes = [permissions.IsAuthenticated]
    serializer_class = UserPasswordSerializer
    queryset = QuerySet()

    def get(self, request, *args, **kwargs):
        ret = {}
        for key in dir(Constant):
            if not key.startswith('_'):
                ret[key] = getattr(Constant, key)
        return Response(ret)

class ImageUploadViewSet(APIView):
    permission_classes = [permissions.AllowAny]

    def post(self, request, *args, **kwargs):

        try:
            if request.method == 'POST' and request.FILES:
                uploaded_file = request.FILES['file']

                full_file_path, file_path = get_upload_file_path(uploaded_file.nar
                self.handle_uploaded_file(uploaded_file, full_file_path)

                response = {
                    'url': file_path
                }
                return Response(response)

        except Exception as e:
            logging.getLogger('default').error(e, exc_info=True)
            raise BaseError(detail='Upload failed', code=status.HTTP_500_INTERNAL_
```

▲ 赞同 1



● 添加评论

↗ 分享

★ 收藏

⚑ 举报

收起 ↗



```
for chunk in f.chunks():
    destination.write(chunk)
destination.close()
```

2.4.3 Blog 相关的 ViewSet 定义

这里的 ViewSet 类通过继承框架提供的基类或者我们自己封装的 BaseViewSet 类，来实现对应的业务接口，如果是非常传统的 CURD 接口，在 ViewSet 里面可能仅仅只需要定义 queryset 的属性就可以完成新增，修改，删除，详情查询，列表查询的接口。

可以看到我们的 Article 对象相关的ViewSet 有 7 个，主要是兼顾了文章在各种维度下的查询和管理需求，比如我们需要按照时间对文章进行查询，需要通过浏览量倒序展示文章列表，需要上线，下架文章，需要不登录就能浏览文章，需要登录管理员才能管理文章等各种需求。

具体代码如下：

```
import datetime

from common.utils import get_year
from django.db.models import QuerySet, Sum, Count
from rest_framework import mixins
from rest_framework.response import Response
from rest_framework.viewsets import GenericViewSet

from blog.models import Article, Comment, Message, Tag, Catalog, Like
from blog.serializers import ArticleSerializer, CommentSerializer, MessageSeriali:
    ArticleListSerializer, CatalogSerializer, ArticleChangeStatusSerializer, LikeS
from common.constants import Constant
from common.views import BaseModelViewSet, BaseViewSetMixin

class ArticleArchiveListViewSet(BaseViewSetMixin, mixins.ListModelMixin, GenericV:
    queryset = Article.objects.all()
    serializer_class = ArticleListSerializer

    def filter_queryset(self, queryset) -> QuerySet:
        queryset = super(ArticleArchiveListViewSet, self).filter_queryset(queryset)
        return queryset.order_by('-create_time')
```

▲ 赞同 1 ▾ 添加评论 分享 收藏 举报 收起 ^

知乎

专栏
Python

```

def list(self, request, *args, **kwargs):
    queryset = self.filter_queryset(self.get_queryset())
    total = len(queryset)
    page_size, page_number = self.get_page_info()
    start_year, end_year = self.get_datetime_range(page_size, page_number)
    queryset = queryset.filter(created_at__gte=start_year).filter(created_at__lt=end_year)
    ret = {
        "count": total,
        "next": None,
        "previous": None,
        'results': []
    }
    years = {}
    for article in queryset.all():
        year = article.created_at.year
        articles = years.get(year)
        if not articles:
            articles = []
        years[year] = articles
        serializer = self.get_serializer(article)
        articles.append(serializer.data)
    for key, value in years.items():
        ret['results'].append({
            'year': key,
            'list': value
        })
    ret['results'].sort(key=lambda i: i['year'], reverse=True)
    return Response(ret)

def get_page_info(self):
    page_size = self.paginator.get_page_size(self.request)
    page_number = self.request.query_params.get(self.paginator.page_query_param)
    return page_size, int(page_number)

@staticmethod
def get_datetime_range(page_size, page_number):
    current_year = get_year(datetime.datetime.now())
    start_year = current_year - page_size * page_number + 1
    start_datetime = '{:d}-01-01 00:00:00'.format(start_year)
    end_datetime = '{:d}-01-01 00:00:00'.format(start_year + page_size)

```

▲ 赞同 1▼添加评论分享收藏举报收起 ^

专栏
Python

GenericViewSet):

```
queryset = Article.objects.all().select_related('catalog', 'author')
serializer_class = ArticleListSerializer

def filter_queryset(self, queryset):
    self.filterset_fields.remove('catalog')
    queryset = super(ArticleListViewSet, self).filter_queryset(queryset)
    if self.is_reader():
        queryset = queryset.exclude(status=Constant.ARTICLE_STATUS_DRAFT)
    params = self.request.query_params
    if 'catalog' in params:
        catalog_id = params.get('catalog', 1)
        catalog = Catalog.objects.get(id=catalog_id)
        catalogs = catalog.get_descendants(include_self=True)
        queryset = queryset.filter(catalog__in=[c.id for c in catalogs])
    return queryset.exclude(status=Constant.ARTICLE_STATUS_DELETED)
```

```
class ArticleViewSet(BaseViewSetMixin,
                     mixins.CreateModelMixin,
                     mixins.RetrieveModelMixin,
                     mixins.UpdateModelMixin,
                     mixins.DestroyModelMixin,
                     GenericViewSet):
    queryset = Article.objects.all()
    serializer_class = ArticleSerializer
```

```
def perform_create(self, serializer):
    extra_infos = self.fill_user(serializer, 'create')
    extra_infos['author'] = self.request.user
    serializer.save(**extra_infos)

def filter_queryset(self, queryset):
    queryset = super(ArticleViewSet, self).filter_queryset(queryset)
    if self.is_reader():
        queryset = queryset.exclude(status=Constant.ARTICLE_STATUS_DRAFT).exclude(
            status=Constant.ARTICLE_STATUS_DELETED)
    return queryset
```

```
def perform_destroy(self, instance: Article):
    instance.status = Constant.ARTICLE_STATUS_DELETED
```

▲ 赞同 1

▼

添加评论

分享

收藏

举报

收起 ^



```
serializer = self.get_serializer(instance)
if self.is_reader():
    instance.views += 1
    instance.save()
return Response(serializer.data)

class ArticlePublishViewSet(BaseViewSetMixin,
                            mixins.UpdateModelMixin,
                            GenericViewSet):
    queryset = Article.objects.all()
    serializer_class = ArticleChangeStatusSerializer

    def filter_queryset(self, queryset):
        queryset = super(ArticlePublishViewSet, self).filter_queryset(queryset)
        return queryset.exclude(status=Constant.ARTICLE_STATUS_DELETED)

    def perform_update(self, serializer):
        extra_infos = self.fill_user(serializer, 'update')
        extra_infos['status'] = Constant.ARTICLE_STATUS_PUBLISHED
        serializer.save(**extra_infos)

class ArticleOfflineViewSet(ArticlePublishViewSet):
    def perform_update(self, serializer):
        extra_infos = self.fill_user(serializer, 'update')
        extra_infos['status'] = Constant.ARTICLE_STATUS_DRAFT
        serializer.save(**extra_infos)

class CommentViewSet(BaseModelViewSet):
    queryset = Comment.objects.all()
    serializer_class = CommentSerializer

    def filter_queryset(self, queryset):
        queryset = super(CommentViewSet, self).filter_queryset(queryset)
        return queryset.filter(reply__isnull=True)

    def perform_create(self, serializer):
        super(CommentViewSet, self).perform_create(serializer)
        article: Article = serializer.validated_data['article']
```

专栏
Python

```
class LikeViewSet(BaseModelViewSet):
    queryset = Like.objects.all()
    serializer_class = LikeSerializer

    def perform_create(self, serializer):
        super(LikeViewSet, self).perform_create(serializer)
        article: Article = serializer.validated_data['article']
        article.likes += 1
        article.save()

class MessageViewSet(BaseModelViewSet):
    queryset = Message.objects.all()
    serializer_class = MessageSerializer

class TagViewSet(BaseModelViewSet):
    queryset = Tag.objects.all()
    serializer_class = TagSerializer

class CatalogViewSet(BaseModelViewSet):
    queryset = Catalog.objects.all()
    serializer_class = CatalogSerializer

    def list(self, request, *args, **kwargs):
        ret = []
        roots = Catalog.objects.filter(id=1).filter(parent__isnull=True)
        if not roots:
            return Response(ret)
        root: Catalog = roots[0]
        root_dict = CatalogSerializer(root).data
        root_dict['children'] = []
        ret.append(root_dict)
        parent_dict = {root.id: root_dict}
        for cls in root.get_descendants():
            data = CatalogSerializer(cls).data
            parent_id = data.get('parent')
            parent = parent_dict.get(parent_id)
            parent['children'].append(data)
```

[赞同 1](#)[添加评论](#)[分享](#)[收藏](#)[举报](#)[收起 ^](#)

专栏
Python

```
    return Response(ret)
```

```
class NumberViewSet(BaseViewSetMixin,
                    mixins.ListModelMixin,
                    GenericViewSet):
    queryset = Article.objects.all()
    serializer_class = ArticleListSerializer

    def list(self, request, *args, **kwargs):
        queryset = self.get_queryset().aggregate(Sum('views'), Sum('likes'), Sum(
            messages = Message.objects.aggregate(Count('id')))

        return Response({
            'views': queryset['views__sum'],
            'likes': queryset['likes__sum'],
            'comments': queryset['comments__sum'],
            'messages': messages['id__count']
        })

class TopArticleViewSet(NumberViewSet):
    def list(self, request, *args, **kwargs):
        queryset = self.filter_queryset(self.get_queryset()).order_by('-views')[::]

        page = self.paginate_queryset(queryset)
        if page is not None:
            serializer = self.get_serializer(page, many=True)
            return self.get_paginated_response(serializer.data)

        serializer = self.get_serializer(queryset, many=True)
        return Response(serializer.data)
```

2.5 定义URL

2.5.1 安装API文档自动生成依赖

通过工具，可以自动生成基于Restful的接口说明。

▲ 赞同 1



● 添加评论

↗ 分享

★ 收藏

⚑ 举报

收起 ^

drf-yasg==1.20.0

2.5.2 common 下的 urls.py

修改 common/urls.py , 最终代码如下代码:

```
from django.conf.urls import url
from django.urls import include, path
from rest_framework import routers

from common import views
from common.views import ImageUploadViewSet

router = routers.DefaultRouter()
router.register('user', views.UserViewSet)

app_name = 'common'

urlpatterns = [
    path('', include(router.urls)),
    url(r'^user/login', views.UserLoginViewSet.as_view()),
    url(r'^user/logout', views.UserLogoutViewSet.as_view()),
    url(r'^user/pwd', views.PasswordUpdateViewSet.as_view()),
    url(r'^dict', views.ConstantViewSet.as_view()),
    url(r'^upload/$', ImageUploadViewSet.as_view()),
]
```

2.5.3 blog 下的 urls.py

在 blog/urls.py 中编写如下代码:

```
from django.urls import include, path
from rest_framework import routers

from blog import views
```



```

router.register('offline', views.ArticleOfflineViewSet)
router.register('archive', views.ArticleArchiveListViewSet)
router.register('tag', views.TagViewSet)
router.register('catalog', views.CatalogViewSet)
router.register('comment', views.CommentViewSet)
router.register('like', views.LikeViewSet)
router.register('message', views.MessageViewSet)
router.register('number', views.NumberViewSet)
router.register('top', views.TopArticleViewSet)

app_name = 'blog'

urlpatterns = [
    path('', include(router.urls)),
]

```

2.5.4 project 下的 urls.py

这里我们使用drf_yasg提供的方法，自动生成接口说明文档，在实际的前后端分类的项目中，这是非常有用的一个工具，可以让前端和后端基于接口约定并行开发，保证前端和后端的开发效率。

修改 project/urls.py，最终代码如下：

```

from django.conf import settings
from django.conf.urls import url
from django.urls import path, re_path, include
from django.views.generic import RedirectView
from django.views.static import serve
from drf_yasg import openapi
from drf_yasg.views import get_schema_view
from rest_framework import permissions

schema_view = get_schema_view(
    openapi.Info(
        title="Blog System API",
        description="Blog site",
        default_version='v1',
        terms_of_service='',

```



```

        permission_classes=(permissions.AllowAny,),
    )

urlpatterns = [
    path('', include('blog.urls', namespace='blog')),
    path('', include('common.urls', namespace='common')),
    url(r'^favicon.ico$', RedirectView.as_view(url=r'static/img/favicon.ico')),
    url(r'^upload/(?P<path>.*$', serve, {'document_root': settings.MEDIA_ROOT}),
    path('api-auth/', include('rest_framework.urls', namespace='rest_framework'))
    re_path(
        r"api/swagger(?P<format>\.json|\.yaml)",
        schema_view.without_ui(cache_timeout=0),
        name="schema-json",
    ),
    path(
        "swagger/",
        schema_view.with_ui("swagger", cache_timeout=0),
        name="schema-swagger-ui",
    ),
    path("docs/", schema_view.with_ui("redoc", cache_timeout=0), name="schema-redoc")
]

```

2.6 配置调整

2.6.1 调整 project/setting.py

在 project/setting.py 中 INSTALLED_APPS 中增加 blog 、 drf_yasg 、 django_filters ，如果不添加，则会出现模板路径找不到的问题

```

INSTALLED_APPS = [
    'django.contrib.admin',
    'django.contrib.auth',
    'django.contrib.contenttypes',
    'django.contrib.sessions',
    'django.contrib.messages',
    'django.contrib.staticfiles',
    'rest_framework',
]

```

]

在 project/setting.py 中 TEMPLATES 调整为：

```
TEMPLATES = [
    {
        'BACKEND': 'django.template.backends.django.DjangoTemplates',
        'DIRS': [os.path.join(BASE_DIR, 'templates')],  

        'APP_DIRS': True,  

        'OPTIONS': {
            'context_processors': [
                'django.template.context_processors.debug',
                'django.template.context_processors.request',
                'django.contrib.auth.context_processors.auth',
                'django.contrib.messages.context_processors.messages',
            ],
        },
    },
]
```

在 project/setting.py 增加关于媒体文件和上传路径的配置

```
MEDIA_ROOT = os.path.join(BASE_DIR, 'upload')
MEDIA_URL = "/upload/"
UPLOAD_URL = 'upload'
STATICFILES_DIRS = (
    os.path.join(BASE_DIR, 'upload'),
)
```

2.6.2 执行模型迁移

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

到此为止个人博客后端部分开发完成。 这里面实际也包含了访客在博客网站上能够访问网站所需要的接口。

▲ 赞同 1

▼

● 添加评论

↗ 分享

★ 收藏

⚑ 举报

收起 ^

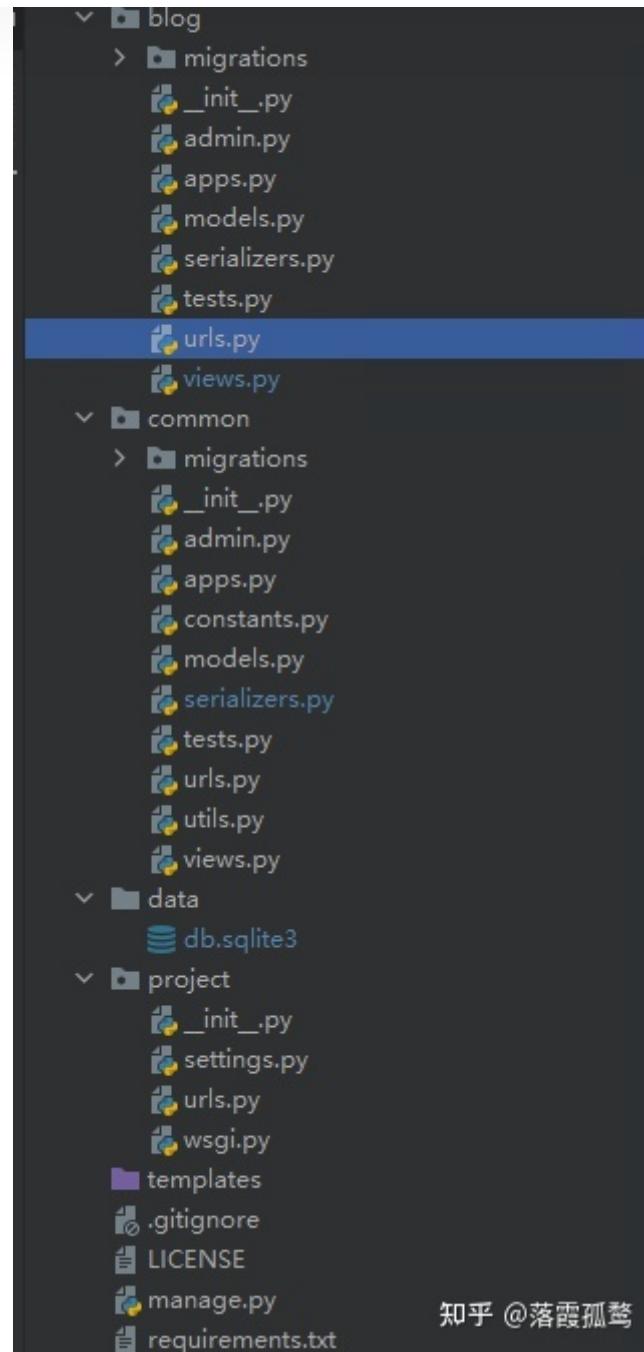


在浏览器中访问 <http://127.0.0.1:8000/swagger/>，效果如下图：

Category	Method	URL	Description
archive	GET	/archive/	archive_list
article	POST	/article/	article_create
	GET	/article/{id}/	article_read
	PUT	/article/{id}/	article_update
	PATCH	/article/{id}/	article_partial_update
	DELETE	/article/{id}/	article_delete
catalog	GET	/catalog/	catalog_list
	POST	/catalog/	catalog_create
	GET	/catalog/{id}/	catalog_read
	PUT	/catalog/{id}/	catalog_update
	PATCH	/catalog/{id}/	catalog_partial_update

3.2 后端文件夹结构

知乎

专栏
Python

发布于 2021-08-24 09:13

▲ 赞同 1 ▾ 添加评论 分享 收藏 举报

收起 ^

Vue3+TypeScript+Django Rest Framework搭建个人博客（二）：用户登录功能

 落霞孤鹜，求知若渴

▲ 赞同 1 ▾ 添加评论 分享 收藏 举报

收起 ^



www.ssyet.com
沙沙野 - 免版权设计素材网

用户登录功能是一个信息系统必不可少的一部分，作为博客网站，同样需要管理员登录管理后台，游客注册后登录评论等

大家好，我是 落霞孤鹜，上一篇我们已经搭建好了前后端的框架的代码，并调通了前后端接口。从这一篇开始，进入到业务功能开发进程中。

首先我们需要实现的功能是用户登录，用户登录功能虽然在系统开发中已经很成熟，但是当我们自己动手做的时候，会发现这个功能是那种典型的说起来容易，做起来复杂的功能，需要考虑和处理的点很多。

一、需求分析

1.1 完整需求

一个完整的用户登录功能，需要考虑的点如下：

- 账号和密码的格式
- 支持邮箱、账号、手机号码登录
- 手机号码支持验证码登录
- 密码错误的次数
- 忘记密码功能

...更多功能

▲ 赞同 1 ▼ ● 添加评论 ↗ 分享 ★ 收藏 📣 举报 收起 ^

- 7天自动登录
- 登录状态保持
- 权限鉴定
- 登出
- 密码修改

在前后端分离的状态下，我们还需要考虑**跨域问题**等

1.2 博客网站需求

考虑到我们的博客系统是个人博客，用户登录的场景主要集中在游客评论，管理员登录管理后台两个场景，所以登录功能可以适当做删减。

该博客系统的登录功能主要实现以下几个点：

- 账号和密码的格式
- 支持邮箱、账号
- 忘记密码功能
- 注册功能
- 登录状态保持
- 权限鉴定
- 登出
- 密码修改

以上功能点，满足博客网站基本需求

- 未登录的游客只能留言，不能评论
- 游客登录后可以评论博客
- 游客登录后可以修改密码
- 管理员登录后可以管理博客后台

二、后端接口开发

用户登录和鉴权实际上在 Django 里面有完整的功能，但是由于我们使用的是前后端分离架构，在 Django 的基础上使用了 Django Rest Framework，因此原有的 Django 登录和鉴权接口需要做改造和调整，以适应前后端分离功能。

3. 密码修改和重置

2.1 配置鉴权模式

这里采用 Django Rest Framework 提供的基于 Django 的 Session 方案，如果你想采用 JWT (介绍) 方案，可以按照官网教程 [Authentication - Django REST framework](#) 进行配置。在 project/settings.py 中的 REST_FRAMEWORK 配置项中修改如下：

```
REST_FRAMEWORK = {
    'DEFAULT_AUTHENTICATION_CLASSES': [
        'rest_framework.authentication.BasicAuthentication',
        'rest_framework.authentication.SessionAuthentication',
    ],
    'DEFAULT_PAGINATION_CLASS': 'rest_framework.pagination.PageNumberPagination',
    'PAGE_SIZE': 10
}
```

2.2 编写登录登出接口

2.2.1 增加 UserLoginSerializer 类

在 common/serializers.py 文件中，增加代码，修改后代码如下：

```
from rest_framework import serializers

from common.models import User


class UserSerializer(serializers.ModelSerializer):
    class Meta:
        model = User
        fields = ['id', 'username', 'avatar', 'email', 'is_active', 'created_at',


class UserLoginSerializer(serializers.ModelSerializer):
    class Meta:
        model = User
```

2.2.2 增加 UserLoginViewSet 类

在 common/views.py 中增加 UserLoginViewSet 类，使用 Django 自带的 authenticate 和 login，完成用户的登录，并返回用户登录信息，在这个过程中，Response 中会创建 Session，保存登录后的 user 信息，生成 Cookies 一并返回。方法修改后代码如下：

```
from django.contrib.auth import authenticate, login
from rest_framework import viewsets, permissions
from rest_framework.generics import GenericAPIView
from rest_framework.response import Response

from common.models import User
from common.serializers import UserSerializer, UserLoginSerializer


class UserViewSet(viewsets.ModelViewSet):
    queryset = User.objects.all().order_by('username')
    serializer_class = UserSerializer
    permission_classes = [permissions.AllowAny]

class UserLoginViewSet(GenericAPIView):
    permission_classes = [permissions.AllowAny]
    serializer_class = UserLoginSerializer
    queryset = User.objects.all()

    def post(self, request, *args, **kwargs):
        username = request.data.get('username', '')
        password = request.data.get('password', '')

        user = authenticate(username=username, password=password)
        if user is not None and user.is_active:
            login(request, user)
            serializer = UserSerializer(user)
            return Response(serializer.data, status=200)
        else:
            ret = {'detail': 'Username or password is wrong'}
```



在 common/views.py 中增加 UserLogoutViewSet 类，使用 Django 自带的 auth_logout，完成用户的登出，并返回登出成功信息，这个过程中，Django 会自动清理 Session 和 Cookies

```
class UserLogoutViewSet(GenericAPIView):
    permission_classes = [permissions.IsAuthenticated]
    serializer_class = UserLoginSerializer

    def get(self, request, *args, **kwargs):
        auth_logout(request)
        return Response({'detail': 'logout successful !'})
```

2.2.4 配置路由

在 common/urls.py 中增加 user/login 和 user/logout 路由，代码如下：

```
from django.conf.urls import url
from django.urls import include, path
from rest_framework import routers

from common import views

router = routers.DefaultRouter()
router.register('user', views.UserViewSet)

app_name = 'common'

urlpatterns = [
    path('', include(router.urls)),
    url(r'^user/login', views.UserLoginViewSet.as_view()),
    url(r'^user/logout', views.UserLogoutViewSet.as_view()),
]
```

2.3 编写修改密码接口

2.3.1 增加 UserPasswordSerializer 类

```

class Meta:
    model = User
    fields = ['id', 'username', 'password', 'new_password']

@staticmethod
def get_new_password(obj):
    return obj.password or ''

```

2.3.2 增加 PasswordUpdateViewSet 类

密码修改的方式有两种，一种是通过修改密码功能修改，这个时候需要知道自己的原密码，然后修改成自己想要的新密码，一种是通过忘记密码功能修改，这个时候不需要知道自己的密码，但需要知道自己绑定的邮箱，新密码发送到邮箱里面。

- 在 common/views.py 中增加一个方法：get_random_password，该方法用来生成一个随即的密码，支撑忘记密码功能

```

def get_random_password():
    import random
    import string
    return ''.join(random.sample(string.ascii_letters + string.digits + string.pur

```

- 安装发送邮件所需要的依赖

```
pip install django-smtp-ssl==1.0
```

- 同时在 requirements.txt 文件中增加依赖

```
django-smtp-ssl==1.0
```

- 在 project/settings.py 中增加邮箱配置，这里的 EMAIL_HOST 和 EMAIL_PORT 是需要依据填写的邮箱做出调整，我这里填写的是网易的 163 邮箱

EMAIL_BACKEND = 'django_smtp_ssl.EmailBackend'

专栏
Python

```
EMAIL_HOST_USER = 'zgj0607@163.com'
EMAIL_HOST_PASSWORD = 'xxxx'
EMAIL_SUBJECT_PREFIX = u'[LSS]'
EMAIL_USE_SSL = True
```

- 在 common/views.py 中增加 PasswordUpdateViewSet，提供请求方式的接口。post 用 来完成修改密码功能。

```
class PasswordUpdateViewSet(GenericAPIView):
    permission_classes = [permissions.IsAuthenticated]
    serializer_class = UserPasswordSerializer
    queryset = User.objects.all()

    def post(self, request, *args, **kwargs):
        user_id = request.user.id
        password = request.data.get('password', '')
        new_password = request.data.get('new_password', '')
        user = User.objects.get(id=user_id)
        if not user.check_password(password):
            ret = {'detail': 'old password is wrong !'}
            return Response(ret, status=403)

        user.set_password(new_password)
        user.save()
        return Response({
            'detail': 'password changed successful !'
        })
```

- 在 UserLoginViewSet 中增加 put 方法，用于完成忘记密码功能，send_mail 使用的是 from django.core.mail import send_mail 语句导入。

将忘记密码的功能放在 LoginViewSet 类下的原因是登录接口和忘记密码的接口均是在不需要登录的情况下调用的接口，因此通过请求方式的不同来区分两种接口。

```
class UserLoginViewSet(GenericAPIView):
    permission_classes = [permissions.AllowAny]
    serializer_class = UserLoginSerializer
    queryset = User.objects.all()
```

知乎

专栏
Python

```
user = authenticate(username=username, password=password)
if user is not None and user.is_active:
    login(request, user)
    serializer = UserSerializer(user)
    return Response(serializer.data, status=200)
else:
    ret = {'detail': 'Username or password is wrong'}
    return Response(ret, status=403)

def put(self, request, *args, **kwargs):
    """
    Parameter: username->user's username who forget old password
    """
    username = request.data.get('username', '')
    users = User.objects.filter(username=username)
    user: User = users[0] if users else None

    if user is not None and user.is_active:
        password = get_random_password()

    try:
        send_mail(subject="New password for Library System",
                  message="Hi: Your new password is: \n{}".format(password),
                  from_email=django.conf.settings.EMAIL_HOST_USER,
                  recipient_list=[user.email],
                  fail_silently=False)
        user.password = make_password(password)
        user.save()
        return Response({
            'detail': 'New password will send to your email!'
        })
    except Exception as e:
        print(e)
        return Response({
            'detail': 'Send New email failed, Please check your email address'
        })
    else:
        ret = {'detail': 'User does not exist(Account is incorrect !)'}
        return Response(ret, status=403)
```



▲ 赞同 1



● 添加评论

↗ 分享

★ 收藏

⚑ 举报

收起 ^

```
from django.conf.urls import url
from django.urls import include, path
from rest_framework import routers

from common import views

router = routers.DefaultRouter()
router.register('user', views.UserViewSet)

app_name = 'common'

urlpatterns = [
    path('', include(router.urls)),
    url(r'^user/login', views.UserLoginViewSet.as_view()),
    url(r'^user/logout', views.UserLogoutViewSet.as_view()),
    url(r'^user/pwd', views.PasswordUpdateViewSet.as_view()),
]
```

至此，后端接口已经编写完成

三、前端页面开发

因为用户登录的场景有两个，一个是管理员登录后台，一个是游客登录博客网站，所以需要在两个地方完成用户的登录。

- 作为管理员登录后台系统，登录后需要进入到管理后台，所以需要单独的登录地址，因此提供一个单独的登录URL和页面。
- 作为游客，正常情况下，可以浏览博客，然后需要评论时，点击登录按钮，完成登录即可，因此需要一个登录对话框即可。

3.1 基于 TypeScript 要求增加 interface 定义

3.1.1 创建 types 文件夹

在 src 下创建文件夹 types，并在 types 下创建文件 index.ts

 赞同 1 添加评论 分享 收藏 举报

收起 ^

```
export interface User {  
    id: number,  
    username: string,  
    email: string,  
    avatar: string | any,  
    nickname: string | any,  
    is_active?: any,  
    is_superuser?: boolean,  
    created_at?: string,  
}  
  
export interface Nav {  
    index: string,  
    path: string,  
    name: string,  
}
```

其中 ? 表示可选属性，可以为空，| 表示属性值类型的可选项，可以多种类型的属性值，any 表示任何类型都可以。

3.2 基于 Vuex 保存 User 登录信息

3.2.1 新增文件夹 store

在 src 下创建文件夹 store，并在 store 文件夹下创建文件 index.ts

3.2.2 定义 User 和 Nav 相关的全局 state

1. 首先定义 state 的接口，目前我们需要用到三个state，一个是用户信息 User，一个是博客页面顶部导航的路由数据 navs，是一个 Nav 的数组，还有一个是当前导航菜单的索引 navIndex，表示当前页面是在哪一个菜单下。
2. 通过 Symbol 定义一个 InjectKey，用于在 Vue3 中通过 useState 获取到我们定义 state
3. 定义 state 在 dispatch 时用到的方法名，这里我们需要用到三个 setUser，clearUser，setNavIndex
4. 定义初始化 User 信息的方法，在登录完成后，我们为了保证用户信息在刷新页面后仍然可以识别用户是已经登录的状态，需要 sessionStorage 中存放登录后的用户信息，所以

src/store/index.ts 中代码如下：

```
import {InjectionKey} from 'vue'
import {createStore, Store} from 'vuex'
import { Nav, User} from "../types";

export interface State {
    user: User,
    navIndex: string,
    navs: Array<Nav>,
}

export const StateKey: InjectionKey<Store<State>> = Symbol();

export const SET_USER = 'setUser';
export const CLEAR_USER = 'clearUser'
export const SET_NAV_INDEX = 'setNavIndex'

export const initDefaultUserInfo = (): User => {
    let user: User = {
        id: 0,
        username: "",
        avatar: "",
        email: '',
        nickname: '',
        is_superuser: false,
    }
    if (window.sessionStorage.userInfo) {
        user = JSON.parse(window.sessionStorage.userInfo);
    }
    return user
}

export const store = createStore<State>({
    state() {
        return {
            user: initDefaultUserInfo(),
            navIndex: '1',
            navs: [
```

知乎

专栏
Python

```
        },
        {
            index: "2",
            path: "/catalog",
            name: "分类",
        },
        {
            index: "3",
            path: "/archive",
            name: "归档",
        },
        {
            index: "4",
            path: "/message",
            name: "留言",
        },
        {
            index: "5",
            path: "/about",
            name: "关于",
        },
    ],
},
},
mutations: {
    setUser(state: object | any, userInfo: object | any) {
        for (const prop in userInfo) {
            state[prop] = userInfo[prop];
        }
    },
    clearUser(state: object | any) {
        state.user = initDefaultUserInfo();
    },
    setNavIndex(state: object | any, navIndex: string) {
        state.navIndex = navIndex
    },
},
})
```

▲ 赞同 1 ▼

● 添加评论

↗ 分享

★ 收藏

⚑ 举报

收起 ^



在 src 下新增文件夹 views , 用于存放可以被 router 定义和管理的页面上

3.3.2 新增 components 文件夹

在 src 下新增文件夹 components , 用于存放页面上的可以复用的组件，这些组件一般不会出现在 router 中，而是通过 import 的方式使用

3.4 增加后端 API 调用方法

由于后端我们使用的 Django 和 Django Rest Framework 两个框架，对接口鉴权模式我们沿用了Django的 Session 模式，因此我们需要处理好跨域访问。

3.4.1 增加 getCookies 工具方法

在 src 下增加 utils 文件夹，在 src/utils 下新增文件 index.js 文件，编写如下代码：

```
export function getCookie(cName: string) {
  if (document.cookie.length > 0) {
    let cStart = document.cookie.indexOf(cName + "=");
    if (cStart !== -1) {
      cStart = cStart + cName.length + 1;
      let cEnd = document.cookie.indexOf(";", cStart);
      if (cEnd === -1) cEnd = document.cookie.length;
      return unescape(document.cookie.substring(cStart, cEnd));
    }
  }
  return "";
}
```

3.4.2 增加请求接口时登录和未登录的处理逻辑

在原来请求后端的定义上，改造 src/api/index.ts ，增加登录和未登录的处理逻辑。

1. Django Rest Framework 使用标准的 Http code 表示未授权登录，所以需要对 Http 的 code 做判断
2. 通过工具方法，在请求接口时，带上 X-CSRFToken

▲ 赞同 1

▼

● 添加评论

↗ 分享

★ 收藏

☞ 举报

收起 ^

知乎

专栏
Python

```
import router from "../router";
import {getCookie} from "../utils";

const request = axios.create({
  baseURL: import.meta.env.MODE !== 'production' ? '/api' : '',
})

request.interceptors.request.use((config: AxiosRequestConfig) => {
  // Django SessionAuthentication need csrf token
  config.headers['X-CSRFToken'] = getCookie('csrftoken')
  return config
})

request.interceptors.response.use(
  (response: AxiosResponse) => {
    const data = response.data
    console.log('response =>', response)
    if (data.status === '401') {
      localStorage.removeItem('user');
      ElMessage({
        message: data.error,
        type: 'error',
        duration: 1.5 * 1000
      })
      return router.push('/login')
    } else if (data.status === 'error') {
      ElMessage({
        message: data.error || data.status,
        type: 'error',
        duration: 1.5 * 1000
      })
    }
  }
)

if (data.success === false && data.msg) {
  ElMessage({
    message: data.msg,
    type: 'error',
    duration: 1.5 * 1000
  })
}
```

▲ 赞同 1



● 添加评论

↗ 分享

★ 收藏

⚑ 举报

收起 ^



```

    },
    ({message, response}) => {
        console.log('err => ', message, response) // for debug
        if (response && response.data && response.data.detail) {
            ElMessage({
                message: response.data.detail,
                type: 'error',
                duration: 2 * 1000
            })
        } else {
            ElMessage({
                message: message,
                type: 'error',
                duration: 2 * 1000
            })
        }
        if (response && (response.status === 403 || response.status === 401)) {
            localStorage.removeItem('user');
            return router.push('/login')
        }
        return Promise.reject(message)
    }
)
export default request;

```

3.4.3 增加后端接口请求方法

在 `src/api/service.ts` 下编写注册、登录、登出方法，代码如下：

```

export async function login(data: any) {
    return request({
        url: '/user/login',
        method: 'post',
        data
    })
}

export function logout() {
    return request({
        url: '/user/logout'
    })
}

```

```
export function register(data: any) {  
    return request({  
        url: '/user/',  
        method: 'post',  
        data  
    })  
}
```

3.5 编写主页和后台登录页面

3.5.1 增加 HelloWorld 的主页

编写一个真正意义上的Hello World 页面在 `src/views` 新增文件 `Home.vue`，后面用于普通用户进入博客网站时看到的第一个页面。代码如下：

```
<template>  
    <h3>HelloWorld</h3>  
</template>  
  
<script lang="ts">  
import { defineComponent, reactive } from "vue";  
export default defineComponent({  
    name: 'Home',  
})  
</script>
```

3.5.2 增加后台登录页面 `Login.vue`

该页面用于管理员登录管理后台，登录成功后进入到后台页面，登录成功后，会通过 `store` 提供的 `dispatch` 方法对全局 `state` 进行修改

3.5.2.1 编写 template 部分

```
<template>  
    <div>...</div>
```

▲ 赞同 1 ▾ 添加评论 分享 收藏 举报 收起 ^

```
:rules="rules"
autocomplete="on"
class="login-form"
label-position="left"
>
<div class="title-container">
    <h3 class="title">博客管理后台</h3>
</div>

<el-form-item prop="account">
    <el-input
        ref="account"
        v-model="state.loginForm.account"
        autocomplete="on"
        name="account"
        placeholder="Account"
        tabindex="1"
        type="text"
    />
</el-form-item>

<el-tooltip
    v-model="state.capsTooltip"
    content="Caps lock is On"
    manual
    placement="right"
>
    <el-form-item prop="password">
        <el-input
            :key="state.passwordType"
            ref="password"
            v-model="state.loginForm.password"
            :type="state.passwordType"
            autocomplete="on"
            name="password"
            placeholder="Password"
            tabindex="2"
            @blur="state.capsTooltip = false"
            @keyup="checkCapslock"
            @keyup.enter="handleLogin"
        />
    </el-form-item>
</el-tooltip>
```

```

<el-button
  :loading="state.loading"
  style="width: 100%; margin-bottom: 30px"
  type="primary"
  @click.prevent="handleLogin"
>
  Login
</el-button>
</el-form>
</div>
</template>

```

3.5.2.2 编写 script 部分

由于我们用的是 TypeScript，所以在 script 后面加上 lang=ts

```

<script lang="ts">
import { defineComponent, reactive } from "vue";
import { forgetPassword, login } from "../api/service";
import { SET_USER } from "../store";
import { User } from "../types";

export default defineComponent({
  name: "Login",
  setup() {
    const validatePassword = (rule: any, value: string, callback: Function) => {
      if (value.length < 6) {
        callback(new Error("The password can not be less than 6 digits"));
      } else {
        callback();
      }
    };
    const state = reactive({
      loginForm: {
        account: "",
        password: "",
      },
      loginRules: {
        account: [
          { required: true, trigger: "blur" }
        ]
      }
    });
    const handleLogin = () => {
      login(state.loginForm).then((res) => {
        if (res.code === 200) {
          SET_USER(res.data);
          router.push("/");
        } else {
          message.error(res.message);
        }
      });
    };
    const handleForget = () => {
      forgetPassword().then((res) => {
        if (res.code === 200) {
          message.success(res.message);
        } else {
          message.error(res.message);
        }
      });
    };
  }
});

```

知乎

专栏
Python

```
        trigger: "blur",
        validator: validatePassword,
    },
],
},
forgetRules: {
    account: [{ required: true, trigger: "blur" }],
},
passwordType: "password",
capsTooltip: false,
loading: false,
isFP: false,
});

return {
    state,
    validatePassword,
};
},
mounted() {
    if (this.state.loginForm.account === "") {
        this.$refs.account.focus();
    } else if (this.state.loginForm.password === "") {
        this.$refs.password.focus();
    }
},
computed: {
    rules() {
        return this.state.isFP ? this.state.forgetRules : this.state.loginRules;
    },
},
methods: {
    checkCapslock(e: KeyboardEvent) {
        const { key } = e;
        this.state.capsTooltip =
            key && key.length === 1 && key >= "A" && key <= "Z";
    },
    handleLogin() {
        this.state.isFP = false;
        this.$refs.loginForm.validate(async (valid: Boolean) => {
            if (valid) {

```

▲ 赞同 1



● 添加评论

↗ 分享

★ 收藏

⚑ 举报

收起 ^

知乎

专栏
Python

```
};

try {
    const data: any = await login(req);
    const user: User = {
        id: data.id,
        username: data.username,
        avatar: data.avatar,
        email: data.email,
        nickname: data.nickname,
    };

    this.$store.commit(SET_USER, {
        user,
    });
    window.sessionStorage.userInfo = JSON.stringify(user);
    await this.$router.push({
        path: "/admin",
    });
    this.state.loading = false;
} catch (e) {
    this.state.loading = false;
}
});

},
startFp() {
    this.state.isFP = true;
    this.$refs.loginForm.clearValidate();
    this.$nextTick(() => {
        this.$refs.loginForm.validate((valid: Boolean) => {
            if (valid) {
                this.$confirm(
                    "We will send a new password to " + this.state.loginForm.account,
                    "Tip",
                ) {
                    confirmButtonText: "OK",
                    cancelButtonText: "Cancel",
                    type: "warning",
                }
            ).then(() => {
                forgetPassword({ account: this.state.loginForm.account }).then(

```

▲ 赞同 1 ▼

● 添加评论

↗ 分享

★ 收藏

⚑ 举报

收起 ^



```
        type: "success",
        duration: 1.5 * 1000,
    });
}
);
});
});
});
},
},
);
</script>
```

3.5.2.3 编写 CSS 部分

由于我们使用的是 less 语法，所以在 style 后面需要加上 lang="less"，同时控制css的作用域，添加 scoped

```
<style lang="less" scoped>
.login-container {
    min-height: 100%;
    width: 100%;
    overflow: hidden;
    background-repeat: no-repeat;
    background-position: center;
    display: flex;
    align-items: center;
    justify-content: center;
    filter: hue-rotate(200deg);
}

.login-form {
    //position: absolute;
    width: 300px;
    max-width: 100%;
    overflow: hidden;
    top: 0;
    left: 0;
    right: 0;
    bottom: 0;
    margin: auto;
    padding: 20px;
    border: 1px solid #ccc;
    border-radius: 5px;
    background-color: #fff;
    box-sizing: border-box;
}
```

▲ 赞同 1

▼

● 添加评论

↗ 分享

★ 收藏

⚑ 举报

收起 ^

知乎

专栏
Python

```
height: 350px;  
}  
  
.tips {  
    font-size: 14px;  
    color: #fff;  
    margin-bottom: 10px;  
}  
  
.tips span:first-of-type {  
    margin-right: 16px;  
}  
  
.svg-container {  
    padding: 6px 5px 6px 15px;  
    color: #889aa4;  
    vertical-align: middle;  
    width: 30px;  
    display: inline-block;  
}  
  
.title-container {  
    position: relative;  
    color: #333;  
}  
  
.title-container .title {  
    font-size: 40px;  
    margin: 0px auto 40px auto;  
    text-align: center;  
    font-weight: bold;  
}  
  
.show-pwd {  
    position: absolute;  
    right: 10px;  
    top: 7px;  
    font-size: 16px;  
    color: #889aa4;  
    cursor: pointer;  
    user-select: none;
```

```
        right: 0;
        bottom: 6px;
    }

    .fp {
        font-size: 12px;
        text-align: right;
        margin-bottom: 10px;
        cursor: pointer;
    }
</style>
```

3.6 定义路由

现在我们已经有了 `Login.vue` 和 `Home.vue` 两个页面了，现在可以定义路由了。

1. 我们采用 `WebHistory` 的方式展示路由，这种方式在浏览器的地址栏中展示的URL更优美
2. 采用 `History` 的方式后，我们需要在 `vite.config.ts` 中定义 `base` 时用这种： `base: '/'`，在 / 前不能增加 .
3. 对首页以外的页面，采用 `import` 懒加载，需要的时候再加载

在 `src/router/index.ts` 文件中编写如下代码：

```
import {createRouter, createWebHistory, RouteRecordRaw} from "vue-router";
import Home from "../views/Home.vue";

const routes: Array<RouteRecordRaw> = [
    {
        path: "/",
        name: "Home",
        component: Home,
        meta: {}
    },
    {
        path: "/login/",
        name: "Login",
        component: () =>
            import(/* webpackChunkName: "login" */ "../views/Login.vue")
    },
]
```

▲ 赞同 1



● 添加评论

↗ 分享

★ 收藏

☞ 举报

收起 ^



```
routes,  
});  
  
export default router;
```

3.7 改造 main.ts

在 main.ts 中我们需要处理如下逻辑：

1. 创建APP
2. 加载 Element-Plus 的组件
3. 加载 Element-Plus 的插件
4. 加载 Vue-Router 的路由
5. 加载 Vuex 的 state

完整代码：

```
import { createApp } from 'vue'  
import App from './App.vue'  
import router from "./router";  
import { StateKey, store } from "./store";  
import 'element-plus/lib/theme-chalk/index.css';  
import 'element-plus/lib/theme-chalk/base.css';  
  
import {  
    ElAffix,  
    ElButton,  
    ElCard,  
    ElCascader,  
    ElCol,  
    ElDescriptions,  
    ElDescriptionsItem,  
    ElDialog,  
    ElDrawer,  
    ElDropdown,  
    ElDropdownItem,  
    ElDropdownMenu,  
    ElForm,
```



```
ElMenu,  
ElMenuItem,  
ElMessage,  
ElMessageBox,  
ElOption,  
ElPagination,  
ElPopconfirm,  
ElProgress,  
ElRow,  
ElSelect,  
ElTable,  
ElTableColumn,  
ElTag,  
ElTimeline,  
ElTimelineItem,  
ElTooltip,  
ElTree,  
ElUpload,  
} from 'element-plus';  
  
const app = createApp(App)  
  
const components = [  
  ElAffix,  
  ElButton,  
  ElCard,  
  ElCascader,  
  ElCol,  
  ElDescriptions,  
  ElDescriptionsItem,  
  ElDialog,  
  ElDrawer,  
  ElDropdown,  
  ElDropdownItem,  
  ElDropdownMenu,  
  ElForm,  
  ElFormItem,  
  ElIcon,  
  ElInput,  
  ElLoading,
```

▲ 赞同 1▼添加评论分享收藏举报收起 ^

```
ElOption,  
ElPagination,  
ElPopconfirm,  
ElProgress,  
ElRow,  
ElSelect,  
ElTable,  
ElTableColumn,  
ElTag,  
ElTimeline,  
ElTimelineItem,  
ElTooltip,  
ElTree,  
ElUpload,  
]  
  
const plugins = [  
    ElLoading,  
    ElMessage,  
    ElMessageBox,  
]  
  
components.forEach(component => {  
    app.component(component.name, component)  
})  
  
plugins.forEach(plugin => {  
    app.use(plugin)  
})  
  
app.use(router).use(store, StateKey).mount('#app')
```

3.8 改造 App.vue

在上一篇中我们为了测试前后端的连通性，将 `App.vue` 直接处理成了一个表格展示用户列表的页面，而实际的项目中，该页面是需要处理路由导航等相关功能的，因此我们先将该页面改造成直接菜单导航的方式。

在游客需要评论的时候，需要完成登录后才可以，所以这里的登录和管理员的登录方式是不一样的。

▲ 赞同 1 ▼ 添加评论 分享 收藏 举报

收起 ^



3. 登录后，不需要做页面跳转，只需要在右上角显示用户的昵称或账号，表示用户已经登录
4. 登录后，可以登出

3.8.1 增加 RegisterAndLogin.vue 组件

这里的注册和登录复用同一个组件，通过按钮点击的不同，展示不同的内容。

1. 需要校验输入的内容
2. 登录成功后，需要更新全局 state 中的用户信息

具体代码如下：

```
<template>
  <el-dialog
    title="登录"
    width="40%"
    v-model="state.dialogModal"
    @close="cancel"
    :show-close="true"
  >
    <el-form>
      <el-formItem label="账号" :label-width="state.formLabelWidth">
        <el-input
          v-model="state.params.username"
          placeholder="请输入有效邮箱"
          autocomplete="off"
        />
      </el-formItem>
      <el-formItem label="密码" :label-width="state.formLabelWidth">
        <el-input
          type="password"
          placeholder="密码"
          v-model="state.params.password"
          autocomplete="off"
        />
      </el-formItem>
      <el-formItem
        v-if="isRegister"
        label="昵称"
      >
    </el-form>
  </el-dialog>
```

知乎

专栏
Python

```
placeholder="用户名或昵称"
autocomplete="off"
/>
</el-formItem>
<el-formItem
  v-if="isRegister"
  label="手机"
  :label-width="state.formLabelWidth"
>
  <el-input
    v-model="state.params.phone"
    placeholder="手机号"
    autocomplete="off"
  />
</el-formItem>
<el-formItem
  v-if="isRegister"
  label="简介"
  :label-width="state.formLabelWidth"
>
  <el-input
    v-model="state.params.desc"
    placeholder="个人简介"
    autocomplete="off"
  />
</el-formItem>
</el-form>
<template v-slot:footer>
  <div class="dialog-footer">
    <el-button
      v-if="isLogin"
      :loading="state.btnLoading"
      type="primary"
      @click="handleOk"
    >
      登录
    </el-button>
    <el-button
      v-if="isRegister"
      :loading="state.btnLoading"
      type="primary"
    >
```

▲ 赞同 1



● 添加评论

↗ 分享

★ 收藏

⚑ 举报

收起 ^



```
</template>
</el-dialog>
</template>

<script lang="ts">
import { defineComponent, reactive, watch } from "vue";
import { useStore } from "vuex";
import { ElMessage } from "element-plus";
import { SET_USER, StateKey } from "../store";
import { login, register } from "../api/service";
import { User } from "../types";

export default defineComponent({
  name: "RegisterAndLogin",
  props: {
    visible: {
      type: Boolean,
      default: false,
    },
    handleFlag: {
      type: String,
      default: false,
    },
  },
  computed: {
    isLoggedIn(): Boolean {
      return this.handleFlag === "login";
    },
    isRegister(): Boolean {
      return this.handleFlag === "register";
    },
  },
  emits: ["ok", "cancel"],
  setup(props, context) {
    const store = useStore(StateKey);
    const state = reactive({
      dialogModal: props.visible,
      btnLoading: false,
      loading: false,
      formLabelWidth: "60px",
      params: {
        type: "register"
      }
    });
    watch(state, () => {
      if (state.loading) {
        ElMessage.error("登录失败");
        context.emit("cancel");
      }
    });
    const handleOk = () => {
      if (props.handleFlag === "register") {
        register(state.params).then(() => {
          ElMessage.success("注册成功");
          store.dispatch(SET_USER, state.params);
          context.emit("ok");
        }).catch(() => {
          ElMessage.error("注册失败");
        });
      } else {
        login(state.params).then(() => {
          ElMessage.success("登录成功");
          store.dispatch(SET_USER, state.params);
          context.emit("ok");
        }).catch(() => {
          ElMessage.error("登录失败");
        });
      }
    };
    const handleCancel = () => {
      context.emit("cancel");
    };
    const handleClose = () => {
      context.emit("close");
    };
    const handleUpdate = (val: string) => {
      state.params.type = val;
    };
    const handleVisibleChange = (val: boolean) => {
      state.dialogModal = val;
    };
    const handleBtnLoadingChange = (val: boolean) => {
      state.btnLoading = val;
    };
    const handleLoadingChange = (val: boolean) => {
      state.loading = val;
    };
    const handleFormLabelWidthChange = (val: string) => {
      state.formLabelWidth = val;
    };
    const handleParamsChange = (val: object) => {
      state.params = val;
    };
  }
})
```

知乎

专栏
Python

```
    phone: "",  
    desc: "",  
,  
});  
  
const submit = async (): Promise<void> => {  
let data: any = "";  
state.btnLoading = true;  
try {  
    if (props.handleFlag === "register") {  
        state.params.email = state.params.username;  
        data = await register(state.params);  
    } else {  
        data = await login(state.params);  
    }  
    state.btnLoading = false;  
  
    const user: User = {  
        id: data.id,  
        username: data.username,  
        avatar: data.avatar,  
        email: data.email,  
        nickname: data.nickname,  
    };  
    store.commit(SET_USER, {  
        user,  
    });  
    window.sessionStorage.userInfo = JSON.stringify(user);  
    context.emit("ok", false);  
    ElMessage({  
        message: "操作成功",  
        type: "success",  
    });  
    state.dialogModal = false;  
} catch (e) {  
    console.error(e);  
    state.btnLoading = false;  
}  
};  
  
const handleOk = (): void => {
```

▲ 赞同 1

▼

● 添加评论

↗ 分享

★ 收藏

⚑ 举报

收起 ^



```
ElMessage({
    message: "账号不能为空!",
    type: "warning",
});
return;
} else if (!reg.test(state.params.username)) {
    ElMessage({
        message: "请输入格式正确的邮箱!",
        type: "warning",
    });
    return;
}
if (props.handleFlag === "register") {
    if (!state.params.password) {
        ElMessage({
            message: "密码不能为空!",
            type: "warning",
        });
        return;
    } else if (!state.params.nickname) {
        ElMessage({
            message: "昵称不能为空!",
            type: "warning",
        });
        return;
    }
    const re = /^(((13[0-9])|(15[0-9])|(17[0-9])|(18[0-9]))+\d{8})$/;
    if (state.params.phone && !re.test(state.params.phone)) {
        ElMessage({
            message: "请输入正确的手机号!",
            type: "warning",
        });
        return;
    }
}
submit();
};

const cancel = (): boolean => {
    context.emit("cancel", false);
    return false;
}
```

[▲ 赞同 1](#)[添加评论](#)[分享](#)[收藏](#)[举报](#)[收起 ^](#)

```
});  
  
    return {  
        state,  
        handleOk,  
        submit,  
        cancel,  
    };  
},  
});  
</script>  
<style scoped>  
.dialog-footer {  
    text-align: right;  
}  
</style>
```

3.8.2 编写 Nav.vue

这个页面处理顶部导航的功能，引用了 RegisterAndLogin.vue 组件。

代码如下：

```
<template>  
<div class="nav">  
    <div class="nav-content">  
        <el-row :gutter="20">  
            <el-col :span="3">  
                <router-link to="/">  
                      
                </router-link>  
            </el-col>  
            <el-col :span="16">  
                <el-menu  
                    :default-active="navIndex"  
                    :router="true"  
                    active-text-color="#409EFF"  
                    class="el-menu-demo"  
                >
```

知乎

专栏
Python

```
:key="r.index"
:index="r.index"
:route="r.path"
>
{{ r.name }}
</el-menuItem>
</el-menu>
</el-col>
<el-col v-if="isLogin" :span="5">
<div class="nav-right">
<el-dropdown>
<span class="el-dropdown-link">
{{ userInfo.nickname ? userInfo.nickname : userInfo.username }}<i class="el-icon-arrow-down el-icon--right"></i>
</span>


<template #dropdown>
<el-dropdown-menu>
<el-dropdown-item @click="handleClick"
>登 出</el-dropdown-item>
</el-dropdown-menu>
</template>
</el-dropdown>
</div>
</el-col>
<el-col v-else :span="4">
<div class="nav-right" v-if="!isLogin">
<el-button
size="small"
```

▲ 赞同 1



● 添加评论

↗ 分享

★ 收藏

⚑ 举报

收起 ^

知乎

专栏
Python

```
>
<el-button
    size="small"
    type="danger"
    @click="handleClick('register')"
>
    注册
</el-button>
</div>
<RegisterAndLogin
    :handle-flag="state.handleFlag"
    :visible="state.visible"
/>
</el-col>
</el-row>
</div>
</div>
</template>

<script lang="ts">
import { defineComponent, reactive } from "vue";
import { User } from "../types";
import { useStore } from "vuex";
import { CLEAR_USER, SET_NAV_INDEX, StateKey } from "../store";
import RegisterAndLogin from "./RegisterAndLogin.vue";
import { logout } from "../api/service";

export default defineComponent({
    name: "Nav",
    components: { RegisterAndLogin },
    computed: {
        userInfo(): User {
            const store = useStore(StateKey);
            return store.state.user;
        },
        isLogin(): Boolean {
            return this.userInfo.id > 0;
        },
        navs(){
            const store = useStore(StateKey);
            return store.state.navs;
        }
    }
})
```

▲ 赞同 1

▼

● 添加评论

↗ 分享

★ 收藏

⚑ 举报

收起 ^

知乎

专栏
Python

```
,  
},  
watch: {  
  $route: {  
    handler(val: any, oldVal: any) {  
      this.routeChange(val, oldVal);  
    },  
    immediate: true,  
  },  
},  
setup() {  
  const state = reactive({  
    handleFlag: "",  
    visible: false,  
    title: "主页",  
  });  
  
  const store = useStore(StateKey);  
  
  const routeChange = (newRoute: any, oldRoute: any): void => {  
    for (let i = 0; i < store.state.navs.length; i++) {  
      const l = store.state.navs[i];  
      if (l.path === newRoute.path) {  
        state.title = l.name;  
        store.commit(SET_NAV_INDEX, l.index);  
        return;  
      }  
    }  
    store.commit(SET_NAV_INDEX, "-1");  
  };  
  
  const handleClick = async (route: string) => {  
    if (["login", "register"].includes(route)) {  
      state.handleFlag = route;  
      state.visible = true;  
    } else {  
      await logout();  
      window.sessionStorage.userInfo = "";  
      store.commit(CLEAR_USER);  
    }  
  };  
};
```

▲ 赞同 1



● 添加评论

↗ 分享

★ 收藏

⚑ 举报

收起 ^

知乎

专栏
Python

```
        routeChange,  
    );  
},  
});  
</script>  
  
<style lang="less">  
.nav {  
    position: fixed;  
    top: 0;  
    left: 0;  
    z-index: 1000;  
    width: 100%;  
    border-bottom: 1px solid #eee;  
    background-color: #fff;  
  
.nav-content {  
    width: 1200px;  
    margin: 0 auto;  
}  
  
.logo {  
    height: 50px;  
    margin: 0;  
    border-radius: 50%;  
    margin-top: 5px;  
}  
  
.el-menu.el-menu--horizontal {  
    border-bottom: none;  
}  
  
.el-menu--horizontal > .el-menu-item {  
    cursor: pointer;  
    color: #333;  
}  
  
.nav-right {  
    position: relative;  
    padding-top: 15px;  
    text-align: right;  
}
```

}

```
.user-img {  
    position: absolute;  
    top: -15px;  
    right: 0;  
    width: 50px;  
    border-radius: 50%;  
}  
}  
}  
  
</style>
```

3.8.3 修改 App.vue

在 `src/App.vue` 下编写如下代码

1. 其中 `Nav` 是用来做导航用的，当浏览器中的地址发生变化时，`router/index.ts` 中定义的路由对应的页面就会渲染到 `router-view` 标签中
2. 通过 Vue 3 的 `defineComponent` 定义 `App` 组件，并导入 `Nav` 组件
3. `css` 部分需要在子组件中生效

```
<template>  
  <div class="container">  
    <Nav/>  
    <div class="layout">  
      <router-view class="view-content"/>  
    </div>  
  </div>  
</template>  
  
<script lang="ts">  
import { defineComponent } from "vue";  
import Nav from "./components/Nav.vue";  
  
export default defineComponent({  
  name: "App",
```

[赞同 1](#)[添加评论](#)[分享](#)[收藏](#)[举报](#)[收起 ^](#)

知乎

专栏
Python

```
</script>

<style lang="less">
body {
    background-color: #f9f9f9;
}

#app {
    font-family: Avenir, Helvetica, Arial, sans-serif;
    -webkit-font-smoothing: antialiased;
    -moz-osx-font-smoothing: grayscale;
    text-align: left;
    padding-top: 50px;
}

.container {
    width: 1200px;
    margin: 0 auto;
}

img {
    vertical-align: bottom;
}

.layout {
    height: auto;
}

.button-container {
    display: flex;
    justify-content: space-between;
    flex: 1;
    margin-bottom: 24px;
}

.view-content {
    margin-top: 12px;
    background-color: #ffffff;
    padding: 12px 24px 24px 24px;
    border-radius: 8px;
}
```



经过这么一波调整后，运行起来的效果如下图：

3.9.1 代码结构

3.9.1.1 前端代码结构

▲ 赞同 1



● 添加评论



分享



收藏



举报

收起 ^

知乎

专栏
Python

A screenshot of a file explorer window showing a project structure. The root directory contains node_modules, public, src, assets, components, less, router, store, types, utils, views, and several configuration files. The src directory is expanded, showing api, assets, components, less, router, store, types, and utils sub-directories. The components directory contains HelloWorld.vue, Nav.vue, and RegisterAndLogin.vue. The views directory contains Home.vue, Login.vue, and App.vue. Configuration files include .gitignore, index.html, package-lock.json, package.json, README.en.md, README.md, tsconfig.json, vite.config.ts, and yarn.lock.

```
node_modules
public
src
  api
    index.ts
    service.ts
  assets
    logo.jpeg
    logo.png
    user.png
  components
    HelloWorld.vue
    Nav.vue
    RegisterAndLogin.vue
  less
    index.less
  router
    index.ts
  store
    index.ts
  types
    index.ts
  utils
    index.ts
  views
    Home.vue
    Login.vue
    App.vue
  main.ts
  shims-vue.d.ts
  vite-env.d.ts
.gitignore
index.html
package-lock.json
package.json
README.en.md
README.md
tsconfig.json
vite.config.ts
yarn.lock
```

知乎 @落霞孤鹜

▲ 赞同 1 ▼

● 添加评论

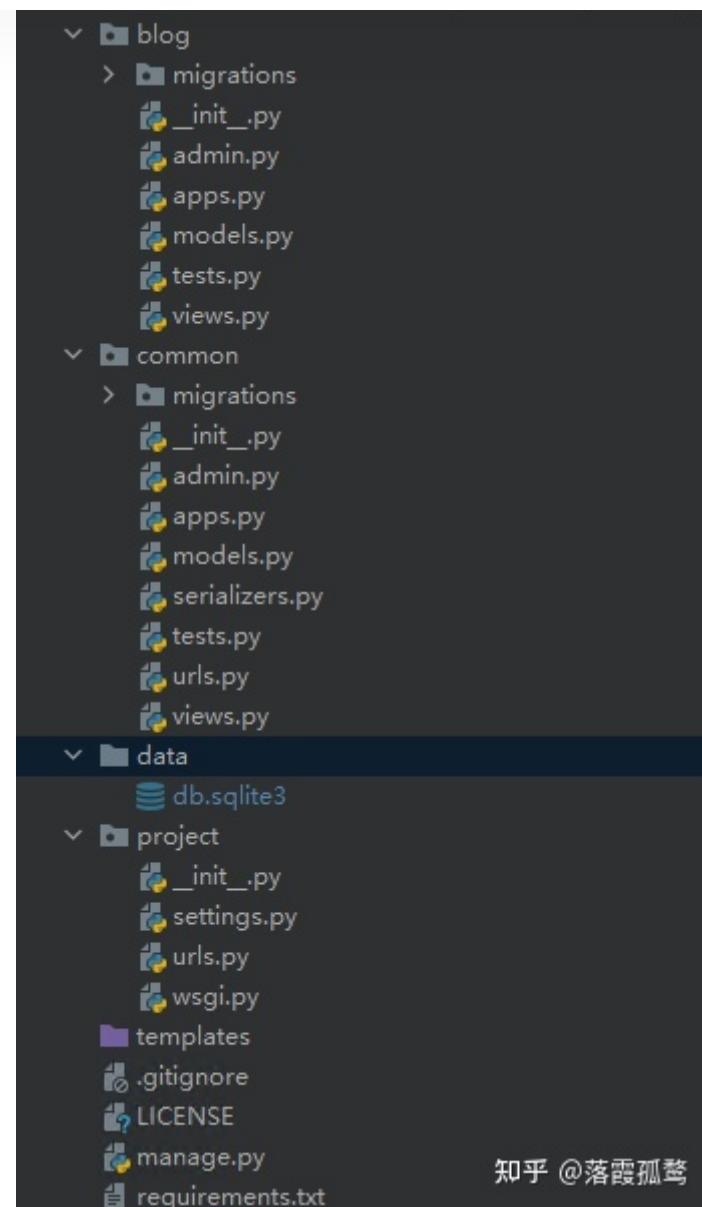
↗ 分享

★ 收藏

☞ 举报

收起 ^

知乎

 专栏
Python

知乎 @落霞孤鹜

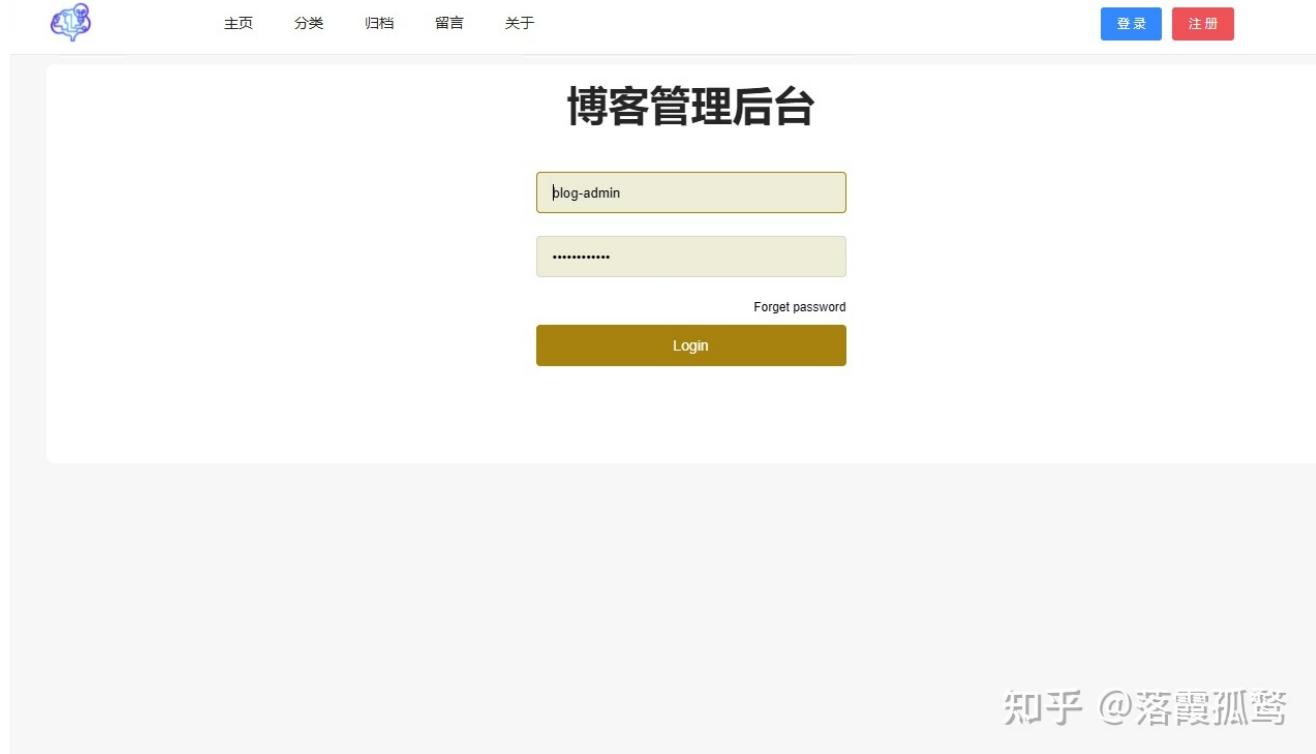
3.9.1 首页效果

[▲ 赞同 1](#)[● 添加评论](#)[↗ 分享](#)[★ 收藏](#)[⚑ 举报](#)[收起 ^](#)



知乎 @落霞孤鹜

3.9.2 管理员登录页面



The screenshot shows a login interface for a blog management system. At the top, there is a navigation bar with links for '主页' (Home), '分类' (Categories), '归档' (Archives), '留言' (Comments), and '关于' (About). On the right side of the navigation bar are two buttons: '登录' (Login) in blue and '注册' (Register) in red. The main area is titled '博客管理后台' (Blog Management Backend). It contains a form with two input fields: the first is pre-filled with 'blog-admin' and the second is a password field with several dots. Below the password field is a 'Forget password' link. At the bottom of the form is a large orange 'Login' button.

3.9.3 游客注册页面

▲ 赞同 1 ▼ ● 添加评论 ↗ 分享 ★ 收藏 📢 举报 收起 ^



登录

账号

密码

昵称

手机

简介

[注册](#)

知乎 @落霞孤鹜

3.9.4 游客登录页面

HelloWorld

登录

账号

密码

[登录](#)

知乎 @落霞孤鹜

编辑于 2021-08-09 21:21

▲ 赞同 3

▼

● 添加评论

▲ 分享

◆ 收藏

⚑ 举报

收起 ▲

▲ 赞同 1

▼

● 添加评论

▲ 分享

★ 收藏

⚑ 举报

收起 ▲

初始化



落霞孤鹜，求知若渴

4 人赞同了该文章



搭建个人博客（一）：框架代码初始化

本文适合对有 Python 语言有一定基础的人群，希望利用 Python 做更多有意思的事情，比如搭建个人博客，记录自己的所思所想，或者想找一个项目实践前后端分离技术等等。跟着本文可以了解和运行项目，本项目是在 Window 10 Professional 系统下开发

大家好，我是 落霞孤鹜，上一篇介绍了开发博客的背景、技术栈，并介绍了如何搭建开发环境。这一篇介绍后端和前端的基础框架代码初始化，基于Django和Vue初始化项目框架代码，跑通Hello world。

一、后端框架代码搭建

后端 Python 代码通过 PyCharm 能比较快速的搭建 Django 项目，因为在 PyCharm 的专业版里面，已经内置了 Django 框架

▲ 赞同 1



● 添加评论

↗ 分享

★ 收藏

⚑ 举报

收起 ^

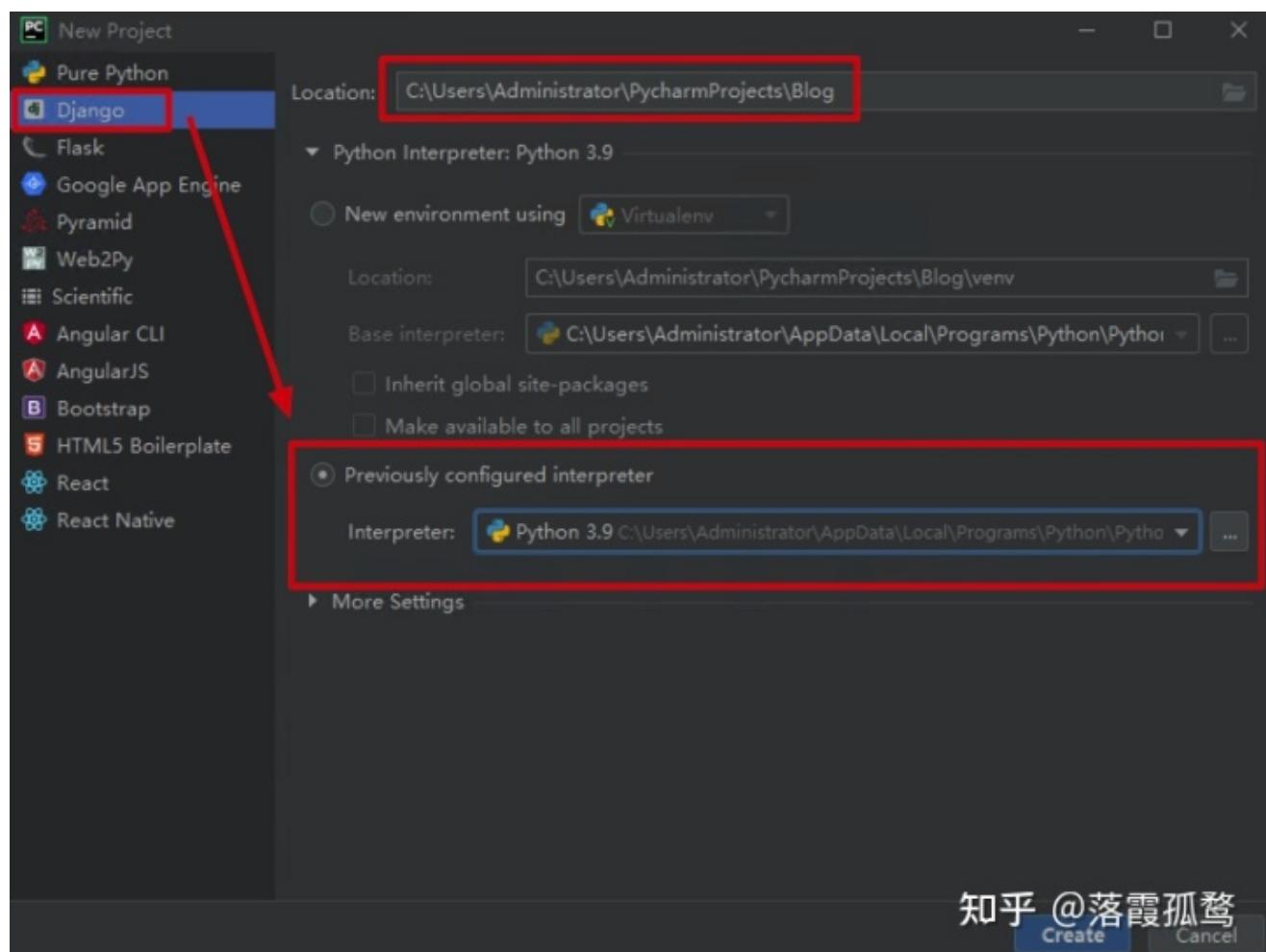
知乎

专栏
Python

为了更好的兼容性，我们自己安装 Django 2 版本，不采用最新版本。在命令行输入如下命令：

```
pip install django==2.2.23
```

1. 在 PyCharm 的首屏界面，点击 New Project 对话框，在左侧选择 Django，在右侧的 Location 中选择项目地址，项目命令为 Blog 并将我们之前安装的 Python 路径选择为 Interpreter，如下图：



1. 点击Create，等待 PyCharm 执行创建。

如果选择的 Python Interpreter 环境中没有安装 Django，PyCharm 会自动安装 Django 最新版本，由于我们已经安装了Django，PyCharm 会自动使用环境中的 Django 版本

▲ 赞同 1 ▼

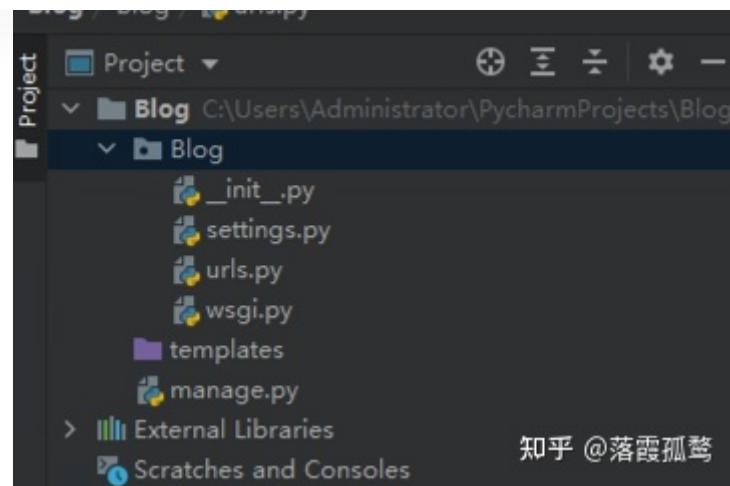
● 添加评论

↗ 分享

★ 收藏

⚑ 举报

收起 ^



1. 在 Pycharm 右下角点击 Terminal，通过 pip 安装 Django Rest Framework

```
pip install djangorestframework==3.12.4
```

1. 验证框架是否可以运行

运行点击 PyCharm 右上角的运行按钮，如果正常，在PyCharm的运行控制台会打印如下信息

```
Performing system checks...
```

```
Watching for file changes with StatReloader
```

```
System check identified no issues (0 silenced).
```

```
You have 17 unapplied migration(s). Your project may not work properly until you run:
```

```
Run 'python manage.py migrate' to apply them.
```

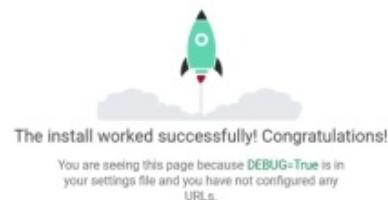
```
July 17, 2021 - 17:42:28
```

```
Django version 2.2.23, using settings 'project.settings'
```

```
Starting development server at http://127.0.0.1:8000/
```

```
Quit the server with CTRL-BREAK.
```

打开 Edge 或 Chrome 浏览器，输入 `http://127.0.0.1:8000`，回车，如下图，说明框架搭建成功



 Django Documentation
Topics, references, & how-tos

 Tutorial: A Polling App
Get started with Django

 Django Community
Connect, get help, or contribute

知乎 @落霞孤鹜

1.2 配置 Django Rest Framework

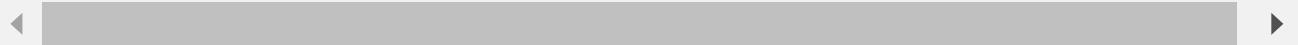
1. 启用 Django Rest Framework

在 Blog 文件夹下，打开 settings.py 文件，在 INSTALLED_APPS 的 list 中增加 rest_framework

```
python INSTALLED_APPS = [ 'django.contrib.admin', 'django.contrib.auth',  
'django.contrib.contenttypes', 'django.contrib.sessions',  
'django.contrib.messages', 'django.contrib.staticfiles', 'rest_framework', ]
```

1. 在 settings.py 中增加 Rest Framework 的配置

```
REST_FRAMEWORK = {  
    'DEFAULT_PAGINATION_CLASS': 'rest_framework.pagination.PageNumberPagination',  
    'PAGE_SIZE': 10  
}
```



1.3 配置 Sqlite 数据库

1. 在项目路径下，创建data文件夹

▲ 赞同 1 ▼ 添加评论 分享 收藏 举报

收起 ^



```
'ENGINE': 'django.db.backends.sqlite3',
'NAME': os.path.join(BASE_DIR, 'data/db.sqlite3'),
}

}
```

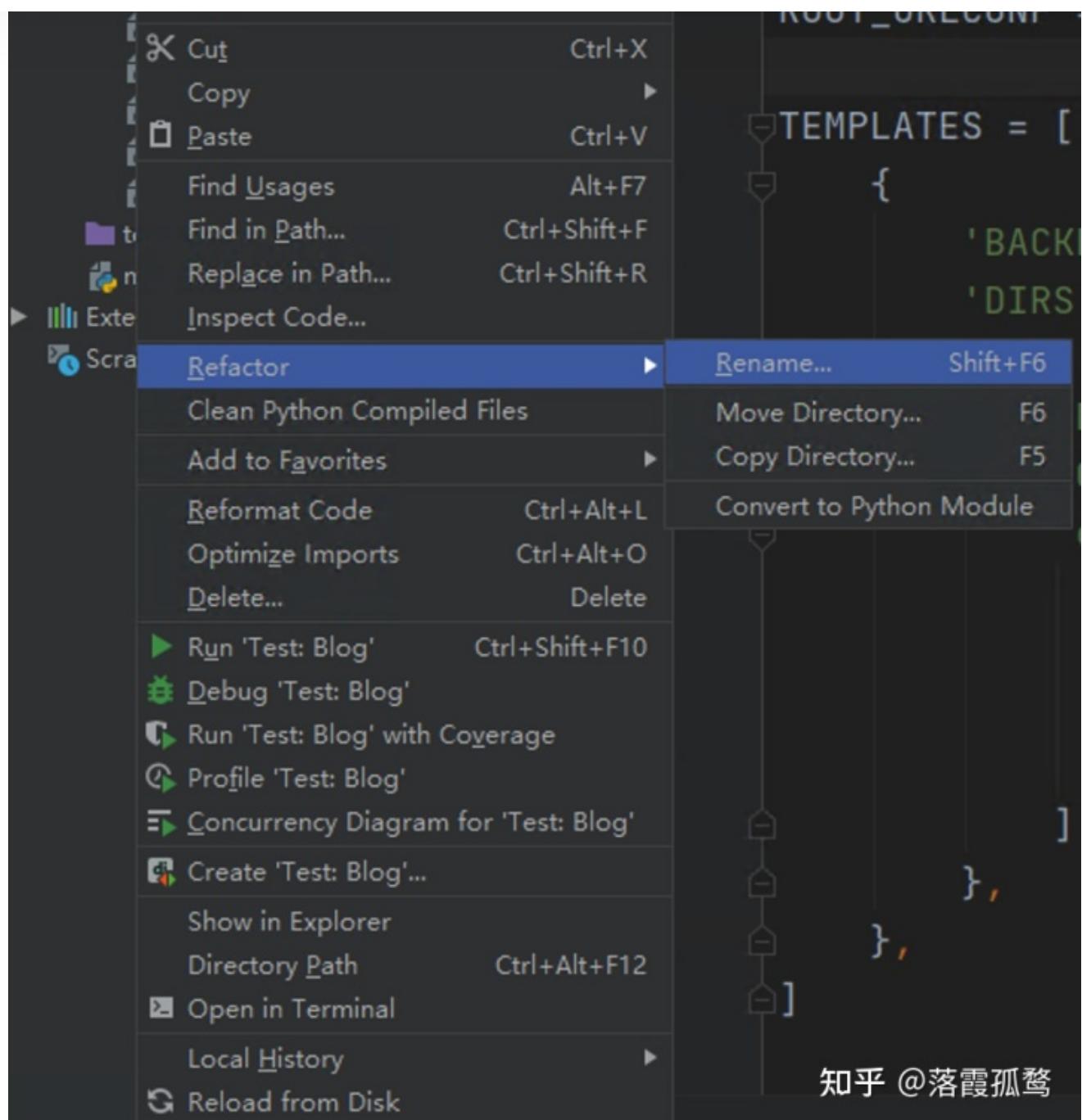
1.4 调整项目结构

1. 修改 Blog 文件夹名称为 project

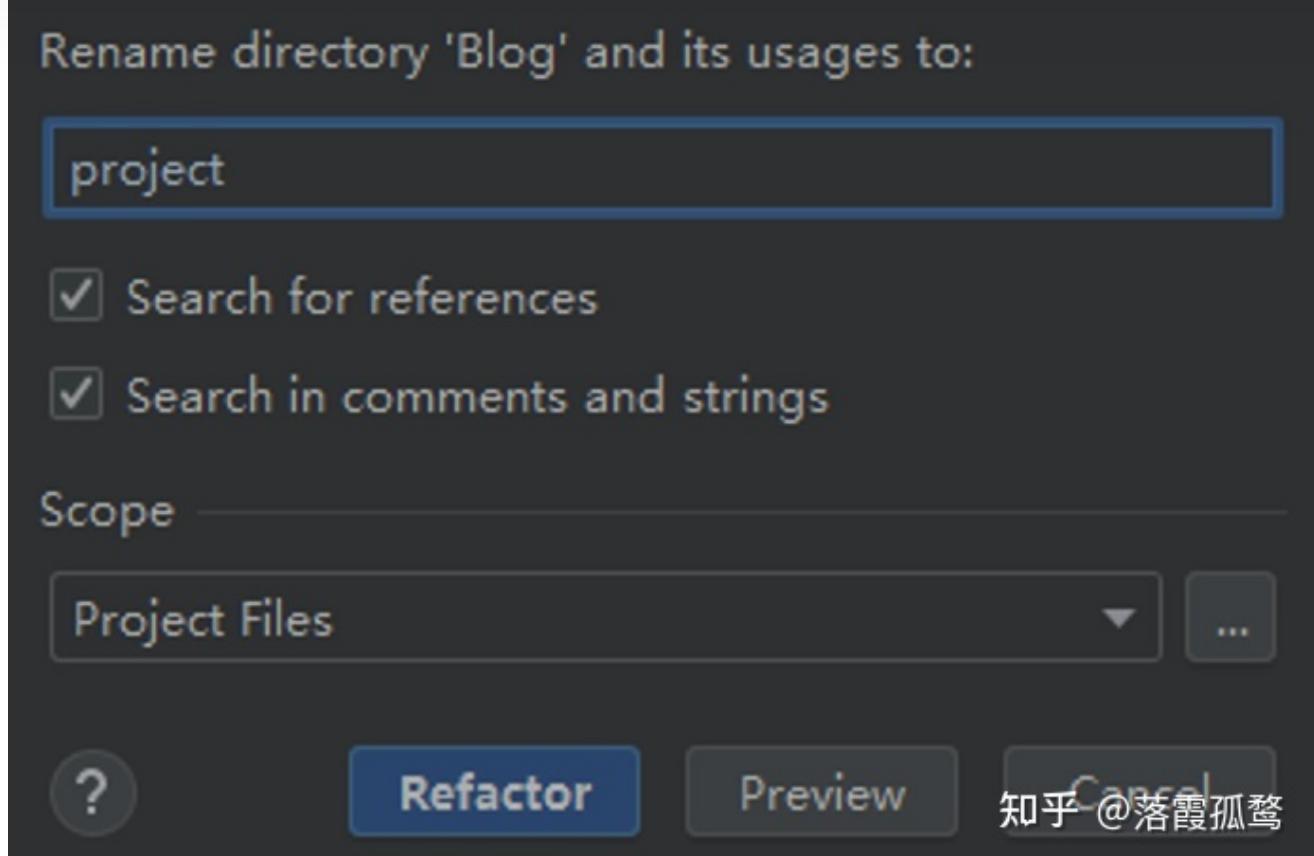
通过 PyCharm 自动生成的项目结构，会自动生成一个和项目名称一样的子文件夹，为了有效的组织后端的各个模块，这里我们将自动生成的 Blog 文件夹修改为 project

操作如图：

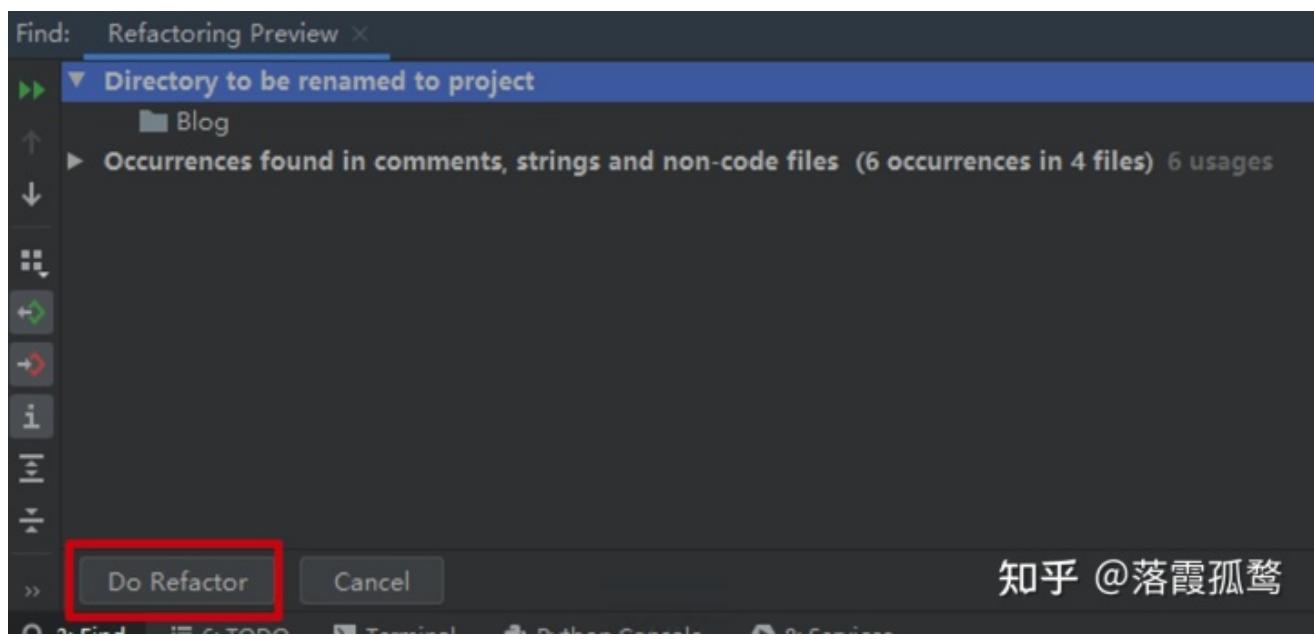
知乎

专栏
Python

知乎 @落霞孤鹜



点击 Refactor，然后点击左下角的 Do Refactor 完成修改。



1. 在 settings.py 文件中，将 ROOT_URLCONF 中的 Blog 修改为 project

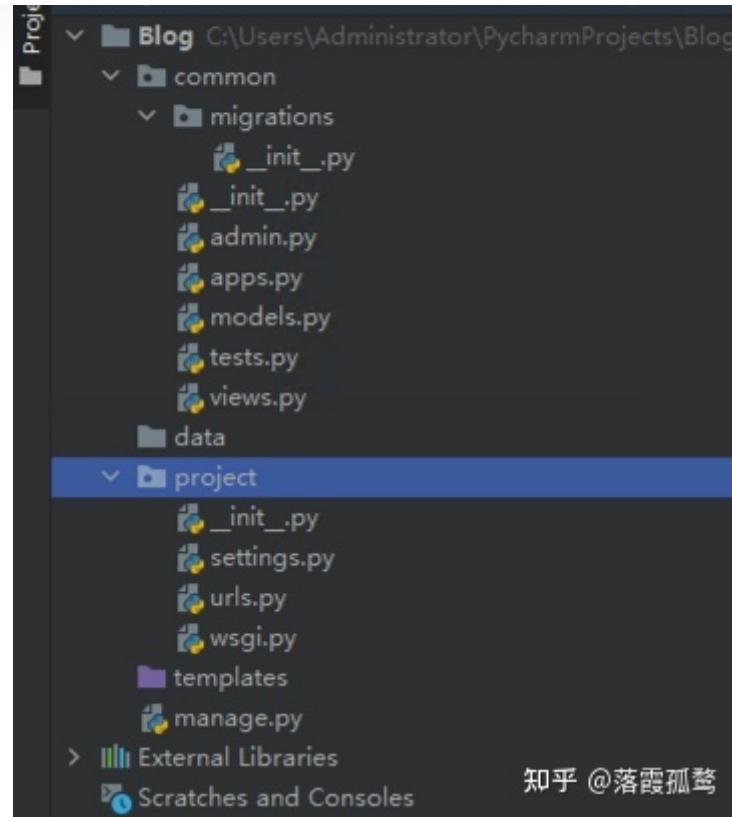
```
'/templates']
```

```
TEMPLATES = [
{
    'BACKEND': 'django.template.backends.django.DjangoTemplates',
    'DIRS': [BASE_DIR + '/templates'],
    'APP_DIRS': True,
    'OPTIONS': {
        'context_processors': [
            'django.template.context_processors.debug',
            'django.template.context_processors.request',
            'django.contrib.auth.context_processors.auth',
            'django.contrib.messages.context_processors.messages',
        ],
    },
},
]
```

1. 创建 common APP, 在 terminal 中输入如下命令:

```
python manage.py startapp common
```

1. 完成后, 整个项目结构如下图:



1.5 编写 User 对象的 API

1. 在 common/models.py 中编写基础模型抽象类 AbstractBaseModel

用来帮助构建所有业务模型自动增加创建人，创建时间，修改人，修改时间

```
from django.contrib.auth.models import AbstractUser
from django.db import models

class AbstractBaseModel(models.Model):
    creator = models.IntegerField('creator', null=True)
    created_at = models.DateTimeField(verbose_name='Created at', auto_now_add=True)

    modifier = models.IntegerField('modifier', null=True)
    modified_at = models.DateTimeField(verbose_name='Modified at', auto_now=True)

    class Meta:
        abstract = True
```

知乎

专栏
Python

- `auto_now_add=True` 表示在新增的时候，自动将该字段的值设置为当前时间
- `auto_now=True` 表示在记录更新的时候，自动设置为当前时间
- `abstract = True` 表示该类是抽象类，不需要生成物理模型
- 在 `common/models.py` 中重写 `User` 类

由于我们需要在博客中记录注册用户的昵称、头像等扩展信息，因此 Django 自带的 `User` 模型字段无法满足，所以通过集成 Django 提供的 `AbstractUser` 来扩展，通过 `Meta` 类中定义我们想要的表名 `blog_user`。

```
class User(AbstractUser, AbstractBaseModel):
    avatar = models.CharField('头像', max_length=1000, blank=True)
    nickname = models.CharField('昵称', null=True, blank=True, max_length=200)

    class Meta(AbstractUser.Meta):
        db_table = 'blog_user'
        swappable = 'AUTH_USER_MODEL'
```

如果不单独定义，则会用 Django 中设定的表名，这样不利于我们有效的识别和管理数据库中的表

1. 在 `common` 下新增 `serializers.py`，在 `serializers.py` 中新增类

`UserSerializer`，继承 Rest Framework 提供的 `serializers.ModelSerializer`

```
from rest_framework import serializers
from common.models import User

class UserSerializer(serializers.ModelSerializer):
    class Meta:
        model = User
        fields = ['id', 'username', 'avatar', 'email', 'is_active', 'created_at',
```

这里定义需要在 API 接口中出现的字段，包括新增、修改、查询接口字段。

1. 在 `common/views.py` 文件中，编写 `UserViewSet` 类，继承 Rest Framework 提供的 `viewsets.ModelViewSet`



```
from common.serializers import UserSerializer

class UserViewSet(viewsets.ModelViewSet):
    queryset = User.objects.all().order_by('username')
    serializer_class = UserSerializer
    permission_classes = [permissions.AllowAny]
```

这里只需要 Override 三个类属性，查询集合 queryset、序列器类 serializer_class，权限校验 permission_classes。

我们这里设置权限校验为 AllowAny，表示这个对象下的接口可以不用登录就可以访问，这么做的目的是为了下面测试接口的连通性。

1. 修改 project/settings.py 配置

在 INSTALLED_APPS 中增加 common

```
INSTALLED_APPS = [
    'django.contrib.admin',
    'django.contrib.auth',
    'django.contrib.contenttypes',
    'django.contrib.sessions',
    'django.contrib.messages',
    'django.contrib.staticfiles',
    'rest_framework',
    'common',
]
```

新增一行代码，定义 User 鉴权对应的模型，因为我们改写的默认的 User 表

```
# User
AUTH_USER_MODEL = 'common.User'
```

1. 定义 API 路由规则

在 common 下新增 urls.py 中，并增加如下代码，这里需要定义 app_name = 'common'，用于路由 Rest Framework 区别路由

```
from common import views

router = routers.DefaultRouter()
router.register('user', views.UserViewSet)

app_name = 'common'

urlpatterns = [
    path('', include(router.urls)),
]
```

在 project/url 中引入 common APP中的路由，并加入 Rest Framework 用户鉴权路由 api-auth/

```
from django.contrib import admin
from django.urls import path, include

urlpatterns = [
    path('admin/', admin.site.urls),
    path('', include('common.urls', namespace='common')),
    path('api-auth/', include('rest_framework.urls', namespace='rest_framework'))
]
```

1.6 模型迁移和配置

1. 通过 Django 做模型迁移

在 PyCharm 提供的 Terminal 中输入如下命令，完成模型创建

```
python manage.py makemigrations
# Migrations for 'common':
#   common\migrations\0001_initial.py
#       - Create model User
python manage.py migrate
# Operations to perform:
#   Apply all migrations: admin, auth, common, contenttypes, sessions
# Running migrations:
```

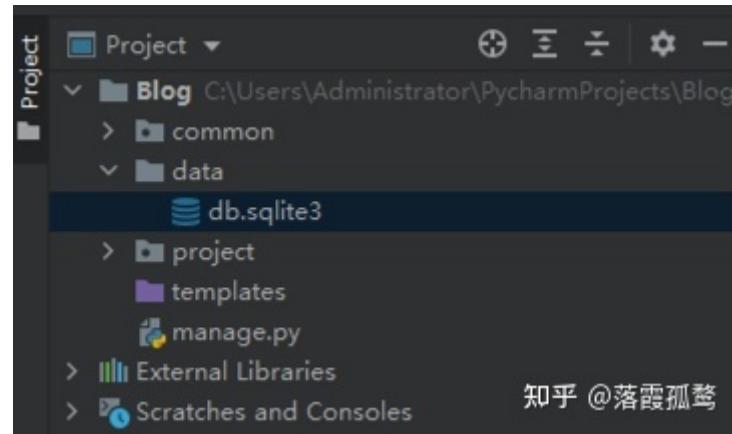
知乎

专栏
Python

```
# Applying auth.0003_alter_user_email_max_length... OK
# Applying auth.0004_alter_user_username_opts... OK
# Applying auth.0005_alter_user_last_login_null... OK
# Applying auth.0006_require_contenttypes_0002... OK
# Applying auth.0007_alter_validators_add_error_messages... OK
# Applying auth.0008_alter_user_username_max_length... OK
# Applying auth.0009_alter_user_last_name_max_length... OK
# Applying auth.0010_alter_group_name_max_length... OK
# Applying auth.0011_update_proxy_permissions... OK
# Applying common.0001_initial... OK
# Applying admin.0001_initial... OK
# Applying admin.0002_logentry_remove_auto_add... OK
# Applying admin.0003_logentry_add_action_flag_choices... OK
# Applying sessions.0001_initial... OK
```

1. 配置数据库管理工具

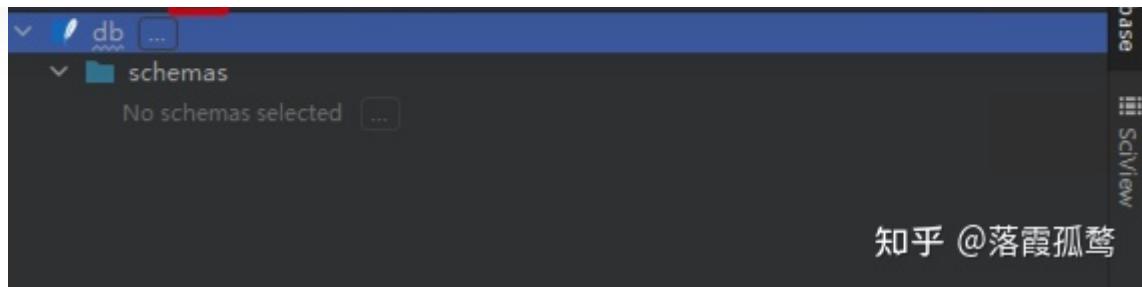
这个时候，可以在 `data` 文件下看到生成的 `Sqlite` 数据库文件 `db.sqlite3`，和我们在 `project/settings.py` 的 `DATABASE` 中定义的名称一致。



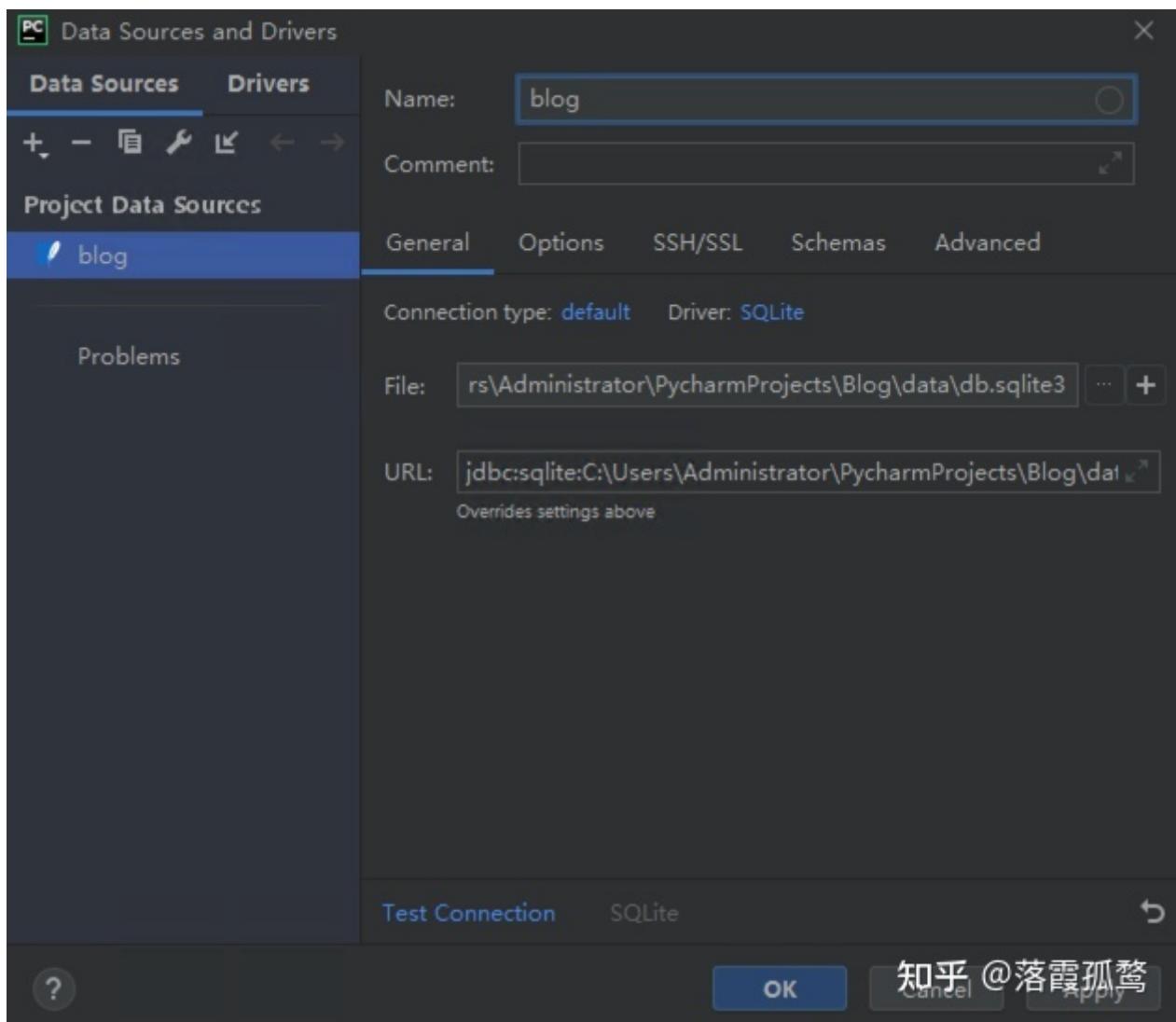
双击这个文件，PyCharm 会自动在右侧的 `Database` 工具类中创建一个 `Sqlite` 的数据库记录。点击下图中表红的按钮，进入数据库配置页面。

知乎

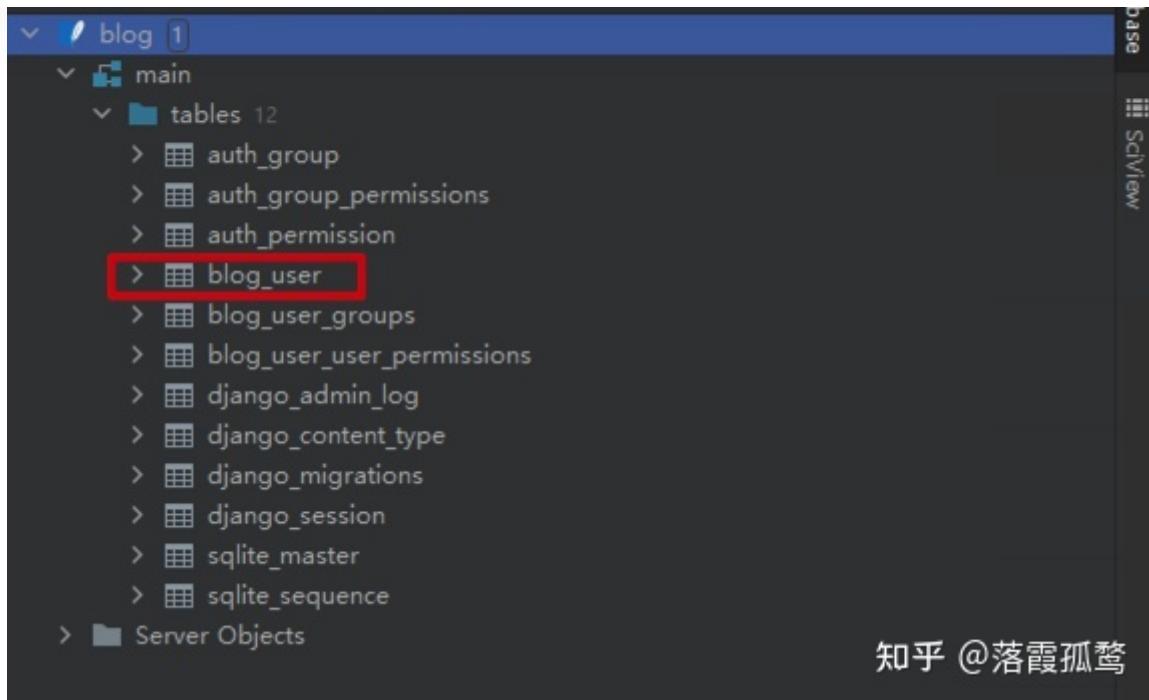
专栏
Python



点击下方的 Download missing driver files , 下载 Sqlite 驱动, 修改 name 为 blog , 在 Schemas 标签页中, 勾选 All schemas , 点击确定。



然后右侧就出现了我们刚刚通过 migrate 命令生成的表:



标红的表就是我们在 `common/models.py` 中定义的 `User` 类映射出来的表，其他的表是 Django 框架内置的表，主要用在管理权限，Session，日志等。

1.7 创建管理员账号

这里通过 Django 自带的命令完成管理员账号的创建，在 PyCharm 提供的 Terminal 中，输入如下命令：

```
python manage.py createsuperuser --email admin@example.com --username blog-admin
# Password:
# Password (again):
# Superuser created successfully.
```

依据提示输入密码 `12345678.Abc`，确认输入密码，回车，完成管理员账号的创建

1.8 测试API

1. 点击 PyCharm 右上角的运行按钮



▲ 赞同 1



● 添加评论

↗ 分享

★ 收藏

⚑ 举报

收起 ▲



看到如下界面，说明API配置已经成功

The screenshot shows a browser window titled "Api Root - Django REST framework" with the URL "127.0.0.1:8000". The page content is titled "Api Root" and describes it as "The default basic root view for DefaultRouter". It shows a single "GET /" entry with the following response details:
HTTP 200 OK
Allow: GET, HEAD, OPTIONS
Content-Type: application/json
Vary: Accept
{
 "user": "http://127.0.0.1:8000/user/"
}

知乎 @落霞孤鹜

1. 测试用户查询接口

点击上图中的 `http://127.0.0.1:8000/user/`

```
HTTP 200 OK
Allow: GET, HEAD, OPTIONS
Content-Type: application/json
Vary: Accept
```

```
{
    "user": "http://127.0.0.1:8000/user/"
}
```

得到如下图界面，说明接口已经完全编写成功，恭喜你，记得给你自己一个大拇指哦



User List

GET /user/

```
HTTP 200 OK
Allow: GET, POST, HEAD, OPTIONS
Content-Type: application/json
Vary: Accept

{
    "count": 1,
    "next": null,
    "previous": null,
    "results": [
        {
            "id": 1,
            "username": "slog-admin",
            "avatar": "",
            "email": "admin@example.com",
            "is_active": true,
            "created_at": "2022-07-17T12:09:12.398993Z",
            "nickname": null
        }
    ]
}
```

OPTIONS

GET

Raw data

HTML form

Username

Required. 150 characters or fewer. Letters, digits and @/./+/-/_ only.

头像

Email address

Active

 Designates whether this user should be treated as active. Unselect this instead of deleting accounts.

知乎 @落霞孤鹜

1.9 Django Rest Framework 官方英文示例教程

<https://www.djangoproject.com/en/2.2/intro/tutorial01/>

二、前端框架代码搭建

在介绍篇，我们已经安装了 Vite，这里我们就通过 Vite 来初始化 Vue 的项目

2.1 通过 Vite 初始化 Vue 项目

1. 在 C:\Users\Administrator 路径下，创建文件夹 VSCodeProjects

```
cd C:\Users\Administrator
mkdir VSCodeProjects
```

1. 在 VSCodeProjects 文件夹下创建 blog 项目

```
cd VSCodeProjects
yarn create vite blog --template vue-ts
```

▲ 赞同 1

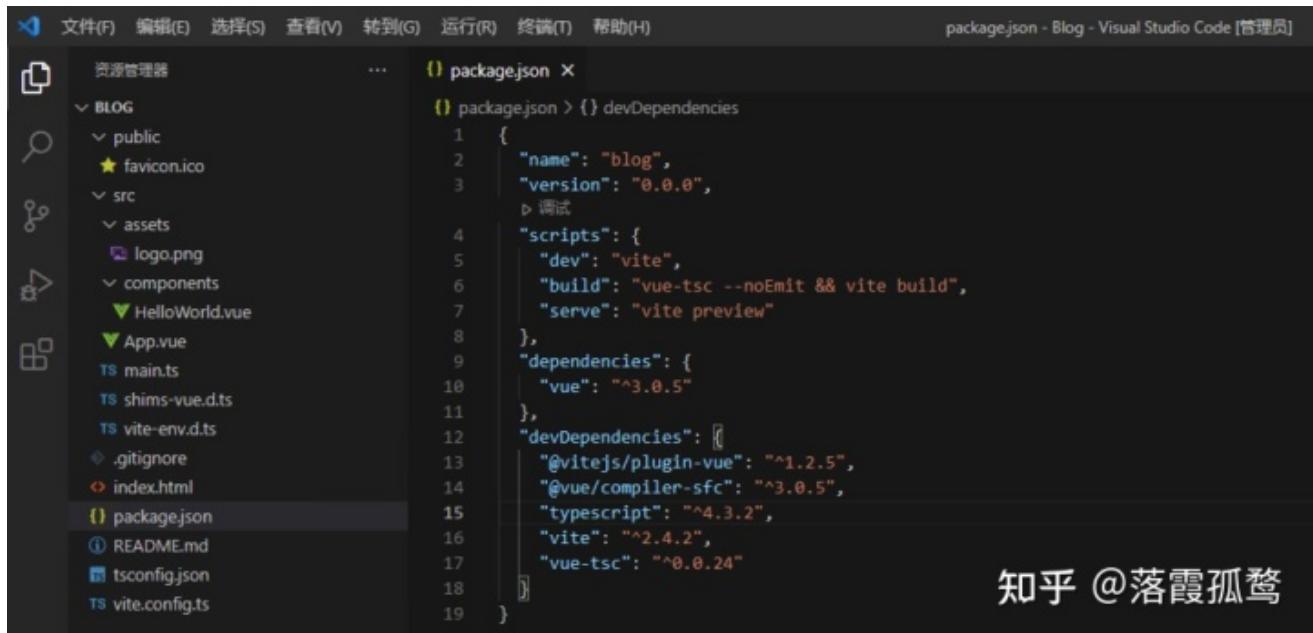
添加评论

分享

收藏

举报

收起 ^



```

{
  "name": "blog",
  "version": "0.0.0",
  "scripts": {
    "dev": "vite",
    "build": "vue-tsc --noEmit && vite build",
    "serve": "vite preview"
  },
  "dependencies": {
    "vue": "^3.0.5"
  },
  "devDependencies": [
    "@vitejs/plugin-vue": "^1.2.5",
    "@vue/compiler-sfc": "^3.0.5",
    "typescript": "^4.3.2",
    "vite": "^2.4.2",
    "vue-tsc": "^0.0.24"
  ]
}

```

知乎 @落霞孤鹜

2.2 依赖安装

1. 安装 Less 依赖

这里的 `-D` 参数表示是在开发阶段的依赖，上线运行时不需要该依赖。

```
yarn add less@4.1.1 -D
```

1. 安装 Element-Plus 依赖

```
yarn add element-plus
```

1. 安装基础依赖，并运行

在 VS Code 中，通过快捷键 `Ctrl + J` 打开 Terminal，输入如下命令：

```
yarn
yarn dev
# yarn run v1.22.10
# warning package.json: No license field
```

```
# Dev server running at:  
# > Network: http://192.168.2.14:3000/  
# > Local:   http://localhost:3000/
```

1. 在浏览器中输入地址 `http://localhost:3000/`，效果如下



Hello Vue 3 + TypeScript + Vite

Recommended IDE setup: [VSCode](#) + [Vetur](#) or [Volar](#) (if using `<script setup>`)

See [README.md](#) for more information.

[Vite Docs](#) | [Vue 3 Docs](#)

count is: 0

Edit `components/HelloWorld.vue` to test hot module replacement

三、前后端代码联调

前后端联调时，需要先在前端配置路由，Vite 代理，Axios 网络请求。

3.1 配置 Axios

1. 安装 Axios，在 VS Code 的 Terminal 中执行命令

在项目根目录下，修改 `vite.config.ts`，代码如下：

```
import vue from '@vitejs/plugin-vue'
import {defineConfig} from 'vite'

export default defineConfig({
  plugins: [
    vue(),
  ],
  base: '/',
  server: {
    host: "localhost",
    port: 3000,
    proxy: {
      '/api': {
        target: 'http://localhost:8000/',
        changeOrigin: true,
        ws: false,
        rewrite: (pathStr) => pathStr.replace('/api', ''),
        timeout: 5000,
      },
    },
  }
});
```

1. 配置 Axios 实例

在 `src` 目录下，新建文件夹 `api`，在 `api` 下新建文件 `index.ts`，编写如下代码：

```
import axios, {AxiosRequestConfig, AxiosResponse} from "axios";

const request = axios.create({
  baseURL: import.meta.env.MODE !== 'production' ? '/api' : '',
})
```



在 api 下新建文件 service.ts , 编写如下代码:

```
import request from "./index";

export function getUserList(params: any) {
    return request({
        url: '/user/',
        method: 'get',
        params,
    })
}
```

3.2 创建用户列表页面

修改在 src 下创建 App.vue , 在文件里面编写如下代码, 获取用户列表的接口调用, 调用后得到的数据加载到表格中展示, 同时通过分页展示列表。

```
<template>
<div>
<div>
<el-table
    :data="state.userList"
    :header-cell-style="{ background: '#eef1f6', color: '#606266' }"
    stripe
>
    <el-table-column type="selection" width="55" />
    <el-table-column label="ID" prop="id" width="80" />
    <el-table-column label="账号" prop="username" width="200" />
    <el-table-column label="昵称" prop="nickname" width="200" />
    <el-table-column label="状态" prop="is_active" />
</el-table>
</div>
<div class="pagination">
<el-pagination
    :page-size="10"
    :total="state.total"
    background
    layout="prev, pager, next"
></el-pagination>
```

▲ 赞同 1

▼

● 添加评论

↗ 分享

★ 收藏

⚑ 举报

收起 ^

知乎

专栏
Python

```
<script lang="ts">

import { defineComponent, reactive } from "vue";
import { getUserList } from "./api/service";

export default defineComponent({
  name: "App",
  setup: function () {
    const state = reactive({
      userList: [],
      params: {
        page: 1,
        page_size: 10,
      },
      total: 0,
    });

    const handleSearch = async (): Promise<void> => {
      try {
        const data: any = await getUserList(state.params);
        state.userList = data.data.results;
        state.total = data.data.count;
      } catch (e) {
        console.error(e);
      }
    };

    handleSearch();
    return {
      state,
      handleSearch,
    };
  },
});

</script>

<style scoped>
.pagination {
  text-align: right;
  margin-top: 12px;
}
</style>
```

▲ 赞同 1

▼

● 添加评论

↗ 分享

★ 收藏

⚑ 举报

收起 ^



1. 安装 vue-router，在 VS Code 的 Terminal 中执行命令

```
yarn add vue-router@next
```

1. 配置 route

在项目 src 目录下面新建 route 目录，并添加 index.ts 文件，文件中添加以下内容

```
import {createRouter, createWebHistory, RouteRecordRaw} from "vue-router";
import App from "../App.vue";

const routes: Array<RouteRecordRaw> = [
  {
    path: "/",
    name: "User",
    component: App,
    meta: {}
  },
]

const router = createRouter({
  history: createWebHistory(import.meta.env.BASE_URL),
  routes,
});

export default router;
```

3.4 调整 main.ts

增加 Element-Plus 的组件，加载 router，代码如下：

```
import { createApp } from 'vue'
import App from './App.vue'
import router from "./route";
import {
  ElPagination,
  ElTable,
```

▲ 赞同 1

▼

● 添加评论

↗ 分享

★ 收藏

⚑ 举报

收起 ^

```
app.component(ElTable.name, ElTable);
app.component(ElTableColumn.name, ElTableColumn);
app.component(ElPagination.name, ElPagination);

app.use(router).mount('#app')
```

3.5 测试界面

1. 在 PyCharm 中启动后端服务
2. 在 VS Code 中启动前端

```
yarn dev
```

1. 在浏览器中输入 `http://127.0.0.1:8080` 得到如下效果



ID	账号	昵称	状态
1	blog-admin		true

到这一步，我们已经完成了前后端联调，第一个接口已经通过界面方式完成调用和数据展示，给自己一个赞哦。

四、代码纳入 Git 版本管理

还记得我们在第一篇文章中创建的代码仓库吧，现在我们要把刚刚创建的前端代码和后台代码提交到代码仓库中。

这里以后端为例，我之前是在Gitee上创建了一个公开仓库：

https://gitee.com/Zhou_Jimmy/blog-backend.git，现在就把后端代码提交到这个仓库中。

命令在 PyCharm 坦仕的 Terminal 中输入

▲ 赞同 1 ▼ 添加评论 分享 收藏 举报

收起 ^



```
git config --global user.name "Zhou_Jimmy"  
git config --global user.email "331352343@qq.com"
```

| 在设置的时候记得修改成你自己的名字和邮箱哦

4.2 本地代码初始化 git

```
git init  
# Initialized empty Git repository in C:/Users/Administrator/PycharmProjects/BLog,  
git remote add origin https://gitee.com/Zhou_Jimmy/blog-backend.git
```

| 在设置的时候，记得修改成你自己的仓库地址

4.3 拉取远程仓库上的代码

```
git pull origin master
```

4.4 建立忽略文件

在项目根路径下增加或编辑 `.gitignore` 文件，忽略不用加入到版本管理的文件，文件中的内容可以参考 Gitee 或 GitHub 提供的模板，这里主要增加 PyCharm 配置信息的文件夹

`.idea`

如果是前端代码，则增加这么一行

`.vscode`

4.5 提交代码

4.5.1 通过 Git 命令提交



```
git push origin master
```

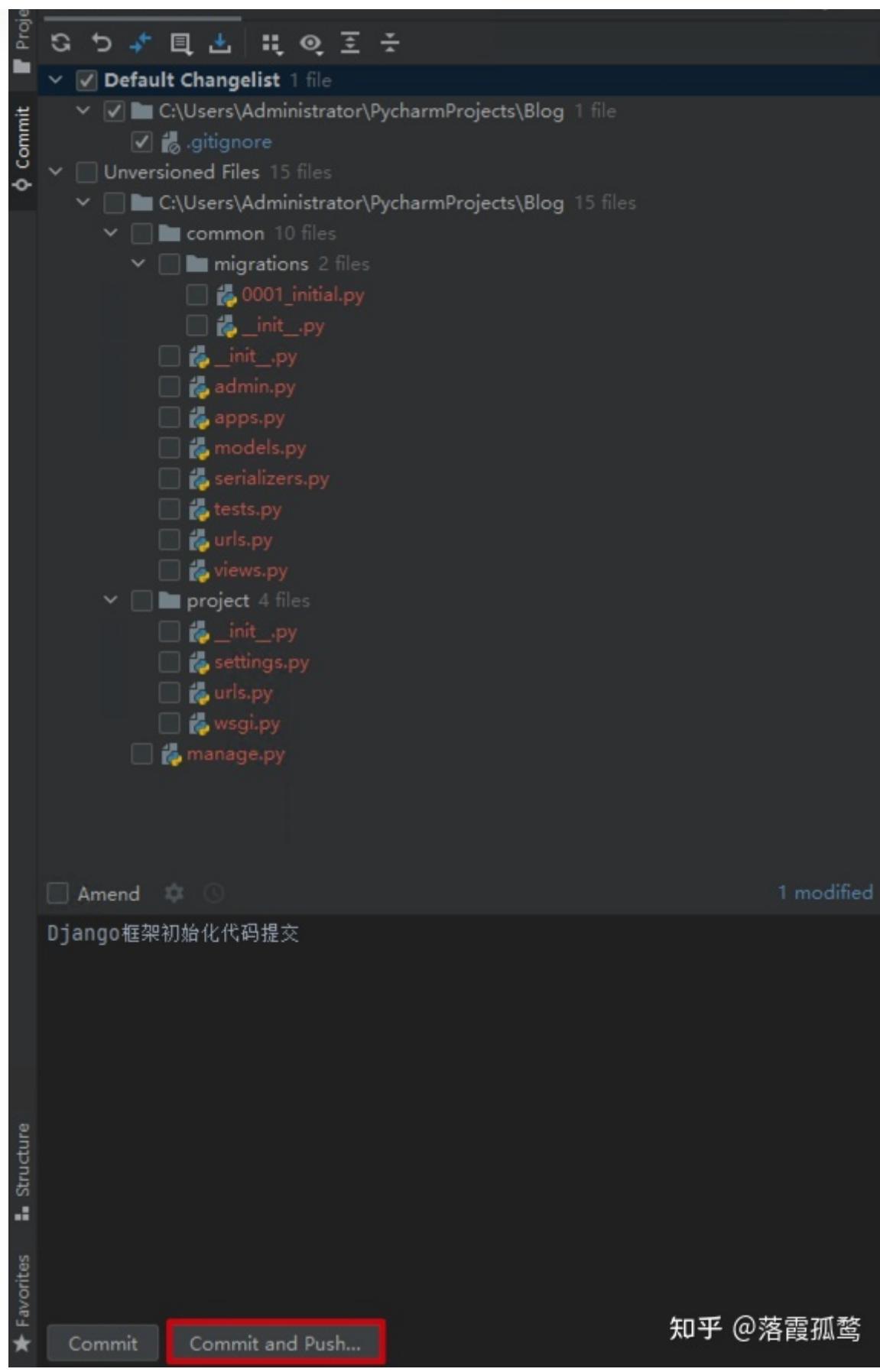
4.5.2 通过 PyCharm 界面功能提交

可以点击 PyCharm 右上角的提交按钮



点击后，左侧出现提交文件选择框，勾选文件，并填写 comment，点击 Commit and Push 按钮，完成提交。

知乎

专栏
Python

▲ 赞同 1

Commit

Commit and Push...

知乎 @落霞孤鹜

知乎

专栏
Python

Zhou_Jimmy Django框架初始化代码提交 07fd264 2 minutes ago

common Django框架初始化代码提交 2 minutes ago

project Django框架初始化代码提交 2 minutes ago

.gitignore Django框架初始化代码提交 2 minutes ago

LICENSE Initial commit 4 hours ago

manage.py Django框架初始化代码提交 2 minutes ago

知乎 @落霞孤鹜 2 minutes ago

前端提交完成后，远程仓库效果如下：

Zhou_Jimmy / Blog-Frontend

This repository are unavailable under any license yet, [Click here to select and create an open source license](#)

master Branches 1 Tags 0 + Pull Request + Issue File Web IDE Clone or download

Zhou_Jimmy Vue框架初始化代码提交 9cb0c3c a minute ago 2 commits

public Vue框架初始化代码提交 a minute ago

src Vue框架初始化代码提交 a minute ago

.gitignore Vue框架初始化代码提交 a minute ago

README.en.md Initial commit 6 minutes ago

README.md Initial commit 6 minutes ago

index.html Vue框架初始化代码提交 a minute ago

package-lock.json Vue框架初始化代码提交 a minute ago

package.json Vue框架初始化代码提交 a minute ago

tsconfig.json Vue框架初始化代码提交 a minute ago

vite.config.ts Vue框架初始化代码提交 a minute ago

yarn-error.log Vue框架初始化代码提交 a minute ago

yarn.lock Vue框架初始化代码提交 a minute ago

知乎 @落霞孤鹜 a minute ago

编辑于 2021-08-09 21:02

▲ 赞同 1



● 添加评论

↗ 分享

★ 收藏

⚑ 举报

收起 ^

Vue3+TypeScript+Django Rest Framework 搭建个人博客



落霞孤鹜，求知若渴

8 人赞同了该文章



Vue3+TypeScript+Django Rest Framework 搭建个人博客

本文适合对有 Python 语言有一定基础的人群，希望利用 Python 做更多有意思的事情，比如搭建个人博客，记录自己的所思所想，或者想找一个项目实践前后端分离技术等等。跟着本文可以了解和运行项目，本项目是在 Window 10 Professional 系统下开发。

大家好！我是 落霞孤鹜 ，从今天开始一步一步搭建一套基于 Python 语言的**个人博客**，这个项目也是我的个人博客搭建过程的记录，在实现过程中踏过了很多坑，这些坑也会在实现具体功能时列举出来，希望能帮助到可能遇到同样问题的你。

整个项目从开始实现到完成，我耗时2个星期，这两个星期是利用晚上下班时间完成的，每天花费1个小时左右，所以合起来大约是个15-20个小时左右，在这里说明时间是想让小伙伴们能对自己完成这个项目的时间有一个初步的评估。

[▲ 赞同 1](#)[● 添加评论](#)[分享](#)[收藏](#)[举报](#)[收起 ^](#)

知乎

专栏
Python

理），后来在简书上写过几篇博客，再后来觉得知乎和公众号也是一个很好的平台。但是今年觉得，虽然这些平台都挺好，终归不是自己的家，希望能给自己的文字找一个家。在这样的想法下催生了搭建一个自己的博客这么一个想法。

自己熟悉 Java 和 Python，两种语言都写过完整的项目，但是还没有用 Python 结合**前后端分离**技术实现一套 Web 系统。所以最终决定用 Python 来开发。

在实现博客的过程中，我借鉴了很多 夜尽天明的博文，基于他的思想和方式，完成了前端代码的构建。

已经实现的博客地址：[落霞孤鹜的个人博客](#)

已经写好的代码仓库：

1. 后端：

Zhou_Jimmy/Blog-Backend
gitee.com/Zhou_Jimmy/blog-backend



1. 前端：

Zhou_Jimmy/Blog-Frontend
gitee.com/Zhou_Jimmy/blog-frontend



效果图：

▲ 赞同 1

▼

● 添加评论

↗ 分享

★ 收藏

⚑ 举报

收起 ^

知乎

专栏
Python

关于我

基于前后端分离技术搭建个人博客

查看 33 评论 8 赞 2 Vue 2021-06-03 13:36:17

SpringBoot搭建外卖网站后台

Gitee自动生成的README.md

查看 19 评论 1 赞 0 Vue 2021-06-03 13:36:17



落霞孤鹜

标签

Vue python

-----我也是有底线的哟-----

技术型产品经理的叨叨



微信搜一搜

Q 微信搜索：微谈小智

打开“微信 / 发现 / 搜一搜”搜索

知乎 @落霞孤鹜

微谈小智 ©2021 Created by 落霞孤鹜

效果图

二、技术栈

博客基于的技术如下：

- **Django 框架**：快速搭建 Web 应用程序的 Python 开源框架，该框架功能齐全，是 Python 世界的 Spring, Lavarel, Ruby On Rail。
- **Django Rest Framework**：用于将Django全栈框架的视图层转换成基于Restful的接口，适配前后端分离架构。
- **Vue 3**：是一套用于构建用户界面的渐进式框架，在前后端分离架构下负责前端页面的开发。
- **Element-Plus**：一套基于 Vue 3.0 的桌面端组件库，可以帮助开发者快速搭建Vue 3的页面。
- **TypeScript**：TypeScript 是微软开发的一个开源的编程语言，通过在 JavaScript 的基础上添加静态类型定义构建而成，帮助 JavaScript 能够胜任开发大型项目，[TypeScript 教程](#)
- **SQLite 数据库**：是一种嵌入式数据库，它的数据库就是一个文件。

顺便提一下开发工具。

▲ 赞同 1



● 添加评论



分享



收藏



举报

收起 ^



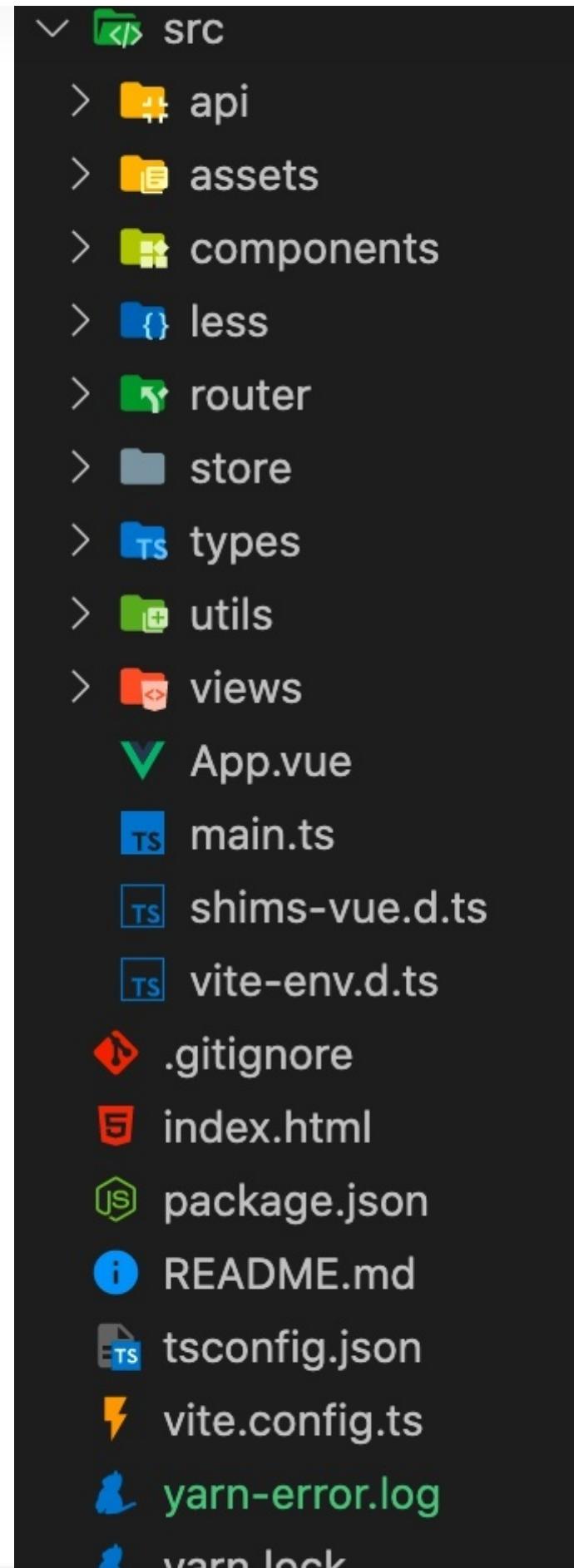
三、项目结构

项目使用前后端分离框架，所以代码项目结构按照前端和后端两部分呈现

3.1 前端项目结构

知乎

专栏
Python



▲ 赞同 1



● 添加评论

↗ 分享

★ 收藏

☞ 举报

收起 ^

专栏
Python

3.1.1 代码目录说明

- `public`：项目的公共文件，一般存放 `fav.icon` 和 `logo.icon`
- `assets`：图片目录，在页面中用到的一些图片文件
- `components`：包含页面中可以复用的组件
- `less`：Less 文件，负责项目的全局样式设置
- `router`：前端页面路由信息
- `store`：Vuex 的集中式存储管理应用的所有组件的状态。
- `types`：TypeScript 需要的类型定义说明
- `utils`：包含各类工具方法
- `views`：所有页面文件，以 `.vue` 结尾
- `App.vue`：项目主页面
- `main.ts`：TypeScript 的项目入口文件
- `shims-vue.d.ts`：TypeScript 对 Vue 的垫片文件，用于配置 TypeScript 支持识别 `.vue` 文件
- `vite-env.d.ts`：Vite 的环境配置文件
- `.gitignore`：Git 版本管理中的排除文件记录，在这个文件中出现的文件，不纳入版本管理，比如 `node_modules`
- `index.html`：项目如何页面，Vue 的挂载入口
- `package.json`：项目依赖和配置文件
- `README.md`：项目说明文件，基于 MarkDown 语法
- `tsconfig.json`：TypeScript 的配置文件
- `vite.config.ts`：Vite 的配置文件
- `yarn-error.log`：Yarn 安装依赖时的错误记录文件
- `yarn.lock`：Yarn 安装依赖后的版本号锁定文件

3.2 后端项目结构

▲ 赞同 1



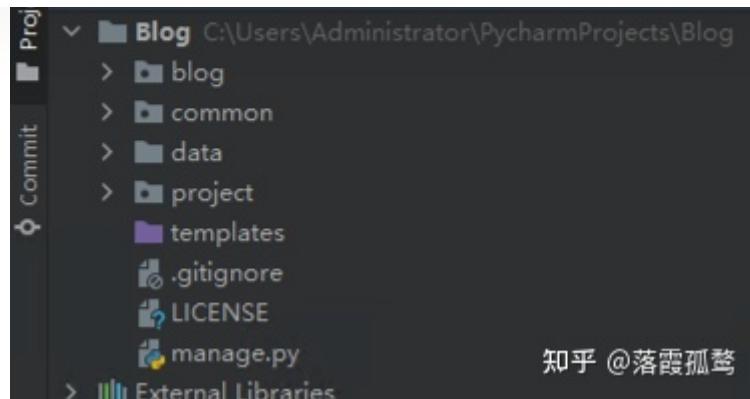
● 添加评论

↗ 分享

★ 收藏

☞ 举报

收起 ^



3.2.1 代码目录说明

- `project` : Django的配置代码和控制代码，比如 `settings.py` , `urls.py` , 同时放置了一些自定义的Django中间件和公共方法。
- `blog` : 博客核心逻辑所在目录，负责博客的CRUD。
- `common` : 处理用户登录，注销，修改密码，图片上传等基础公共功能，同时包含一些工具类和代码常量。
- `data` : 存放 `Sqlite` 数据库文件。
- `logs` : 存放项目的日志，按照天进行风分割。
- `upload` : 用于存放上传的文件，按照日期方式的文件夹进行组织。
- `manage.py` : Django的管理和控制代码，通过执行 `python manage.py XXX` 方式完成对整个项目的管理
- `requirements.txt` : 记录 Python 项目的所有依赖包和版本号

博客项目的结构简单清晰，想把它运行起来也超级简单。你是不是开始手痒痒了，那接下来我们一起让它运行起来吧。

四、开发环境搭建

环境准备分类前端开发环境准备、后端环境准备、代码仓库创建。开发环境基于Windows 10 搭建。

需要的开发环境：

- **Python 3.9.6**
- **Node.js 14.17.3**

▲ 赞同 1

▼

● 添加评论

↗ 分享

★ 收藏

☞ 举报

收起 ^

4.1 前端开发环境搭建

1. 下载和安装 Node.js 下载链接如下：

<https://nodejs.org/dist/v14.17.3/node-v14.17.3-x86.msi>

1. 安装 Node 后，会自动安装 NPM，在命令行中输入如下命令，验证是否安装成功

```
node -v  
# v14.17.3  
npm -v  
# 6.14.13
```

1. NPM 全换淘宝镜像源，加速依赖安装速度

```
npm config set registry https://registry.npm.taobao.org  
npm config get registry  
#https://registry.npm.taobao.org/
```

1. 全局安装 Yarn

```
npm install -g yarn  
yarn version  
# yarn version v1.22.10
```

1. Yarn 切换淘宝镜像源，加速依赖安装速度

```
yarn config set registry https://registry.npm.taobao.org/  
# yarn config v1.22.10  
# success Set "registry" to "https://registry.npm.taobao.org/".  
# Done in 0.13s.  
yarn config get registry  
# https://registry.npm.taobao.org/
```

1. 安装Vite

▲ 赞同 1 ▼

● 添加评论

↗ 分享

★ 收藏

☞ 举报

收起 ^

```
# [1/4] Resolving packages...
# warning create-vite-app@1.21.0: create-vite-app has been deprecated. run `npm i
# [2/4] Fetching packages...
# [3/4] Linking dependencies...
# [4/4] Building fresh packages...

# success Installed "create-vite-app@1.21.0" with binaries:
#   - create-vite-app
#   - cva
# Done in 1.47s.
```

1. 下载和安装 VS Code , 下载链接如下

<https://code.visualstudio.com/Download>

1. 安装 VS Code 所需要的插件

直接在 VS Code 的插件市场中搜索并安装

- Vetur
- ESLint

4.2 后端开发环境搭建

1. 下载 Python 3.9

<https://www.python.org/ftp/python/3.9.6/python-3.9.6-amd64.exe>

在下载完成后安装的时候，记得勾选将Python加入的Path中



Install Python 3.9.6 (64-BIT)

Select Install Now to install Python with default settings, or choose Customize to enable or disable features.

Install Now

C:\Users\ai\AppData\Local\Programs\Python\Python39

Includes IDLE, pip and documentation

Creates shortcuts and file associations

Customize installation

Choose location and features

Install launcher for all users (recommended)

Add Python 3.9 to PATH

知乎 @洛雷孙酱

1. 验证是否安装成功

按 `win+r`，输入 `cmd`，回车，输入 `python` 验证是否安装成功，如果成功，会显示如图所示，可以正确显示 Python 版本信息

```
python -V
# Python 3.9.6
pip -V
# pip 21.1.3 from c:\users\administrator\appdata\local\programs\python\python39\l...
```

1. pip 更换国内源

```
pip config set global.index-url https://pypi.douban.com/simple
#Writing to C:\Users\Administrator\AppData\Roaming\pip\pip.ini
pip config list
#global.index-url='https://pypi.douban.com/simple'
```

1. 下载和安装 Pycharm，安装教程如下链接。

<https://www.runoob.com/w3cnote/pycharm-windows-install.html>

▲ 赞同 1

▼

● 添加评论

↗ 分享

★ 收藏

➥ 举报

收起 ^



1. 下载和安装Git，版本管理软件，下载链接如下。

<https://git-scm.com/download/win>

1. 安装完成后，在cmd中输入 git 命令，如果出现如下显示，在说明安装成功

```
C:\Users\ai> git version  
# git version 2.32.0.windows.2
```

如果显示如下错误，则需要设置系统环境变量

```
C:\Users\ai> git version  
# 'git' 不是内部或外部命令，也不是可运行的程序  
# 或批处理文件。
```

1. 创建代码仓库。

在 GitHub 或者 Gitee 上创建仓库，由于 GitHub 的网速问题，我选择在 Gitee 上创建仓库，这里需要两个仓库，一个是前端代码仓库 Blog-Frontend，一个是后端代码仓库 Blog-Backend。

创建仓库的教程链接：

- [创建你的第一个仓库 - Gitee](#)
- [关于Github私有仓库的创建 - 知乎 \(zhihu.com\)](#)

4.4 设置系统环境变量

在搭建开发环境的时候，需要频繁的设置环境变量，这里是Win10设置环境变量的教程。

<https://jingyan.baidu.com/article/00a07f3876cd0582d128dc55.html>

五、博客功能介绍

该功能区展示了文章的互动信息，包括点赞数、评论数、分享数、收藏数和举报数。

▲ 赞同 1

▼

● 添加评论

↗ 分享

★ 收藏

⚑ 举报

收起 ^

3. 看博客 (博客网站)

5.1 用户管理

作为用户管理功能，我们需要处理管理员和不同用户两种不同角色，以此来区分用户登录后进入到哪一个页面。

作为个人博客，管理员就是我们自己，因此只需要一个账号，而且这个账号属于系统内置。

普通用户就是进入到博客网站浏览我们博客的人，可以是注册的用户，也可以是游客。

注册用户可以评论博客，登录后可以改密码，也可以退出登录。

5.2 写博客 (管理后台)

管理后台主要是管理博客的内容，比如发表博客，下架博客，修改博客，分类，查看评论，查看用户等。

5.3 看博客

游客可以查看所有发布的博客，可以通过注册功能注册成博客用户，然后发布博客的评论。

至此，所有准备条件已经全部具备，接下来的章节，我们开始通过代码一步一步构建博客。

编辑于 2021-11-06 13:15

▲ 赞同 8

▼

● 4 条评论

↗ 分享

★ 收藏

⚑ 举报

收起 ^

▲ 赞同 1

▼

● 添加评论

↗ 分享

★ 收藏

⚑ 举报

收起 ^