[toc]

API：整合修订（1.1）

- 统一了命名规范，并划分了对应板块

1.账户板块（登录，登出，注册，注销）

（0）用户的类

```python

# 继承AbstractUser,自带id,username和password

class User(AbstractUser): email_code = models.IntegerField(null=True, blank=True) #邮箱验证码 reputation = models.IntegerField(default=100) #信誉值 all_likes = models.IntegerField(default=0) #收获的总点赞数 all_views = models.IntegerField(default=0) #收获的总浏览量 influence = models.IntegerField(default=0) #影响力因子 master = models.BooleanField(default=False) #是否是管理员 block = models.BooleanField(default=False) #是否被封禁 block_end_time = models.DateTimeField(null=True, blank=True) #封禁结束时间 blocklist = models.ManyToManyField('self', symmetrical=False, related_name='blocked_by', through='BlockList') #黑名单 profile_url = models.CharField(max_length=255, null=True, blank=True) #头像的地址 def **str**(self): return self.username ```

（1）注册

**url : /index/register**

**<1> POST /register**

**描述：** 此接口用于注册，包括用户名、密码、电子邮件和邮箱验证码的提交。

**请求参数：**

| 参数名 | 类型 | 必填 | 描述 |
| --- | --- | --- | --- |
| `user_name` | string | 是 | 用户名，需唯一。 |
| `pass_word` | string | 是 | 用户密码。 |
| `email` | string | 是 | 用户邮箱，需唯一。 |
| `email_code` | string | 是 | 邮箱验证码，通过 `/register?send_code=1&email=email` 获取。 |

**响应参数（注：之后的响应参数，如无特殊说明均使用status+message的形式）：**

| 参数名 | 类型 | 描述 |
| --- | --- | --- |
| `status` | int | 状态码。 |
| `message` | string | 返回信息。 |
```

- **200 OK**: 注册成功；
- **400 Bad Request**: 注册请求的参数错误或缺失；
- **409 Conflict**: 注册冲突，例如用户名或邮箱已存在；
- **429 Too Many Requests**: 注册请求过多，例如短时间内多次尝试注册；
- **430 Invalid Email**: 邮箱格式有误，非山大邮箱；
- **431 Block Email：**邮箱注销冷却中；
- **500 Internal Server Error**: 服务器内部错误；

<2> GET /register?send_code=1&email=email

描述：
此接口用于向指定的邮箱发送验证码，用于验证用户邮箱。

请求参数：

参数名	类型	必填	描述
`email`	string	是	目标邮箱地址。

响应参数：

- **200 OK**: 验证码获取成功；
- **404 Not Found**: 验证码未找到或已过期。
- **429 Too Many Requests**: 验证码请求过多，例如短时间内多次请求验证码；
- **500 Internal Server Error**: 服务器内部错误；

（2）用户名和密码登录

url：/index/login_passwd

POST /login_passwd

描述：用户登录接口。此接口用于登录，用于提交用户的用户名和密码。

请求参数：

参数名	类型	必填	描述
`user_name`	string	是	用户名，需唯一。
`pass_word`	string	是	用户密码。

响应参数：

- **200 OK**: 登录成功。
- **401 Unauthorized**: 用户名或密码错误。

- **423 Locked**: 账号被锁定，例如因多次登录失败。
- **429 Too Many Requests**: 登录请求过多，例如短时间内多次尝试登录。

（3）邮箱和验证码登录

url：/index/login_email

<1> POST /login_email

描述： 此接口用于登录，提交用户的邮箱和密码。

请求参数：

参数名	类型	必填	描述
`email`	string	是	用户邮箱，需唯一。
`email_code`	string	是	邮箱验证码，通过 `/login_email?send_code=1&email=email` 获取。

响应参数：

- **200 OK**: 登录成功。
- **401 Unauthorized**: 邮箱或验证码错误。
- **423 Locked**: 账号被锁定，例如因多次登录失败。
- **429 Too Many Requests**: 登录请求过多，例如短时间内多次尝试登录。

<2> GET /login_email?send_code=1&email=email

描述： 此接口用于向指定的邮箱发送验证码，用于验证用户邮箱。

请求参数：

参数名	类型	必填	描述
`email`	string	是	目标邮箱地址。

响应参数：

- **200 OK**: 验证码获取成功；
- **404 Not Found**: 验证码未找到或已过期。
- **429 Too Many Requests**: 验证码请求过多，例如短时间内多次请求验证码；
- **500 Internal Server Error**: 服务器内部错误；

（4）登出

url：/index/logout

POST /logout

描述： 此接口用于退出登录。退出登录后自动关闭会话，无需请求参数。

响应参数：

- **200 OK**: 登出成功；
- **500 Internal Server Error**: 服务器内部错误；

（5）注销账户

url：/index/delete_account

<1> POST /delete_account

描述： 此接口用于注销账户，提交用户的用户名和邮箱。

请求参数：

参数名	类型	必填	描述
`user_name`	string	是	用户名，需唯一。
`email`	string	是	目标邮箱地址。
`email_code`	string	是	邮箱验证码，通过`/delete_account?send_code=1&email=email`获取

响应参数：

- **200 OK**: 注销成功。
- **404 Not Found**: 邮箱或验证码错误。
- **401 Unauthorized**: 用户不存在。

<2> GET /delete_account?send_code=1&email=email

描述： 此接口用于向指定的邮箱发送验证码，用于验证用户邮箱。

请求参数：

参数名	类型	必填	描述
`email`	string	是	目标邮箱地址。

响应参数：

- **200 OK**: 验证码获取成功；
- **404 Not Found**: 验证码未找到或已过期。
- **429 Too Many Requests**: 验证码请求过多，例如短时间内多次请求验证码；
- **500 Internal Server Error**: 服务器内部错误；

（5）重置密码

url：/index/reset_password

<1>POST /reset_password

描述： 此接口用于重置密码，提交用户的用户名和邮箱。

请求参数：

参数名	类型	必填	描述
`email`	string	是	目标邮箱地址。
`pass_word`	string	是	原密码。
`new_pass_word`	string	是	新密码。
`email_code`	string	是	重置密码的验证码，通过 `/reset_password?send_code=1&email=email` 获取。

响应参数：

- **200 OK**: 重置密码成功。
- **401 Unauthorized**: 邮箱或验证码错误。
- **423 Locked**: 短时间内频繁修改，已锁定。

<2>GET /reset_password?send_code=1&email=email

描述： 此接口用于向指定的邮箱发送验证码，用于验证用户邮箱。

请求参数：

参数名	类型	必填	描述
`email`	string	是	目标邮箱地址。

响应参数：

- **200 OK**: 验证码获取成功；
- **404 Not Found**: 验证码未找到或已过期。
- **429 Too Many Requests**: 验证码请求过多，例如短时间内多次请求验证码；
- **500 Internal Server Error**: 服务器内部错误；

2.黑名单板块

（0）黑名单类

```python class BlockList(models.Model): from_user = models.ForeignKey(User, related_name='blocking', on_delete=models.CASCADE) to_user = models.ForeignKey(User, related_name='blocked', on_delete=models.CASCADE) created_at = models.DateTimeField(auto_now_add=True)
```

```
class Meta:
    unique_together = ('from_user', 'to_user')
```

```

### （1）拉黑用户

**url：/index/block**

**POST /block**

**描述：** 此接口用于拉黑其他用户。

**请求参数：**

| 参数名 | 类型 | 必填 | 描述 |
| --- | --- | --- | --- |
| `to_user_id` | int | 是 | 被拉黑者的id。 |

**响应参数：**

- **200 OK**: 拉黑用户成功。
- **409 Conflict**: 已经拉黑过该用户。
- **404 Not Found**: 用户未找到或已注销。
- **500 Internal Server Error**: 服务器内部错误；

### （2）解除拉黑

**url：/index/unblock**

**POST /unblock**

**描述：** 此接口用于解除对用户的拉黑。

**请求参数：**

| 参数名 | 类型 | 必填 | 描述 |
| --- | --- | --- | --- |
| `to_user_id` | int | 是 | 被拉黑者的id。 |

**响应参数：**

- **200 OK**: 解除拉黑成功。
- **401 Unauthorized**：尚未拉黑该用户，无需解除。
- **404 Not Found**: 用户未找到或已注销。
- **500 Internal Server Error**: 服务器内部错误；

## 3.Article板块
```

（0）Article的类

```python class Article(models.Model): id = models.AutoField(primary_key=True) article_title = models.CharField(max_length=255) author = models.ForeignKey(User, on_delete=models.CASCADE, related_name='articles') content = models.TextField() tags = models.CharField(max_length=255) stars = models.IntegerField(default=0) likes = GenericRelation(Like) views = models.IntegerField(default=0) block = models.BooleanField(default=False) publish_time = models.DateTimeField(auto_now_add=True)

```
def __str__(self):
    return self.article_title
```

```

（1）创建article

url：/index/article/create

POST /article/create

请求参数：

参数名	类型	必填	描述
`article_title`	string	是	文章标题。
`content`	string	是	文章内容。
`tags`	string	否	文章标签。以逗号分隔。
`author_id`	int	是	作者ID。
`article_type`	string	否	文章类型：原创或转载。
`origin_link`	string	否	转载时的原文链接。

响应参数：

- **200 OK**: 创建文章成功。
- **500 Internal Server Error**: 服务器内部错误；

（2）编辑article

url：/index/article/edit

POST /article/edit

请求参数：

参数名	类型	必填	描述

参数名	类型	必填	描述
`id`	int	是	文章的id。
`article_title`	string	否	文章标题。
`content`	string	否	文章内容。
`tags`	string	否	文章标签。以#号分隔。
`article_type`	string	否	文章类型：原创或转载。
`origin_link`	string	否	转载时的原文链接。

响应参数：

- **200 OK**: 编辑文章成功。
- **500 Internal Server Error**: 服务器内部错误；

（3）删除article

url：/index/article/delete

POST /article/delete

描述： 此接口用于删除article。

请求参数：

参数名	类型	必填	描述
`id`	int	是	文章的id。

响应参数：

- **200 OK**: 删除文章成功。
- **500 Internal Server Error**: 服务器内部错误；

（4）通过id获取Article的详细信息

url：/index/article/detail

GET /article/detail?id=id

请求参数：

参数名	类型	必填	描述
`id`	int	是	文章的id。

响应参数：

参数名	类型	描述
`status`	int	状态码。
`message`	string	返回信息。
`article_detail`	list	文章详情。

- **200 OK**: 获取文章详情成功。
- **500 Internal Server Error**: 服务器内部错误；

若获取成功，则`article_detail`的内容如下：

参数名	类型	描述
`article_id`	int	文章id。
`article_title`	string	文章题目。
`article_content`	string	文章内容。
`article_type`	string	转载/原创。
`origin_link`	string	若为转载，则标注原帖。
`article_tags`	array	文章的标签。
`author_name`	string	作者的名字。
`author_profile_url`	string	作者头像的url。
`star_count`	int	总收藏数。
`view_count`	int	总浏览量。
`reply_count`	int	总回复量（包括文章评论及回复）。
`source_url`	string	资源链接。
`publish_time`	time	文章发布时间。

---

*（5）根据id分页获取Article下的Post列表：试行版

url：/index/article/post_list

GET /article/post_list?id=id&page_index=page_index&page_size=page_size

描述： 此接口用于获取指定文章（通过文章ID）下，按分页要求返回的Post列表。

请求参数：

参数名	类型	必填	描述
`id`	int	是	文章ID。
`page_index`	int	否	第几页，默认为第一页。

参数名	类型	必填	描述
`page_size`	int	否	每页显示的Post数量，默认为20。

响应参数：

参数名	类型	描述
`status`	int	状态码。
`message`	string	返回信息。
`post_list`	array	Post列表。

- **200 OK**: 获取Post列表成功。
- **500 Internal Server Error**: 服务器内部错误。

若获取成功，则`post_list`中的每个Post对象的内容如下：

参数名	类型	描述
`post_id`	int	帖子ID。
`post_title`	string	帖子标题。
`post_content`	string	帖子内容。
`poster_name`	string	发帖人名称。
`poster_profile_url`	string	发帖人头像URL。
`view_count`	int	帖子的浏览量。
`like_count`	int	帖子的点赞数。
`reply_count`	int	帖子的回复数。
`tags`	array	帖子的标签。
`publish_time`	time	帖子的发布时间。

（6）分页获取article列表

url：/index/article/list

GET /article/list?page_index=page_index&page_size=page_size

描述： 此接口用于提交分页要求，并获取article列表的对应页。

请求参数：

参数名	类型	必填	描述
`page_index`	int	否	第几页，默认为第一页。

参数名	类型	必填	描述
`page_size`	int	否	每页显示的文章数量，默认为20。

响应参数：

参数名	类型	描述
`status`	int	状态码。
`message`	string	返回信息。
`article_list`	array	文章列表。

- **200 OK**: 获取文章列表成功。
- **500 Internal Server Error**: 服务器内部错误。

若获取成功，则`article_list`中的每个文章对象的内容如下：

参数名	类型	描述
`article_id`	int	文章ID。
`article_title`	string	文章标题。
`author_name`	string	作者名称。
`author_profile_url`	string	作者头像的URL。
`star_count`	int	文章的收藏数。
`view_count`	int	文章的浏览量。
`like_count`	int	文章的点赞数。
`tags`	array	文章标签。
`publish_time`	time	文章发布时间。

4.Post与Reply板块

（0）Post与Reply类

```python class Post(models.Model): id = models.AutoField(primary_key=True) post_title = models.CharField(max_length=255) poster = models.ForeignKey(User, on_delete=models.CASCADE, related_name='posts') content = models.TextField() tags = models.CharField(max_length=255) views = models.IntegerField(default=0) likes = GenericRelation(Like) block = models.BooleanField(default=False) top = models.BooleanField(default=False) publish_time = models.DateTimeField(auto_now_add=True) article = models.ForeignKey(Article, on_delete=models.CASCADE, related_name='posts', null=True, blank=True) course = models.ForeignKey(Course, on_delete=models.CASCADE, related_name='posts', null=True, blank=True)
```

```
    def send_notification(self, mentioned_user):
        Notification.objects.create(
            user=mentioned_user,
            message=f"Your article have a new post, title:
    {self.post_title[:50]}, content:{self.content[:50]}"  # 通知消息可以包含帖子内
    容的前50个字符
        )

    def __str__(self):
        return self.post_title
```

class Reply(models.Model): id = models.AutoField(primary_key=True) reply_content = models.TextField() reply_time = models.DateTimeField(auto_now_add=True) post = models.ForeignKey(Post, on_delete=models.CASCADE, related_name='replies') replier = models.ForeignKey(User, on_delete=models.CASCADE, related_name='replies') likes = GenericRelation(Like)

```
    def send_notification(self, mentioned_user):
        Notification.objects.create(
            user=mentioned_user,
            message=f"You were mentioned in a reply: {self.reply_content[:50]}"
    # 通知消息可以包含回复内容的前50个字符
        )

    def __str__(self):
        return f'Reply to {self.post.post_title} by {self.replier.username}'
```

```

（1）在Article下发帖

**url：/index/post/article_post**

**POST /post/article_post**

**描述：** 此接口用于在文章下创建post。

**请求参数：**

| 参数名 | 类型 | 必填 | 描述 |
| --- | --- | --- | --- |
| `post_title` | string | 否 | 帖子标题。 |
| `post_content` | string | 是 | 帖子内容。 |

**响应参数：**

- **201 Created**: 帖子创建成功。
- **400 Bad Request**: 请求参数不完整或格式错误。
```

*（2）在Course下发帖：试行版

此处发帖实际上表示的是提问，不涉及评分，故不支持修改

url：/index/post/course_post

POST /post/course_post

描述： 此接口用于在课程下创建post。

请求参数：

参数名	类型	必填	描述
`post_title`	string	否	帖子标题。
`post_content`	string	是	帖子内容。

响应参数：

- **201 Created**: 帖子创建成功。
- **400 Bad Request**: 请求参数不完整或格式错误。

（3）通过id获取Post的详细信息

url：/index/post/detail

GET /post/detail?id=id

描述： 此接口用于获取指定Post的详细信息。

请求参数：

参数名	类型	必填	描述
`id`	int	是	帖子的id。

响应参数：

参数名	类型	描述
`status`	int	状态码。
`message`	string	返回信息。
`post_detail`	list	帖子详情。

- **200 OK**: 获取Post详情成功，返回详细信息。
- **404 Not Found**: 未找到指定的帖子。
- **400 Bad Request**: 请求参数不合法。

若获取成功，则`post_detail`的参数如下：

参数名	类型	描述
`post_id`	int	帖子id。
`post_title`	string	帖子标题。
`post_content`	string	帖子内容。
`poster_name`	string	发帖人的名字。
`poster_profile_url`	string	发帖人头像的url。
`view_count`	int	帖子的浏览量。
`like_count`	int	帖子的点赞数。
`reply_count`	int	帖子的回复数。
`tags`	array	帖子的标签。
`publish_time`	time	帖子的发布时间。

（4）根据id分页获取Post下的Reply列表

url：/index/post/reply_list

GET /post/reply_list?id=id&page_index=page_index&page_size=page_size

描述： 此接口用于根据Post的id分页获取Reply列表。

请求参数：

参数名	类型	必填	描述
`id`	int	是	post对应id。
`page_index`	int	否	第几页，默认为第一页。
`page_size`	int	否	每页显示的回复数量，默认为20。

响应参数：

参数名	类型	描述
`status`	int	状态码。
`message`	string	返回信息。
`reply_list`	array	回复列表。

- **200 OK**: 获取Reply列表成功。
- **404 Not Found**: 未找到指定的帖子或回复。
- **400 Bad Request**: 请求参数不合法。

若获取成功，则`reply_list`中的每个对象内容如下：

参数名	类型	描述
`reply_id`	int	回复ID。
`reply_content`	string	回复内容。
`replier_name`	string	回复者名字。
`replier_profile_url`	string	回复者头像的URL。
`like_count`	int	回复的点赞数。
`publish_time`	time	回复的发布时间。

（5）删除Post

url：/index/post/delete

POST /post/delete

描述： 此接口用于删除指定的Post。

请求参数：

参数名	类型	必填	描述
`id`	int	是	帖子的id。

响应参数：

- **200 OK**: 帖子删除成功。
- **404 Not Found**: 未找到指定的帖子。

（6）在Post下发表Reply

url：/index/reply/create

POST /reply/create

描述： 此接口用于在指定Post下发表回复。

请求参数：

参数名	类型	必填	描述
`id`	int	是	帖子的id。
`reply_content`	string	是	回复内容。

响应参数：

- **201 Created**: 回复成功。
- **400 Bad Request**: 请求参数不完整或格式错误。

（7）删除Reply

url：/index/reply/delete

POST /reply/delete

描述： 此接口用于删除指定的Reply。

请求参数：

参数名	类型	必填	描述
`id`	int	是	回复的id。

响应参数：

- **200 OK**: 回复删除成功。
- **404 Not Found**: 未找到指定的回复。

（8）获取reply详情

url：/index/reply/detail

GET /reply/detail?id=id

描述： 此接口用于获取指定reply的详细信息。

请求参数：

参数名	类型	必填	描述
`id`	int	是	回复的id。

响应参数：

参数名	类型	描述
`status`	int	状态码。
`message`	string	返回信息。
`reply_detail`	list	回复详情。

- **200 OK**: 获取reply详情成功，返回详细信息。
- **404 Not Found**: 未找到指定的回复。
- **400 Bad Request**: 请求参数不合法。

若获取成功，则`reply_detail`的参数如下：

参数名	类型	描述

参数名	类型	描述
`reply_id`	int	回复id。
`reply_content`	string	回复内容。
`replier_name`	string	回复者名字。
`replier_profile_url`	string	回复者头像的url。
`like_count`	int	回复的点赞数。
`publish_time`	time	回复的发布时间。

5.课程板块

（0）Course类

```python class Course(models.Model): COURSE_TYPE_CHOICES = [ ('compulsory', 'Compulsory'), # 必修课 ('elective', 'Elective'), # 选修课 ('restricted_elective', 'Restricted Elective'), # 限选课 ]

```
COURSE_METHOD_CHOICES = [
    ('online', 'Online'), # 线上
    ('offline', 'Offline'), # 线下
    ('hybrid', 'Hybrid'), # 混合
]

id = models.AutoField(primary_key=True) # 自增id, 自动设为主键
course_name = models.CharField(max_length=255) # 课程名
course_type = models.CharField(max_length=50, choices=COURSE_TYPE_CHOICES)
# 课程类型
college = models.CharField(max_length=255) # 开设大学
credits = models.DecimalField(max_digits=4, decimal_places=2) # 学分
course_teacher = models.CharField(max_length=255) # 课程老师
course_method = models.CharField(max_length=50,
choices=COURSE_METHOD_CHOICES) # 教学方式
assessment_method = models.CharField(max_length=255) # 考核方式
likes = GenericRelation(Like)
score = models.DecimalField(max_digits=3, decimal_places=2, default=0.00) #
评分
all_score = models.DecimalField(max_digits=3, decimal_places=2,
default=0.00) # 总评分
all_people = models.IntegerField(default=0) # 总评分人数
relative_articles = models.ManyToManyField(Article, related_name='courses')
publish_time = models.DateTimeField(auto_now_add=True)
```

```

（1）创建Course

url：/index/course/create

POST /course/create

描述： 此接口用于创建新的课程。

请求参数：

参数名	类型	必填	描述
`course_name`	string	是	课程名。
`course_type`	string	是	课程类型（必修、选修、限选）。
`college`	string	是	开设大学。
`credits`	decimal	是	课程学分。
`course_teacher`	string	是	课程教师。
`course_method`	string	是	教学方式（线上、线下、混合）。
`assessment_method`	string	是	考核方式。

响应参数：

- **200 OK**: 课程创建成功。
- **400 Bad Request**: 请求参数不完整或格式错误。

（2）编辑Course

url：/index/course/edit

POST /course/edit

描述： 此接口用于编辑已存在的课程。

请求参数：

参数名	类型	必填	描述
`id`	int	是	课程ID。
`course_name`	string	否	课程名。
`course_type`	string	否	课程类型（必修、选修、限选）。
`college`	string	否	开设大学。
`credits`	decimal	否	课程学分。
`course_teacher`	string	否	课程教师。
`course_method`	string	否	教学方式（线上、线下、混合）。
`assessment_method`	string	否	考核方式。

响应参数：

- **200 OK**: 课程编辑成功。
- **404 Not Found**: 未找到指定的课程。
- **400 Bad Request**: 请求参数不合法。

（3）删除Course

url：/index/course/delete

POST /course/delete

描述： 此接口用于删除指定的课程。

请求参数：

参数名	类型	必填	描述
`id`	int	是	课程ID。

响应参数：

- **200 OK**: 课程删除成功。
- **404 Not Found**: 未找到指定的课程。

---

*（4）对课程评分并评价：待修改

也许可以考虑直接合并到“Course下发帖”，打分同时发表post

url：/index/course/rate

POST /course/rate

描述： 此接口用于为课程打分并进行评价。

请求参数：

参数名	类型	必填	描述
`course_id`	int	是	课程ID。
`score`	decimal	是	评分（0.00 到 5.00）。
`comment`	string	否	评价内容。

响应参数：

- **201 Created**: 评分和评价提交成功。
- **400 Bad Request**: 请求参数不完整或格式错误。

---

*（5）修改课程评分或评价：待修改

同上

url：/index/course/edit_rating

POST /course/edit_rating

描述： 此接口用于修改已提交的课程评分或评价。

请求参数：

参数名	类型	必填	描述
`course_id`	int	是	课程ID。
`score`	decimal	否	修改后的评分（0.00 到 5.00）。
`comment`	string	否	修改后的评价内容。

响应参数：

- **200 OK**: 评分和评价修改成功。
- **404 Not Found**: 未找到指定的课程评分或评价。
- **400 Bad Request**: 请求参数不合法。

（6）通过id获取Course的详细信息

url：/index/course/detail

GET /course/detail?id=id

请求参数：

参数名	类型	必填	描述
`id`	int	是	课程的id。

响应参数：

参数名	类型	描述
`status`	int	状态码。
`message`	string	返回信息。
`course_detail`	list	课程详情。

- **200 OK**: 获取课程详情成功，返回详细信息。
- **404 Not Found**: 未找到指定的课程。
- **400 Bad Request**: 请求参数不合法。

若获取成功，则`course_detail`内容如下：

参数名	类型	描述
`course_id`	int	课程id。
`course_name`	string	课程名。
`course_type`	string	课程类型（必修、选修、限选）。
`college`	string	开设大学。
`credits`	decimal	学分。
`course_teacher`	string	教师名称。
`course_method`	string	教学方式（线上、线下、混合）。
`assessment_method`	string	考核方式。
`score`	decimal	评分（0.00 到 5.00）。
`all_score`	decimal	总评分。
`all_people`	int	总评分人数。
`relative_articles`	array	相关的文章列表。
`publish_time`	time	课程发布的时间。

（7）根据id分页获取Course下的Post列表

url：/index/course/post_list

GET /course/post_list?id=id&page_index=page_index&page_size=page_size

描述： 此接口用于获取指定课程（通过课程ID）下，按分页要求返回的Post列表。

请求参数：

参数名	类型	必填	描述
`id`	int	是	课程ID。
`page_index`	int	否	第几页，默认为第一页。
`page_size`	int	否	每页显示的Post数量，默认为20。

响应参数：

参数名	类型	描述
`status`	int	状态码。
`message`	string	返回信息。
`post_list`	array	Post列表。

- **200 OK**: 获取Post列表成功。

- **500 Internal Server Error**: 服务器内部错误。

若获取成功，则`post_list`中的每个Post对象的内容如下：

参数名	类型	描述
`post_id`	int	帖子ID。
`post_title`	string	帖子标题。
`post_content`	string	帖子内容。
`poster_name`	string	发帖人名称。
`poster_profile_url`	string	发帖人头像URL。
`view_count`	int	帖子的浏览量。
`like_count`	int	帖子的点赞数。
`reply_count`	int	帖子的回复数。
`tags`	array	帖子的标签。
`publish_time`	time	帖子的发布时间。

（8）分页获取course列表

url：/index/course/list

GET /course/list?page_index=page_index&page_size=page_size

描述： 此接口用于提交分页要求，并获取course列表的对应页。

请求参数：

参数名	类型	必填	描述
`page_index`	int	否	第几页，无此字段默认首页
`page_size`	int	否	每页项目数，无此字段默认20

响应参数：

参数名	类型	描述
`status`	int	状态码。
`message`	string	返回信息。
`course_list`	array	post列表。

- **200 OK**: 获取课程列表成功。
- **500 Internal Server Error**: 服务器内部错误；

若获取成功，则`course_list`中的每个list对象的内容如下：

参数名	类型	描述
`course_id`	int	课程id。
`course_name`	string	课程名。
`course_type`	string	课程类型（必修、选修、限选）。
`college`	string	开设大学。
`credits`	decimal	学分。
`course_teacher`	string	教师名称。
`course_method`	string	教学方式（线上、线下、混合）。
`assessment_method`	string	考核方式。
`score`	decimal	评分（0.00 到 5.00）。
`all_score`	decimal	总评分。
`all_people`	int	总评分人数。
`relative_articles`	array	相关的文章列表。
`publish_time`	time	课程发布的时间。

7.私信与通知板块

还没开始写

8.收藏夹板块

（0）Star类

（1）收藏文章

（2）收藏课程

9.点赞板块

（0）Like类

（1）给Article点赞

（2）给Post点赞

（3）给Reply点赞

10.图片API板块

（0）Image类

（1）上传image

11.封禁与屏蔽板块

还没开始写