

5 - 用户传图 - 智能协同云图库项目教程 - 编程导航教程

本节重点之前为了方便和安全性，只支持管理员上传图片，如果能让平台素材更丰富，也要允许用户自主上传图片。

本节重点

之前为了方便和安全性，只支持管理员上传图片，如果能让平台素材更丰富，也要允许用户自主上传图片。

本节我们就重点开发用户传图能力，并支持更多传图的方式，大纲：

- 支持用户上传图片和审核功能
- 通过 URL 导入图片
- 批量抓取和创建图片

一、用户上传图片及审核

需求分析

之前我们已经开发了管理员上传图片功能，想实现用户上传图片就比较简单了，但是我们要考虑到一点“用户上传的内容可能是不安全的”。

一般只要涉及到“用户上传内容”（俗称 UGC）的场景，就要增加审核功能。

具体分析每个需求：

1) 用户上传创建图片：需要开放权限，允许用户上传图片，功能和流程跟之前管理员上传图片一致，也要增加文件校验。

2) 管理员审核图片：管理员可以查看和 **筛选** 所有待审核的图片，并标记为通过或拒绝，可填写通过或拒绝的具体原因。此外，需要记录审核人和审核时间作为日志，如果发现误审的情况也可以追责。

方案设计

方案设计阶段我们需要确认：

- 审核的具体逻辑
- 库表设计

1、审核逻辑

1) 管理员可以操作审核的状态流转：

- 默认为“待审核”，可以设置为“审核通过”或“审核拒绝”
- 已拒绝的图片可以重新审核为通过
- 已通过的图片可以撤销为拒绝状态

2) 管理员自动审核：管理员上传 / 更新图片时，图片自动审核通过，并且自动填充审核参数 —— 设置审核人为创建人、审核时间为当前时间、审核原因为“管理员自动过审”。

3) 用户操作需要审核：用户上传或编辑图片时，图片的状态会被重置为“待审核”。

重复审核时，既可以选择重置 **所有** 审核参数，也可以仅重置审核状态。其余参数在前端不展示，但是在后端保留，以便管理员参考历史审核信息。

4) 控制内容可见性：对于用户来说，应该只能看见“审核通过”状态的数据；管理员可以在图片管理页面看到所有数据，并且根据审核状态筛选图片。

Q：是否要考虑并发问题呢？

A：由于审核操作为管理员手动执行，不涉及复杂的奖励机制或并发高频请求，误审核或重复审核对系统影响不大，因此无需过度考虑并发问题。

2、库表设计

为了支持审核功能，我们在 picture 图片表中新增审核相关字段，同时优化索引设计以提升查询性能。

修改表的 SQL 如下：

```
ALTER TABLE picture

    ADD COLUMN reviewStatus INT DEFAULT 0 NOT NULL COMMENT '审核状态：
    ADD COLUMN reviewMessage VARCHAR(512) NULL COMMENT '审核信息'，
    ADD COLUMN reviewerId BIGINT NULL COMMENT '审核人 ID'，
    ADD COLUMN reviewTime DATETIME NULL COMMENT '审核时间'；

CREATE INDEX idx_reviewStatus ON picture (reviewStatus);
```



注意事项：

- 1) 审核状态：reviewStatus 使用整数（0、1、2）表示不同的审核状态，而不是用字符串，可以节约表的空间、提升查找效率。
- 2) 索引设计：由于要根据审核状态筛选图片，所以给该字段添加索引，提升查询性能。

下面我们进行开发，先后端再前端。

后端开发

1、数据模型开发

由于新增了一些审核相关的字段，要对原有的数据模型（实体类、包装类等）进行修改。

- 1) 实体类 Picture 新增：

```
private Integer reviewStatus;
```

```
private String reviewMessage;
```

```
private Long reviewerId;
```

```
private Date reviewTime;
```

2) 图片查询请求类 PictureQueryRequest 新增:

```
private Integer reviewStatus;
```

```
private String reviewMessage;
```

```
private Long reviewerId;
```

3) 新建审核状态枚举类:

```
@Getter
public enum PictureReviewStatusEnum {
    REVIEWING("待审核", 0),
    PASS("通过", 1),
    REJECT("拒绝", 2);

    private final String text;
    private final int value;

    PictureReviewStatusEnum(String text, int value) {
        this.text = text;
        this.value = value;
    }

    public static PictureReviewStatusEnum getEnumByValue(Integer value) {
        if (ObjUtil.isEmpty(value)) {
            return null;
        }
        for (PictureReviewStatusEnum pictureReviewStatusEnum : PictureReviewStatusEnum.values()) {
            if (pictureReviewStatusEnum.value == value) {
                return pictureReviewStatusEnum;
            }
        }
        return null;
    }
}
```

2、管理员审核功能

1) 开发请求包装类，注意不需要增加 reviewerId 和 reviewTime 字段，这两个是由系统自动填充的，而不是由前端传递。

```
@Data
public class PictureReviewRequest implements Serializable {

    private Long id;

    private Integer reviewStatus;

    private String reviewMessage;

    private static final long serialVersionUID = 1L;
}
```

2) 开发审核服务

接口：

```
void doPictureReview(PictureReviewRequest pictureReviewRequest, User
```



实现类：

```
@Override
public void doPictureReview(PictureReviewRequest pictureReviewRequest) {
    Long id = pictureReviewRequest.getId();
    Integer reviewStatus = pictureReviewRequest.getReviewStatus();
    PictureReviewStatusEnum reviewStatusEnum = PictureReviewStatusEnum
    if (id == null || reviewStatusEnum == null || PictureReviewStatusEnum
        throw new BusinessException(ErrorCode.PARAMS_ERROR);
    }

    Picture oldPicture = this.getById(id);
    ThrowUtils.throwIf(oldPicture == null, ErrorCode.NOT_FOUND_ERROR)

    if (oldPicture.getReviewStatus().equals(reviewStatus)) {
        throw new BusinessException(ErrorCode.PARAMS_ERROR, "请勿重复审
    }

    Picture updatePicture = new Picture();
    BeanUtils.copyProperties(pictureReviewRequest, updatePicture);
    updatePicture.setReviewerId(loginUser.getId());
    updatePicture.setReviewTime(new Date());
}
```

```

        boolean result = this.updateById(updatePicture);
        ThrowUtils.throwIf(!result, ErrorCode.OPERATION_ERROR);
    }

```

3) 开发审核接口，注意权限设置为仅管理员可用：

```

@PostMapping("/review")
@AuthCheck(mustRole = UserConstant.ADMIN_ROLE)
public BaseResponse<Boolean> doPictureReview(@RequestBody PictureReviewRequest request,
                                              HttpServletRequest req) {
    ThrowUtils.throwIf(request == null, ErrorCode.PARAMS_ERROR);
    User loginUser = userService.getLoginUser(request);
    pictureService.doPictureReview(request, loginUser);
    return ResultUtils.success(true);
}

```

3、审核状态设置

1) 权限控制

首先取消上传图片接口（uploadPicture）的权限校验注解，但是注意，由于图片上传功能是支持图片编辑的，所以需要做好编辑权限控制——仅本人或管理员可编辑。

修改 PictureService 的 uploadPicture 方法，补充权限校验逻辑：

```

    if (pictureId != null) {
        Picture oldPicture = this.getById(pictureId);
        ThrowUtils.throwIf(oldPicture == null, ErrorCode.NOT_FOUND_ERROR);

        if (!oldPicture.getUserId().equals(loginUser.getId()) && !userService.isAdmin(loginUser)) {
            throw new BusinessException(ErrorCode.NO_AUTH_ERROR);
        }
    }
}

```

2) 设置审核状态：管理员自动过审并且填充审核参数；用户上传或编辑图片时，图片的状态会被重置为“待审核”。

由于图片上传、用户编辑、管理员更新这 3 个操作都需要设置审核状态，所以我们可以先编写一个通用的“补充审核参数”的方法，根据用户的角色给图片对象填充审核字段的值。

```

@Override
public void fillReviewParams(Picture picture, User loginUser) {
    if (userService.isAdmin(loginUser)) {

        picture.setReviewStatus(PictureReviewStatusEnum.PASS.getValue);
        picture.setReviewerId(loginUser.getId());
        picture.setReviewMessage("管理员自动过审");
        picture.setReviewTime(new Date());
    } else {

        picture.setReviewStatus(PictureReviewStatusEnum.REVIEWING.get
    }
}

```



分别给 3 个操作补充审核参数。图片更新接口：

```

public BaseResponse<Boolean> updatePicture(@RequestBody PictureUpdate
    , HttpServletRequest request) {

    Picture oldPicture = pictureService.getById(id);
    ThrowUtils.throwIf(oldPicture == null, ErrorCode.NOT_FOUND_ERROR)

    User loginUser = userService.getLoginUser(request);
    pictureService.fillReviewParams(picture, loginUser);

    boolean result = pictureService.updateById(picture);
    ThrowUtils.throwIf(!result, ErrorCode.OPERATION_ERROR);
    return ResultUtils.success(true);
}

```



图片修改接口：

```

public BaseResponse<Boolean> editPicture(@RequestBody PictureEditRequ

    if (!oldPicture.getUserId().equals(loginUser.getId()) && !userSer
        throw new BusinessException(ErrorCode.NO_AUTH_ERROR);
    }

    pictureService.fillReviewParams(picture, loginUser);

    boolean result = pictureService.updateById(picture);
    ThrowUtils.throwIf(!result, ErrorCode.OPERATION_ERROR);
    return ResultUtils.success(true);
}

```



上传图片服务：

```

@Override
public PictureVO uploadPicture(Object inputSource, PictureUploadReque

    picture.setPicFormat(uploadPictureResult.getPicFormat());
    picture.setUserId(loginUser.getId());

    fillReviewParams(picture, loginUser);

    if (pictureId != null) {

        picture.setId(pictureId);
        picture.setEditTime(new Date());
    }

}

```

4、控制内容可见性

目前我们只有主页给用户查看图片列表，所以需要修改主页调用的 listPictureVOByPage 接口，补充查询条件即可，默认只能查看已过审的数据：

```

pictureQueryRequest.setReviewStatus(PictureReviewStatusEnum.PASS.getV

Page<Picture> picturePage = pictureService.page(new Page<>(current, s
    pictureService.getQueryWrapper(pictureQueryRequest));

```

需要同步更改 PictureService 的 getQueryWrapper 方法，支持根据审核字段进行查询：

```

Integer reviewStatus = pictureQueryRequest.getReviewStatus();
String reviewMessage = pictureQueryRequest.getReviewMessage();
Long reviewerId = pictureQueryRequest.getReviewerId();
queryWrapper.eq(ObjUtil.isNotEmpty(reviewStatus), "reviewStatus", rev
queryWrapper.like(StrUtil.isNotBlank(reviewMessage), "reviewMessage",
queryWrapper.eq(ObjUtil.isNotEmpty(reviewerId), "reviewerId", reviewe

```

这样一来，后端就同时支持了“管理员筛选审核状态”的功能。

至此，用户上传图片及审核的后端就开发完成了。

Q：根据 id 查询图片的接口需要做同样的限制么？

A: 对目前咱们的系统来说, 用户正常情况下不会得到未过审图片的 id, 影响面较小, 可以暂时不做, 感兴趣的同学按需优化即可。

前端开发

控制首页未过审的图片不可见已经通过后端接口实现, 我们只需开发管理员审核和按审核状态筛选图片功能即可。

1、定义审核常量

和后端一样, 前端也有很多地方要用到审核状态信息, 可以定义为一个常量。

在 `constants` 目录下新建 `picture.ts` 常量文件, 定义枚举信息、对应的中文映射、以及后续筛选审核状态时要用到的选项数组:

```
export const PIC_REVIEW_STATUS_ENUM = {
  REVIEWING: 0,
  PASS: 1,
  REJECT: 2,
}

export const PIC_REVIEW_STATUS_MAP = {
  0: '待审核',
  1: '通过',
  2: '拒绝',
}

export const PIC_REVIEW_STATUS_OPTIONS = Object.keys(PIC_REVIEW_STATU
  return {
    label: PIC_REVIEW_STATUS_MAP[key],
    value: key,
  }
})
```



2、管理员审核功能

1) 表格列新增审核信息:

```
const columns = [

  {
    title: '审核信息',
    dataIndex: 'reviewMessage',
  },
],
```

]

2) 自定义审核信息列要展示的内容:

```
<!-- 审核信息 -->
<template v-if="column.dataIndex === 'reviewMessage'">
  <div>审核状态: {{ PIC_REVIEW_STATUS_MAP[record.reviewStatus] }}</div>
  <div>审核信息: {{ record.reviewMessage }}</div>
  <div>审核人: {{ recordReviewerId }}</div>
</template>
```



3) 新增审核通过和拒绝的操作按钮:

```
<template v-else-if="column.key === 'action'">
  <a-space wrap>
    <a-button
      v-if="record.reviewStatus !== PIC_REVIEW_STATUS_ENUM.PASS"
      type="link"
      @click="handleReview(record, PIC_REVIEW_STATUS_ENUM.PASS)"
    >
      通过
    </a-button>
    <a-button
      v-if="record.reviewStatus !== PIC_REVIEW_STATUS_ENUM.REJECT"
      type="link"
      danger
      @click="handleReview(record, PIC_REVIEW_STATUS_ENUM.REJECT)"
    >
      拒绝
    </a-button>
    <a-button type="link" :href="`/add_picture?id=${record.id}`" targ
    >编辑
    </a-button>
    <a-button type="link" danger @click="doDelete(record.id)">删除</a
  </a-space>
</template>
```



4) 编写审核函数, 调用后端完成操作:

```
const handleReview = async (record: API.Picture, reviewStatus: number) => {
  const reviewMessage = reviewStatus === PIC_REVIEW_STATUS_ENUM.PASS
  const res = await doPictureReviewUsingPost({
    id: record.id,
    reviewStatus,
    reviewMessage,
  })
  if (res.data.code === 0) {
    message.success('审核操作成功')

    fetchData()
  }
}
```



```
    } else {  
      message.error('审核操作失败，' + res.data.message)  
    }  
  }  
}
```

3、按审核状态筛选

只需要在原来的搜索表单中补充一个表单项即可，使用下拉框组件，传入定义好的审核状态常量作为选项：

```
<a-form-item label="审核状态" name="reviewStatus">  
  <a-select  
    v-model:value="searchParams.reviewStatus"  
    :options="PIC_REVIEW_STATUS_OPTIONS"  
    placeholder="请输入审核状态"  
  
    allow-clear  
  />  
</a-form-item>
```

前端扩展

- 1) 审核通过或拒绝时可以填写原因，可以利用 [模态框组件](#) 实现
- 2) 可以在详情页添加审核通过、审核拒绝的快捷操作按钮，仅管理员可见
- 3) 删除和拒绝属于危险操作，可以点击后提示确认框，确认后才执行操作。可以利用 [组件](#) 实现

测试

测试流程：

1. 普通用户上传图片，此时主页看不到这张图片
2. 管理员操作过审，主页就可以看到这张图片
3. 管理员操作拒绝，主页将无法看到这张图片

运行效果如图：

鱼皮云图库

🏠 主页

📄 创建图片

👤 用户管理

🖼️ 图片管理

📑 编程导航

👤 程序员鱼皮

图片管理

关键词: 从名称和简介搜索

类型: 请选择类型

标签: 请选择标签

审核状态: 请选择审核状态

搜索

id	图片	名称	简介	类型	标签	图片信息	用户 id	审核信息	创建时间	编辑时间	操作
186241 87910 69589 505		秋招热题	面试鸭刷题平...	海报	高清 热门	格式: PNG 宽度: 3264 高度: 1150 宽高比: 1.9 大小: 489.50KB	186168 23429 80448 257	审核状态: 拒绝 审核信息: 拒绝 员操作通过 审核人: 18616823429 80448257	2024-11-29 16:50:16	2024-11-29 21:09:04	通过 编辑 删除
186241 52775 22100 226		323617302614 35_pic_hjd	鱼皮新书出版	素材	生活 高清	格式: JPEG 宽度: 3264 高度: 2448 宽高比: 1.33 大小: 1482.02KB	186168 23429 80448 257	审核状态: 通过 审核信息: 管理 员操作通过 审核人: 18616823429 80448257	2024-11-29 16:36:18	2024-11-29 17:16:55	拒绝 编辑 删除
186212 57883 67081 474		鱼皮生活照	25 岁的鱼皮...	素材	生活 搞笑 艺术 高清	格式: JPEG 宽度: 1280 高度: 1707 宽高比: 0.75 大小: 720.68KB	186168 23429 80448 257	审核状态: 通过 审核信息: 管理 员操作通过 审核人: 18616823429 80448257	2024-11-28 21:25:59	2024-11-29 17:17:31	拒绝 编辑 删除

鱼皮云图库

🏠 主页

📄 创建图片

👤 用户管理

🖼️ 图片管理

📑 编程导航

👤 程序员鱼皮

图片管理

关键词: 从名称和简介搜索

类型: 请选择类型

标签: 请选择标签

审核状态: 请选择审核状态

搜索

id	图片	名称	简介	类型	标签	图片信息	用户 id	审核信息	创建时间	编辑时间	操作
186241 87910 69589 505		秋招热题	面试鸭刷题平...	海报	高清 热门	格式: PNG 宽度: 2184 高度: 1150 宽高比: 1.9 大小: 489.50KB	186168 23429 80448 257	审核状态: 拒绝 审核信息: 管理 员操作通过 审核人: 18616823429 80448257	2024-11-29 16:50:16	2024-11-29 21:09:04	通过 编辑 删除
186241 52775 22100 226		323617302614 35_pic_hjd	鱼皮新书出版	素材	生活 高清	格式: JPEG 宽度: 3264 高度: 2448 宽高比: 1.33 大小: 1482.02KB	186168 23429 80448 257	审核状态: 通过 审核信息: 管理 员操作通过 审核人: 18616823429 80448257	2024-11-29 16:36:18	2024-11-29 17:16:55	拒绝 编辑 删除
186212 57883 67081 474		鱼皮生活照	25 岁的鱼皮...	素材	生活 搞笑 艺术 高清	格式: JPEG 宽度: 1280 高度: 1707 宽高比: 0.75 大小: 720.68KB	186168 23429 80448 257	审核状态: 通过 审核信息: 管理 员操作通过 审核人: 18616823429 80448257	2024-11-28 21:25:59	2024-11-29 17:17:31	拒绝 编辑 删除

Q：有同学说，用户现在看不到自己被拒绝的图片啊？怎么修改？

A：本项目后续会带大家开发用户个人空间，到时候会完成这个功能。

扩展

1、更多审核策略

在实际企业中，为了提高审核效率、减少垃圾内容，同时保证用户体验和平台的安全性，常常会结合技术手段和业务策略来优化审核流程。比如下面几点，大家可以按需扩展：

1. 内容安全审核服务：借助专业的第三方平台的内容审核服务来实现自动审核，像腾讯云、阿里云等基本都支持图片、文本、音视频等内容的审核。
2. AI 审核：可以将文本内容和审核规则输入给 AI，让 AI 返回是否合规。

3. 分级审核策略：区分普通用户与高信誉用户，高信誉用户可减少或免除审核流程，比如 VIP 用户自动过审，也可以提高部分效率。
4. 实名信息和内容溯源：通过用户实名或者手机号注册，提高用户行为的责任感，减少垃圾内容的产生。
5. 举报机制：通过给平台增加举报机制，还可以给举报行为一些奖励，让用户帮忙维护平台。

2、审核通知

当管理员完成审核后，系统可以通过消息中心或邮件通知用户审核结果。

二、通过 URL 导入图片

需求分析

为了提高上传图片的效率，除了支持上传本地文件外，还可以支持输入一个远程 URL，直接将网上已有的图片导入到我们的系统中。

方案设计

实现原理很简单，但是有一些细节需要注意：

1) 下载图片：后端服务器从指定的远程 URL 下载图片到本地临时存储。对于 Java 项目，可以直接使用 Hutool 的 `HttpUtil.downloadFile` 方法一行代码完成。

2) 校验图片：跟验证本地文件一样，需要校验图片的格式、大小等。

传统的校验思路是先把文件下载到本地，再对本地文件进行校验，有没有更节省资源的方法呢？

** 其实可以先对 URL 本身进行校验。 ** 首先是校验 URL 字符串本身的合法性，比如要是合理的 URL 地址。此外，可以先使用 `HEAD` 请求来获取 URL 对应文件的元信息（如文件大小、格式等）。HEAD 请求仅返回 HTTP 响应头信息，而不会下载文件的内容，大大降低了网络流量的消耗。

注意此处不能使用 GET 请求，它会获取完整文件。

3) 上传图片：将校验通过的图片上传到对象存储服务，生成存储 URL。

之后的流程就都可以复用从本地上传图片的流程了。

后端开发

1、服务开发

先编写通过 URL 上传文件的方法，为了便于开发，直接在 FileManager 类中编写，绝大多数代码跟之前的 uploadPicture 方法一致，只需要改动以下 4 处位置：

1. 方法接受的参数：之前是 MultipartFile 文件类型，现在是 String 字符串类型
2. 校验图片：之前是校验文件，现在是校验 URL
3. 获取文件名称：之前是根据文件获取，现在是根据 URL 获取
4. 保存临时文件：之前是将 MultipartFile 写入到临时文件，现在是从 URL 下载文件

代码如下：

```
public UploadPictureResult uploadPictureByUrl(String fileUrl, String

    validPicture(fileUrl);

    String uuid = RandomUtil.randomString(16);

    String originFilename = FileUtil.mainName(fileUrl);
    String uploadFilename = String.format("%s_%s.%s", DateUtil.format
        FileUtil.getSuffix(originFilename));
    String uploadPath = String.format("/%s/%s", uploadPathPrefix, upl
    File file = null;
    try {

        file = File.createTempFile(uploadPath, null);

        HttpUtil.downloadFile(fileUrl, file);

    } catch (Exception e) {
```

```

        log.error("图片上传到对象存储失败", e);
        throw new BusinessException(ErrorCode.SYSTEM_ERROR, "上传失败")
    } finally {
        this.deleteTempFile(file);
    }
}

```

2、校验 URL 图片

编写校验 URL 图片的方法，分别校验 URL 格式、协议、文件是否存在、文件格式、文件大小。

代码如下：

```

private void validPicture(String fileUrl) {
    ThrowUtils.throwIf(StrUtil.isBlank(fileUrl), ErrorCode.PARAMS_ERROR)

    try {

        new URL(fileUrl);
    } catch (MalformedURLException e) {
        throw new BusinessException(ErrorCode.PARAMS_ERROR, "文件地址错误")
    }

    ThrowUtils.throwIf(!(fileUrl.startsWith("http://") || fileUrl.startsWith("https://")),
        ErrorCode.PARAMS_ERROR, "仅支持 HTTP 或 HTTPS 协议的文件地址")

    HttpResponse response = null;
    try {
        response = HttpUtil.createRequest(Method.HEAD, fileUrl).execute()

        if (response.getStatus() != HttpStatus.HTTP_OK) {
            return;
        }

        String contentType = response.header("Content-Type");
        if (StrUtil.isNotBlank(contentType)) {

            final List<String> ALLOW_CONTENT_TYPES = Arrays.asList("image/jpeg", "image/png", "image/gif", "image/webp");
            ThrowUtils.throwIf(!ALLOW_CONTENT_TYPES.contains(contentType),
                ErrorCode.PARAMS_ERROR, "文件类型错误");
        }

        String contentLengthStr = response.header("Content-Length");
        if (StrUtil.isNotBlank(contentLengthStr)) {
            try {
                long contentLength = Long.parseLong(contentLengthStr);
                final long TWO_MB = 2 * 1024 * 1024L;
                ThrowUtils.throwIf(contentLength > TWO_MB, ErrorCode.PARAMS_ERROR, "文件大小超过 2MB");
            } catch (NumberFormatException e) {
                throw new BusinessException(ErrorCode.PARAMS_ERROR, "文件大小校验失败")
            }
        }
    } finally {
        if (response != null) {
            response.close();
        }
    }
}

```

```
        response.close();
    }
}
```

上述代码中，注意 2 点：

1. 注意发送 HTTP 请求后，需要即时释放资源
2. 有些 URL 地址可能不支持通过 HEAD 请求访问，为了提高导入成功率，即使 HEAD 请求访问失败，也不会报错，并且不用执行后续的校验。**仅对能获取到的信息进行校验。**

3、优化代码 - 模板方法模式

目前我们的 FileManager 文件内写了两种不同的上传文件的方法，但是我们会发现，这两种方法的 **流程完全一致**、而且大多数代码都是相同的。

这种情况下，我们就要想要运用设计模式 —— **模板方法模式** 对代码进行优化。

模板方法模式是行为型设计模式，适用于具有通用处理流程、但处理细节不同的情况。通过定义一个抽象模板类，提供通用的业务流程处理逻辑，并将不同的部分定义为抽象方法，由子类具体实现。

在我们的场景中，两种文件上传方法的流程都是：

1. 校验文件
2. 获取上传地址
3. 获取本地临时文件
4. 上传到对象存储
5. 封装解析得到的图片信息
6. 清理临时文件

可以将这些流程抽象为一套模板（抽象类），将每个实现不一样的步骤都定义为一个抽象方法，比如：

1. 校验图片
2. 获取文件名称
3. 保存临时文件

下面开始开发，先在 `manager` 包下新建 `upload` 包，将模板方法有关的代码全部放在该包下统一管理。

1) 新建图片上传模板 **抽象类** `PictureUploadTemplate`，代码如下：

```
@Slf4j
public abstract class PictureUploadTemplate {

    @Resource
    protected CosManager cosManager;

    @Resource
    protected CosClientConfig cosClientConfig;

    public final UploadPictureResult uploadPicture(Object inputSource) {

        validPicture(inputSource);

        String uuid = RandomUtil.randomString(16);
        String originFilename = getOriginFilename(inputSource);
        String uploadFilename = String.format("%s_%s.%s", DateUtil.format("yyyyMMddHHmmss", DateUtil.now()),
            FileUtil.getSuffix(originFilename));
        String uploadPath = String.format("/%s/%s", uploadPathPrefix, uuid);

        File file = null;
        try {

            file = File.createTempFile(uploadPath, null);

            processFile(inputSource, file);

            PutObjectResult putObjectResult = cosManager.putObject(uploadPath, file);
            ImageInfo imageInfo = putObjectResult.getCiUploadResult();

            return buildResult(originFilename, file, uploadPath, imageInfo);
        } catch (Exception e) {
            log.error("图片上传到对象存储失败", e);
            throw new BusinessException(ErrorCodes.SYSTEM_ERROR, "上传图片失败");
        } finally {

            deleteTempFile(file);
        }
    }

    protected abstract void validPicture(Object inputSource);
}
```

```

        protected abstract String getOriginFilename(Object inputSource);

        protected abstract void processFile(Object inputSource, File file);

        private UploadPictureResult buildResult(String originFilename, File file) {
            UploadPictureResult uploadPictureResult = new UploadPictureResult();
            int picWidth = imageInfo.getWidth();
            int picHeight = imageInfo.getHeight();
            double picScale = NumberUtil.round(picWidth * 1.0 / picHeight);
            uploadPictureResult.setPicName(FileUtil.mainName(originFilename));
            uploadPictureResult.setPicWidth(picWidth);
            uploadPictureResult.setPicHeight(picHeight);
            uploadPictureResult.setPicScale(picScale);
            uploadPictureResult.setPicFormat(imageInfo.getFormat());
            uploadPictureResult.setPicSize(FileUtil.size(file));
            uploadPictureResult.setUrl(cosClientConfig.getHost() + "/" + FileUtil.subName(originFilename));
            return uploadPictureResult;
        }

        public void deleteTempFile(File file) {
            if (file == null) {
                return;
            }
            boolean deleteResult = file.delete();
            if (!deleteResult) {
                log.error("file delete error, filepath = {}", file.getAbsolutePath());
            }
        }
    }
}

```

上述代码中，我们把每个步骤都封装为了一个单独的方法，公共的实现（比如 `deleteTempFile`）可以直接放到模板中，而不用放到具体的实现类中。

注意，为了让模板同时兼容 `MultiPartFile` 和 `String` 类型的文件参数，直接将这两种情况统一为 `Object` 类型的 `inputSource` 输入源。

2) 新建本地图片上传子类 `FilePictureUpload`，继承模板，并且打上 `@Service` 注解生成 Bean 实例：

```

@Service
public class FilePictureUpload extends PictureUploadTemplate {

    @Override
    protected void validPicture(Object inputSource) {
        MultiPartFile multipartFile = (MultiPartFile) inputSource;
        ThrowUtils.throwIf(multipartFile == null, ErrorCode.PARAMS_ERROR);
    }
}

```



```

        long fileSize = multipartFile.getSize();
        final long ONE_M = 1024 * 1024L;
        ThrowUtils.throwIf(fileSize > 2 * ONE_M, ErrorCode.PARAMS_ERR);

        String fileSuffix = FileUtil.getSuffix(multipartFile.getOriginalFilename());

        final List<String> ALLOW_FORMAT_LIST = Arrays.asList("jpeg", "png", "gif", "bmp");
        ThrowUtils.throwIf(!ALLOW_FORMAT_LIST.contains(fileSuffix), ErrorCode.PARAMS_ERR);
    }

    @Override
    protected String getOriginFilename(Object inputSource) {
        MultipartFile multipartFile = (MultipartFile) inputSource;
        return multipartFile.getOriginalFilename();
    }

    @Override
    protected void processFile(Object inputSource, File file) throws IOException {
        MultipartFile multipartFile = (MultipartFile) inputSource;
        multipartFile.transferTo(file);
    }
}

```

3) 新建 URL 图片上传子类 `UrlPictureUpload`，继承模板，并且打上 `@Service` 注解生成 Bean 实例：

```

@Service
public class UrlPictureUpload extends PictureUploadTemplate {
    @Override
    protected void validPicture(Object inputSource) {
        String fileUrl = (String) inputSource;
        ThrowUtils.throwIf(StrUtil.isBlank(fileUrl), ErrorCode.PARAMS_ERR);
    }

    @Override
    protected String getOriginFilename(Object inputSource) {
        String fileUrl = (String) inputSource;

        return FileUtil.mainName(fileUrl);
    }

    @Override
    protected void processFile(Object inputSource, File file) throws IOException {
        String fileUrl = (String) inputSource;

        HttpUtil.downloadFile(fileUrl, file);
    }
}

```



优化完后，可以还原 `FileManager` 文件，并添加 `@Deprecated` 注解表示已废弃，后续将直接使用文件上传模板类

PictureUploadTemplate。

@Deprecated

4、图片上传服务支持 URL 上传

由于图片上传的逻辑还是比较复杂的，尽量让 URL 上传复用之前的代码。

但是之前图片上传服务的 uploadPicture 方法接受的是文件类型的参数，现在要支持 URL 上传，怎么办呢？

可以将输入参数跟上述模板一样，改为 Object 类型的 inputSource，然后在代码中可以根据 inputSource 的实际类型，来选择对应的图片上传子类。代码如下：

```
@Resource
private FilePictureUpload filePictureUpload;

@Resource
private UrlPictureUpload urlPictureUpload;

public PictureVO uploadPicture(Object inputSource, PictureUploadReque
    if (inputSource == null) {
        throw new BusinessException(ErrorCode.PARAMS_ERROR, "图片为空"
    }

    String uploadPathPrefix = String.format("public/%s", loginUser.ge

    PictureUploadTemplate pictureUploadTemplate = filePictureUpload;
    if (inputSource instanceof String) {
        pictureUploadTemplate = urlPictureUpload;
    }
    UploadPictureResult uploadPictureResult = pictureUploadTemplate.u

}
```

💡 除了通过对象类型判断外，也可以通过传一个业务参数（如 type）来区分不同的上传方式。

5、接口开发

1) 在请求封装类 PictureUploadRequest 中新增 fileUrl 文件地址：

```

@Data
public class PictureUploadRequest implements Serializable {

    private Long id;

    private String fileUrl;

    private static final long serialVersionUID = 1L;
}

```

2) 在 PictureContoller 中新增接口，通过 URL 上传图片：

```

@PostMapping("/upload/url")
public BaseResponse<PictureVO> uploadPictureByUrl(
    @RequestBody PictureUploadRequest pictureUploadRequest,
    HttpServletRequest request) {
    User loginUser = userService.getLoginUser(request);
    String fileUrl = pictureUploadRequest.getFileUrl();
    PictureVO pictureVO = pictureService.uploadPicture(fileUrl, pictu
    return ResultUtils.success(pictureVO);
}

```



然后通过 Swagger 接口文档测试本地文件图片和 URL 图片的上传，示例图片 URL：

<https://www.codefather.cn/logo.png>

前端开发

和本地上传图片的开发流程一样，我们先来开发一个 URL 上传图片的组件，绝大多数代码都可以复用本地上传图片组件。

1、URL 上传组件

URL 上传组件 = 文本输入框 + 提交按钮

可以使用组件库的 [复合输入框组件](#)：

```

<div>
  <a-input-group compact>
    <a-input v-model:value="fileUrl" placeholder="请输入图片 URL" />
    <a-button type="primary" :loading="loading" @click="handleUpload" />
  </a-input-group>
  

```

</div>

开发上传操作函数，需要将用户输入的 fileUrl 提交到后端：

```
const loading = ref<boolean>(false)
const fileUrl = ref<string>()

const handleUpload = async () => {
  loading.value = true
  try {
    const params: API.PictureUploadRequest = { fileUrl: fileUrl.value
    if (props.picture) {
      params.id = props.picture.id
    }
    const res = await uploadPictureByUrlUsingPost(params)
    if (res.data.code === 0 && res.data.data) {
      message.success('图片上传成功')

      props.onSuccess?.(res.data.data)
    } else {
      message.error('图片上传失败，' + res.data.message)
    }
  } catch (error) {
    message.error('图片上传失败')
  } finally {
    loading.value = false
  }
}
```



2、开发创建页面

之前已经开发了创建图片页面，可以在上传图片时增加一个 [Tabs 选项组件](#)，让用户自己选择上传方式。

```
<!-- 选择上传方式 -->
<a-tabs v-model:activeKey="uploadType"
  >>
  <a-tab-pane key="file" tab="文件上传">
    <PictureUpload :picture="picture" :onSuccess="onSuccess" />
  </a-tab-pane>
  <a-tab-pane key="url" tab="URL 上传" force-render>
    <UrlPictureUpload :picture="picture" :onSuccess="onSuccess" />
  </a-tab-pane>
</a-tabs>
```



定义上传类型变量：

```
const uploadType = ref<'file' | 'url'>('file')
```

其他代码都不需要调整。你会发现只要开发思路清晰、代码结构良好，新功能的扩展是很快的~

测试

没上传图片时，效果如图：



创建图片

文件上传 URL 上传

请输入图片 URL

提交

编程导航
codefather.cn

上传图片后，效果如图：



创建图片

文件上传 URL 上传

https://pic.code-nav.cn/mianshiya/other_pi

提交

快速入门少走弯路

JOIN US

编程导航

编程学习 交流答疑 项目实战 求职辅导

10+ 项目教程
100+ 技术直播
1000+ 精华文章

名称

IFfRNFyU_bcdhchanggui

编程导航
codefather.cn

除了创建外，最好也测试下修改图片，防止优化代码的过程中出现了疏漏。

批量抓取和创建图片

需求分析

为了帮助管理员快速丰富图片库，冷启动项目，需要提供批量从网络抓取并创建图片的功能。

**** 但是要注意，不建议将该功能开放给普通用户！ **** 主要是为了防止滥用导致的版权问题、低质量内容的上传、服务器资源消耗和安全问题。因为我们要从网络批量抓取图片（爬虫），如果功能开放给用户，相当于所有用户都在使用我们的服务器作为爬虫源头，容易导致我们的服务器 IP 被封禁。

方案设计

方案设计的重点包括：

- 如何抓取图片
- 抓取和导入规则

1、如何抓取图片？

思考 2 个问题：从哪里抓取图片？怎么抓取图片呢？

绝大多数的图片素材网站，都是有版权保护的，不建议大家操作，容易被封禁 IP 和账号。比较安全的方法是从搜索引擎中抓取图片，仅学习使用、不商用的话基本不会有什么风险。

这里我们选择从 bing 搜索获取图片，首先进入 [bing 图片网站](#)，可以看到很多图片，但是如何获取这些图片呢？



有 2 种常见的做法，第一种是请求到完整的页面内容后，对页面的 HTML 结构进行解析，提取到图片的地址，再通过 URL 下载；还有一种是直接调用后端获取图片地址的接口拿到图片数据。

要使用哪种方式，还是要具体情况具体分析，比如在调研过程中，我们会发现直接从 bing 图片的首页抓取数据，可能会出现获取不到图片的情况。所以我们换一种策略，尝试去找图片接口。

按 F12 打开网络请求控制台，向下滚动图片时会触发新一波图片的加载，就能看到获取图片数据的接口了：

<https://cn.bing.com/images/async?q=%s&mmsync=1>

注意，URL 地址必须要添加 mmsync=1 参数！否则加载条数不对



注意，图片的地址后面有很多附加参数，比如 `?w=199&h=180`，在导入图片时一定要移除！否则会影响图片的质量，还有可能导致上传到对象存储的文件包含被转义的特殊字符，引发无法访问等问题。



2、抓取和导入规则

可以在抓取时，让管理员填写以下参数：

- 搜索关键词：便于找到需要的数据
- 抓取数量：单次要抓取的条数，不建议超过 30 条（接口单次返回的图片有限）

后端开发

1、定义请求体

在 `model.dto.picture` 包下新建 `PictureUploadByBatchRequest`：

```
@Data
public class PictureUploadByBatchRequest {
```

```

        private String searchText;

        private Integer count = 10;
    }

```

2、开发服务

1) 引入 [jsoup 库](#)，此处选 v1.15.3 版本，使用的人较多：

```

<dependency>
    <groupId>org.jsoup</groupId>
    <artifactId>jsoup</artifactId>
    <version>1.15.3</version>
</dependency>

```

2) 编写批量抓取和创建图片方法

接口：

```

Integer uploadPictureByBatch(
    PictureUploadByBatchRequest pictureUploadByBatchRequest,
    User loginUser
);

```

实现类：

```

@Override
public int uploadPictureByBatch(PictureUploadByBatchRequest pictureUp
    String searchText = pictureUploadByBatchRequest.getSearchText();

    Integer count = pictureUploadByBatchRequest.getCount();
    ThrowUtils.throwIf(count > 30, ErrorCode.PARAMS_ERROR, "最多 30 条

    String fetchUrl = String.format("https://cn.bing.com/images/async
    Document document;
    try {
        document = Jsoup.connect(fetchUrl).get();
    } catch (IOException e) {
        log.error("获取页面失败", e);
        throw new BusinessException(ErrorCode.OPERATION_ERROR, "获取页
    }
    Element div = document.getElementsByClass("dgControl").first();
    if (ObjUtil.isNull(div)) {
        throw new BusinessException(ErrorCode.OPERATION_ERROR, "获取元
    }
    Elements imgElementList = div.select("img.mimg");
    int uploadCount = 0;

```

```

for (Element imgElement : imgElementList) {
    String fileUrl = imgElement.attr("src");
    if (StrUtil.isBlank(fileUrl)) {
        log.info("当前链接为空，已跳过: {}", fileUrl);
        continue;
    }

    int questionMarkIndex = fileUrl.indexOf("?");
    if (questionMarkIndex > -1) {
        fileUrl = fileUrl.substring(0, questionMarkIndex);
    }

    PictureUploadRequest pictureUploadRequest = new PictureUpload
    try {
        PictureVO pictureVO = this.uploadPicture(fileUrl, picture
        log.info("图片上传成功, id = {}", pictureVO.getId());
        uploadCount++;
    } catch (Exception e) {
        log.error("图片上传失败", e);
        continue;
    }
    if (uploadCount >= count) {
        break;
    }
}
return uploadCount;
}

```

上述代码中，我们添加了很多日志记录和异常处理逻辑，使得单张图片抓取或导入失败时任务还能够继续执行，最终返回创建成功的图片数。

💡 如果抓取的内容数量较多，可以适当使用 `Thread.sleep` 阻塞等待一段时间，减少服务器被封禁的概率。

3、开发接口

在 Controller 中新增接口，注意限制仅管理员可用：

```

@PostMapping("/upload/batch")
@AuthCheck(mustRole = UserConstant.ADMIN_ROLE)
public BaseResponse<Integer> uploadPictureByBatch(
    @RequestBody PictureUploadByBatchRequest pictureUploadByBatch
    HttpServletRequest request
) {
    ThrowUtils.throwIf(pictureUploadByBatchRequest == null, ErrorCode
    User loginUser = userService.getLoginUser(request);
    int uploadCount = pictureService.uploadPictureByBatch(pictureUplo
    return ResultUtils.success(uploadCount);
}

```



4、扩展功能 - 批量设置属性

之前我们导入系统的图片名称都是由对方的 URL 决定的，名称可能乱七八糟，而且不利于我们得知数据是在那一批被导入的。

因此我们可以让管理员在执行任务前指定 **名称前缀**，即导入到系统中的图片名称。比如前缀为“鱼皮”，得到的图片名称就是“鱼皮 1”、“鱼皮 2”。。。

相当于支持抓取和创建图片时批量对某批图片命名，名称前缀默认等于搜索关键词。

下面来开发实现：

1) 给 PictureUploadByBatchRequest 请求包装类补充 namePrefix 参数：

```
private String namePrefix;
```

2) 由于图片名称是在 uploadPicture 方法中传入并设置给 Picture 图片对象的，所以需要给该方法接受的参数 PictureUploadRequest 类中补充 picName 参数：

```
private String picName;
```

3) 修改 uploadPicture 服务方法，在构造入库图片信息时，可以通过 pictureUploadRequest 对象获取到要手动设置的图片名称，而不是完全依赖于解析的结果：

```
Picture picture = new Picture();
picture.setUrl(uploadPictureResult.getUrl());
String picName = uploadPictureResult.getPicName();
if (pictureUploadRequest != null && StrUtil.isNotBlank(pictureUploadRequest.getPicName())) {
    picName = pictureUploadRequest.getPicName();
}
picture.setName(picName);
```



4) 修改批量抓取和导入图片的服务方法

uploadPictureByBatch, 补充图片名称生成逻辑:

```
String namePrefix = pictureUploadByBatchRequest.getNamePrefix();
if (StrUtil.isBlank(namePrefix)) {
    namePrefix = searchText;
}

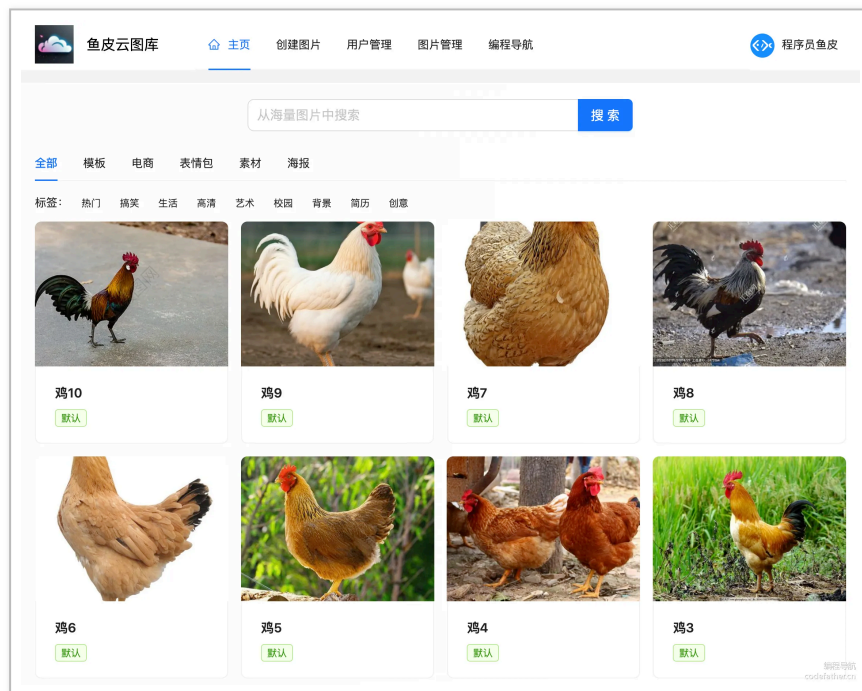
PictureUploadRequest pictureUploadRequest = new PictureUploadRequest(
    if (StrUtil.isNotBlank(namePrefix)) {

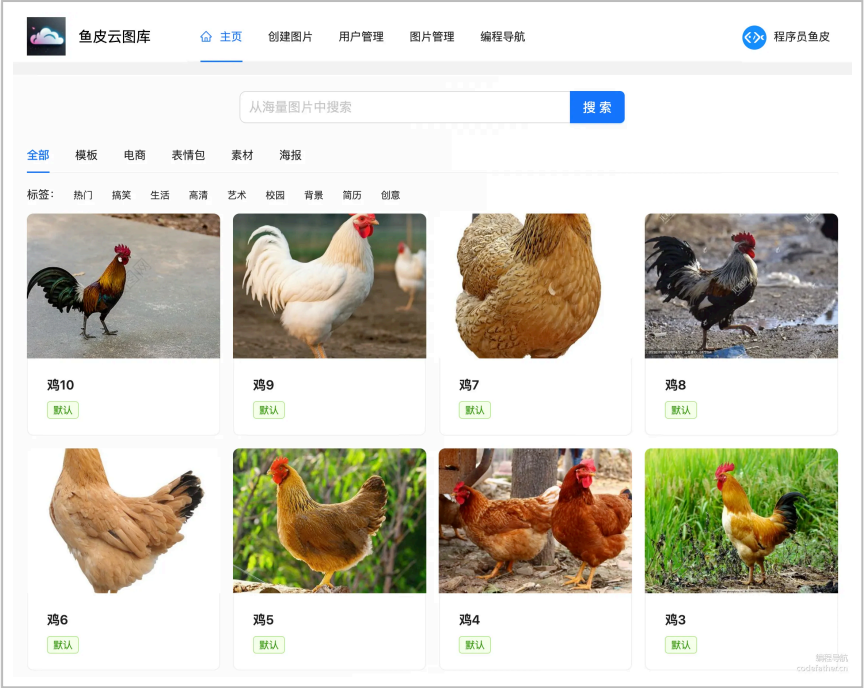
        pictureUploadRequest.setPicName(namePrefix + (uploadCount + 1));
    }
}
```



5、接口测试

可以通过 Swagger 测试批量抓取和创建图片功能, 效果如图:





前端开发

可以新建一个批量创建图片页面，并且在图片管理页面补充跳转到该页面的按钮。

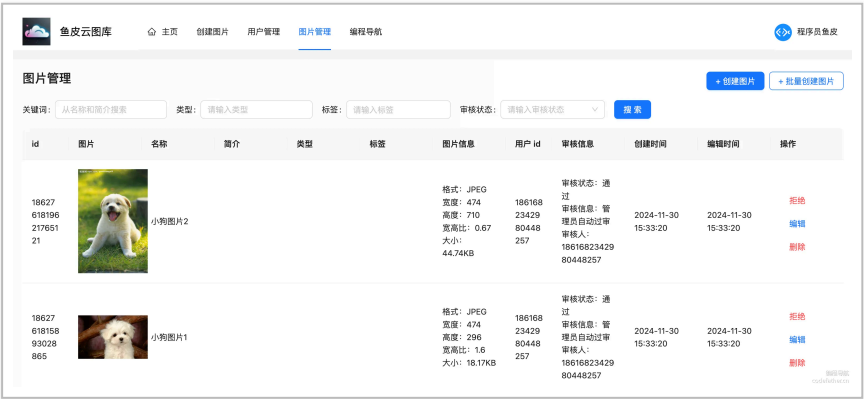
1、图片管理页面补充按钮

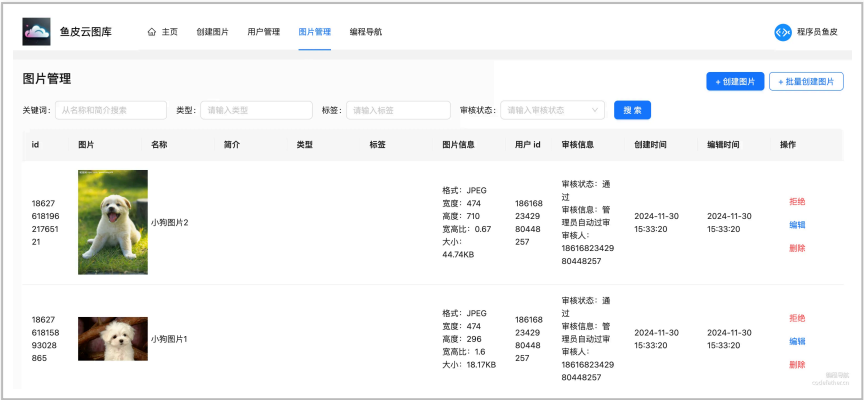
管理页面补充“批量创建图片”按钮，代码如下：

```
<a-space>
  <a-button type="primary" href="/add_picture" target="_blank">+ 创建
  <a-button type="primary" href="/add_picture/batch" target="_blank"
</a-space>
```



效果如图：





2、批量创建图片页面

1) 新建页面文件 `AddPictureBatchPage.vue`（复制创建图片页面），并添加路由：

```
{
  path: '/add_picture/batch',
  name: '批量创建图片',
  component: AddPictureBatchPage,
}
```

正常情况下，普通用户是看不见该页面的，即使看见了，也会因为后端的限制无法使用。

2) 该页面主体是一个表单，和创建图片页面极为相似，先修改表单项：

```
<div>
  <h2>批量创建图片</h2>
  <a-form layout="vertical" :model="formData" @finish="handleSubmit">
    <a-form-item label="关键词" name="searchText">
      <a-input v-model:value="formData.searchText" placeholder="请输入">
    </a-form-item>
    <a-form-item label="抓取数量" name="count">
      <a-input-number
        v-model:value="formData.count"
        placeholder="请输入数量"

        :min="1"
        :max="30"
        allow-clear
      />
    </a-form-item>
    <a-form-item label="名称前缀" name="namePrefix">
      <a-input v-model:value="formData.namePrefix" placeholder="请输入">
    </a-form-item>
    <a-form-item>
      <a-button type="primary" html-type="submit" :loading="loading">
        执行任务
      </a-button>
    </a-form-item>
  </a-form>
</div>
```

```
    </a-form>
  </div>
```

注意，由于批量抓取任务是同步的，可能比较慢，所以需要添加 loading 效果，防止点击过快重复执行。

定义表单项结构和 loading 变量：

```
const formData = reactive<API.PictureUploadByBatchRequest>({
  count: 10,
})
const loading = ref(false)
```

3) 编写提交函数，抓取成功后会输出信息并跳转到主页：

```
const handleSubmit = async (values: any) => {
  loading.value = true;
  const res = await uploadPictureByBatchUsingPost({
    ...formData,
  })
  if (res.data.code === 0 && res.data.data) {
    message.success(`创建成功，共 ${res.data.data} 条`)
    router.push({
      path: '/',
    })
  } else {
    message.error('创建失败，' + res.data.message)
  }
  loading.value = false;
}
```

测试

批量创建页面效果如图：

 鱼皮云图库 [主页](#) [创建图片](#) [...](#)  程序员鱼皮

批量创建图片

关键词

请输入关键词

抓取数量

10

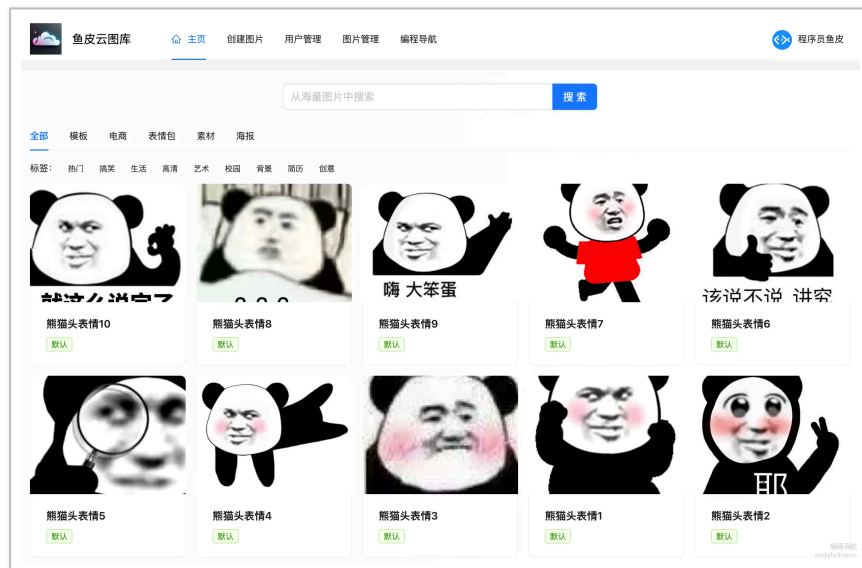
名称前缀

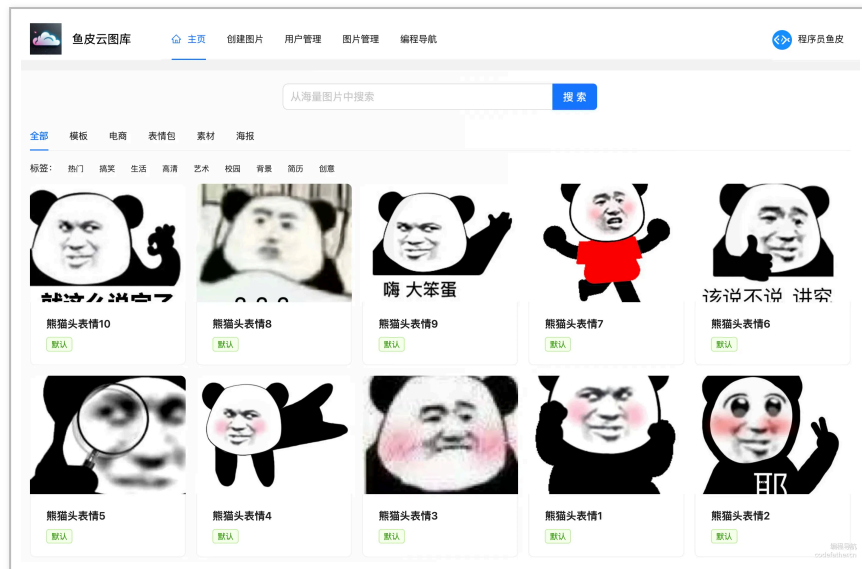
请输入名称前缀，会自动补充序号

执行任务

编程导航
codefather.cn

可以随意输入关键词进行测试，这下爽了，无论你想做表情包网站、设计素材网站、海报网站、Logo 网站、还是壁纸网站，全都可以轻松完成！





友情提示，虽然本项目在功能上的目标是商业级平台，但一定要注意版权问题，不能随意拿别人的素材来商用。

扩展

- 1) 支持管理员填写每批抓取图片的偏移量，防止重复抓取。
- 2) 系统内部记录原始图片 URL，便于内部复盘归档，但是注意不需要暴露给用户。
- 3) 和批量设置名称一样，支持批量设置抓取到的图片的分类和标签等。
- 4) 我们目前抓取到的图片清晰度有限，可以尝试能否获取到质量更高的图片。

最后

讲到这里，好像我们的公开图库平台功能已经比较完善了，现在就可以上线给别人用了嘛！

如果你也是这个想法，那真的是。。。

太惨啦！

现在上线，得亏死啊！在下一节教程中，鱼皮会带大家做一些重要的优化点，加油学习吧！

本文由 简悦 SimpRead 优化，用以提升阅读体验

使用了 全新的简悦词法分析引擎^{beta}，[点击查看详细说明](#)

