

自然写教学资源管理与内容分发系统软件 V1.0

软件著作权鉴别材料 — 源程序

权利人：深圳自然写科技有限公司

版本号：V1.0

源程序目录结构

```
13-writech-resource-platform/  
├─ WritechResourceApplication.java  
├─ controller/  
│   ├── DotCodeController.java  
│   └─ ResourceController.java  
├─ model/  
│   └─ Resource.java  
└─ service/  
    ├── AuditService.java  
    ├── CdnService.java  
    ├── DotCodeService.java  
    ├── ResourceService.java  
    └─ SearchService.java
```

源程序文件清单

(根目录)

WritechResourceApplication.java

```
/*  
 * 自然写教学资源管理与内容分发系统软件 V1.0  
 * WritechResourceApplication.java - Spring Boot启动类与全局配置  
 */  
package com.writech.resource;  
  
import org.springframework.boot.SpringApplication;  
import org.springframework.boot.autoconfigure.SpringBootApplication;
```

```

import org.springframework.boot.context.properties.EnableConfigurationProperties;
import org.springframework.cache.annotation.EnableCaching;
import org.springframework.context.annotation.Bean;
import org.springframework.context.annotation.Configuration;
import org.springframework.scheduling.annotation.EnableAsync;
import org.springframework.scheduling.annotation.EnableScheduling;
import org.springframework.web.servlet.config.annotation.CorsRegistry;
import org.springframework.web.servlet.config.annotation.InterceptorRegistry;
import org.springframework.web.servlet.config.annotation.WebMvcConfigurer;

import javax.annotation.PostConstruct;
import javax.annotation.PreDestroy;
import java.util.TimeZone;
import java.util.logging.Logger;

/**
 * 自然写教学资源管理与内容分发系统启动类
 *
 * 功能概述：
 * - 课件/字帖/试卷模板管理
 * - 点阵码资源生成与管理
 * - 内容审核与版本控制
 * - 多终端资源分发与CDN缓存
 * - 教师自定义内容上传
 * - 按年级/学科/出版社分类检索
 * - 资源使用统计
 */
@SpringBootApplication
@EnableCaching
@EnableAsync
@EnableScheduling
@EnableConfigurationProperties
public class WritetechResourceApplication {

    private static final Logger logger =
        Logger.getLogger(WritetechResourceApplication.class.getName());

    public static void main(String[] args) {
        // 设置默认时区为东八区
        TimeZone.setDefault(TimeZone.getTimeZone("Asia/Shanghai"));
        SpringApplication.run(WritetechResourceApplication.class, args);
        logger.info("自然写资源管理平台已启动");
    }

    @PostConstruct
    public void init() {
        logger.info("资源平台初始化：检查OSS连接、ES索引、CDN配置...");
        // 初始化OSS连接
        // 检查Elasticsearch索引是否存在，不存在则创建
        // 预热CDN缓存配置
    }

    @PreDestroy
    public void cleanup() {
        logger.info("资源平台关闭：释放连接资源...");
    }
}

```

```

/**
 * Web MVC配置：CORS跨域、拦截器
 */
@Configuration
static class WebConfig implements WebMvcConfigurer {

    @Override
    public void addCorsMappings(CorsRegistry registry) {
        registry.addMapping("/api/**")
            .allowedOrigins(
                "https://admin.writech.com",
                "https://teacher.writech.com"
            )
            .allowedMethods("GET", "POST", "PUT", "DELETE")
            .allowedHeaders("*")
            .allowCredentials(true)
            .maxAge(3600);
    }

    @Override
    public void addInterceptors(InterceptorRegistry registry) {
        // 审计日志拦截器：记录所有资源操作
        // registry.addInterceptor(new AuditLogInterceptor())
        //     .addPathPatterns("/api/**");

        // 权限校验拦截器：按学校/区域授权
        // registry.addInterceptor(new PermissionInterceptor())
        //     .addPathPatterns("/api/**")
        //     .excludePathPatterns("/api/v1/health");
    }
}
}

```

controller/

controller/DotCodeController.java

```

/**
 * 自然写教学资源管理与内容分发系统软件 V1.0
 * controller/DotCodeController.java - 点阵码生成API
 * controller/AuditController.java - 内容审核API
 */
package com.writech.resource.controller;

import java.util.*;
import java.util.logging.Logger;

/**
 * 点阵码生成控制器
 *
 * 提供点阵码资源的生成、查询、绑定等API接口。
 * 点阵码是自然写系统的核心技术资源。
 */

```

```

public class DotCodeController {

    private static final Logger logger =
        Logger.getLogger(DotCodeController.class.getName());

    /**
     * 生成点阵码资源包 POST /api/v1/dotcode/generate
     *
     * 为指定资源（字帖/试卷/课件）生成配套的点阵码。
     * 点阵码ID全局唯一分配，生成后可叠加到PDF模板上。
     */
    public Map<String, Object> generateDotCode(Map<String, Object> request) {
        String resourceId = (String) request.get("resource_id");
        int pageCount = (int) request.getOrDefault("page_count", 1);
        double pageWidth = (double) request.getOrDefault("page_width", 210.0);
        double pageHeight = (double) request.getOrDefault("page_height", 297.0);
        boolean overlay = (boolean) request.getOrDefault("overlay_on_template", false);

        logger.info(String.format(
            "点阵码生成请求: resource=%s, pages=%d, size=%.0fx%.0f, overlay=%b",
            resourceId, pageCount, pageWidth, pageHeight, overlay
        ));

        // 参数校验
        if (resourceId == null || resourceId.isEmpty()) {
            Map<String, Object> err = new HashMap<>();
            err.put("code", 400);
            err.put("message", "resource_id不能为空");
            return err;
        }
        if (pageCount < 1 || pageCount > 500) {
            Map<String, Object> err = new HashMap<>();
            err.put("code", 400);
            err.put("message", "页数须在1-500之间");
            return err;
        }

        // 调用点阵码生成服务
        // DotCodeService.DotCodeGenerateRequest genReq = new
        DotCodeService.DotCodeGenerateRequest();
        // genReq.setResourceId(resourceId);
        // genReq.setPageCount(pageCount);
        // DotCodeService.DotCodeGenerateResult result =
        dotCodeService.generateDotCodes(genReq);

        Map<String, Object> result = new HashMap<>();
        result.put("code", 0);
        result.put("message", "点阵码生成成功");
        result.put("data", Map.of(
            "resource_id", resourceId,
            "page_count", pageCount,
            "dot_code_ids", new ArrayList<>(),
            "output_file_url", ""
        ));
        return result;
    }
}

```

```

/**
 * 查询点阵码绑定信息 GET /api/v1/dotcode/binding/{dotCodeId}
 *
 * 根据点阵码ID查询其绑定的资源和页面信息。
 * 用于点阵笔采集到坐标后定位到具体页面。
 */
public Map<String, Object> queryBinding(long dotCodeId) {
    logger.info("查询点阵码绑定: dotCodeId=" + dotCodeId);

    // DotCodeBinding binding = dotCodeService.queryBinding(dotCodeId);
    // if (binding == null) {
    //     return ErrorResponse(404, "点阵码绑定信息不存在");
    // }

    Map<String, Object> result = new HashMap<>();
    result.put("code", 0);
    result.put("data", Map.of(
        "dot_code_id", dotCodeId,
        "resource_id", "",
        "page_index", 0,
        "area_type", "full_page",
        "area", Map.of("x", 0, "y", 0, "width", 210, "height", 297)
    ));
    return result;
}

/**
 * 查询资源关联的所有点阵码 GET /api/v1/dotcode/resource/{resourceId}
 */
public Map<String, Object> queryByResource(String resourceId) {
    logger.info("查询资源点阵码: resource=" + resourceId);

    // List<DotCodeBinding> bindings = dotCodeService.queryByResourceId(resourceId);

    Map<String, Object> result = new HashMap<>();
    result.put("code", 0);
    result.put("data", Map.of(
        "resource_id", resourceId,
        "bindings", new ArrayList<>()
    ));
    return result;
}

/**
 * 撤销点阵码绑定 DELETE /api/v1/dotcode/binding/{dotCodeId}
 *
 * 撤销后该点阵码ID可被重新分配。
 * 仅管理员可执行此操作。
 */
public Map<String, Object> revokeBinding(long dotCodeId, String operatorId) {
    logger.info(String.format(
        "撤销点阵码绑定: dotCodeId=%d, operator=%s", dotCodeId, operatorId
    ));

    // dotCodeService.revokeBinding(dotCodeId);

    Map<String, Object> result = new HashMap<>();

```

```

        result.put("code", 0);
        result.put("message", "点阵码绑定已撤销");
        return result;
    }
}

/**
 * 内容审核控制器
 *
 * 提供资源内容审核的完整流程API:
 * - 待审核资源列表查询
 * - 审核通过/驳回/退回操作
 * - 批量审核
 * - 审核记录查询
 * - 审核统计仪表盘
 */
class AuditController {

    private static final Logger logger =
        Logger.getLogger(AuditController.class.getName());

    /**
     * 获取待审核资源列表 GET /api/v1/resource/audit/pending
     *
     * 按上传时间排序，支持按类型和学科过滤。
     */
    public Map<String, Object> getPendingList(
        String type,
        String subject,
        int page,
        int pageSize
    ) {
        logger.info(String.format(
            "待审核列表: type=%s, subject=%s, page=%d", type, subject, page
        ));

        // 查询MySQL: status = 'PENDING'
        // List<Resource> pending = resourceMapper.selectByStatus("PENDING", type,
subject, page, pageSize);
        // int total = resourceMapper.countByStatus("PENDING", type, subject);

        Map<String, Object> result = new HashMap<>();
        result.put("code", 0);
        result.put("data", Map.of(
            "total", 0,
            "page", page,
            "items", new ArrayList<>()
        ));
        return result;
    }

    /**
     * 执行审核操作 PUT /api/v1/resource/audit/{id}
     *
     * 审核通过后资源自动进入CDN分发，可被终端检索下载。
     * 驳回后通知上传者修改。
     */

```

```

public Map<String, Object> performAudit(
    String resourceId,
    Map<String, Object> auditData
) {
    String action = (String) auditData.get("action");
    String comment = (String) auditData.get("comment");
    String auditorId = (String) auditData.get("auditor_id");

    logger.info(String.format(
        "审核操作: resource=%s, action=%s, auditor=%s",
        resourceId, action, auditorId
    ));

    // 校验审核动作合法性
    Set<String> validActions = new HashSet<>(Arrays.asList(
        "APPROVE", "REJECT", "RETURN"
    ));
    if (!validActions.contains(action)) {
        Map<String, Object> err = new HashMap<>();
        err.put("code", 400);
        err.put("message", "不合法的审核操作: " + action);
        return err;
    }

    // 调用审核服务
    // AuditService.AuditRequest req = new AuditService.AuditRequest();
    // req.setResourceId(resourceId);
    // req.setAction(AuditService.AuditAction.valueOf(action));
    // req.setComment(comment);
    // req.setAuditorId(auditorId);
    // return auditService.performAudit(req);

    Map<String, Object> result = new HashMap<>();
    result.put("code", 0);
    result.put("message", "审核操作成功");
    result.put("data", Map.of(
        "resource_id", resourceId,
        "action", action,
        "new_status", "APPROVE".equals(action) ? "APPROVED" : "REJECTED"
    ));
    return result;
}

/**
 * 批量审核 POST /api/v1/resource/audit/batch
 */
public Map<String, Object> batchAudit(Map<String, Object> batchRequest) {
    List<String> resourceIds = (List<String>) batchRequest.get("resource_ids");
    String action = (String) batchRequest.get("action");
    String comment = (String) batchRequest.get("comment");
    String auditorId = (String) batchRequest.get("auditor_id");

    logger.info(String.format(
        "批量审核: count=%d, action=%s", resourceIds.size(), action
    ));

    // return auditService.batchAudit(resourceIds, action, comment, auditorId);
}

```

```

        Map<String, Object> result = new HashMap<>();
        result.put("code", 0);
        result.put("data", Map.of(
            "total", resourceIds.size(),
            "success", resourceIds.size(),
            "failed", 0
        ));
        return result;
    }

    /**
     * 查询审核记录 GET /api/v1/resource/audit/records
     */
    public Map<String, Object> getAuditRecords(
        String resourceId,
        String auditorId,
        int page,
        int pageSize
    ) {
        logger.info(String.format(
            "审核记录查询: resource=%s, auditor=%s", resourceId, auditorId
        ));

        // return auditService.queryAuditRecords(resourceId, auditorId, page, pageSize);

        Map<String, Object> result = new HashMap<>();
        result.put("code", 0);
        result.put("data", Map.of(
            "total", 0,
            "page", page,
            "items", new ArrayList<>()
        ));
        return result;
    }

    /**
     * 审核统计仪表盘 GET /api/v1/resource/audit/stats
     *
     * 返回待审核数量、今日已审核数量、通过率等统计。
     */
    public Map<String, Object> getAuditStats() {
        // return auditService.getAuditStats();

        Map<String, Object> result = new HashMap<>();
        result.put("code", 0);
        result.put("data", Map.of(
            "pending_count", 0,
            "approved_today", 0,
            "rejected_today", 0,
            "approval_rate", 0.0,
            "avg_audit_hours", 0.0
        ));
        return result;
    }
}

```


controller/ResourceController.java

```
/*
 * 自然写教学资源管理与内容分发系统软件 V1.0
 * controller/ResourceController.java - 资源CRUD与检索API
 */
package com.writech.resource.controller;

import java.util.*;
import java.util.logging.Logger;

/**
 * 资源管理控制器
 *
 * 提供教学资源（课件、字帖、试卷、模板）的增删改查接口，
 * 支持按年级/学科/出版社分类检索（Elasticsearch全文检索），
 * CDN签名URL下载，教师自定义上传，版本管理等功能。
 */
public class ResourceController {

    private static final Logger logger =
        Logger.getLogger(ResourceController.class.getName());

    // =====
    // 数据模型
    // =====

    /** 资源类型枚举 */
    public enum ResourceType {
        COURSEWARE,    // 课件 (PPT/PDF)
        COPYBOOK,      // 字帖模板
        EXAM_PAPER,    // 试卷
        TEMPLATE,      // 通用模板
        DOT_CODE,      // 点阵码资源
        VIDEO,         // 教学视频
        AUDIO,         // 音频资料
        IMAGE          // 图片素材
    }

    /** 审核状态枚举 */
    public enum AuditStatus {
        PENDING,       // 待审核
        APPROVED,      // 已通过
        REJECTED,      // 已驳回
        WITHDRAWN      // 已撤回
    }

    /** 资源元数据模型（对应MySQL resource表） */
    public static class ResourceMetadata {
        private String id;
        private String name;
        private String description;
        private ResourceType type;
        private String subject;    // 学科
        private String grade;     // 年级
    }
}
```

```

private String publisher;    // 出版社
private String version;     // 版本号
private AuditStatus auditStatus;
private String fileKey;     // OSS文件Key
private long fileSize;      // 文件大小 (字节)
private String mimeType;    // MIME类型
private String thumbnailUrl; // 缩略图URL
private String uploaderId;  // 上传者ID
private String uploaderName; // 上传者姓名
private String schoolId;    // 所属学校
private String tags;        // 标签 (逗号分隔)
private int downloadCount;  // 下载次数
private int useCount;       // 使用次数
private Date createdAt;
private Date updatedAt;

// Getter/Setter
public String getId() { return id; }
public void setId(String id) { this.id = id; }
public String getName() { return name; }
public void setName(String name) { this.name = name; }
public String getDescription() { return description; }
public void setDescription(String desc) { this.description = desc; }
public ResourceType getType() { return type; }
public void setType(ResourceType type) { this.type = type; }
public String getSubject() { return subject; }
public void setSubject(String subject) { this.subject = subject; }
public String getGrade() { return grade; }
public void setGrade(String grade) { this.grade = grade; }
public String getPublisher() { return publisher; }
public void setPublisher(String publisher) { this.publisher = publisher; }
public AuditStatus getAuditStatus() { return auditStatus; }
public void setAuditStatus(AuditStatus s) { this.auditStatus = s; }
public String getFileKey() { return fileKey; }
public void setFileKey(String key) { this.fileKey = key; }
public long getFileSize() { return fileSize; }
public void setFileSize(long size) { this.fileSize = size; }
public String getSchoolId() { return schoolId; }
public void setSchoolId(String schoolId) { this.schoolId = schoolId; }
public int getDownloadCount() { return downloadCount; }
public int getUseCount() { return useCount; }
public Date getCreatedAt() { return createdAt; }
public Date getUpdatedAt() { return updatedAt; }
}

/** 分类目录树节点 */
public static class CategoryNode {
    private String id;
    private String name;
    private String parentId;
    private int level;        // 层级 (1=年级, 2=学科, 3=出版社)
    private int sortOrder;
    private List<CategoryNode> children;

    public CategoryNode(String id, String name, String parentId, int level) {
        this.id = id;
        this.name = name;
    }
}

```

```

        this.parentId = parentId;
        this.level = level;
        this.children = new ArrayList<>();
    }

    public String getId() { return id; }
    public String getName() { return name; }
    public List<CategoryNode> getChildren() { return children; }
    public void addChild(CategoryNode child) { children.add(child); }
}

/** 资源搜索请求 */
public static class SearchRequest {
    private String keyword;        // 搜索关键词
    private String subject;        // 学科过滤
    private String grade;          // 年级过滤
    private String publisher;      // 出版社过滤
    private ResourceType type;     // 资源类型过滤
    private String schoolId;       // 学校授权范围
    private int page;
    private int pageSize;
    private String sortBy;         // 排序字段
    private String sortOrder;     // ASC/DESC

    public String getKeyword() { return keyword; }
    public void setKeyword(String kw) { this.keyword = kw; }
    public String getSubject() { return subject; }
    public void setSubject(String s) { this.subject = s; }
    public String getGrade() { return grade; }
    public void setGrade(String g) { this.grade = g; }
    public String getPublisher() { return publisher; }
    public void setPublisher(String p) { this.publisher = p; }
    public ResourceType getType() { return type; }
    public void setType(ResourceType t) { this.type = t; }
    public int getPage() { return page > 0 ? page : 1; }
    public int getPageSize() { return pageSize > 0 ? Math.min(pageSize, 100) : 20; }
}

/** 搜索结果 */
public static class SearchResult {
    private long total;
    private int page;
    private List<ResourceMetadata> items;
    private Map<String, List<Map<String, Object>>> facets; // 聚合面

    public SearchResult(long total, int page, List<ResourceMetadata> items) {
        this.total = total;
        this.page = page;
        this.items = items;
    }

    public long getTotal() { return total; }
    public List<ResourceMetadata> getItems() { return items; }
    public void setFacets(Map<String, List<Map<String, Object>>> f) { this.facets =
f; }
}

```

```
// =====
//  API接口实现
// =====

/**
 * 资源检索接口 GET /api/v1/resource/search
 *
 * 使用Elasticsearch进行全文检索，支持：
 * - 关键词匹配（资源名称、描述、标签）
 * - 多条件组合过滤（年级+学科+出版社+类型）
 * - 聚合面统计（各分类维度的资源数量）
 * - 分页排序
 *
 * 权限控制：教师仅可搜索本校已授权资源
 */
public Map<String, Object> searchResources(SearchRequest request) {
    logger.info(String.format(
        "资源检索: keyword=%s, subject=%s, grade=%s, publisher=%s",
        request.getKeyword(), request.getSubject(),
        request.getGrade(), request.getPublisher()
    ));

    // 构建Elasticsearch查询
    // BoolQueryBuilder boolQuery = QueryBuilders.boolQuery();

    // 关键词全文匹配（multi_match查询名称+描述+标签）
    // if (request.getKeyword() != null && !request.getKeyword().isEmpty()) {
    //     boolQuery.must(QueryBuilders.multiMatchQuery(
    //         request.getKeyword(), "name", "description", "tags"
    //     ).type(MultiMatchQueryBuilder.Type.BEST_FIELDS));
    // }

    // 条件过滤
    // if (request.getSubject() != null) {
    //     boolQuery.filter(QueryBuilders.termQuery("subject",
request.getSubject()));
    // }
    // if (request.getGrade() != null) {
    //     boolQuery.filter(QueryBuilders.termQuery("grade", request.getGrade()));
    // }
    // if (request.getPublisher() != null) {
    //     boolQuery.filter(QueryBuilders.termQuery("publisher",
request.getPublisher()));
    // }
    // if (request.getType() != null) {
    //     boolQuery.filter(QueryBuilders.termQuery("type",
request.getType().name()));
    // }

    // 学校授权过滤（仅返回该校已授权的资源）
    // boolQuery.filter(QueryBuilders.termQuery("school_id",
request.getSchoolId()));

    // 仅返回审核通过的资源
    // boolQuery.filter(QueryBuilders.termQuery("audit_status", "APPROVED"));

    // 聚合统计（按学科/年级/出版社/类型分组计数）

```

```

        // AggregationBuilder subjectAgg =
AggregationBuilders.terms("by_subject").field("subject");
        // AggregationBuilder gradeAgg =
AggregationBuilders.terms("by_grade").field("grade");

        // 执行搜索
        // SearchResponse response = elasticsearchClient.search(searchRequest);

        Map<String, Object> result = new HashMap<>();
        result.put("code", 0);
        result.put("message", "success");
        result.put("data", new SearchResult(0, request.getPage(), new ArrayList<>()));
        return result;
    }

    /**
     * 资源下载接口 GET /api/v1/resource/download/{id}
     *
     * 生成CDN签名URL返回给客户端，签名URL有效期30分钟。
     * 同时记录下载次数，用于使用统计。
     *
     * 安全措施：
     * - Referrer校验：仅允许来自writech.com域名的请求
     * - 签名URL：包含过期时间和HMAC签名，防盗链
     * - 数字水印：下载时可选添加水印（包含学校/教师标识）
     */
    public Map<String, Object> downloadResource(String resourceId, String userId) {
        logger.info(String.format("资源下载: id=%s, user=%s", resourceId, userId));

        // 查询资源元数据
        // ResourceMetadata resource = resourceMapper.selectById(resourceId);
        // if (resource == null) {
        //     return ErrorResponse(404, "资源不存在");
        // }

        // 权限校验：检查用户是否有权访问该资源
        // if (!permissionService.canAccess(userId, resource.getSchoolId())) {
        //     return ErrorResponse(403, "无权访问此资源");
        // }

        // 生成CDN签名下载URL
        // String signedUrl = cdnService.generateSignedUrl(
        //     resource.getFileKey(),
        //     30 * 60 // 有效期30分钟
        // );

        // 异步更新下载计数
        // asyncUpdateDownloadCount(resourceId);

        Map<String, Object> result = new HashMap<>();
        result.put("code", 0);
        result.put("data", Map.of(
            "resource_id", resourceId,
            "download_url", "",
            "expires_in", 1800,
            "file_name", "",
            "file_size", 0
        ));
    }

```

```

    ));
    return result;
}

/**
 * 教师上传资源接口 POST /api/v1/resource/upload
 *
 * 教师可上传自定义教学资源，上传后状态为PENDING（待审核），
 * 需管理员审核通过后才可被其他教师检索和使用。
 *
 * 上传流程：
 * 1. 前端分片上传到OSS（使用STS临时凭证）
 * 2. 上传完成后调用此接口创建资源元数据
 * 3. 系统自动生成缩略图
 * 4. 同步索引到Elasticsearch
 */
public Map<String, Object> uploadResource(Map<String, Object> uploadRequest) {
    String name = (String) uploadRequest.get("name");
    String description = (String) uploadRequest.get("description");
    String fileKey = (String) uploadRequest.get("file_key");
    String subject = (String) uploadRequest.get("subject");
    String grade = (String) uploadRequest.get("grade");
    String typeStr = (String) uploadRequest.get("type");

    logger.info(String.format(
        "教师上传资源: name=%s, subject=%s, grade=%s, type=%s",
        name, subject, grade, typeStr
    ));

    // 参数校验
    if (name == null || name.trim().isEmpty()) {
        Map<String, Object> err = new HashMap<>();
        err.put("code", 400);
        err.put("message", "资源名称不能为空");
        return err;
    }

    // 创建资源元数据记录（状态为PENDING待审核）
    ResourceMetadata resource = new ResourceMetadata();
    resource.setId(UUID.randomUUID().toString().replace("-", ""));
    resource.setName(name);
    resource.setDescription(description);
    resource.setSubject(subject);
    resource.setGrade(grade);
    resource.setFileKey(fileKey);
    resource.setAuditStatus(AuditStatus.PENDING);

    // 插入MySQL
    // resourceMapper.insert(resource);

    // 异步生成缩略图
    // asyncGenerateThumbnail(resource.getId(), fileKey);

    // 同步到Elasticsearch索引
    // elasticsearchService.indexResource(resource);

    Map<String, Object> result = new HashMap<>();

```

```

        result.put("code", 0);
        result.put("message", "上传成功, 等待审核");
        result.put("data", Map.of("resource_id", resource.getId()));
        return result;
    }

    /**
     * 获取资源版本历史 GET /api/v1/resource/versions/{id}
     *
     * 返回资源的所有历史版本列表, 支持查看和回滚。
     */
    public Map<String, Object> getResourceVersions(String resourceId) {
        logger.info("查询资源版本: id=" + resourceId);

        // 查询版本历史
        // List<ResourceVersion> versions =
        versionMapper.selectByResourceId(resourceId);

        Map<String, Object> result = new HashMap<>();
        result.put("code", 0);
        result.put("data", Map.of(
            "resource_id", resourceId,
            "versions", new ArrayList<>()
        ));
        return result;
    }

    /**
     * 获取分类目录树 GET /api/v1/resource/categories
     *
     * 返回三级分类目录树: 年级 → 学科 → 出版社
     */
    public Map<String, Object> getCategoryTree() {
        // 从MySQL查询分类数据并构建树形结构
        // List<CategoryNode> roots = buildCategoryTree();

        Map<String, Object> result = new HashMap<>();
        result.put("code", 0);
        result.put("data", new ArrayList<CategoryNode>());
        return result;
    }

    /**
     * 获取资源使用统计 GET /api/v1/stat/resource/{id}
     *
     * 从ClickHouse查询资源的使用统计数据 (下载量、使用次数、终端分布)
     */
    public Map<String, Object> getResourceStats(String resourceId) {
        logger.info("资源统计查询: id=" + resourceId);

        // 从ClickHouse查询使用统计
        // ResourceStats stats = clickhouseClient.queryResourceStats(resourceId);

        Map<String, Object> result = new HashMap<>();
        result.put("code", 0);
        result.put("data", Map.of(
            "resource_id", resourceId,

```

```

        "download_count", 0,
        "use_count", 0,
        "terminal_distribution", Map.of(
            "pad", 0, "pc", 0, "mobile", 0, "board", 0
        ),
        "daily_trend", new ArrayList<>()
    ));
    return result;
}
}

```

model/

model/Resource.java

```

/*
 * 自然写教学资源管理与内容分发系统软件 V1.0
 * model/Resource.java - 资源数据模型
 * model/DotPattern.java - 点阵码模型
 * model/AuditRecord.java - 审核记录模型
 * config/OssConfig.java - OSS对象存储配置
 * config/ElasticsearchConfig.java - ES配置
 * security/ResourceSecurity.java - 资源安全（防盗链+水印）
 */
package com.writech.resource.model;

import java.util.*;

/**
 * 资源数据模型（对应MySQL resource表）
 */
public class Resource {

    /** 资源ID (UUID) */
    private String id;

    /** 资源名称 */
    private String name;

    /** 资源描述 */
    private String description;

    /** 资源类型 (COURSEWARE/COPYBOOK/EXAM_PAPER/TEMPLATE/DOT_CODE/VIDEO) */
    private String type;

    /** 学科 */
    private String subject;

    /** 适用年级 */
    private String grade;

    /** 出版社 */
    private String publisher;

```



```
/** 版本号 */
private String version;

/** 审核状态 (PENDING/APPROVED/REJECTED/WITHDRAWN) */
private String auditStatus;

/** OSS文件存储Key */
private String fileKey;

/** 文件大小 (字节) */
private long fileSize;

/** MIME类型 */
private String mimeType;

/** 缩略图URL */
private String thumbnailUrl;

/** 上传者ID */
private String uploaderId;

/** 上传者姓名 */
private String uploaderName;

/** 所属学校ID */
private String schoolId;

/** 标签 (逗号分隔) */
private String tags;

/** 下载次数 */
private int downloadCount;

/** 使用次数 */
private int useCount;

/** 创建时间 */
private Date createdAt;

/** 更新时间 */
private Date updatedAt;

/** 是否已删除 (软删除标记) */
private boolean deleted;

// =====
//  Getter / Setter
//  =====

public String getId() { return id; }
public void setId(String id) { this.id = id; }
public String getName() { return name; }
public void setName(String name) { this.name = name; }
public String getDescription() { return description; }
public void setDescription(String description) { this.description = description; }
public String getType() { return type; }
```

```

    public void setType(String type) { this.type = type; }
    public String getSubject() { return subject; }
    public void setSubject(String subject) { this.subject = subject; }
    public String getGrade() { return grade; }
    public void setGrade(String grade) { this.grade = grade; }
    public String getPublisher() { return publisher; }
    public void setPublisher(String publisher) { this.publisher = publisher; }
    public String getVersion() { return version; }
    public void setVersion(String version) { this.version = version; }
    public String getAuditStatus() { return auditStatus; }
    public void setAuditStatus(String auditStatus) { this.auditStatus = auditStatus; }
    public String getFileKey() { return fileKey; }
    public void setFileKey(String fileKey) { this.fileKey = fileKey; }
    public long getFileSize() { return fileSize; }
    public void setFileSize(long fileSize) { this.fileSize = fileSize; }
    public String getMimeType() { return mimeType; }
    public void setMimeType(String mimeType) { this.mimeType = mimeType; }
    public String getThumbnailUrl() { return thumbnailUrl; }
    public void setThumbnailUrl(String thumbnailUrl) { this.thumbnailUrl = thumbnailUrl;
}

    public String getUploaderId() { return uploaderId; }
    public void setUploaderId(String uploaderId) { this.uploaderId = uploaderId; }
    public String getSchoolId() { return schoolId; }
    public void setSchoolId(String schoolId) { this.schoolId = schoolId; }
    public String getTags() { return tags; }
    public void setTags(String tags) { this.tags = tags; }
    public int getDownloadCount() { return downloadCount; }
    public int getUseCount() { return useCount; }
    public Date getCreatedAt() { return createdAt; }
    public Date getUpdatedAt() { return updatedAt; }
    public boolean isDeleted() { return deleted; }
    public void setDeleted(boolean deleted) { this.deleted = deleted; }

    @Override
    public String toString() {
        return "Resource{id='" + id + "', name='" + name + "', type='" + type +
            "', subject='" + subject + "', grade='" + grade + "'}";
    }
}

/**
 * 点阵码模型 (对应MySQL dot_pattern表 + OSS文件)
 */
class DotPattern {

    /** 点阵码ID (全局唯一) */
    private long dotCodeId;

    /** 关联的资源ID */
    private String resourceId;

    /** 页面序号 */
    private int pageIndex;

    /** 区域类型 */
    private String areaType;

```

```

/** 区域坐标和尺寸 (mm) */
private double areaX;
private double areaY;
private double areaWidth;
private double areaHeight;

/** 点阵码图案文件OSS Key */
private String patternFileKey;

/** 生成参数JSON */
private String generateParams;

/** 状态 (ACTIVE/REVOKED) */
private String status;

/** 创建时间 */
private Date createdAt;

public long getDotCodeId() { return dotCodeId; }
public void setDotCodeId(long id) { this.dotCodeId = id; }
public String getResourceId() { return resourceId; }
public void setResourceId(String rid) { this.resourceId = rid; }
public int getPageIndex() { return pageIndex; }
public void setPageIndex(int idx) { this.pageIndex = idx; }
public String getAreaType() { return areaType; }
public void setAreaType(String type) { this.areaType = type; }
public double getAreaX() { return areaX; }
public double getAreaY() { return areaY; }
public double getAreaWidth() { return areaWidth; }
public double getAreaHeight() { return areaHeight; }
public String getPatternFileKey() { return patternFileKey; }
public String getStatus() { return status; }
public void setStatus(String status) { this.status = status; }
public Date getCreatedAt() { return createdAt; }
}

/**
 * 审核记录模型 (对应MySQL audit_record表)
 */
class AuditRecord {

    /** 记录ID */
    private String id;

    /** 关联的资源ID */
    private String resourceId;

    /** 审核人ID */
    private String auditorId;

    /** 审核人姓名 */
    private String auditorName;

    /** 审核操作 (APPROVE/REJECT/RETURN/WITHDRAW) */
    private String action;

    /** 审核意见 */

```

```

private String comment;

/** 审核前状态 */
private String preStatus;

/** 审核后状态 */
private String postStatus;

/** 审核时间 */
private Date createdAt;

public String getId() { return id; }
public void setId(String id) { this.id = id; }
public String getResourceId() { return resourceId; }
public void setResourceId(String rid) { this.resourceId = rid; }
public String getAuditorId() { return auditorId; }
public void setAuditorId(String aid) { this.auditorId = aid; }
public String getAction() { return action; }
public void setAction(String action) { this.action = action; }
public String getComment() { return comment; }
public void setComment(String comment) { this.comment = comment; }
public String getPreStatus() { return preStatus; }
public String getPostStatus() { return postStatus; }
public Date getCreatedAt() { return createdAt; }
}

/**
 * OSS对象存储配置
 *
 * 多副本冗余存储（99.99%数据持久性），
 * 支持STS临时凭证直传，生命周期管理。
 */
class OssConfig {

    /** OSS区域端点 */
    private String endpoint;

    /** 存储桶名称 */
    private String bucketName;

    /** AccessKey ID */
    private String accessKeyId;

    /** AccessKey Secret（加密存储） */
    private String accessKeySecret;

    /** STS角色ARN（用于前端直传临时授权） */
    private String stsRoleArn;

    /** STS会话名称 */
    private String stsSessionName;

    /** 资源前缀路径 */
    private String resourcePrefix;

    /** 缩略图前缀路径 */
    private String thumbnailPrefix;

```

```

/** 临时文件过期天数 */
private int tempFileExpireDays;

public OssConfig() {
    this.endpoint = "https://oss-cn-hangzhou.aliyuncs.com";
    this.bucketName = "writech-resources";
    this.resourcePrefix = "resources/";
    this.thumbnailPrefix = "thumbnails/";
    this.tempFileExpireDays = 7;
    this.stsSessionName = "writech-upload-session";
}

public String getEndpoint() { return endpoint; }
public void setEndpoint(String ep) { this.endpoint = ep; }
public String getBucketName() { return bucketName; }
public void setBucketName(String name) { this.bucketName = name; }
public String getAccessKeyId() { return accessKeyId; }
public void setAccessKeyId(String id) { this.accessKeyId = id; }
public String getAccessKeySecret() { return accessKeySecret; }
public void setAccessKeySecret(String secret) { this.accessKeySecret = secret; }
public String getStsRoleArn() { return stsRoleArn; }
public void setStsRoleArn(String arn) { this.stsRoleArn = arn; }
public String getResourcePrefix() { return resourcePrefix; }
public String getThumbnailPrefix() { return thumbnailPrefix; }
public int getTempFileExpireDays() { return tempFileExpireDays; }
}

/**
 * Elasticsearch配置
 *
 * ES集群部署，索引按学科/年级分片，
 * 支持IK中文分词器。
 */
class ElasticsearchConfig {

    /** ES集群节点列表 */
    private List<String> nodes;

    /** 连接超时（毫秒） */
    private int connectTimeout;

    /** 读取超时（毫秒） */
    private int socketTimeout;

    /** 索引名称 */
    private String indexName;

    /** 分片数 */
    private int shards;

    /** 副本数 */
    private int replicas;

    /** 用户名（X-Pack安全） */
    private String username;

```

```

/** 密码 */
private String password;

public ElasticsearchConfig() {
    this.nodes = Arrays.asList("localhost:9200");
    this.connectTimeout = 5000;
    this.socketTimeout = 30000;
    this.indexName = "writech_resources";
    this.shards = 3;
    this.replicas = 1;
}

public List<String> getNodes() { return nodes; }
public void setNodes(List<String> nodes) { this.nodes = nodes; }
public int getConnectTimeout() { return connectTimeout; }
public int getSocketTimeout() { return socketTimeout; }
public String getIndexName() { return indexName; }
public int getShards() { return shards; }
public int getReplicas() { return replicas; }
public String getUsername() { return username; }
public void setUsername(String u) { this.username = u; }
public String getPassword() { return password; }
public void setPassword(String p) { this.password = p; }
}

/**
 * 资源安全服务
 *
 * 负责:
 * - 防盗链 (Referer校验 + 签名URL)
 * - 数字水印 (PDF/图片添加学校教师标识水印)
 * - 权限控制 (按学校/区域授权)
 * - 点阵码安全 (全局唯一分配防冲突)
 */
class ResourceSecurity {

    /** 防盗链Referer白名单 */
    private static final Set<String> REFERER_WHITELIST = new HashSet<>(Arrays.asList(
        "/*.writech.com",
        "localhost"
    ));

    /**
     * 校验资源访问权限
     *
     * 规则:
     * - 管理员: 可访问本校所有资源
     * - 教师: 可访问本校已授权资源
     * - 学生/家长: 仅可访问已分配的资源
     */
    public boolean checkPermission(
        String userId,
        String userRole,
        String userSchoolId,
        String resourceSchoolId
    ) {
        // 超级管理员无限制

```

```

        if ("super_admin".equals(userRole)) {
            return true;
        }

        // 校级管理员和教师：必须同校
        if ("admin".equals(userRole) || "teacher".equals(userRole)) {
            return userSchoolId != null && userSchoolId.equals(resourceSchoolId);
        }

        // 学生/家长：需要额外的资源分配记录校验
        // return resourceAssignmentMapper.isAssigned(userId, resourceId);
        return false;
    }

    /**
     * 验证Referer防盗链
     */
    public boolean validateReferer(String referer) {
        if (referer == null || referer.isEmpty()) {
            return false;
        }
        for (String pattern : REFERER_WHITELIST) {
            if (pattern.startsWith("*.")) {
                String domain = pattern.substring(2);
                if (referer.contains(domain)) return true;
            } else {
                if (referer.contains(pattern)) return true;
            }
        }
        return false;
    }

    /**
     * 生成水印文本
     * 格式：学校名称 + 教师姓名 + 日期
     */
    public String generateWatermarkText(
        String schoolName, String teacherName
    ) {
        String dateStr = new java.text.SimpleDateFormat("yyyy-MM-dd")
            .format(new Date());
        return String.format("%s %s %s", schoolName, teacherName, dateStr);
    }
}

```

service/

service/AuditService.java

```

/**
 * 自然写教学资源管理与内容分发系统软件 V1.0
 * service/AuditService.java - 内容审核服务
 */

```

```

package com.writech.resource.service;

import java.util.*;
import java.util.logging.Logger;

/**
 * 内容审核服务
 *
 * 教师上传的资源需经过管理员审核后才能被其他用户检索和使用。
 * 审核流程支持：
 * - 自动预审（AI内容安全检测）
 * - 人工审核（管理员审核通过/驳回/退回修改）
 * - 审核记录全程留痕
 * - 批量审核
 */
public class AuditService {

    private static final Logger logger =
        Logger.getLogger(AuditService.class.getName());

    // =====
    // 审核数据模型
    // =====

    /** 审核操作类型 */
    public enum AuditAction {
        APPROVE,    // 审核通过
        REJECT,     // 驳回
        RETURN,     // 退回修改
        WITHDRAW    // 上传者撤回
    }

    /** 审核记录（对应MySQL audit_record表） */
    public static class AuditRecord {
        private String id;
        private String resourceId;
        private String resourceName;
        private String auditorId;        // 审核人ID
        private String auditorName;      // 审核人姓名
        private AuditAction action;
        private String comment;          // 审核意见
        private String preStatus;        // 审核前状态
        private String postStatus;       // 审核后状态
        private Date createdAt;

        // Getter/Setter
        public String getId() { return id; }
        public void setId(String id) { this.id = id; }
        public String getResourceId() { return resourceId; }
        public void setResourceId(String rid) { this.resourceId = rid; }
        public String getResourceName() { return resourceName; }
        public void setResourceName(String name) { this.resourceName = name; }
        public String getAuditorId() { return auditorId; }
        public void setAuditorId(String id) { this.auditorId = id; }
        public AuditAction getAction() { return action; }
        public void setAction(AuditAction action) { this.action = action; }
        public String getComment() { return comment; }
    }
}

```



```

        public void setComment(String comment) { this.comment = comment; }
        public Date getCreatedAt() { return createdAt; }
    }

    /** 审核请求 */
    public static class AuditRequest {
        private String resourceId;
        private AuditAction action;
        private String comment;
        private String auditorId;

        public String getResourceId() { return resourceId; }
        public void setResourceId(String id) { this.resourceId = id; }
        public AuditAction getAction() { return action; }
        public void setAction(AuditAction action) { this.action = action; }
        public String getComment() { return comment; }
        public void setComment(String c) { this.comment = c; }
        public String getAuditorId() { return auditorId; }
        public void setAuditorId(String id) { this.auditorId = id; }
    }

    /** 自动预审结果 */
    public static class PreAuditResult {
        private boolean safe;
        private double safeScore;          // 安全评分 (0-1)
        private List<String> warnings;     // 警告信息
        private String category;           // 内容分类

        public PreAuditResult(boolean safe, double score) {
            this.safe = safe;
            this.safeScore = score;
            this.warnings = new ArrayList<>();
        }

        public boolean isSafe() { return safe; }
        public double getSafeScore() { return safeScore; }
        public List<String> getWarnings() { return warnings; }
        public void addWarning(String w) { this.warnings.add(w); }
    }

    // =====
    // 审核业务方法
    // =====

    /**
     * 执行审核操作 PUT /api/v1/resource/audit/{id}
     *
     * @param request 审核请求
     * @return 审核结果
     */
    public Map<String, Object> performAudit(AuditRequest request) {
        logger.info(String.format(
            "执行审核: resource=%s, action=%s, auditor=%s",
            request.getResourceId(), request.getAction(), request.getAuditorId()
        ));

        // 查询资源当前状态

```

```

        // ResourceMetadata resource =
resourceMapper.selectById(request.getResourceId());
        // if (resource == null) {
        //     return ErrorResponse(404, "资源不存在");
        // }

        // 状态机校验：只有PENDING状态可被审核
        // if (resource.getAuditStatus() != AuditStatus.PENDING) {
        //     return ErrorResponse(400, "当前状态不可审核");
        // }

        // 创建审核记录
        AuditRecord record = new AuditRecord();
        record.setId(UUID.randomUUID().toString().replace("-", ""));
        record.setResourceId(request.getResourceId());
        record.setAuditorId(request.getAuditorId());
        record.setAction(request.getAction());
        record.setComment(request.getComment());
        record.setPreStatus("PENDING");

        // 根据审核动作更新资源状态
        String newStatus;
        switch (request.getAction()) {
            case APPROVE:
                newStatus = "APPROVED";
                // 审核通过后，同步更新Elasticsearch索引状态
                // updateEsAuditStatus(request.getResourceId(), "APPROVED");
                // 预热CDN缓存（使资源可被终端下载）
                // cdnService.preheatResource(request.getResourceId());
                break;
            case REJECT:
                newStatus = "REJECTED";
                break;
            case RETURN:
                newStatus = "PENDING"; // 退回修改后重新提交
                break;
            default:
                newStatus = "PENDING";
        }

        record.setPostStatus(newStatus);

        // 持久化
        // auditRecordMapper.insert(record);
        // resourceMapper.updateAuditStatus(request.getResourceId(), newStatus);

        // 通知上传者审核结果（消息推送）
        // notifyUploader(request.getResourceId(), request.getAction(),
        request.getComment());

        Map<String, Object> result = new HashMap<>();
        result.put("code", 0);
        result.put("message", "审核操作成功");
        result.put("data", Map.of(
            "resource_id", request.getResourceId(),
            "new_status", newStatus,
            "audit_record_id", record.getId()
        ));
    }
}

```

```

    ));
    return result;
}

/**
 * 批量审核
 *
 * @param resourceIds 资源ID列表
 * @param action 审核动作
 * @param comment 审核意见
 * @param auditorId 审核人
 * @return 批量审核结果
 */
public Map<String, Object> batchAudit(
    List<String> resourceIds,
    AuditAction action,
    String comment,
    String auditorId
) {
    logger.info(String.format(
        "批量审核: count=%d, action=%s", resourceIds.size(), action
    ));

    int successCount = 0;
    int failCount = 0;
    List<String> failedIds = new ArrayList<>();

    for (String resourceId : resourceIds) {
        try {
            AuditRequest request = new AuditRequest();
            request.setResourceId(resourceId);
            request.setAction(action);
            request.setComment(comment);
            request.setAuditorId(auditorId);

            Map<String, Object> result = performAudit(request);
            if ((int) result.get("code") == 0) {
                successCount++;
            } else {
                failCount++;
                failedIds.add(resourceId);
            }
        } catch (Exception e) {
            failCount++;
            failedIds.add(resourceId);
            logger.warning("批量审核失败: resource=" + resourceId + ", error=" +
e.getMessage());
        }
    }

    Map<String, Object> result = new HashMap<>();
    result.put("code", 0);
    result.put("data", Map.of(
        "total", resourceIds.size(),
        "success", successCount,
        "failed", failCount,
        "failed_ids", failedIds
    ));
}

```

```

    ));
    return result;
}

/**
 * AI自动预审
 *
 * 在人工审核前，自动进行内容安全检测：
 * - 文本内容是否包含违禁词
 * - 图片是否包含不当内容
 * - 文件格式是否合规
 * - 文件大小是否超限
 *
 * @param resourceId 资源ID
 * @return 预审结果
 */
public PreAuditResult performPreAudit(String resourceId) {
    logger.info("AI预审: resource=" + resourceId);

    PreAuditResult result = new PreAuditResult(true, 1.0);

    // 1. 文件格式和大小检查
    // ResourceMetadata resource = resourceMapper.selectById(resourceId);
    // if (resource.getFileSize() > MAX_FILE_SIZE) {
    //     result = new PreAuditResult(false, 0.0);
    //     result.addWarning("文件大小超过限制");
    //     return result;
    // }

    // 2. 文本内容安全检测（提取PDF/PPT中的文字进行违禁词检查）
    // String textContent = extractTextContent(resource.getFileKey());
    // ContentSafetyResult textSafety = contentSafetyApi.checkText(textContent);
    // if (!textSafety.isSafe()) {
    //     result.addWarning("文本内容包含敏感词: " + textSafety.getDetails());
    // }

    // 3. 图片内容安全检测（提取文档中的图片进行AI审核）
    // List<byte[]> images = extractImages(resource.getFileKey());
    // for (byte[] image : images) {
    //     ImageSafetyResult imageSafety = contentSafetyApi.checkImage(image);
    //     if (!imageSafety.isSafe()) {
    //         result.addWarning("图片内容不合规: " + imageSafety.getCategory());
    //     }
    // }

    // 综合评分
    if (!result.getWarnings().isEmpty()) {
        double penalty = result.getWarnings().size() * 0.2;
        double finalScore = Math.max(0.0, 1.0 - penalty);
        result = new PreAuditResult(finalScore >= 0.6, finalScore);
    }

    logger.info(String.format(
        "预审完成: resource=%s, safe=%b, score=%.2f",
        resourceId, result.isSafe(), result.getSafeScore()
    ));
}

```

```

        return result;
    }

    /**
     * 查询审核记录列表
     *
     * @param resourceId 资源ID (可选, 为空则查所有)
     * @param auditorId 审核人ID (可选)
     * @param page 页码
     * @param pageSize 每页大小
     * @return 审核记录列表
     */
    public Map<String, Object> queryAuditRecords(
        String resourceId,
        String auditorId,
        int page,
        int pageSize
    ) {
        logger.info(String.format(
            "查询审核记录: resource=%s, auditor=%s, page=%d",
            resourceId, auditorId, page
        ));

        // List<AuditRecord> records = auditRecordMapper.selectByCondition(
        //     resourceId, auditorId, page, pageSize
        // );
        // int total = auditRecordMapper.countByCondition(resourceId, auditorId);

        Map<String, Object> result = new HashMap<>();
        result.put("code", 0);
        result.put("data", Map.of(
            "total", 0,
            "page", page,
            "items", new ArrayList<>()
        ));
        return result;
    }

    /**
     * 获取待审核资源数量 (仪表盘统计用)
     */
    public Map<String, Object> getAuditStats() {
        // int pendingCount = resourceMapper.countByStatus("PENDING");
        // int approvedToday = auditRecordMapper.countTodayByAction("APPROVE");
        // int rejectedToday = auditRecordMapper.countTodayByAction("REJECT");

        Map<String, Object> result = new HashMap<>();
        result.put("code", 0);
        result.put("data", Map.of(
            "pending_count", 0,
            "approved_today", 0,
            "rejected_today", 0
        ));
        return result;
    }
}

```

service/CdnService.java

```
/*
 * 自然写教学资源管理与内容分发系统软件 V1.0
 * service/CdnService.java - CDN分发与缓存管理服务
 */
package com.writech.resource.service;

import java.util.*;
import java.util.logging.Logger;
import javax.crypto.Mac;
import javax.crypto.spec.SecretKeySpec;
import java.nio.charset.StandardCharsets;
import java.security.InvalidKeyException;
import java.security.NoSuchAlgorithmException;

/**
 * CDN分发与缓存管理服务
 *
 * 负责教学资源的CDN加速分发，包括：
 * - 签名URL生成（防盗链）
 * - CDN缓存预热与刷新
 * - 资源分发策略管理
 * - 下载流量统计
 */
public class CdnService {

    private static final Logger logger =
        Logger.getLogger(CdnService.class.getName());

    // =====
    //  CDN配置
    //  =====

    /** CDN域名 */
    private static final String CDN_DOMAIN = "https://cdn.writech.com";

    /** CDN签名密钥 */
    private String cdnSignKey;

    /** 签名URL默认有效期（秒） */
    private static final int DEFAULT_EXPIRE_SECONDS = 1800;

    /** Referer白名单 */
    private static final Set<String> REFERER_WHITELIST = new HashSet<>(Arrays.asList(
        "*.writech.com",
        "localhost",
        "127.0.0.1"
    ));

    /** CDN缓存策略（按资源类型配置TTL） */
    private static final Map<String, Integer> CACHE_TTL_MAP = new HashMap<>();
    static {
        CACHE_TTL_MAP.put("pdf", 86400 * 30); // PDF资源缓存30天
        CACHE_TTL_MAP.put("image", 86400 * 90); // 图片缓存90天
    }
}
```

```

        CACHE_TTL_MAP.put("video", 86400 * 7);           // 视频缓存7天
        CACHE_TTL_MAP.put("template", 86400 * 30);      // 模板缓存30天
        CACHE_TTL_MAP.put("dotcode", 86400 * 365);      // 点阵码缓存1年（不变内容）
    }

    public CdnService(String signKey) {
        this.cdnSignKey = signKey;
        logger.info("CDN服务初始化: domain=" + CDN_DOMAIN);
    }

    // =====
    //  签名URL生成（防盗链核心）
    //  =====

    /**
     * 生成CDN签名下载URL
     *
     * 签名算法（TypeA鉴权）：
     * 1. 计算签名原文：path-timestamp-rand-uid
     * 2. HMAC-SHA256(原文，密钥)
     * 3. 拼接签名URL：domain/path?auth_key=timestamp-rand-uid-signature
     *
     * @param objectKey OSS对象Key
     * @param expireSeconds 有效期（秒）
     * @return 签名后的CDN下载URL
     */
    public String generateSignedUrl(String objectKey, int expireSeconds) {
        if (expireSeconds <= 0) {
            expireSeconds = DEFAULT_EXPIRE_SECONDS;
        }

        long timestamp = System.currentTimeMillis() / 1000 + expireSeconds;
        String rand = UUID.randomUUID().toString().replace("-", "").substring(0, 8);
        String uid = "0"; // 用户标识（可选）
        String path = "/" + objectKey;

        // 签名原文
        String signContent = String.format("%s-%d-%s-%s", path, timestamp, rand, uid);

        // HMAC-SHA256计算签名
        String signature = hmacSha256(signContent, cdnSignKey);

        // 拼接签名URL
        String authKey = String.format("%d-%s-%s-%s", timestamp, rand, uid, signature);
        String signedUrl = String.format("%s%s?auth_key=%s", CDN_DOMAIN, path, authKey);

        logger.info(String.format(
            "生成签名URL: key=%s, expire=%ds", objectKey, expireSeconds
        ));

        return signedUrl;
    }

    /**
     * 验证签名URL是否有效
     *
     * @param url 待验证的URL
     */

```

```

    * @return 验证结果
    */
    public boolean verifySignedUrl(String url) {
        try {
            // 解析auth_key参数
            String authKey = extractParam(url, "auth_key");
            if (authKey == null) return false;

            String[] parts = authKey.split("-", 4);
            if (parts.length != 4) return false;

            long timestamp = Long.parseLong(parts[0]);
            String rand = parts[1];
            String uid = parts[2];
            String receivedSignature = parts[3];

            // 检查是否过期
            if (System.currentTimeMillis() / 1000 > timestamp) {
                return false;
            }

            // 重新计算签名对比
            String path = extractPath(url);
            String signContent = String.format("%s-%d-%s-%s", path, timestamp, rand,
uid);

            String expectedSignature = hmacSha256(signContent, cdnSignKey);

            return expectedSignature.equals(receivedSignature);
        } catch (Exception e) {
            logger.warning("签名验证异常: " + e.getMessage());
            return false;
        }
    }

    /**
     * HMAC-SHA256签名计算
     */
    private String hmacSha256(String data, String key) {
        try {
            Mac mac = Mac.getInstance("HmacSHA256");
            SecretKeySpec secretKey = new SecretKeySpec(
                key.getBytes(StandardCharsets.UTF_8), "HmacSHA256"
            );
            mac.init(secretKey);
            byte[] hash = mac.doFinal(data.getBytes(StandardCharsets.UTF_8));

            // 转换为十六进制字符串
            StringBuilder sb = new StringBuilder();
            for (byte b : hash) {
                sb.append(String.format("%02x", b));
            }
            return sb.toString();
        } catch (NoSuchAlgorithmException | InvalidKeyException e) {
            throw new RuntimeException("HMAC-SHA256计算失败", e);
        }
    }
}

```



```
// =====
//  CDN缓存管理
// =====

/**
 * 预热CDN缓存
 *
 * 将指定资源推送到CDN所有边缘节点，确保用户首次访问也能快速响应。
 * 通常在资源审核通过后触发预热。
 *
 * @param objectKeys 要预热的资源Key列表
 */
public void preheatResources(List<String> objectKeys) {
    logger.info(String.format("CDN缓存预热: %d个资源", objectKeys.size()));

    List<String> urls = new ArrayList<>();
    for (String key : objectKeys) {
        urls.add(CDN_DOMAIN + "/" + key);
    }

    // 调用CDN API预热
    // PushObjectCacheRequest request = new PushObjectCacheRequest();
    // request.setObjectPath(String.join("\n", urls));
    // cdnClient.pushObjectCache(request);

    logger.info("CDN预热任务已提交");
}

/**
 * 刷新CDN缓存
 *
 * 资源更新或删除后，需要刷新CDN缓存使旧版本失效。
 *
 * @param objectKeys 要刷新的资源Key列表
 */
public void refreshCache(List<String> objectKeys) {
    logger.info(String.format("CDN缓存刷新: %d个资源", objectKeys.size()));

    List<String> urls = new ArrayList<>();
    for (String key : objectKeys) {
        urls.add(CDN_DOMAIN + "/" + key);
    }

    // 调用CDN API刷新
    // RefreshObjectCachesRequest request = new RefreshObjectCachesRequest();
    // request.setObjectPath(String.join("\n", urls));
    // cdnClient.refreshObjectCaches(request);

    logger.info("CDN刷新任务已提交");
}

/**
 * 刷新目录缓存（用于整个类别的批量更新）
 */
public void refreshDirectoryCache(String directoryPath) {
    logger.info("CDN目录缓存刷新: " + directoryPath);
    // RefreshObjectCachesRequest request = new RefreshObjectCachesRequest();

```

```

        // request.setObjectPath(CDN_DOMAIN + "/" + directoryPath);
        // request.setObjectType("Directory");
        // cdnClient.refreshObjectCaches(request);
    }

    // =====
    // Referer防盗链校验
    // =====

    /**
     * 校验请求Referer是否在白名单中
     *
     * @param referer 请求头中的Referer
     * @return 是否允许访问
     */
    public boolean validateReferer(String referer) {
        if (referer == null || referer.isEmpty()) {
            return false; // 空Referer拒绝
        }

        for (String pattern : REFERER_WHITELIST) {
            if (pattern.startsWith("*.") ) {
                // 通配符匹配
                String domain = pattern.substring(2);
                if (referer.contains(domain)) {
                    return true;
                }
            } else {
                if (referer.contains(pattern)) {
                    return true;
                }
            }
        }

        logger.warning("Referer校验失败: " + referer);
        return false;
    }

    // =====
    // 流量统计
    // =====

    /**
     * 记录资源下载事件（异步写入ClickHouse）
     *
     * @param resourceId 资源ID
     * @param userId 下载用户ID
     * @param terminal 终端类型 (pad/pc/mobile/board)
     * @param fileSize 文件大小 (字节)
     */
    public void recordDownloadEvent(
        String resourceId,
        String userId,
        String terminal,
        long fileSize
    ) {
        // 异步写入ClickHouse使用统计表
    }

```

```

        // Map<String, Object> event = new HashMap<>();
        // event.put("resource_id", resourceId);
        // event.put("user_id", userId);
        // event.put("terminal", terminal);
        // event.put("file_size", fileSize);
        // event.put("download_at", new Date());
        // event.put("cdn_node", getCdnNodeId());

        // clickhouseClient.insert("usage_stat", event);
    }

    /**
     * 查询资源下载统计
     */
    public Map<String, Object> getDownloadStats(
        String resourceId, String startDate, String endDate
    ) {
        // 从ClickHouse查询聚合统计
        // SELECT count() as downloads, sum(file_size) as total_bytes,
        //         uniq(user_id) as unique_users
        // FROM usage_stat
        // WHERE resource_id = ? AND download_at BETWEEN ? AND ?

        Map<String, Object> stats = new HashMap<>();
        stats.put("resource_id", resourceId);
        stats.put("total_downloads", 0);
        stats.put("total_bytes", 0L);
        stats.put("unique_users", 0);
        stats.put("by_terminal", new HashMap<>());
        stats.put("daily_trend", new ArrayList<>());
        return stats;
    }

    // =====
    // 辅助方法
    // =====

    /** 从URL中提取指定参数 */
    private String extractParam(String url, String paramName) {
        int start = url.indexOf(paramName + "=");
        if (start < 0) return null;
        start += paramName.length() + 1;
        int end = url.indexOf("&", start);
        return end > 0 ? url.substring(start, end) : url.substring(start);
    }

    /** 从URL中提取路径部分 */
    private String extractPath(String url) {
        int start = url.indexOf("/", url.indexOf("//") + 2);
        int end = url.indexOf("?");
        return end > 0 ? url.substring(start, end) : url.substring(start);
    }
}

```

service/DotCodeService.java

```

/*
 * 自然写教学资源管理与内容分发系统软件 V1.0
 * service/DotCodeService.java - 点阵码生成引擎服务
 */
package com.writech.resource.service;

import java.util.*;
import java.util.concurrent.ConcurrentHashMap;
import java.util.logging.Logger;
import java.security.MessageDigest;
import java.security.NoSuchAlgorithmException;

/**
 * 点阵码生成引擎服务
 *
 * 负责点阵码资源的生成、分配和管理。
 * 点阵码是自然写系统的核心技术，每个点阵码对应一个唯一的
 * 页面/区域标识，配合点阵笔可精确定位书写位置。
 *
 * 功能：
 * - 点阵码ID全局唯一分配（防冲突）
 * - 点阵码图案生成（OGP编码）
 * - 点阵码与页面/课件的绑定关系管理
 * - 批量生成点阵码资源包
 * - 点阵码PDF合成（叠加到字帖/试卷模板上）
 */
public class DotCodeService {

    private static final Logger logger =
        Logger.getLogger(DotCodeService.class.getName());

    // =====
    // 点阵码常量与配置
    // =====

    /** OGP点阵码编码参数 */
    private static final int DOT_GRID_SIZE = 6;           // 每组点阵6x6
    private static final double DOT_SPACING_MM = 0.3;     // 点间距0.3mm
    private static final double DOT_OFFSET_MM = 0.1;      // 点偏移量0.1mm
    private static final int DOTS_PER_PAGE = 10000;       // 每页约10000个点

    /** 点阵码ID分配范围 */
    private static final long ID_RANGE_START = 1_000_000_000L;
    private static final long ID_RANGE_END = 9_999_999_999L;

    /** 当前已分配的最大ID（原子操作保证线程安全） */
    private long currentMaxId = ID_RANGE_START;

    /** 点阵码-页面绑定关系缓存 */
    private final Map<Long, DotCodeBinding> bindingCache = new ConcurrentHashMap<>();

    // =====
    // 数据模型
    // =====

    /** 点阵码绑定关系 */

```

```

public static class DotCodeBinding {
    private long dotCodeId;           // 点阵码ID
    private String resourceId;         // 绑定的资源ID
    private int pageIndex;            // 页面序号
    private String areaType;          // 区域类型 (full_page/answer_area/title_area)
    private double areaX;              // 区域起始X坐标 (mm)
    private double areaY;              // 区域起始Y坐标 (mm)
    private double areaWidth;          // 区域宽度 (mm)
    private double areaHeight;         // 区域高度 (mm)
    private Date createdAt;

    public DotCodeBinding() {}

    public DotCodeBinding(long dotCodeId, String resourceId, int pageIndex) {
        this.dotCodeId = dotCodeId;
        this.resourceId = resourceId;
        this.pageIndex = pageIndex;
        this.createdAt = new Date();
    }

    public long getDotCodeId() { return dotCodeId; }
    public void setDotCodeId(long id) { this.dotCodeId = id; }
    public String getResourceId() { return resourceId; }
    public void setResourceId(String rid) { this.resourceId = rid; }
    public int getPageIndex() { return pageIndex; }
    public void setPageIndex(int idx) { this.pageIndex = idx; }
    public String getAreaType() { return areaType; }
    public void setAreaType(String type) { this.areaType = type; }
    public double getAreaX() { return areaX; }
    public double getAreaY() { return areaY; }
    public double getAreaWidth() { return areaWidth; }
    public double getAreaHeight() { return areaHeight; }
}

/** 点阵码生成请求 */
public static class DotCodeGenerateRequest {
    private String resourceId;         // 关联资源ID
    private int pageCount;             // 页数
    private double pageWidth;          // 页面宽度 (mm)
    private double pageHeight;         // 页面高度 (mm)
    private String outputFormat;       // 输出格式 (pdf/png/svg)
    private boolean overlayOnTemplate; // 是否叠加到模板上
    private String templateFileKey;    // 模板文件OSS Key

    public String getResourceId() { return resourceId; }
    public void setResourceId(String id) { this.resourceId = id; }
    public int getPageCount() { return pageCount; }
    public void setPageCount(int count) { this.pageCount = count; }
    public double getPageWidth() { return pageWidth > 0 ? pageWidth : 210.0; }
    public double getPageHeight() { return pageHeight > 0 ? pageHeight : 297.0; }
    public String getOutputFormat() { return outputFormat != null ? outputFormat :
"pdf"; }
    public boolean isOverlayOnTemplate() { return overlayOnTemplate; }
    public String getTemplateFileKey() { return templateFileKey; }
}

/** 点阵码生成结果 */

```

```

public static class DotCodeGenerateResult {
    private String taskId;
    private String resourceId;
    private List<Long> dotCodeIds;          // 分配的点阵码ID列表
    private String outputFileKey;          // 生成的文件OSS Key
    private int pageCount;
    private long totalDots;
    private String status;                  // processing/completed/failed

    public String getTaskId() { return taskId; }
    public void setTaskId(String id) { this.taskId = id; }
    public List<Long> getDotCodeIds() { return dotCodeIds; }
    public void setDotCodeIds(List<Long> ids) { this.dotCodeIds = ids; }
    public String getOutputFileKey() { return outputFileKey; }
    public void setOutputFileKey(String key) { this.outputFileKey = key; }
    public String getStatus() { return status; }
    public void setStatus(String s) { this.status = s; }
}

// =====
// 核心方法实现
// =====

/**
 * 批量生成点阵码资源包 POST /api/v1/dotcode/generate
 *
 * 流程：
 * 1. 分配全局唯一的点阵码ID范围
 * 2. 为每页生成OGP编码的点阵图案
 * 3. 如果需要叠加模板，合成到模板PDF上
 * 4. 上传生成结果到OSS
 * 5. 记录绑定关系到MySQL
 */
public DotCodeGenerateResult generateDotCodes(DotCodeGenerateRequest request) {
    logger.info(String.format(
        "生成点阵码: resource=%s, pages=%d, size=%.0fx%.0fmm",
        request.getResourceId(), request.getPageCount(),
        request.getPageWidth(), request.getPageHeight()
    ));

    DotCodeGenerateResult result = new DotCodeGenerateResult();
    result.setTaskId(UUID.randomUUID().toString().replace("-", "").substring(0,
16));
    result.setStatus("processing");

    // 1. 分配点阵码ID
    List<Long> allocatedIds = allocateDotCodeIds(request.getPageCount());
    result.setDotCodeIds(allocatedIds);

    // 2. 为每页生成点阵码图案
    for (int i = 0; i < request.getPageCount(); i++) {
        long dotCodeId = allocatedIds.get(i);

        // 生成OGP编码点阵图案
        byte[][] dotPattern = generateOGPPattern(
            dotCodeId,
            request.getPageWidth(),

```

```

        request.getPageHeight()
    );

    // 记录绑定关系
    DotCodeBinding binding = new DotCodeBinding(
        dotCodeId, request.getResourceId(), i
    );
    binding.setAreaType("full_page");
    binding.setAreaX(0);
    binding.setAreaY(0);
    binding.setAreaWidth(request.getPageWidth());
    binding.setAreaHeight(request.getPageHeight());

    bindingCache.put(dotCodeId, binding);

    // 持久化到MySQL
    // dotCodeMapper.insertBinding(binding);
}

// 3. 如果叠加模板, 合成PDF
if (request.isOverlayOnTemplate() && request.getTemplateFileKey() != null) {
    // 下载模板PDF
    // byte[] templatePdf = ossClient.getObject(request.getTemplateFileKey());
    // 叠加点阵码图层
    // byte[] mergedPdf = pdfMerger.overlayDotCodes(templatePdf, dotPatterns);
    // 上传合成后的PDF
    // String outputKey = ossClient.putObject(mergedPdf, ...);
    // result.setOutputFileKey(outputKey);
}

result.setStatus("completed");
result.setPageCount(request.getPageCount());
result.setTotalDots((long) request.getPageCount() * DOTS_PER_PAGE);

logger.info(String.format(
    "点阵码生成完成: task=%s, ids=[%d~%d], dots=%d",
    result.getTaskId(),
    allocatedIds.get(0),
    allocatedIds.get(allocatedIds.size() - 1),
    result.getTotalDots()
));

return result;
}

/**
 * 分配全局唯一的点阵码ID
 *
 * 使用原子递增方式保证ID全局唯一, 防止多服务器实例间冲突。
 * 生产环境使用Redis分布式ID生成器。
 *
 * @param count 需要分配的ID数量
 * @return 分配的ID列表
 */
public synchronized List<Long> allocateDotCodeIds(int count) {
    List<Long> ids = new ArrayList<>();

```

```

        if (currentMaxId + count > ID_RANGE_END) {
            throw new RuntimeException("点阵码ID已耗尽, 请联系管理员扩容");
        }

        for (int i = 0; i < count; i++) {
            currentMaxId++;
            ids.add(currentMaxId);
        }

        // 持久化当前最大ID (Redis或数据库)
        // redisTemplate.set("dot_code_max_id", String.valueOf(currentMaxId));

        logger.info(String.format(
            "分配点阵码ID: count=%d, range=[%d, %d]",
            count, ids.get(0), ids.get(ids.size() - 1)
        ));

        return ids;
    }

    /**
     * 生成OGP编码的点阵图案
     *
     * OGP (Optical Glyph Pattern) 编码原理:
     * 将点阵码ID编码为点的微小位移方向 (上下左右4个方向),
     * 每组6x6点阵编码一组信息, 整页覆盖实现全页面位置编码。
     *
     * @param dotCodeId 点阵码ID
     * @param pageWidthMm 页面宽度 (毫米)
     * @param pageHeightMm 页面高度 (毫米)
     * @return 点阵图案 (2D数组, 0=无偏移, 1=上, 2=右, 3=下, 4=左)
     */
    public byte[][] generateOGPPattern(
        long dotCodeId,
        double pageWidthMm,
        double pageHeightMm
    ) {
        // 计算网格尺寸
        int gridCols = (int) (pageWidthMm / DOT_SPACING_MM);
        int gridRows = (int) (pageHeightMm / DOT_SPACING_MM);

        byte[][] pattern = new byte[gridRows][gridCols];

        // 将点阵码ID编码为二进制位流
        long encodedId = dotCodeId;
        byte[] idBits = new byte[40]; // 40位足以表示10位十进制数
        for (int i = 0; i < 40; i++) {
            idBits[i] = (byte) ((encodedId >> (39 - i)) & 1);
        }

        // 填充点阵图案
        for (int row = 0; row < gridRows; row++) {
            for (int col = 0; col < gridCols; col++) {
                // 每个点的偏移方向由其位置和ID编码共同决定
                int groupRow = row / DOT_GRID_SIZE;
                int groupCol = col / DOT_GRID_SIZE;
                int localRow = row % DOT_GRID_SIZE;

```



```

        int localCol = col % DOT_GRID_SIZE;

        // 位置编码 + ID编码 混合
        int bitIndex = ((groupRow * (gridCols / DOT_GRID_SIZE) + groupCol)
            * DOT_GRID_SIZE * DOT_GRID_SIZE
            + localRow * DOT_GRID_SIZE + localCol) % 40;

        // 偏移方向: 0=无, 1=上, 2=右, 3=下, 4=左
        int positionHash = (row * 7 + col * 13 + (int) dotCodeId) % 5;
        pattern[row][col] = (byte) ((positionHash + idBits[bitIndex]) % 5);
    }
}

// 添加校验码区域 (边缘4行/列作为同步标记和校验)
addSyncMarkers(pattern, gridRows, gridCols);

return pattern;
}

/**
 * 在点阵图案边缘添加同步标记和校验码
 * 摄像头采集后需要同步标记来确定方向和位置
 */
private void addSyncMarkers(byte[][] pattern, int rows, int cols) {
    // 顶部同步行: 交替0和1
    for (int col = 0; col < cols; col++) {
        pattern[0][col] = (byte) (col % 2 == 0 ? 1 : 3);
        pattern[1][col] = (byte) (col % 2 == 0 ? 3 : 1);
    }

    // 左侧同步列
    for (int row = 0; row < rows; row++) {
        pattern[row][0] = (byte) (row % 2 == 0 ? 2 : 4);
        pattern[row][1] = (byte) (row % 2 == 0 ? 4 : 2);
    }

    // 右下角放置4x4校验码块
    // 校验码 = CRC-8(页面ID的低8位)
    // 用于摄像头快速验证解码是否正确
}

/**
 * 根据点阵码ID查询绑定的资源和页面信息
 *
 * @param dotCodeId 点阵码ID
 * @return 绑定关系 (如果存在)
 */
public DotCodeBinding queryBinding(long dotCodeId) {
    // 先查缓存
    DotCodeBinding cached = bindingCache.get(dotCodeId);
    if (cached != null) {
        return cached;
    }

    // 缓存未命中, 查数据库
    // DotCodeBinding binding = dotCodeMapper.selectByDotCodeId(dotCodeId);
    // if (binding != null) {

```

```

        //      bindingCache.put(dotCodeId, binding);
        // }
        // return binding;

        return null;
    }

    /**
     * 查询资源关联的所有点阵码
     */
    public List<DotCodeBinding> queryByResourceId(String resourceId) {
        // return dotCodeMapper.selectByResourceId(resourceId);
        return new ArrayList<>();
    }

    /**
     * 计算点阵码的SHA-256指纹（用于校验完整性）
     */
    public String calculatePatternFingerprint(byte[][] pattern) {
        try {
            MessageDigest digest = MessageDigest.getInstance("SHA-256");
            for (byte[] row : pattern) {
                digest.update(row);
            }
            byte[] hash = digest.digest();
            StringBuilder sb = new StringBuilder();
            for (byte b : hash) {
                sb.append(String.format("%02x", b));
            }
            return sb.toString();
        } catch (NoSuchAlgorithmException e) {
            throw new RuntimeException("SHA-256不可用", e);
        }
    }
}

```

service/ResourceService.java

```

/*
 * 自然写教学资源管理与内容分发系统软件 V1.0
 * service/ResourceService.java - 资源管理业务服务
 */
package com.writech.resource.service;

import java.util.*;
import java.util.logging.Logger;
import java.util.stream.Collectors;

/**
 * 资源管理业务服务
 *
 * 负责资源的完整生命周期管理：
 * - 资源元数据CRUD（MySQL）
 * - 文件存储管理（OSS/MinIO对象存储）

```

```

* - 全文索引管理 (Elasticsearch)
* - CDN缓存管理
* - 版本控制
* - 数字水印
*/
public class ResourceService {

    private static final Logger logger =
        Logger.getLogger(ResourceService.class.getName());

    // =====
    // 配置常量
    // =====

    /** 支持的文件类型及最大大小 (MB) */
    private static final Map<String, Integer> ALLOWED_FILE_TYPES = new HashMap<>();
    static {
        ALLOWED_FILE_TYPES.put("application/pdf", 100);
        ALLOWED_FILE_TYPES.put("application/vnd.ms-powerpoint", 200);
        ALLOWED_FILE_TYPES.put("application/vnd.openxmlformats-officedocument.presentationml.presentation", 200);
        ALLOWED_FILE_TYPES.put("image/jpeg", 20);
        ALLOWED_FILE_TYPES.put("image/png", 20);
        ALLOWED_FILE_TYPES.put("image/svg+xml", 10);
        ALLOWED_FILE_TYPES.put("video/mp4", 500);
        ALLOWED_FILE_TYPES.put("audio/mpeg", 50);
    }

    /** OSS存储桶名称 */
    private static final String OSS_BUCKET = "writech-resources";

    /** 缩略图存储前缀 */
    private static final String THUMBNAIL_PREFIX = "thumbnails/";

    /** Elasticsearch索引名称 */
    private static final String ES_INDEX = "writech_resources";

    // =====
    // 数据模型
    // =====

    /** 资源版本记录 */
    public static class ResourceVersion {
        private String id;
        private String resourceId;
        private int versionNumber;
        private String fileKey;
        private long fileSize;
        private String changeLog;
        private String operatorId;
        private Date createdAt;

        public String getId() { return id; }
        public void setId(String id) { this.id = id; }
        public String getResourceId() { return resourceId; }
        public void setResourceId(String rid) { this.resourceId = rid; }
        public int getVersionNumber() { return versionNumber; }
    }
}

```

```

    public void setVersionNumber(int v) { this.versionNumber = v; }
    public String getFileKey() { return fileKey; }
    public void setFileKey(String key) { this.fileKey = key; }
    public String getChangeLog() { return changeLog; }
    public void setChangeLog(String log) { this.changeLog = log; }
    public Date getCreatedAt() { return createdAt; }
}

/** 数字水印配置 */
public static class WatermarkConfig {
    private String text;           // 水印文字 (学校名+教师名)
    private float opacity;         // 透明度 (0.0-1.0)
    private int fontSize;          // 字号
    private float rotation;        // 旋转角度
    private String position;       // 位置: center/bottom-right/tiled

    public WatermarkConfig(String text) {
        this.text = text;
        this.opacity = 0.15f;
        this.fontSize = 24;
        this.rotation = -30.0f;
        this.position = "tiled";
    }

    public String getText() { return text; }
    public float getOpacity() { return opacity; }
    public int getFontSize() { return fontSize; }
    public float getRotation() { return rotation; }
    public String getPosition() { return position; }
}

/** STS临时上传凭证 */
public static class UploadCredential {
    private String accessKeyId;
    private String accessKeySecret;
    private String securityToken;
    private String bucket;
    private String objectKeyPrefix;
    private long expireTimeSeconds;

    public String getAccessKeyId() { return accessKeyId; }
    public String getAccessKeySecret() { return accessKeySecret; }
    public String getSecurityToken() { return securityToken; }
    public String getBucket() { return bucket; }
    public String getObjectKeyPrefix() { return objectKeyPrefix; }
    public long getExpireTimeSeconds() { return expireTimeSeconds; }
}

// =====
// 业务方法
// =====

/**
 * 获取STS临时上传凭证
 *
 * 前端使用STS凭证直接上传到OSS，避免文件经过应用服务器。
 * STS凭证限制：仅允许PUT到指定前缀路径，有效期15分钟。

```

```

*
* @param uploaderId 上传者ID
* @param fileType 文件MIME类型
* @return STS临时凭证
*/
public UploadCredential getUploadCredential(String uploaderId, String fileType) {
    logger.info(String.format("获取上传凭证: user=%s, type=%s", uploaderId,
fileType));

    // 校验文件类型
    if (!ALLOWED_FILE_TYPES.containsKey(fileType)) {
        throw new IllegalArgumentException("不支持的文件类型: " + fileType);
    }

    // 生成上传路径前缀: resources/{uploaderId}/{year}/{month}/
    Calendar cal = Calendar.getInstance();
    String prefix = String.format(
        "resources/%s/%d/%02d/",
        uploaderId,
        cal.get(Calendar.YEAR),
        cal.get(Calendar.MONTH) + 1
    );

    // 调用OSS STS服务获取临时凭证
    // AssumeRoleResponse response = stsClient.assumeRole(
    //     roleArn, policy, sessionName, 900 // 15分钟
    // );

    UploadCredential credential = new UploadCredential();
    // credential.accessKeyId = response.getCredentials().getAccessKeyId();
    // credential.accessKeySecret = response.getCredentials().getAccessKeySecret();
    // credential.securityToken = response.getCredentials().getSecurityToken();
    // credential.bucket = OSS_BUCKET;
    // credential.objectKeyPrefix = prefix;
    // credential.expireTimeSeconds = 900;

    return credential;
}

/**
* 创建资源记录 (上传完成后调用)
*
* @param metadata 资源元数据
* @return 创建的资源ID
*/
public String createResource(Map<String, Object> metadata) {
    String name = (String) metadata.get("name");
    String fileKey = (String) metadata.get("file_key");
    String mimeType = (String) metadata.get("mime_type");

    logger.info(String.format("创建资源: name=%s, key=%s", name, fileKey));

    // 生成资源ID
    String resourceId = UUID.randomUUID().toString().replace("-", "");

    // 自动生成缩略图
    generateThumbnailAsync(resourceId, fileKey, mimeType);

```

```

        // 插入MySQL元数据
        // resourceMapper.insert(resource);

        // 创建初始版本记录
        createVersion(resourceId, fileKey, "初始版本", (String)
metadata.get("uploader_id"));

        // 同步索引到Elasticsearch
        indexToElasticsearch(resourceId, metadata);

        logger.info("资源创建成功: id=" + resourceId);
        return resourceId;
    }

    /**
     * 更新资源（新版本上传）
     *
     * 资源更新不删除旧版本，而是创建新版本记录，
     * 支持版本回滚。更新后需刷新CDN缓存。
     */
    public void updateResource(String resourceId, Map<String, Object> updateData) {
        logger.info("更新资源: id=" + resourceId);

        String newFileKey = (String) updateData.get("file_key");
        String changeLog = (String) updateData.get("change_log");
        String operatorId = (String) updateData.get("operator_id");

        // 创建新版本
        if (newFileKey != null) {
            createVersion(resourceId, newFileKey, changeLog, operatorId);
        }

        // 更新MySQL元数据
        // resourceMapper.update(resourceId, updateData);

        // 更新Elasticsearch索引
        updateElasticsearchIndex(resourceId, updateData);

        // 刷新CDN缓存
        refreshCdnCache(resourceId);
    }

    /**
     * 创建版本记录
     */
    private void createVersion(
        String resourceId, String fileKey, String changeLog, String operatorId
    ) {
        // 查询当前最大版本号
        // int maxVersion = versionMapper.selectMaxVersion(resourceId);

        ResourceVersion version = new ResourceVersion();
        version.setId(UUID.randomUUID().toString().replace("-", ""));
        version.setResourceId(resourceId);
        version.setVersionNumber(1); // maxVersion + 1
        version.setFileKey(fileKey);
    }

```

```

        version.setChangeLog(changeLog);

        // versionMapper.insert(version);
        logger.info(String.format(
            "创建版本: resource=%s, version=%d", resourceId, version.getVersionNumber()
        ));
    }

    /**
     * 异步生成缩略图
     *
     * 根据文件类型采用不同策略:
     * - PDF: 渲染第一页为图片
     * - PPT: 提取封面幻灯片
     * - 图片: 直接缩放
     * - 视频: 提取关键帧
     */
    private void generateThumbnailAsync(String resourceId, String fileKey, String
    mimeType) {
        // @Async 异步执行
        logger.info(String.format(
            "生成缩略图: resource=%s, type=%s", resourceId, mimeType
        ));

        // 根据MIME类型选择缩略图生成策略
        // if (mimeType.equals("application/pdf")) {
        //     PDDocument doc = PDDocument.load(ossClient.getObject(fileKey));
        //     PDFRenderer renderer = new PDFRenderer(doc);
        //     BufferedImage image = renderer.renderImageWithDPI(0, 150);
        //     // 缩放为缩略图尺寸(320x240)
        //     BufferedImage thumb = ImageUtils.resize(image, 320, 240);
        //     // 上传缩略图到OSS
        //     ossClient.putObject(THUMBNAIL_PREFIX + resourceId + ".jpg", thumb);
        // }

    }

    /**
     * 索引资源到Elasticsearch
     *
     * 索引字段: 名称、描述、标签、学科、年级、出版社、类型
     * 支持中文分词 (IK分词器)
     */
    private void indexToElasticsearch(String resourceId, Map<String, Object> metadata) {
        logger.info("索引资源到ES: id=" + resourceId);

        // Map<String, Object> document = new HashMap<>();
        // document.put("id", resourceId);
        // document.put("name", metadata.get("name"));
        // document.put("description", metadata.get("description"));
        // document.put("tags", metadata.get("tags"));
        // document.put("subject", metadata.get("subject"));
        // document.put("grade", metadata.get("grade"));
        // document.put("publisher", metadata.get("publisher"));
        // document.put("type", metadata.get("type"));
        // document.put("school_id", metadata.get("school_id"));
        // document.put("audit_status", "PENDING");
        // document.put("created_at", new Date());
    }

```

```

        // IndexRequest request = new IndexRequest(ES_INDEX)
        //     .id(resourceId)
        //     .source(document);
        // elasticsearchClient.index(request);
    }

    /**
     * 更新Elasticsearch索引
     */
    private void updateElasticsearchIndex(String resourceId, Map<String, Object>
updateData) {
        // UpdateRequest request = new UpdateRequest(ES_INDEX, resourceId)
        //     .doc(updateData);
        // elasticsearchClient.update(request);
    }

    /**
     * 刷新CDN缓存
     *
     * 资源更新后需要刷新CDN节点缓存，确保终端获取最新版本。
     */
    private void refreshCdnCache(String resourceId) {
        logger.info("刷新CDN缓存: resource=" + resourceId);
        // String cdnUrl = String.format("https://cdn.writech.com/resources/%s",
resourceId);
        // cdnClient.refreshObjectCaches(Collections.singletonList(cdnUrl));
    }

    /**
     * 添加数字水印
     *
     * 下载资源时可选添加数字水印，水印包含学校和教师标识，
     * 用于版权保护和追踪。
     *
     * @param fileBytes 原始文件字节
     * @param config 水印配置
     * @return 添加水印后的文件字节
     */
    public byte[] addWatermark(byte[] fileBytes, WatermarkConfig config) {
        logger.info("添加数字水印: text=" + config.getText());

        // PDF水印添加
        // PDDocument doc = PDDocument.load(fileBytes);
        // for (PDPage page : doc.getPages()) {
        //     PDPageContentStream cs = new PDPageContentStream(doc, page, APPEND,
true);
        //     cs.setFont(PDType1Font.HELVETICA, config.getFontSize());
        //     cs.setNonStrokingColor(200, 200, 200); // 浅灰色
        //     // 平铺水印
        //     for (float y = 0; y < page.getMediaBox().getHeight(); y += 100) {
        //         for (float x = 0; x < page.getMediaBox().getWidth(); x += 200) {
        //             cs.beginText();
        //             Matrix matrix = Matrix.getRotateInstance(
        //                 Math.toRadians(config.getRotation()), x, y
        //             );
        //             cs.setTextMatrix(matrix);

```



```

        //          cs.showText(config.getText());
        //          cs.endText();
        //      }
        //  }
        //  cs.close();
        // }

        return fileBytes;
    }

    /**
     * 删除资源（软删除）
     *
     * 不物理删除文件，仅标记为已删除状态。
     * OSS文件通过生命周期策略定期清理。
     */
    public void deleteResource(String resourceId, String operatorId) {
        logger.info(String.format(
            "删除资源: id=%s, operator=%s", resourceId, operatorId
        ));

        // 软删除：更新状态
        // resourceMapper.updateStatus(resourceId, "DELETED");

        // 从ES索引中移除
        // elasticSearchClient.delete(new DeleteRequest(ES_INDEX, resourceId));

        // 刷新CDN
        refreshCdnCache(resourceId);
    }
}

```

service/SearchService.java

```

/*
 * 自然写教学资源管理与内容分发系统软件 V1.0
 * service/SearchService.java - Elasticsearch全文检索服务
 */
package com.writech.resource.service;

import java.util.*;
import java.util.logging.Logger;

/**
 * Elasticsearch全文检索服务
 *
 * 负责教学资源的全文检索能力：
 * - 索引创建与管理（按学科/年级分片）
 * - 中文分词（IK分词器）
 * - 多条件组合检索
 * - 聚合统计（Facet搜索）
 * - 搜索建议（Suggest）
 * - 相关资源推荐
 */

```

```

public class SearchService {

    private static final Logger logger =
        Logger.getLogger(SearchService.class.getName());

    /** ES索引名称 */
    private static final String INDEX_NAME = "writech_resources";

    /** 索引分片数 */
    private static final int NUMBER_OF_SHARDS = 3;

    /** 索引副本数 */
    private static final int NUMBER_OF_REPLICAS = 1;

    /** 搜索结果高亮标签 */
    private static final String HIGHLIGHT_PRE_TAG = "<em>";
    private static final String HIGHLIGHT_POST_TAG = "</em>";

    /**
     * 创建资源索引（系统初始化时调用）
     *
     * 索引映射字段：
     * - name: text（IK中文分词）+ keyword子字段
     * - description: text（IK中文分词）
     * - tags: keyword数组
     * - subject/grade/publisher/type/school_id/audit_status: keyword
     * - download_count/use_count: integer
     * - created_at/updated_at: date
     */
    public void createIndex() {
        logger.info("创建ES索引: " + INDEX_NAME);

        Map<String, Object> settings = new HashMap<>();
        settings.put("number_of_shards", NUMBER_OF_SHARDS);
        settings.put("number_of_replicas", NUMBER_OF_REPLICAS);

        // IK分词器配置
        Map<String, Object> analysis = new HashMap<>();
        Map<String, Object> analyzers = new HashMap<>();
        analyzers.put("ik_max", Map.of("type", "custom", "tokenizer", "ik_max_word"));
        analyzers.put("ik_smart", Map.of("type", "custom", "tokenizer", "ik_smart"));
        analysis.put("analyzer", analyzers);
        settings.put("analysis", analysis);

        // 字段映射定义
        Map<String, Object> properties = new LinkedHashMap<>();

        // 名称字段: 主搜索字段
        Map<String, Object> nameField = new HashMap<>();
        nameField.put("type", "text");
        nameField.put("analyzer", "ik_max_word");
        nameField.put("search_analyzer", "ik_smart");
        nameField.put("fields", Map.of("keyword", Map.of("type", "keyword")));
        properties.put("name", nameField);

        // 描述字段
        properties.put("description", Map.of("type", "text", "analyzer",

```

```

        "ik_max_word"));
        properties.put("tags", Map.of("type", "keyword"));
        properties.put("subject", Map.of("type", "keyword"));
        properties.put("grade", Map.of("type", "keyword"));
        properties.put("publisher", Map.of("type", "keyword"));
        properties.put("type", Map.of("type", "keyword"));
        properties.put("school_id", Map.of("type", "keyword"));
        properties.put("audit_status", Map.of("type", "keyword"));
        properties.put("download_count", Map.of("type", "integer"));
        properties.put("use_count", Map.of("type", "integer"));
        properties.put("created_at", Map.of("type", "date"));

        logger.info("ES索引映射已定义: " + properties.size() + "个字段");
    }

    /**
     * 全文检索资源
     *
     * 搜索策略:
     * 1. 关键词multi_match跨name+description+tags字段
     * 2. 分类term精确过滤subject/grade/publisher
     * 3. 权限过滤 (仅审核通过+本校授权)
     * 4. 相关性+热度综合排序 (function_score)
     * 5. 聚合统计各分类维度资源数量
     * 6. 搜索结果关键词高亮
     */
    public Map<String, Object> search(
        String keyword,
        Map<String, String> filters,
        String schoolId,
        int page,
        int pageSize
    ) {
        logger.info(String.format(
            "资源搜索: keyword=%s, school=%s, page=%d", keyword, schoolId, page
        ));

        // 构建Bool查询
        // BoolQueryBuilder boolQuery = QueryBuilders.boolQuery();

        // 关键词匹配 (boost权重: name:3 > tags:2 > description:1)
        // if (keyword != null && !keyword.trim().isEmpty()) {
        //     boolQuery.must(QueryBuilders.multiMatchQuery(keyword)
        //         .field("name", 3.0f)
        //         .field("tags", 2.0f)
        //         .field("description", 1.0f)
        //         .type(MultiMatchQueryBuilder.Type.BEST_FIELDS)
        //         .minimumShouldMatch("70%"));
        // }

        // 分类过滤
        // if (filters != null) {
        //     filters.forEach((key, value) -> {
        //         if (value != null) boolQuery.filter(termQuery(key, value));
        //     });
        // }
    }

```

```

// 权限过滤: 仅返回审核通过的资源
// boolQuery.filter(termQuery("audit_status", "APPROVED"));
// boolQuery.filter(termQuery("school_id", schoolId));

// function_score: 相关性*0.7 + log(download_count+1)*0.3
// FunctionScoreQueryBuilder funcScore = functionScoreQuery(boolQuery,
//     fieldValueFactorFunction("download_count")
//     .modifier(Modifier.LOG1P).factor(0.3f)
// ).scoreMode(ScoreMode.SUM);

// 聚合统计
// 按subject/grade/publisher/type分组统计数量

// 高亮配置
// HighlightBuilder highlight = new HighlightBuilder()
//     .preTags(HIGHLIGHT_PRE_TAG).postTags(HIGHLIGHT_POST_TAG)
//     .field("name").field("description");

Map<String, Object> result = new HashMap<>();
result.put("total", 0);
result.put("page", page);
result.put("items", new ArrayList<>());
result.put("facets", Map.of(
    "by_subject", new ArrayList<>(),
    "by_grade", new ArrayList<>(),
    "by_publisher", new ArrayList<>(),
    "by_type", new ArrayList<>()
));
return result;
}

/**
 * 搜索建议 (输入补全)
 * 用户输入时实时返回匹配的资源名称建议
 */
public List<String> suggest(String prefix, int size) {
    if (prefix == null || prefix.trim().isEmpty()) {
        return Collections.emptyList();
    }
    logger.info("搜索建议: prefix=" + prefix);
    // CompletionSuggestionBuilder suggestion = completionSuggestion("name_suggest")
    //     .prefix(prefix).size(size);
    return new ArrayList<>();
}

/**
 * 相关资源推荐 (More Like This查询)
 * 基于内容相似度推荐同类资源
 */
public List<Map<String, Object>> recommend(String resourceId, int size) {
    logger.info(String.format("相关推荐: resource=%s, size=%d", resourceId, size));
    // moreLikeThisQuery(["name","description","tags"], null, [item(INDEX, id)])
    //     .minTermFreq(1).maxQueryTerms(12)
    return new ArrayList<>();
}

/** 索引单个资源文档 */

```

```

public void indexDocument(String resourceId, Map<String, Object> doc) {
    logger.info("索引资源: id=" + resourceId);
}

/** 更新索引文档（部分更新） */
public void updateDocument(String resourceId, Map<String, Object> partialDoc) {
    logger.info("更新索引: id=" + resourceId);
}

/** 删除索引文档 */
public void deleteDocument(String resourceId) {
    logger.info("删除索引: id=" + resourceId);
}

/**
 * 批量重建索引
 * 从MySQL全量加载资源元数据，重新构建ES索引
 */
public int rebuildIndex() {
    logger.info("开始重建ES索引...");
    // 1. 删除旧索引
    // 2. 重新创建索引（含映射）
    createIndex();
    // 3. 从MySQL批量查询所有审核通过的资源
    // 4. 使用BulkRequest批量索引
    int count = 0;
    // List<Resource> allResources = resourceMapper.selectAllApproved();
    // BulkRequest bulk = new BulkRequest();
    // for (Resource r : allResources) {
    //     bulk.add(new IndexRequest(INDEX_NAME).id(r.getId()).source(toDoc(r)));
    //     count++;
    //     if (count % 500 == 0) {
    //         elasticsearchClient.bulk(bulk);
    //         bulk = new BulkRequest();
    //     }
    // }
    // if (bulk.numberOfActions() > 0) elasticsearchClient.bulk(bulk);
    logger.info("ES索引重建完成: " + count + "条");
    return count;
}
}

```