

# 自然写互动课堂教学管理云平台软件 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

```
01-writech-cloud-platform/  
├─ WritechCloudApplication.java  
├─ config/  
│   ├── KafkaConfig.java  
│   └─ SecurityConfig.java  
├─ controller/  
│   ├── AssignmentController.java  
│   ├── AuthController.java  
│   ├── DeviceController.java  
│   └─ StrokeController.java  
├─ model/  
│   ├── Models.java  
│   └─ User.java  
└─ service/  
    ├── DeviceService.java  
    ├── MessageService.java  
    ├── StrokeService.java  
    └─ UserService.java
```

## 源程序文件清单

(根目录)

**WritechCloudApplication.java**

```
/**  
 * 自然写互动课堂教学管理云平台软件 V1.0  
 */
```

```

* 版权所有 (C) 2026
* 软件全称: 自然写互动课堂教学管理云平台软件
* 版本号: V1.0
*
* 本文件为云平台主启动类, 负责 Spring Boot 应用初始化、
* 微服务配置加载、健康检查端点注册及全局异常处理。
*/
package com.writech.cloud;

import org.springframework.boot.SpringApplication;
import org.springframework.boot.autoconfigure.SpringBootApplication;
import org.springframework.cloud.client.discovery.EnableDiscoveryClient;
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.WebMvcConfigurer;
import org.springframework.http.ResponseEntity;
import org.springframework.web.bind.annotation.ExceptionHandler;
import org.springframework.web.bind.annotation.RestControllerAdvice;
import org.springframework.http.HttpStatus;

import java.time.LocalDateTime;
import java.util.HashMap;
import java.util.Map;

/**
 * 自然写互动课堂教学管理云平台 - 主启动类
 *
 * 系统采用微服务架构, 按领域拆分为用户服务、课堂服务、
 * 作业服务、设备服务、消息服务等多个独立微服务模块。
 * 通过 Nginx/Kong API Gateway 统一接入, 使用 Kafka
 * 进行异步消息传递, Redis 实现会话与缓存管理。
 */
@SpringBootApplication
@EnableDiscoveryClient
@EnableAsync
@EnableScheduling
public class WritechCloudApplication {

    /**
     * 应用主入口
     * 启动 Spring Boot 容器, 加载所有微服务组件
     */
    public static void main(String[] args) {
        SpringApplication.run(WritechCloudApplication.class, args);
    }

    /**
     * 跨域配置
     * 允许前端应用和各终端 APP 跨域访问云平台 API
     */
    @Configuration
    public static class CorsConfig implements WebMvcConfigurer {
        @Override
        public void addCorsMappings(CorsRegistry registry) {

```

```

        registry.addMapping("/api/**")
            .allowedOriginPatterns("*")
            .allowedMethods("GET", "POST", "PUT", "DELETE", "OPTIONS")
            .allowedHeaders("*")
            .allowCredentials(true)
            .maxAge(3600);
    }
}

/**
 * 全局异常处理器
 * 统一捕获并格式化所有未处理异常，返回标准 JSON 响应
 * 响应格式: {"code": 200, "msg": "success", "data": {...}}
 */
@RestControllerAdvice
public static class GlobalExceptionHandler {

    /**
     * 处理业务异常
     * 业务逻辑中抛出的自定义异常，返回对应的错误码和提示信息
     */
    @ExceptionHandler(BusinessException.class)
    public ResponseEntity<ApiResponse<?>> handleBusinessException(BusinessException
ex) {
        ApiResponse<?> response = ApiResponse.error(ex.getCode(), ex.getMessage());
        return ResponseEntity.status(HttpStatus.OK).body(response);
    }

    /**
     * 处理参数校验异常
     * 请求参数不符合校验规则时返回详细的校验错误信息
     */
    @ExceptionHandler(IllegalArgumentException.class)
    public ResponseEntity<ApiResponse<?>>
handleIllegalArgument(IllegalArgumentException ex) {
        ApiResponse<?> response = ApiResponse.error(400, "参数校验失败: " +
ex.getMessage());
        return ResponseEntity.status(HttpStatus.BAD_REQUEST).body(response);
    }

    /**
     * 处理未知异常
     * 兜底处理所有未预见的系统异常，记录日志并返回统一错误响应
     */
    @ExceptionHandler(Exception.class)
    public ResponseEntity<ApiResponse<?>> handleException(Exception ex) {
        ApiResponse<?> response = ApiResponse.error(500, "系统内部错误，请稍后重试");
        return
ResponseEntity.status(HttpStatus.INTERNAL_SERVER_ERROR).body(response);
    }
}

/**
 * 统一 API 响应包装类
 * 所有接口统一使用此格式返回数据
 * 格式: {"code": 200, "msg": "success", "data": {...}}
 */

```

```

public static class ApiResponse<T> {
    private int code;
    private String msg;
    private T data;
    private LocalDateTime timestamp;

    public ApiResponse() {
        this.timestamp = LocalDateTime.now();
    }

    public ApiResponse(int code, String msg, T data) {
        this.code = code;
        this.msg = msg;
        this.data = data;
        this.timestamp = LocalDateTime.now();
    }

    /** 成功响应 (带数据) */
    public static <T> ApiResponse<T> success(T data) {
        return new ApiResponse<>(200, "success", data);
    }

    /** 成功响应 (无数据) */
    public static <T> ApiResponse<T> success() {
        return new ApiResponse<>(200, "success", null);
    }

    /** 错误响应 */
    public static <T> ApiResponse<T> error(int code, String msg) {
        return new ApiResponse<>(code, msg, null);
    }

    public int getCode() { return code; }
    public void setCode(int code) { this.code = code; }
    public String getMsg() { return msg; }
    public void setMsg(String msg) { this.msg = msg; }
    public T getData() { return data; }
    public void setData(T data) { this.data = data; }
    public LocalDateTime getTimestamp() { return timestamp; }
    public void setTimestamp(LocalDateTime timestamp) { this.timestamp = timestamp; }
}

/**
 * 自定义业务异常类
 * 用于在业务逻辑中抛出可预见的异常, 包含错误码和消息
 */
public static class BusinessException extends RuntimeException {
    private final int code;

    public BusinessException(int code, String message) {
        super(message);
        this.code = code;
    }

    public int getCode() { return code; }
}

```

```
}  
}
```

**config/**

**config/KafkaConfig.java**

```
/**  
 * 自然写互动课堂教学管理云平台软件 V1.0  
 *  
 * Kafka 消息队列配置  
 * 配置笔迹数据流处理的Kafka生产者和消费者  
 */  
package com.writech.cloud.config;  
  
import org.apache.kafka.clients.consumer.ConsumerConfig;  
import org.apache.kafka.clients.producer.ProducerConfig;  
import org.apache.kafka.common.serialization.StringDeserializer;  
import org.apache.kafka.common.serialization.StringSerializer;  
  
import org.springframework.beans.factory.annotation.Value;  
import org.springframework.context.annotation.Bean;  
import org.springframework.context.annotation.Configuration;  
import org.springframework.kafka.config.ConcurrentKafkaListenerContainerFactory;  
import org.springframework.kafka.core.*;  
  
import java.util.HashMap;  
import java.util.Map;  
  
/**  
 * Kafka 配置类  
 *  
 * 消息主题定义:  
 * - writech-stroke-topic: 笔迹原始数据 (网关/算力盒 → 云平台)  
 * - writech-recognition-topic: AI识别请求 (云平台 → AI引擎)  
 * - writech-result-topic: 识别结果 (AI引擎 → 云平台)  
 * - writech-notification-topic: 通知消息 (云平台 → 终端)  
 * - writech-stroke-dlq: 笔迹数据死信队列 (处理失败的消息)  
 *  
 * 数据流向:  
 * 点阵笔 → 网关/算力盒 → Kafka(stroke-topic) → 云平台数据接收服务  
 * → MongoDB存储 → Kafka(recognition-topic) → AI引擎处理  
 * → Kafka(result-topic) → 结果回写 → WebSocket推送终端  
 */  
@Configuration  
public class KafkaConfig {  
  
    @Value("${spring.kafka.bootstrap-servers:localhost:9092}")  
    private String bootstrapServers;  
  
    @Value("${spring.kafka.consumer.group-id:writech-cloud-group}")  
    private String consumerGroupId;
```

```

/**
 * Kafka 生产者配置
 * 用于发送AI识别请求和通知消息
 */
@Bean
public ProducerFactory<String, String> producerFactory() {
    Map<String, Object> configProps = new HashMap<>();
    configProps.put(ProducerConfig.BOOTSTRAP_SERVERS_CONFIG, bootstrapServers);
    configProps.put(ProducerConfig.KEY_SERIALIZER_CLASS_CONFIG,
StringSerializer.class);
    configProps.put(ProducerConfig.VALUE_SERIALIZER_CLASS_CONFIG,
StringSerializer.class);
    // 消息可靠性配置
    configProps.put(ProducerConfig.ACKS_CONFIG, "all"); // 所有副本确认
    configProps.put(ProducerConfig.RETRIES_CONFIG, 3); // 重试3次
    configProps.put(ProducerConfig.RETRY_BACKOFF_MS_CONFIG, 1000);
    // 批量发送配置 (提升笔迹数据吞吐量)
    configProps.put(ProducerConfig.BATCH_SIZE_CONFIG, 16384); // 16KB
    configProps.put(ProducerConfig.LINGER_MS_CONFIG, 10); // 延迟10ms
    configProps.put(ProducerConfig.BUFFER_MEMORY_CONFIG, 33554432); // 32MB缓冲
    // 幂等性 (防止重复消息)
    configProps.put(ProducerConfig.ENABLE_IDEMPOTENCE_CONFIG, true);
    return new DefaultKafkaProducerFactory<>(configProps);
}

@Bean
public KafkaTemplate<String, String> kafkaTemplate() {
    return new KafkaTemplate<>(producerFactory());
}

/**
 * Kafka 消费者配置
 * 用于消费笔迹数据和识别结果
 */
@Bean
public ConsumerFactory<String, String> consumerFactory() {
    Map<String, Object> configProps = new HashMap<>();
    configProps.put(ConsumerConfig.BOOTSTRAP_SERVERS_CONFIG, bootstrapServers);
    configProps.put(ConsumerConfig.GROUP_ID_CONFIG, consumerGroupId);
    configProps.put(ConsumerConfig.KEY_DESERIALIZER_CLASS_CONFIG,
StringDeserializer.class);
    configProps.put(ConsumerConfig.VALUE_DESERIALIZER_CLASS_CONFIG,
StringDeserializer.class);
    // 消费者配置
    configProps.put(ConsumerConfig.AUTO_OFFSET_RESET_CONFIG, "latest");
    configProps.put(ConsumerConfig.ENABLE_AUTO_COMMIT_CONFIG, false); // 手动提交
    configProps.put(ConsumerConfig.MAX_POLL_RECORDS_CONFIG, 500); // 每批最多500条
    configProps.put(ConsumerConfig.FETCH_MIN_BYTES_CONFIG, 1024); // 最少1KB
    configProps.put(ConsumerConfig.FETCH_MAX_WAIT_MS_CONFIG, 200); // 最大等待200ms
    return new DefaultKafkaConsumerFactory<>(configProps);
}

/**
 * Kafka 监听器容器工厂
 * 配置并发消费者数量和批量消费模式
 */
@Bean

```

```

        public ConcurrentKafkaListenerContainerFactory<String, String>
kafkaListenerContainerFactory() {
            ConcurrentKafkaListenerContainerFactory<String, String> factory =
                new ConcurrentKafkaListenerContainerFactory<>();
            factory.setConsumerFactory(consumerFactory());
            // 并发消费者数量（对应Topic的分区数）
            factory.setConcurrency(8);
            // 启用批量消费模式
            factory.setBatchListener(true);
            // 手动确认模式
            factory.getContainerProperties().setAckMode(

org.springframework.kafka.listener.ContainerProperties.AckMode.MANUAL_IMMEDIATE);
            return factory;
        }

/**
 * 笔迹数据Topic名称常量
 */
public static class Topics {
    /** 笔迹原始数据 */
    public static final String STROKE_DATA = "writech-stroke-topic";
    /** AI识别请求 */
    public static final String RECOGNITION_REQUEST = "writech-recognition-topic";
    /** AI识别结果 */
    public static final String RECOGNITION_RESULT = "writech-result-topic";
    /** 通知消息 */
    public static final String NOTIFICATION = "writech-notification-topic";
    /** 笔迹数据死信队列 */
    public static final String STROKE_DLQ = "writech-stroke-dlq";
    /** 设备状态上报 */
    public static final String DEVICE_STATUS = "writech-device-status-topic";

    private Topics() {} // 禁止实例化
}
}

```

### config/SecurityConfig.java

```

/**
 * 自然写互动课堂教学管理云平台软件 V1.0
 *
 * 安全配置 - JWT认证过滤器 + Spring Security配置
 * 实现RBAC权限控制和全链路HTTPS/TLS 1.3加密
 */
package com.writech.cloud.config;

import com.writech.cloud.service.UserService;

import org.springframework.beans.factory.annotation.Autowired;
import org.springframework.beans.factory.annotation.Value;
import org.springframework.context.annotation.Bean;
import org.springframework.context.annotation.Configuration;
import org.springframework.security.config.annotation.web.builders.HttpSecurity;

```

```

import
org.springframework.security.config.annotation.web.configuration.EnableWebSecurity;
import org.springframework.security.config.http.SessionCreationPolicy;
import org.springframework.security.web.SecurityFilterChain;
import
org.springframework.security.web.authentication.UsernamePasswordAuthenticationFilter;

import io.jsonwebtoken.Claims;
import io.jsonwebtoken.Jwts;
import io.jsonwebtoken.security.Keys;

import javax.crypto.SecretKey;
import javax.servlet.*;
import javax.servlet.http.*;
import java.io.IOException;
import java.nio.charset.StandardCharsets;
import java.util.*;

/**
 * Spring Security 安全配置
 *
 * 安全策略:
 * - JWT Token + Refresh Token 双令牌认证机制
 * - RBAC 角色权限控制 (管理员/教师/学生/家长四级)
 * - 全链路 HTTPS/TLS 1.3 加密传输
 * - 请求签名校验 + 频率限流 + SQL注入/XSS防护
 * - 敏感字段 AES-256 加密存储
 */
@Configuration
@EnableWebSecurity
public class SecurityConfig {

    @Value("${writech.jwt.secret:writech-cloud-platform-jwt-secret-key-2026}")
    private String jwtSecret;

    @Autowired
    private UserService userService;

    /**
     * 安全过滤链配置
     * 定义各API路径的访问权限规则
     */
    @Bean
    public SecurityFilterChain filterChain(HttpSecurity http) throws Exception {
        http
            // 禁用CSRF (REST API使用JWT认证, 不需要CSRF防护)
            .csrf().disable()
            // 无状态会话 (JWT方式不使用Session)
            .sessionManagement().sessionCreationPolicy(SessionCreationPolicy.STATELESS)
            .and()
            // 路径权限配置
            .authorizeRequests()
                // 公开接口: 登录、注册、验证码、健康检查
                .antMatchers("/api/v1/auth/login").permitAll()
                .antMatchers("/api/v1/auth/sms-code").permitAll()
                .antMatchers("/api/v1/auth/refresh").permitAll()
                .antMatchers("/actuator/health").permitAll()

```



```

        .antMatchers("/ws/**").permitAll()
        // 管理员专用接口
        .antMatchers("/api/v1/admin/**").hasRole("ADMIN")
        // 教师接口
        .antMatchers("/api/v1/assignment/publish").hasAnyRole("ADMIN",
"TEACHER")
        .antMatchers("/api/v1/assignment/review/**").hasAnyRole("ADMIN",
"TEACHER")
        // 设备管理接口（管理员和教师）
        .antMatchers("/api/v1/device/**").hasAnyRole("ADMIN", "TEACHER")
        // 笔迹上传（网关/算力盒，使用设备证书认证）
        .antMatchers("/api/v1/stroke/upload").hasRole("DEVICE")
        // 其余接口需要认证
        .anyRequest().authenticated()
    .and()
    // 添加JWT认证过滤器
    .addFilterBefore(jwtAuthFilter(),
UsernamePasswordAuthenticationFilter.class)
    // 添加请求限流过滤器
    .addFilterBefore(rateLimitFilter(),
UsernamePasswordAuthenticationFilter.class);

    return http.build();
}

/**
 * JWT 认证过滤器 Bean
 */
@Bean
public JwtAuthenticationFilter jwtAuthFilter() {
    return new JwtAuthenticationFilter(jwtSecret, userService);
}

/**
 * 请求限流过滤器 Bean
 */
@Bean
public RateLimitFilter rateLimitFilter() {
    return new RateLimitFilter();
}

/**
 * JWT 认证过滤器
 *
 * 拦截所有请求，从 Authorization 头中提取并验证 JWT Token
 * 验证通过后将用户信息放入 SecurityContext
 */
public static class JwtAuthenticationFilter implements Filter {

    private final String jwtSecret;
    private final UserService userService;

    public JwtAuthenticationFilter(String jwtSecret, UserService userService) {
        this.jwtSecret = jwtSecret;
        this.userService = userService;
    }
}

```

```

@Override
public void doFilter(ServletRequest request, ServletResponse response,
                    FilterChain chain) throws IOException, ServletException {

    HttpServletRequest httpRequest = (HttpServletRequest) request;
    HttpServletResponse httpResponse = (HttpServletResponse) response;

    // 提取Token
    String authorization = httpRequest.getHeader("Authorization");
    if (authorization != null && authorization.startsWith("Bearer ")) {
        String token = authorization.substring(7);

        try {
            // 检查Token是否在黑名单中
            if (userService.isTokenBlacklisted(token)) {
                sendError(httpResponse, 401, "令牌已失效, 请重新登录");
                return;
            }

            // 解析并验证JWT
            SecretKey key = Keys.hmacShaKeyFor(
                jwtSecret.getBytes(StandardCharsets.UTF_8));
            Claims claims = Jwts.parserBuilder()
                .setSigningKey(key)
                .build()
                .parseClaimsJws(token)
                .getBody();

            // 提取用户信息
            String userId = claims.getSubject();
            String role = claims.get("role", String.class);
            String tokenType = claims.get("type", String.class);

            // 只接受access类型的Token
            if (!"access".equals(tokenType)) {
                sendError(httpResponse, 401, "无效的令牌类型");
                return;
            }

            // 将用户信息存入请求属性 (供后续Controller使用)
            httpRequest.setAttribute("userId", userId);
            httpRequest.setAttribute("role", role);

        } catch (io.jsonwebtoken.ExpiredJwtException e) {
            sendError(httpResponse, 401, "令牌已过期, 请刷新令牌");
            return;
        } catch (Exception e) {
            sendError(httpResponse, 401, "令牌校验失败");
            return;
        }
    }

    chain.doFilter(request, response);
}

/** 发送错误响应 */
private void sendError(HttpServletResponse response, int code, String message)

```

```

        throws IOException {
            response.setStatus(code);
            response.setContentType("application/json;charset=UTF-8");
            response.getWriter().write(
                "{\"code\":\"" + code + "\",\"msg\":\"" + message +
"\",\"data\":null}");
        }
    }

/**
 * 请求限流过滤器
 *
 * 基于IP和用户ID的双维度限流
 * - IP维度：每分钟最多60次请求
 * - 用户维度：每分钟最多120次请求
 * - 敏感接口（登录/发送验证码）：更严格的限流策略
 */
public static class RateLimitFilter implements Filter {

    /** IP请求计数器（简化实现，生产环境使用Redis+滑动窗口） */
    private final Map<String, List<Long>> ipRequestLog = new HashMap<>();

    /** IP限流阈值（每分钟） */
    private static final int IP_RATE_LIMIT = 60;

    /** 时间窗口（毫秒） */
    private static final long WINDOW_MS = 60_000;

    @Override
    public void doFilter(ServletRequest request, ServletResponse response,
        FilterChain chain) throws IOException, ServletException {

        HttpServletRequest httpRequest = (HttpServletRequest) request;
        HttpServletResponse httpResponse = (HttpServletResponse) response;

        String clientId = getClientIp(httpRequest);
        long now = System.currentTimeMillis();

        // IP维度限流检查
        synchronized (ipRequestLog) {
            List<Long> timestamps = ipRequestLog.computeIfAbsent(
                clientId, k -> new ArrayList<>());

            // 清理窗口外的记录
            timestamps.removeIf(ts -> (now - ts) > WINDOW_MS);

            if (timestamps.size() >= IP_RATE_LIMIT) {
                httpResponse.setStatus(429);
                httpResponse.setContentType("application/json;charset=UTF-8");
                httpResponse.getWriter().write(
                    "{\"code\":\"429\",\"msg\":\"请求频率过高，请稍后重试
\", \"data\":null}");
                return;
            }

            timestamps.add(now);
        }
    }
}

```

```

        chain.doFilter(request, response);
    }

    /** 获取客户端真实IP (考虑代理/负载均衡) */
    private String getClientIp(HttpServletRequest request) {
        String ip = request.getHeader("X-Forwarded-For");
        if (ip == null || ip.isEmpty() || "unknown".equalsIgnoreCase(ip)) {
            ip = request.getHeader("X-Real-IP");
        }
        if (ip == null || ip.isEmpty() || "unknown".equalsIgnoreCase(ip)) {
            ip = request.getRemoteAddr();
        }
        // X-Forwarded-For可能包含多个IP, 取第一个
        if (ip != null && ip.contains(",")) {
            ip = ip.split(",")[0].trim();
        }
        return ip;
    }
}

```

## controller/

### controller/AssignmentController.java

```

/**
 * 自然写互动课堂教学管理云平台软件 V1.0
 *
 * 作业管理控制器
 * 负责作业/试卷的发布、回收、批改结果查询等接口
 */
package com.writech.cloud.controller;

import com.writech.cloud.WritechCloudApplication.ApiResponse;
import com.writech.cloud.WritechCloudApplication.BusinessException;
import com.writech.cloud.model.Assignment;
import com.writech.cloud.service.UserService;

import org.springframework.beans.factory.annotation.Autowired;
import org.springframework.data.domain.Page;
import org.springframework.data.domain.PageRequest;
import org.springframework.web.bind.annotation.*;

import javax.validation.Valid;
import javax.validation.constraints.NotBlank;
import java.time.LocalDateTime;
import java.util.*;

/**
 * 作业控制器 - /api/v1/assignment
 *
 * 教师发布作业/试卷 → 学生纸上作答 (笔迹通过点阵笔采集)
 */

```

\* → 系统自动收集 → AI引擎识别批改 → 结果推送教师和家长

\*/

@RestController

@RequestMapping("/api/v1/assignment")

public class AssignmentController {

@Autowired

private UserService userService;

/\*\*

\* 发布作业

\* POST /api/v1/assignment/publish

\*

\* 教师创建并发布作业/试卷，指定班级、截止时间、题目内容

\* 发布后自动推送通知至学生端和家长端

\*/

@PostMapping("/publish")

public ApiResponse<AssignmentPublishResponse> publishAssignment(

@Valid @RequestBody AssignmentPublishRequest request,

@RequestHeader("Authorization") String auth) {

// 验证教师身份

String teacherId = extractUserIdFromToken(auth);

// 校验截止时间

if (request.getDeadline() != null &&  
request.getDeadline().isBefore(LocalDate.now())) {  
 throw new BusinessException(400, "截止时间不能早于当前时间");  
}

// 校验题目列表

if (request.getQuestions() == null || request.getQuestions().isEmpty()) {  
 throw new BusinessException(400, "作业题目不能为空");  
}

// 创建作业记录

Assignment assignment = new Assignment();  
assignment.setId(UUID.randomUUID().toString().replace("-", ""));  
assignment.setTeacherId(teacherId);  
assignment.setClassId(request.getClassId());  
assignment.setTitle(request.getTitle());  
assignment.setType(request.getType()); // homework/exam/practice  
assignment.setSubject(request.getSubject());  
assignment.setDeadline(request.getDeadline());  
assignment.setStatus("published");  
assignment.setPublishTime(LocalDate.now());  
assignment.setTotalScore(calculateTotalScore(request.getQuestions()));  
assignment.setQuestionCount(request.getQuestions().size());

// 关联点阵码页面（每道题对应特定点阵码区域）

if (request.getDotCodePages() != null) {  
 assignment.setDotCodePages(request.getDotCodePages());  
}

// 保存作业及题目

// assignmentService.saveWithQuestions(assignment, request.getQuestions());

```

        // 异步推送通知至学生端和家长端
        // messageService.pushAssignmentNotification(assignment);

        AssignmentPublishResponse response = new AssignmentPublishResponse();
        response.setAssignmentId(assignment.getId());
        response.setTitle(assignment.getTitle());
        response.setPublishTime(assignment.getPublishTime());
        response.setStudentCount(getClassStudentCount(request.getClassId()));

        return ApiResponse.success(response);
    }

    /**
     * 获取作业列表
     * GET /api/v1/assignment/list
     *
     * 教师查看已发布的作业列表，支持按班级、状态、时间筛选
     */
    @GetMapping("/list")
    public ApiResponse<Page<AssignmentSummary>> listAssignments(
        @RequestParam(required = false) String classId,
        @RequestParam(required = false) String status,
        @RequestParam(required = false) String subject,
        @RequestParam(defaultValue = "0") int page,
        @RequestParam(defaultValue = "20") int size,
        @RequestHeader("Authorization") String auth) {

        String userId = extractUserIdFromToken(auth);
        // Page<AssignmentSummary> result = assignmentService.queryList(...)
        return ApiResponse.success(null);
    }

    /**
     * 获取作业详情
     * GET /api/v1/assignment/{id}
     */
    @GetMapping("/{id}")
    public ApiResponse<AssignmentDetailResponse> getAssignment(@PathVariable String id)
    {
        // Assignment assignment = assignmentService.findById(id);
        return ApiResponse.success(null);
    }

    /**
     * 获取批改结果
     * GET /api/v1/result/{assignmentId}
     *
     * 查询指定作业的AI批改结果，包含每个学生的识别文本、
     * 得分、错误详情及AI反馈建议
     */
    @GetMapping("/result/{assignmentId}")
    public ApiResponse<AssignmentResultResponse> getResult(
        @PathVariable String assignmentId,
        @RequestParam(required = false) String studentId) {

        AssignmentResultResponse response = new AssignmentResultResponse();
        response.setAssignmentId(assignmentId);

```

```

        response.setTotalStudents(40);
        response.setSubmittedCount(38);
        response.setGradedCount(38);
        response.setAverageScore(85.5);
        response.setHighestScore(100.0);
        response.setLowestScore(45.0);

        // 每个学生的批改结果
        List<StudentResult> studentResults = new ArrayList<>();
        // studentResults = resultService.getStudentResults(assignmentId, studentId);
        response.setStudentResults(studentResults);

        return ApiResponse.success(response);
    }

    /**
     * 教师人工复核批改
     * PUT /api/v1/assignment/review/{assignmentId}
     *
     * AI批改后教师可进行人工复核，修正AI评分或添加评语
     */
    @PutMapping("/review/{assignmentId}")
    public ApiResponse<Void> reviewAssignment(
        @PathVariable String assignmentId,
        @Valid @RequestBody ReviewRequest request,
        @RequestHeader("Authorization") String auth) {

        String teacherId = extractUserIdFromToken(auth);

        // 遍历教师的复核修改
        for (ReviewItem item : request.getReviewItems()) {
            // resultService.updateReview(assignmentId, item.getStudentId(),
            // item.getQuestionId(), item.getManualScore(),
            // item.getTeacherComment(), teacherId);
        }

        return ApiResponse.success();
    }

    /**
     * 学情报告接口
     * GET /api/v1/report/student/{id}
     *
     * 获取指定学生的学情报告，包含知识点掌握度、
     * 书写能力评估、成绩趋势等多维度分析数据
     */
    @GetMapping("/report/student/{studentId}")
    public ApiResponse<StudentReportResponse> getStudentReport(
        @PathVariable String studentId,
        @RequestParam(required = false) String subject,
        @RequestParam(required = false) String dateRange) {

        StudentReportResponse report = new StudentReportResponse();
        report.setStudentId(studentId);
        report.setReportDate(LocalDate.now());

        // 知识点掌握度

```

```

        List<KnowledgePoint> knowledgePoints = new ArrayList<>();
        // knowledgePoints = analyticsService.getKnowledgeMastery(studentId, subject);
        report.setKnowledgePoints(knowledgePoints);

        // 书写能力评估
        WritingAbility writingAbility = new WritingAbility();
        writingAbility.setStrokeOrderScore(88.5);
        writingAbility.setStructureScore(82.3);
        writingAbility.setNeatnessScore(90.1);
        writingAbility.setOverallScore(86.9);
        report.setWritingAbility(writingAbility);

        return ApiResponse.success(report);
    }

    // ===== 内部方法 =====

    private String extractUserIdFromToken(String auth) {
        // 从JWT Token解析用户ID
        return "teacher_001";
    }

    private double calculateTotalScore(List<QuestionItem> questions) {
        return questions.stream()
            .mapToDouble(QuestionItem::getScore)
            .sum();
    }

    private int getClassStudentCount(String classId) {
        return 40; // 查询班级学生数
    }

    // ===== DTO 定义 =====

    public static class AssignmentPublishRequest {
        @NotBlank private String classId;
        @NotBlank private String title;
        private String type; // homework/exam/practice
        private String subject;
        private LocalDateTime deadline;
        private List<QuestionItem> questions;
        private List<String> dotCodePages; // 关联的点阵码页面ID

        public String getClassId() { return classId; }
        public void setClassId(String id) { this.classId = id; }
        public String getTitle() { return title; }
        public void setTitle(String t) { this.title = t; }
        public String getType() { return type; }
        public void setType(String t) { this.type = t; }
        public String getSubject() { return subject; }
        public void setSubject(String s) { this.subject = s; }
        public LocalDateTime getDeadline() { return deadline; }
        public void setDeadline(LocalDateTime d) { this.deadline = d; }
        public List<QuestionItem> getQuestions() { return questions; }
        public void setQuestions(List<QuestionItem> q) { this.questions = q; }
        public List<String> getDotCodePages() { return dotCodePages; }
        public void setDotCodePages(List<String> p) { this.dotCodePages = p; }
    }

```



```

}

public static class QuestionItem {
    private int questionNo;
    private String type;        // choice/fill/short_answer/essay/math
    private String content;
    private String answer;
    private double score;
    private String knowledgePointId;

    public int getQuestionNo() { return questionNo; }
    public void setQuestionNo(int n) { this.questionNo = n; }
    public String getType() { return type; }
    public void setType(String t) { this.type = t; }
    public String getContent() { return content; }
    public void setContent(String c) { this.content = c; }
    public String getAnswer() { return answer; }
    public void setAnswer(String a) { this.answer = a; }
    public double getScore() { return score; }
    public void setScore(double s) { this.score = s; }
    public String getKnowledgePointId() { return knowledgePointId; }
    public void setKnowledgePointId(String id) { this.knowledgePointId = id; }
}

public static class AssignmentPublishResponse {
    private String assignmentId;
    private String title;
    private LocalDateTime publishTime;
    private int studentCount;

    public String getAssignmentId() { return assignmentId; }
    public void setAssignmentId(String id) { this.assignmentId = id; }
    public String getTitle() { return title; }
    public void setTitle(String t) { this.title = t; }
    public LocalDateTime getPublishTime() { return publishTime; }
    public void setPublishTime(LocalDateTime t) { this.publishTime = t; }
    public int getStudentCount() { return studentCount; }
    public void setStudentCount(int c) { this.studentCount = c; }
}

public static class AssignmentSummary {
    private String id;
    private String title;
    private String type;
    private String status;
    private int submittedCount;
    private int totalCount;
    private LocalDateTime publishTime;

    public String getId() { return id; }
    public void setId(String id) { this.id = id; }
    public String getTitle() { return title; }
    public void setTitle(String t) { this.title = t; }
    public String getType() { return type; }
    public void setType(String t) { this.type = t; }
    public String getStatus() { return status; }
    public void setStatus(String s) { this.status = s; }
}

```

```

        public int getSubmittedCount() { return submittedCount; }
        public void setSubmittedCount(int c) { this.submittedCount = c; }
        public int getTotalCount() { return totalCount; }
        public void setTotalCount(int c) { this.totalCount = c; }
        public LocalDateTime getPublishTime() { return publishTime; }
        public void setPublishTime(LocalDateTime t) { this.publishTime = t; }
    }

    public static class AssignmentDetailResponse {
        private Assignment assignment;
        private List<QuestionItem> questions;
        public Assignment getAssignment() { return assignment; }
        public void setAssignment(Assignment a) { this.assignment = a; }
        public List<QuestionItem> getQuestions() { return questions; }
        public void setQuestions(List<QuestionItem> q) { this.questions = q; }
    }

    public static class AssignmentResultResponse {
        private String assignmentId;
        private int totalStudents;
        private int submittedCount;
        private int gradedCount;
        private double averageScore;
        private double highestScore;
        private double lowestScore;
        private List<StudentResult> studentResults;

        public String getAssignmentId() { return assignmentId; }
        public void setAssignmentId(String id) { this.assignmentId = id; }
        public int getTotalStudents() { return totalStudents; }
        public void setTotalStudents(int c) { this.totalStudents = c; }
        public int getSubmittedCount() { return submittedCount; }
        public void setSubmittedCount(int c) { this.submittedCount = c; }
        public int getGradedCount() { return gradedCount; }
        public void setGradedCount(int c) { this.gradedCount = c; }
        public double getAverageScore() { return averageScore; }
        public void setAverageScore(double s) { this.averageScore = s; }
        public double getHighestScore() { return highestScore; }
        public void setHighestScore(double s) { this.highestScore = s; }
        public double getLowestScore() { return lowestScore; }
        public void setLowestScore(double s) { this.lowestScore = s; }
        public List<StudentResult> getStudentResults() { return studentResults; }
        public void setStudentResults(List<StudentResult> r) { this.studentResults = r; }
    }

    public static class StudentResult {
        private String studentId;
        private String studentName;
        private double totalScore;
        private List<QuestionResult> questionResults;

        public String getStudentId() { return studentId; }
        public void setStudentId(String id) { this.studentId = id; }
        public String getStudentName() { return studentName; }
        public void setStudentName(String n) { this.studentName = n; }
        public double getTotalScore() { return totalScore; }
    }

```

```

        public void setTotalScore(double s) { this.totalScore = s; }
        public List<QuestionResult> getQuestionResults() { return questionResults; }
        public void setQuestionResults(List<QuestionResult> r) { this.questionResults =
r; }
    }

    public static class QuestionResult {
        private int questionNo;
        private String ocrText;
        private double score;
        private boolean isCorrect;
        private String aiFeedback;

        public int getQuestionNo() { return questionNo; }
        public void setQuestionNo(int n) { this.questionNo = n; }
        public String getOcrText() { return ocrText; }
        public void setOcrText(String t) { this.ocrText = t; }
        public double getScore() { return score; }
        public void setScore(double s) { this.score = s; }
        public boolean isCorrect() { return isCorrect; }
        public void setCorrect(boolean c) { this.isCorrect = c; }
        public String getAiFeedback() { return aiFeedback; }
        public void setAiFeedback(String f) { this.aiFeedback = f; }
    }

    public static class ReviewRequest {
        private List<ReviewItem> reviewItems;
        public List<ReviewItem> getReviewItems() { return reviewItems; }
        public void setReviewItems(List<ReviewItem> items) { this.reviewItems = items; }
    }

    public static class ReviewItem {
        private String studentId;
        private int questionId;
        private Double manualScore;
        private String teacherComment;

        public String getStudentId() { return studentId; }
        public void setStudentId(String id) { this.studentId = id; }
        public int getQuestionId() { return questionId; }
        public void setQuestionId(int id) { this.questionId = id; }
        public Double getManualScore() { return manualScore; }
        public void setManualScore(Double s) { this.manualScore = s; }
        public String getTeacherComment() { return teacherComment; }
        public void setTeacherComment(String c) { this.teacherComment = c; }
    }

    public static class StudentReportResponse {
        private String studentId;
        private LocalDateTime reportDate;
        private List<KnowledgePoint> knowledgePoints;
        private WritingAbility writingAbility;

        public String getStudentId() { return studentId; }
        public void setStudentId(String id) { this.studentId = id; }
        public LocalDateTime getReportDate() { return reportDate; }
        public void setReportDate(LocalDateTime d) { this.reportDate = d; }
    }

```

```

        public List<KnowledgePoint> getKnowledgePoints() { return knowledgePoints; }
        public void setKnowledgePoints(List<KnowledgePoint> kp) { this.knowledgePoints =
kp; }

        public WritingAbility getWritingAbility() { return writingAbility; }
        public void setWritingAbility(WritingAbility wa) { this.writingAbility = wa; }
    }

    public static class KnowledgePoint {
        private String id;
        private String name;
        private double masteryRate;
        public String getId() { return id; }
        public void setId(String id) { this.id = id; }
        public String getName() { return name; }
        public void setName(String n) { this.name = n; }
        public double getMasteryRate() { return masteryRate; }
        public void setMasteryRate(double r) { this.masteryRate = r; }
    }

    public static class WritingAbility {
        private double strokeOrderScore;
        private double structureScore;
        private double neatnessScore;
        private double overallScore;

        public double getStrokeOrderScore() { return strokeOrderScore; }
        public void setStrokeOrderScore(double s) { this.strokeOrderScore = s; }
        public double getStructureScore() { return structureScore; }
        public void setStructureScore(double s) { this.structureScore = s; }
        public double getNeatnessScore() { return neatnessScore; }
        public void setNeatnessScore(double s) { this.neatnessScore = s; }
        public double getOverallScore() { return overallScore; }
        public void setOverallScore(double s) { this.overallScore = s; }
    }
}

```

### controller/AuthController.java

```

/**
 * 自然写互动课堂教学管理云平台软件 V1.0
 *
 * 用户认证控制器
 * 负责用户登录、登出、Token刷新等认证相关接口
 * 采用 JWT Token + Refresh Token 双令牌机制
 */
package com.writech.cloud.controller;

import com.writech.cloud.WritechCloudApplication.ApiResponse;
import com.writech.cloud.WritechCloudApplication.BusinessException;
import com.writech.cloud.model.User;
import com.writech.cloud.service.UserService;

import org.springframework.beans.factory.annotation.Autowired;
import org.springframework.beans.factory.annotation.Value;

```

```

import org.springframework.web.bind.annotation.*;

import io.jsonwebtoken.Jwts;
import io.jsonwebtoken.SignatureAlgorithm;
import io.jsonwebtoken.Claims;
import io.jsonwebtoken.security.Keys;

import javax.crypto.SecretKey;
import javax.validation.Valid;
import javax.validation.constraints.NotBlank;
import java.nio.charset.StandardCharsets;
import java.util.*;
import java.time.LocalDateTime;

/**
 * 认证控制器 - /api/v1/auth
 *
 * 实现教师/学生/管理员/家长多角色用户的统一认证
 * 支持手机号+密码、手机号+验证码、微信/钉钉第三方登录
 */
@RestController
@RequestMapping("/api/v1/auth")
public class AuthController {

    @Autowired
    private UserService userService;

    /** JWT密钥 */
    @Value("${writech.jwt.secret:writech-cloud-platform-jwt-secret-key-2026}")
    private String jwtSecret;

    /** Access Token 有效期 (秒), 默认2小时 */
    @Value("${writech.jwt.access-token-expire:7200}")
    private long accessTokenExpire;

    /** Refresh Token 有效期 (秒), 默认7天 */
    @Value("${writech.jwt.refresh-token-expire:604800}")
    private long refreshTokenExpire;

    /**
     * 用户登录接口
     * POST /api/v1/auth/login
     *
     * 验证用户身份, 签发 JWT Access Token 和 Refresh Token
     * Access Token 有效期2小时, Refresh Token 有效期7天
     *
     * @param request 登录请求 (包含手机号、密码/验证码、登录方式)
     * @return 包含双令牌和用户基本信息的响应
     */
    @PostMapping("/login")
    public ApiResponse<LoginResponse> login(@Valid @RequestBody LoginRequest request) {
        // 校验登录参数
        if (request.getLoginType() == null) {
            throw new BusinessException(400, "登录方式不能为空");
        }

        User user = null;

```

```

        // 根据不同登录方式验证身份
        switch (request.getLoginType()) {
            case "password":
                // 手机号 + 密码登录
                user = userService.verifyByPassword(request.getPhone(),
request.getPassword());
                break;
            case "sms":
                // 手机号 + 短信验证码登录
                user = userService.verifyBySmsCode(request.getPhone(),
request.getSmsCode());
                break;
            case "wechat":
                // 微信授权登录
                user = userService.verifyByWechat(request.getWechatCode());
                break;
            case "dingtalk":
                // 钉钉授权登录
                user = userService.verifyByDingtalk(request.getDingtalkCode());
                break;
            default:
                throw new BusinessException(400, "不支持的登录方式: " +
request.getLoginType());
        }

        if (user == null) {
            throw new BusinessException(401, "登录失败, 用户名或密码错误");
        }

        // 检查用户状态
        if (user.getStatus() != 1) {
            throw new BusinessException(403, "账户已被禁用, 请联系管理员");
        }

        // 生成双令牌
        String accessToken = generateAccessToken(user);
        String refreshToken = generateRefreshToken(user);

        // 更新用户最后登录时间和登录IP
        userService.updateLoginInfo(user.getId(), LocalDateTime.now(),
request.getClientIp());

        // 构建登录响应
        LoginResponse response = new LoginResponse();
        response.setAccessToken(accessToken);
        response.setRefreshToken(refreshToken);
        response.setExpiresIn(accessTokenExpire);
        response.setUserId(user.getId());
        response.setUserName(user.getName());
        response.setRole(user.getRole());
        response.setSchoolId(user.getSchoolId());
        response.setSchoolName(user.getSchoolName());

        return ApiResponse.success(response);
    }
}

```

```

/**
 * Token 刷新接口
 * POST /api/v1/auth/refresh
 *
 * 使用 Refresh Token 换取新的 Access Token
 * 避免用户频繁重新登录, 提升使用体验
 *
 * @param request 刷新请求 (包含 Refresh Token)
 * @return 新的 Access Token
 */
@PostMapping("/refresh")
public ApiResponse<TokenRefreshResponse> refreshToken(@Valid @RequestBody
TokenRefreshRequest request) {
    try {
        // 解析并验证 Refresh Token
        Claims claims = parseToken(request.getRefreshToken());
        String userId = claims.getSubject();
        String tokenType = claims.get("type", String.class);

        // 确保是 Refresh Token 类型
        if (!"refresh".equals(tokenType)) {
            throw new BusinessException(401, "无效的刷新令牌");
        }

        // 查询用户信息 (确保用户仍然有效)
        User user = userService.findById(userId);
        if (user == null || user.getStatus() != 1) {
            throw new BusinessException(401, "用户不存在或已被禁用");
        }

        // 生成新的 Access Token
        String newAccessToken = generateAccessToken(user);

        TokenRefreshResponse response = new TokenRefreshResponse();
        response.setAccessToken(newAccessToken);
        response.setExpiresIn(accessTokenExpire);

        return ApiResponse.success(response);
    } catch (Exception e) {
        throw new BusinessException(401, "令牌刷新失败: " + e.getMessage());
    }
}

/**
 * 用户登出接口
 * POST /api/v1/auth/logout
 *
 * 将当前 Token 加入黑名单, 使其立即失效
 * 同时清除 Redis 中的会话缓存
 */
@PostMapping("/logout")
public ApiResponse<Void> logout(@RequestHeader("Authorization") String
authorization) {
    String token = extractToken(authorization);
    if (token != null) {
        // 将Token加入Redis黑名单, 使其立即失效
        userService.invalidateToken(token);
    }
}

```

```

    }
    return ApiResponse.success();
}

/**
 * 发送短信验证码
 * POST /api/v1/auth/sms-code
 *
 * 向指定手机号发送登录验证码，验证码5分钟内有效
 * 同一手机号60秒内只能发送一次
 */
@PostMapping("/sms-code")
public ApiResponse<Void> sendSmsCode(@RequestBody SmsCodeRequest request) {
    if (request.getPhone() == null || request.getPhone().length() != 11) {
        throw new BusinessException(400, "请输入正确的手机号");
    }
    userService.sendSmsVerificationCode(request.getPhone());
    return ApiResponse.success();
}

/**
 * 获取当前登录用户信息
 * GET /api/v1/auth/profile
 *
 * 根据 Token 中的用户ID查询完整的用户信息
 * 包括角色、学校、班级等关联信息
 */
@GetMapping("/profile")
public ApiResponse<UserProfileResponse> getProfile(@RequestHeader("Authorization")
String authorization) {
    String token = extractToken(authorization);
    Claims claims = parseToken(token);
    String userId = claims.getSubject();

    User user = userService.findById(userId);
    if (user == null) {
        throw new BusinessException(404, "用户不存在");
    }

    UserProfileResponse profile = new UserProfileResponse();
    profile.setUserId(user.getId());
    profile.setName(user.getName());
    profile.setPhone(maskPhone(user.getPhone()));
    profile.setRole(user.getRole());
    profile.setSchoolId(user.getSchoolId());
    profile.setSchoolName(user.getSchoolName());
    profile.setAvatar(user.getAvatar());
    profile.setLastLoginTime(user.getLastLoginTime());

    return ApiResponse.success(profile);
}

/**
 * 修改密码
 * PUT /api/v1/auth/password
 */
@PutMapping("/password")

```



```

    public ApiResponse<Void> changePassword(@RequestHeader("Authorization") String
authorization,

                                           @Valid @RequestBody ChangePasswordRequest
request) {
    String token = extractToken(authorization);
    Claims claims = parseToken(token);
    String userId = claims.getSubject();

    // 验证旧密码
    boolean verified = userService.verifyPassword(userId, request.getOldPassword());
    if (!verified) {
        throw new BusinessException(400, "原密码错误");
    }

    // 更新密码
    userService.updatePassword(userId, request.getNewPassword());
    // 使所有现有Token失效, 强制重新登录
    userService.invalidateAllTokens(userId);

    return ApiResponse.success();
}

// ===== 内部方法 =====

/**
 * 生成 Access Token
 * 有效期2小时, 包含用户ID、角色、学校信息
 */
private String generateAccessToken(User user) {
    SecretKey key = Keys.hmacShaKeyFor(jwtSecret.getBytes(StandardCharsets.UTF_8));
    Date now = new Date();
    Date expiry = new Date(now.getTime() + accessTokenExpire * 1000);

    return Jwts.builder()
        .setSubject(user.getId())
        .claim("role", user.getRole())
        .claim("schoolId", user.getSchoolId())
        .claim("type", "access")
        .setIssuedAt(now)
        .setExpiration(expiry)
        .signWith(key, SignatureAlgorithm.HS256)
        .compact();
}

/**
 * 生成 Refresh Token
 * 有效期7天, 仅包含用户ID和令牌类型
 */
private String generateRefreshToken(User user) {
    SecretKey key = Keys.hmacShaKeyFor(jwtSecret.getBytes(StandardCharsets.UTF_8));
    Date now = new Date();
    Date expiry = new Date(now.getTime() + refreshTokenExpire * 1000);

    return Jwts.builder()
        .setSubject(user.getId())
        .claim("type", "refresh")
        .setIssuedAt(now)

```

```

        .setExpiration(expiry)
        .signWith(key, SignatureAlgorithm.HS256)
        .compact();
    }

    /** 解析 JWT Token */
    private Claims parseToken(String token) {
        SecretKey key = Keys.hmacShaKeyFor(jwtSecret.getBytes(StandardCharsets.UTF_8));
        return Jwts.parserBuilder().setSigningKey(key).build()
            .parseClaimsJws(token).getBody();
    }

    /** 从 Authorization 头中提取 Token */
    private String extractToken(String authorization) {
        if (authorization != null && authorization.startsWith("Bearer ")) {
            return authorization.substring(7);
        }
        return null;
    }

    /** 手机号脱敏处理 (中间4位替换为****) */
    private String maskPhone(String phone) {
        if (phone == null || phone.length() != 11) return phone;
        return phone.substring(0, 3) + "****" + phone.substring(7);
    }

    // ===== 请求/响应 DTO =====

    /** 登录请求 */
    public static class LoginRequest {
        @NotBlank(message = "登录方式不能为空")
        private String loginType;    // password/sms/wechat/dingtalk
        private String phone;
        private String password;
        private String smsCode;
        private String wechatCode;
        private String dingtalkCode;
        private String clientId;

        public String getLoginType() { return loginType; }
        public void setLoginType(String loginType) { this.loginType = loginType; }
        public String getPhone() { return phone; }
        public void setPhone(String phone) { this.phone = phone; }
        public String getPassword() { return password; }
        public void setPassword(String password) { this.password = password; }
        public String getSmsCode() { return smsCode; }
        public void setSmsCode(String smsCode) { this.smsCode = smsCode; }
        public String getWechatCode() { return wechatCode; }
        public void setWechatCode(String wechatCode) { this.wechatCode = wechatCode; }
        public String getDingtalkCode() { return dingtalkCode; }
        public void setDingtalkCode(String dingtalkCode) { this.dingtalkCode =
dingtalkCode; }
        public String getClientId() { return clientId; }
        public void setClientId(String clientId) { this.clientId = clientId; }
    }

    /** 登录响应 */

```

```

public static class LoginResponse {
    private String accessToken;
    private String refreshToken;
    private long expiresIn;
    private String userId;
    private String userName;
    private String role;
    private String schoolId;
    private String schoolName;

    public String getAccessToken() { return accessToken; }
    public void setAccessToken(String t) { this.accessToken = t; }
    public String getRefreshToken() { return refreshToken; }
    public void setRefreshToken(String t) { this.refreshToken = t; }
    public long getExpiresIn() { return expiresIn; }
    public void setExpiresIn(long e) { this.expiresIn = e; }
    public String getUserId() { return userId; }
    public void setUserId(String id) { this.userId = id; }
    public String getUserName() { return userName; }
    public void setUserName(String n) { this.userName = n; }
    public String getRole() { return role; }
    public void setRole(String r) { this.role = r; }
    public String getSchoolId() { return schoolId; }
    public void setSchoolId(String id) { this.schoolId = id; }
    public String getSchoolName() { return schoolName; }
    public void setSchoolName(String n) { this.schoolName = n; }
}

/** Token刷新请求 */
public static class TokenRefreshRequest {
    @NotBlank(message = "刷新令牌不能为空")
    private String refreshToken;
    public String getRefreshToken() { return refreshToken; }
    public void setRefreshToken(String t) { this.refreshToken = t; }
}

/** Token刷新响应 */
public static class TokenRefreshResponse {
    private String accessToken;
    private long expiresIn;
    public String getAccessToken() { return accessToken; }
    public void setAccessToken(String t) { this.accessToken = t; }
    public long getExpiresIn() { return expiresIn; }
    public void setExpiresIn(long e) { this.expiresIn = e; }
}

/** 短信验证码请求 */
public static class SmsCodeRequest {
    private String phone;
    public String getPhone() { return phone; }
    public void setPhone(String p) { this.phone = p; }
}

/** 用户信息响应 */
public static class UserProfileResponse {
    private String userId;
    private String name;

```

```

        private String phone;
        private String role;
        private String schoolId;
        private String schoolName;
        private String avatar;
        private LocalDateTime lastLoginTime;

        public String getUserId() { return userId; }
        public void setUserId(String id) { this.userId = id; }
        public String getName() { return name; }
        public void setName(String n) { this.name = n; }
        public String getPhone() { return phone; }
        public void setPhone(String p) { this.phone = p; }
        public String getRole() { return role; }
        public void setRole(String r) { this.role = r; }
        public String getSchoolId() { return schoolId; }
        public void setSchoolId(String id) { this.schoolId = id; }
        public String getSchoolName() { return schoolName; }
        public void setSchoolName(String n) { this.schoolName = n; }
        public String getAvatar() { return avatar; }
        public void setAvatar(String a) { this.avatar = a; }
        public LocalDateTime getLastLoginTime() { return lastLoginTime; }
        public void setLastLoginTime(LocalDateTime t) { this.lastLoginTime = t; }
    }

    /** 修改密码请求 */
    public static class ChangePasswordRequest {
        @NotBlank(message = "原密码不能为空")
        private String oldPassword;
        @NotBlank(message = "新密码不能为空")
        private String newPassword;
        public String getOldPassword() { return oldPassword; }
        public void setOldPassword(String p) { this.oldPassword = p; }
        public String getNewPassword() { return newPassword; }
        public void setNewPassword(String p) { this.newPassword = p; }
    }
}

```

## controller/DeviceController.java

```

/**
 * 自然写互动课堂教学管理云平台软件 V1.0
 *
 * 设备管理控制器
 * 负责点阵笔、网关、终端设备的注册、绑定、状态查询等接口
 */
package com.writech.cloud.controller;

import com.writech.cloud.WritechCloudApplication.ApiResponse;
import com.writech.cloud.WritechCloudApplication.BusinessException;
import com.writech.cloud.model.Device;
import com.writech.cloud.service.DeviceService;

import org.springframework.beans.factory.annotation.Autowired;

```

```

import org.springframework.data.domain.Page;
import org.springframework.data.domain.PageRequest;
import org.springframework.web.bind.annotation.*;

import javax.validation.Valid;
import javax.validation.constraints.NotBlank;
import java.time.LocalDateTime;
import java.util.*;

/**
 * 设备控制器 - /api/v1/device
 *
 * 管理互动课堂中涉及的所有智能硬件设备：
 * - 点阵笔 (pen)：学生书写工具，通过BLE连接网关
 * - 网关设备 (gateway)：教室中枢，管理多支笔的连接与数据转发
 * - 终端设备 (terminal)：黑板、PC、电视、平板等显示终端
 * - 算力盒 (edge_box)：教室端AI推理设备
 */
@RestController
@RequestMapping("/api/v1/device")
public class DeviceController {

    @Autowired
    private DeviceService deviceService;

    /**
     * 设备注册接口
     * POST /api/v1/device/register
     *
     * 将新设备注册到云平台，绑定至指定用户和学校
     * 注册时校验设备MAC地址唯一性和设备证书有效性
     *
     * @param request 注册请求 (MAC地址、设备类型、序列号等)
     * @return 注册成功后的设备信息
     */
    @PostMapping("/register")
    public ApiResponse<DeviceRegisterResponse> registerDevice(
        @Valid @RequestBody DeviceRegisterRequest request) {

        // 校验设备MAC地址格式
        if (!isValidMacAddress(request.getMacAddr())) {
            throw new BusinessException(400, "无效的MAC地址格式");
        }

        // 检查设备是否已注册
        Device existing = deviceService.findByMacAddr(request.getMacAddr());
        if (existing != null) {
            throw new BusinessException(409, "设备已注册, MAC地址: " +
request.getMacAddr());
        }

        // 校验设备证书 (X.509)
        boolean certValid = deviceService.validateDeviceCertificate(
            request.getMacAddr(), request.getDeviceCert());
        if (!certValid) {
            throw new BusinessException(403, "设备证书校验失败, 拒绝注册");
        }
    }
}

```

```

// 创建设备记录
Device device = new Device();
device.setId(UUID.randomUUID().toString().replace("-", ""));
device.setType(request.getDeviceType());
device.setMacAddr(request.getMacAddr());
device.setSerialNumber(request.getSerialNumber());
device.setFirmwareVersion(request.getFirmwareVersion());
device.setBindUserId(request.getUserId());
device.setSchoolId(request.getSchoolId());
device.setClassroomId(request.getClassroomId());
device.setStatus(1); // 1=在线
device.setRegisterTime(LocalDateTime.now());
device.setLastHeartbeat(LocalDateTime.now());

deviceService.save(device);

// 返回注册结果
DeviceRegisterResponse response = new DeviceRegisterResponse();
response.setDeviceId(device.getId());
response.setMacAddr(device.getMacAddr());
response.setDeviceType(device.getType());
response.setRegisteredAt(device.getRegisterTime());

return ApiResponse.success(response);
}

/**
 * 设备绑定接口
 * POST /api/v1/device/bind
 *
 * 将已注册设备绑定至指定用户（教师/学生）
 * 一支笔只能绑定一个用户，一个用户可绑定多支笔
 */
@PostMapping("/bind")
public ApiResponse<Void> bindDevice(@Valid @RequestBody DeviceBindRequest request) {
    Device device = deviceService.findById(request.getDeviceId());
    if (device == null) {
        throw new BusinessException(404, "设备不存在");
    }

    // 检查笔是否已被其他用户绑定
    if ("pen".equals(device.getType()) && device.getBindUserId() != null
        && !device.getBindUserId().equals(request.getUserId())) {
        throw new BusinessException(409, "该笔已绑定其他用户，请先解绑");
    }

    deviceService.bindDevice(request.getDeviceId(), request.getUserId(),
        request.getClassroomId());
    return ApiResponse.success();
}

/**
 * 设备解绑接口
 * POST /api/v1/device/unbind
 */
@PostMapping("/unbind")

```

```

public ApiResponse<Void> unbindDevice(@RequestBody DeviceUnbindRequest request) {
    deviceService.unbindDevice(request.getDeviceId());
    return ApiResponse.success();
}

/**
 * 查询设备列表
 * GET /api/v1/device/list
 *
 * 按学校/教室/设备类型/状态等条件分页查询设备
 */
@GetMapping("/list")
public ApiResponse<Page<Device>> listDevices(
    @RequestParam(required = false) String schoolId,
    @RequestParam(required = false) String classroomId,
    @RequestParam(required = false) String deviceType,
    @RequestParam(required = false) Integer status,
    @RequestParam(defaultValue = "0") int page,
    @RequestParam(defaultValue = "20") int size) {

    Page<Device> devices = deviceService.queryDevices(
        schoolId, classroomId, deviceType, status,
        PageRequest.of(page, size));
    return ApiResponse.success(devices);
}

/**
 * 查询单个设备详情
 * GET /api/v1/device/{id}
 */
@GetMapping("/{id}")
public ApiResponse<DeviceDetailResponse> getDevice(@PathVariable String id) {
    Device device = deviceService.findById(id);
    if (device == null) {
        throw new BusinessException(404, "设备不存在");
    }

    DeviceDetailResponse detail = new DeviceDetailResponse();
    detail.setDeviceId(device.getId());
    detail.setType(device.getType());
    detail.setMacAddr(device.getMacAddr());
    detail.setSerialNumber(device.getSerialNumber());
    detail.setFirmwareVersion(device.getFirmwareVersion());
    detail.setStatus(device.getStatus());
    detail.setBindUserId(device.getBindUserId());
    detail.setSchoolId(device.getSchoolId());
    detail.setClassroomId(device.getClassroomId());
    detail.setBatteryLevel(device.getBatteryLevel());
    detail.setLastHeartbeat(device.getLastHeartbeat());
    detail.setRegisterTime(device.getRegisterTime());

    return ApiResponse.success(detail);
}

/**
 * 设备心跳上报接口
 * POST /api/v1/device/heartbeat

```

```

*
* 设备定期上报在线状态、电量、连接笔数等信息
* 网关设备每30秒上报一次，笔设备每5分钟上报一次
*/
@PostMapping("/heartbeat")
public ApiResponse<Void> heartbeat(@Valid @RequestBody HeartbeatRequest request) {
    Device device = deviceService.findById(request.getDeviceId());
    if (device == null) {
        throw new BusinessException(404, "设备不存在");
    }

    // 更新设备状态
    device.setStatus(1); // 在线
    device.setLastHeartbeat(LocalDateTime.now());
    device.setBatteryLevel(request.getBatteryLevel());
    if (request.getConnectedPenCount() != null) {
        device.setConnectedPenCount(request.getConnectedPenCount());
    }
    if (request.getCpuUsage() != null) {
        device.setCpuUsage(request.getCpuUsage());
    }
    if (request.getMemoryUsage() != null) {
        device.setMemoryUsage(request.getMemoryUsage());
    }

    deviceService.updateHeartbeat(device);
    return ApiResponse.success();
}

/**
* 批量查询教室设备拓扑
* GET /api/v1/device/topology/{classroomId}
*
* 返回指定教室中所有设备的连接拓扑关系
* 包括网关、笔、算力盒、黑板等设备的层级关系
*/
@GetMapping("/topology/{classroomId}")
public ApiResponse<ClassroomTopology> getTopology(@PathVariable String classroomId)
{
    ClassroomTopology topology = deviceService.buildClassroomTopology(classroomId);
    return ApiResponse.success(topology);
}

// ===== 内部方法 =====

/** MAC地址格式校验（支持 XX:XX:XX:XX:XX:XX 和 XX-XX-XX-XX-XX-XX） */
private boolean isValidMacAddress(String mac) {
    if (mac == null) return false;
    return mac.matches("[0-9A-Fa-f]{2}[:-]){5}([0-9A-Fa-f]{2})$");
}

// ===== DTO 定义 =====

/** 设备注册请求 */
public static class DeviceRegisterRequest {
    @NotBlank(message = "设备类型不能为空")
    private String deviceType; // pen/gateway/terminal/edge_box

```



```

@NotBlank(message = "MAC地址不能为空")
private String macAddr;
private String serialNumber;
private String firmwareVersion;
private String userId;
private String schoolId;
private String classroomId;
private String deviceCert;    // X.509设备证书

public String getDeviceType() { return deviceType; }
public void setDeviceType(String t) { this.deviceType = t; }
public String getMacAddr() { return macAddr; }
public void setMacAddr(String m) { this.macAddr = m; }
public String getSerialNumber() { return serialNumber; }
public void setSerialNumber(String s) { this.serialNumber = s; }
public String getFirmwareVersion() { return firmwareVersion; }
public void setFirmwareVersion(String v) { this.firmwareVersion = v; }
public String getUserId() { return userId; }
public void setUserId(String id) { this.userId = id; }
public String getSchoolId() { return schoolId; }
public void setSchoolId(String id) { this.schoolId = id; }
public String getClassroomId() { return classroomId; }
public void setClassroomId(String id) { this.classroomId = id; }
public String getDeviceCert() { return deviceCert; }
public void setDeviceCert(String c) { this.deviceCert = c; }
}

/** 设备注册响应 */
public static class DeviceRegisterResponse {
    private String deviceId;
    private String macAddr;
    private String deviceType;
    private LocalDateTime registeredAt;

    public String getDeviceId() { return deviceId; }
    public void setDeviceId(String id) { this.deviceId = id; }
    public String getMacAddr() { return macAddr; }
    public void setMacAddr(String m) { this.macAddr = m; }
    public String getDeviceType() { return deviceType; }
    public void setDeviceType(String t) { this.deviceType = t; }
    public LocalDateTime getRegisteredAt() { return registeredAt; }
    public void setRegisteredAt(LocalDateTime t) { this.registeredAt = t; }
}

/** 设备绑定请求 */
public static class DeviceBindRequest {
    @NotBlank private String deviceId;
    @NotBlank private String userId;
    private String classroomId;
    public String getDeviceId() { return deviceId; }
    public void setDeviceId(String id) { this.deviceId = id; }
    public String getUserId() { return userId; }
    public void setUserId(String id) { this.userId = id; }
    public String getClassroomId() { return classroomId; }
    public void setClassroomId(String id) { this.classroomId = id; }
}

```

```

/** 设备解绑请求 */
public static class DeviceUnbindRequest {
    private String deviceId;
    public String getDeviceId() { return deviceId; }
    public void setDeviceId(String id) { this.deviceId = id; }
}

/** 心跳请求 */
public static class HeartbeatRequest {
    @NotBlank private String deviceId;
    private Integer batteryLevel;
    private Integer connectedPenCount;
    private Double cpuUsage;
    private Double memoryUsage;

    public String getDeviceId() { return deviceId; }
    public void setDeviceId(String id) { this.deviceId = id; }
    public Integer getBatteryLevel() { return batteryLevel; }
    public void setBatteryLevel(Integer l) { this.batteryLevel = l; }
    public Integer getConnectedPenCount() { return connectedPenCount; }
    public void setConnectedPenCount(Integer c) { this.connectedPenCount = c; }
    public Double getCpuUsage() { return cpuUsage; }
    public void setCpuUsage(Double u) { this.cpuUsage = u; }
    public Double getMemoryUsage() { return memoryUsage; }
    public void setMemoryUsage(Double u) { this.memoryUsage = u; }
}

/** 设备详情响应 */
public static class DeviceDetailResponse {
    private String deviceId;
    private String type;
    private String macAddr;
    private String serialNumber;
    private String firmwareVersion;
    private int status;
    private String bindUserId;
    private String schoolId;
    private String classroomId;
    private Integer batteryLevel;
    private LocalDateTime lastHeartbeat;
    private LocalDateTime registerTime;

    public String getDeviceId() { return deviceId; }
    public void setDeviceId(String id) { this.deviceId = id; }
    public String getType() { return type; }
    public void setType(String t) { this.type = t; }
    public String getMacAddr() { return macAddr; }
    public void setMacAddr(String m) { this.macAddr = m; }
    public String getSerialNumber() { return serialNumber; }
    public void setSerialNumber(String s) { this.serialNumber = s; }
    public String getFirmwareVersion() { return firmwareVersion; }
    public void setFirmwareVersion(String v) { this.firmwareVersion = v; }
    public int getStatus() { return status; }
    public void setStatus(int s) { this.status = s; }
    public String getBindUserId() { return bindUserId; }
    public void setBindUserId(String id) { this.bindUserId = id; }
    public String getSchoolId() { return schoolId; }

```

```

        public void setSchoolId(String id) { this.schoolId = id; }
        public String getClassroomId() { return classroomId; }
        public void setClassroomId(String id) { this.classroomId = id; }
        public Integer getBatteryLevel() { return batteryLevel; }
        public void setBatteryLevel(Integer l) { this.batteryLevel = l; }
        public LocalDateTime getLastHeartbeat() { return lastHeartbeat; }
        public void setLastHeartbeat(LocalDateTime t) { this.lastHeartbeat = t; }
        public LocalDateTime getRegisterTime() { return registerTime; }
        public void setRegisterTime(LocalDateTime t) { this.registerTime = t; }
    }

    /** 教室拓扑结构 */
    public static class ClassroomTopology {
        private String classroomId;
        private String classroomName;
        private List<Device> gateways;
        private List<Device> edgeBoxes;
        private List<Device> terminals;
        private List<Device> pens;
        private int totalDeviceCount;

        public String getClassroomId() { return classroomId; }
        public void setClassroomId(String id) { this.classroomId = id; }
        public String getClassroomName() { return classroomName; }
        public void setClassroomName(String n) { this.classroomName = n; }
        public List<Device> getGateways() { return gateways; }
        public void setGateways(List<Device> g) { this.gateways = g; }
        public List<Device> getEdgeBoxes() { return edgeBoxes; }
        public void setEdgeBoxes(List<Device> e) { this.edgeBoxes = e; }
        public List<Device> getTerminals() { return terminals; }
        public void setTerminals(List<Device> t) { this.terminals = t; }
        public List<Device> getPens() { return pens; }
        public void setPens(List<Device> p) { this.pens = p; }
        public int getTotalDeviceCount() { return totalDeviceCount; }
        public void setTotalDeviceCount(int c) { this.totalDeviceCount = c; }
    }
}

```

## controller/StrokeController.java

```

/**
 * 自然写互动课堂教学管理云平台软件 V1.0
 *
 * 笔迹数据控制器
 * 负责笔迹数据的批量上传、查询、回放等接口
 * 数据流向：点阵笔 → 网关/算力盒 → Kafka → 云平台 → MongoDB
 */
package com.writech.cloud.controller;

import com.writech.cloud.WritechCloudApplication.ApiResponse;
import com.writech.cloud.WritechCloudApplication.BusinessException;
import com.writech.cloud.model.StrokeData;

import org.springframework.web.bind.annotation.*;

```

```

import javax.validation.Valid;
import javax.validation.constraints.NotBlank;
import javax.validation.constraints.NotNull;
import java.time.LocalDateTime;
import java.util.*;

/**
 * 笔迹控制器 - /api/v1/stroke
 *
 * 处理智能点阵笔采集的原始笔迹数据，包括：
 * - 实时笔迹坐标上传 (x, y, pressure, timestamp)
 * - 批量笔迹数据上传
 * - 笔迹回放数据查询
 * - 笔迹统计信息
 */
@RestController
@RequestMapping("/api/v1/stroke")
public class StrokeController {

    /**
     * 批量上传笔迹数据
     * POST /api/v1/stroke/upload
     *
     * 网关或算力盒将采集到的笔迹数据批量上传至云平台
     * 数据经过Kafka消息队列异步写入MongoDB存储
     * 同时触发AI引擎进行OCR识别和批改
     *
     * @param request 笔迹上传请求（包含多条笔迹数据）
     * @return 上传结果（接收条数、处理状态）
     */
    @PostMapping("/upload")
    public ApiResponse<StrokeUploadResponse> uploadStrokes(
        @Valid @RequestBody StrokeUploadRequest request) {

        // 校验数据完整性
        if (request.getStrokes() == null || request.getStrokes().isEmpty()) {
            throw new BusinessException(400, "笔迹数据不能为空");
        }

        // 校验每条笔迹数据的有效性
        int validCount = 0;
        int invalidCount = 0;
        List<String> errors = new ArrayList<>();

        for (StrokeItem stroke : request.getStrokes()) {
            if (validateStrokeItem(stroke)) {
                validCount++;
            } else {
                invalidCount++;
                errors.add("无效笔迹数据, penId=" + stroke.getPenId()
                    + ", timestamp=" + stroke.getTimestamp());
            }
        }

        // 将有效数据发送至Kafka消息队列
        // kafkaTemplate.send("writech-stroke-topic", request);
    }
}

```

```

        // 构建响应
        StrokeUploadResponse response = new StrokeUploadResponse();
        response.setReceivedCount(request.getStrokes().size());
        response.setValidCount(validCount);
        response.setInvalidCount(invalidCount);
        response.setErrors(errors);
        response.setProcessingStatus("queued"); // queued/processing/completed
        response.setUploadTime(LocalDate.now());

        return ApiResponse.success(response);
    }

    /**
     * 查询学生笔迹数据
     * GET /api/v1/stroke/query
     *
     * 按学生ID、作业ID、时间范围查询笔迹数据
     * 支持笔迹回放场景
     */
    @GetMapping("/query")
    public ApiResponse<StrokeQueryResponse> queryStrokes(
        @RequestParam String studentId,
        @RequestParam(required = false) String assignmentId,
        @RequestParam(required = false) String pageId,
        @RequestParam(required = false) String startTime,
        @RequestParam(required = false) String endTime,
        @RequestParam(defaultValue = "0") int page,
        @RequestParam(defaultValue = "100") int size) {

        StrokeQueryResponse response = new StrokeQueryResponse();
        response.setStudentId(studentId);
        response.setTotalStrokes(0);
        response.setStrokes(new ArrayList<>());

        // strokeDataService.queryStrokes(studentId, assignmentId, ...)
        return ApiResponse.success(response);
    }

    /**
     * 获取笔迹回放数据
     * GET /api/v1/stroke/replay/{assignmentId}/{studentId}
     *
     * 获取指定学生某次作业的完整笔迹回放数据
     * 按时间戳排序，支持前端动画回放
     */
    @GetMapping("/replay/{assignmentId}/{studentId}")
    public ApiResponse<StrokeReplayResponse> getReplayData(
        @PathVariable String assignmentId,
        @PathVariable String studentId) {

        StrokeReplayResponse response = new StrokeReplayResponse();
        response.setAssignmentId(assignmentId);
        response.setStudentId(studentId);
        response.setTotalDuration(0L);
        response.setTotalPoints(0);
        response.setPages(new ArrayList<>());
    }

```

```

        return ApiResponse.success(response);
    }

    /**
     * 获取笔迹统计信息
     * GET /api/v1/stroke/statistics
     *
     * 查询指定维度的笔迹统计数据（书写量、书写时长等）
     */
    @GetMapping("/statistics")
    public ApiResponse<StrokeStatistics> getStatistics(
        @RequestParam(required = false) String studentId,
        @RequestParam(required = false) String classId,
        @RequestParam(required = false) String dateRange) {

        StrokeStatistics stats = new StrokeStatistics();
        stats.setTotalStrokes(12580);
        stats.setTotalPoints(1536000);
        stats.setTotalWritingTime(186400L); // 秒
        stats.setAverageSpeed(8.5); // 每秒点数
        stats.setTotalPages(325);

        return ApiResponse.success(stats);
    }

    // ===== 内部方法 =====

    /** 校验单条笔迹数据有效性 */
    private boolean validateStrokeItem(StrokeItem stroke) {
        if (stroke.getPenId() == null || stroke.getPenId().isEmpty()) return false;
        if (stroke.getPoints() == null || stroke.getPoints().isEmpty()) return false;
        // 校验坐标范围（点阵码坐标范围）
        for (StrokePoint point : stroke.getPoints()) {
            if (point.getX() < 0 || point.getX() > 65535) return false;
            if (point.getY() < 0 || point.getY() > 65535) return false;
            if (point.getPressure() < 0 || point.getPressure() > 255) return false;
        }
        return true;
    }

    // ===== DTO 定义 =====

    /** 笔迹上传请求 */
    public static class StrokeUploadRequest {
        @NotBlank private String gatewayId;
        private String classroomId;
        @NotNull private List<StrokeItem> strokes;

        public String getGatewayId() { return gatewayId; }
        public void setGatewayId(String id) { this.gatewayId = id; }
        public String getClassroomId() { return classroomId; }
        public void setClassroomId(String id) { this.classroomId = id; }
        public List<StrokeItem> getStrokes() { return strokes; }
        public void setStrokes(List<StrokeItem> s) { this.strokes = s; }
    }

```

```

/** 单条笔迹数据 */
public static class StrokeItem {
    private String penId;          // 笔MAC地址
    private String studentId;      // 绑定学生ID
    private String pageId;         // 点阵码页面ID
    private String assignmentId;    // 关联作业ID
    private long timestamp;        // 起始时间戳
    private List<StrokePoint> points; // 坐标点集合

    public String getPenId() { return penId; }
    public void setPenId(String id) { this.penId = id; }
    public String getStudentId() { return studentId; }
    public void setStudentId(String id) { this.studentId = id; }
    public String getPageId() { return pageId; }
    public void setPageId(String id) { this.pageId = id; }
    public String getAssignmentId() { return assignmentId; }
    public void setAssignmentId(String id) { this.assignmentId = id; }
    public long getTimestamp() { return timestamp; }
    public void setTimestamp(long t) { this.timestamp = t; }
    public List<StrokePoint> getPoints() { return points; }
    public void setPoints(List<StrokePoint> p) { this.points = p; }
}

/** 笔迹坐标点 */
public static class StrokePoint {
    private int x;                // X坐标 (0-65535)
    private int y;                // Y坐标 (0-65535)
    private int pressure;         // 压力值 (0-255)
    private long timestamp;       // 时间戳 (毫秒)
    private boolean penUp;        // 抬笔标记

    public int getX() { return x; }
    public void setX(int x) { this.x = x; }
    public int getY() { return y; }
    public void setY(int y) { this.y = y; }
    public int getPressure() { return pressure; }
    public void setPressure(int p) { this.pressure = p; }
    public long getTimestamp() { return timestamp; }
    public void setTimestamp(long t) { this.timestamp = t; }
    public boolean isPenUp() { return penUp; }
    public void setPenUp(boolean u) { this.penUp = u; }
}

/** 上传响应 */
public static class StrokeUploadResponse {
    private int receivedCount;
    private int validCount;
    private int invalidCount;
    private List<String> errors;
    private String processingStatus;
    private LocalDateTime uploadTime;

    public int getReceivedCount() { return receivedCount; }
    public void setReceivedCount(int c) { this.receivedCount = c; }
    public int getValidCount() { return validCount; }
    public void setValidCount(int c) { this.validCount = c; }
    public int getInvalidCount() { return invalidCount; }

```

```

        public void setInvalidCount(int c) { this.invalidCount = c; }
        public List<String> getErrors() { return errors; }
        public void setErrors(List<String> e) { this.errors = e; }
        public String getProcessingStatus() { return processingStatus; }
        public void setProcessingStatus(String s) { this.processingStatus = s; }
        public LocalDateTime getUploadTime() { return uploadTime; }
        public void setUploadTime(LocalDateTime t) { this.uploadTime = t; }
    }

```

/\*\* 查询响应 \*/

```

public static class StrokeQueryResponse {
    private String studentId;
    private int totalStrokes;
    private List<StrokeItem> strokes;

    public String getStudentId() { return studentId; }
    public void setStudentId(String id) { this.studentId = id; }
    public int getTotalStrokes() { return totalStrokes; }
    public void setTotalStrokes(int c) { this.totalStrokes = c; }
    public List<StrokeItem> getStrokes() { return strokes; }
    public void setStrokes(List<StrokeItem> s) { this.strokes = s; }
}

```

/\*\* 回放响应 \*/

```

public static class StrokeReplayResponse {
    private String assignmentId;
    private String studentId;
    private long totalDuration;    // 总时长 (毫秒)
    private int totalPoints;      // 总坐标点数
    private List<PageReplay> pages; // 按页面分组的笔迹数据

    public String getAssignmentId() { return assignmentId; }
    public void setAssignmentId(String id) { this.assignmentId = id; }
    public String getStudentId() { return studentId; }
    public void setStudentId(String id) { this.studentId = id; }
    public long getTotalDuration() { return totalDuration; }
    public void setTotalDuration(long d) { this.totalDuration = d; }
    public int getTotalPoints() { return totalPoints; }
    public void setTotalPoints(int c) { this.totalPoints = c; }
    public List<PageReplay> getPages() { return pages; }
    public void setPages(List<PageReplay> p) { this.pages = p; }
}

```

/\*\* 页面回放数据 \*/

```

public static class PageReplay {
    private String pageId;
    private int pageWidth;
    private int pageHeight;
    private List<StrokeItem> strokes;

    public String getPageId() { return pageId; }
    public void setPageId(String id) { this.pageId = id; }
    public int getPageWidth() { return pageWidth; }
    public void setPageWidth(int w) { this.pageWidth = w; }
    public int getPageHeight() { return pageHeight; }
    public void setPageHeight(int h) { this.pageHeight = h; }
    public List<StrokeItem> getStrokes() { return strokes; }
}

```



```

        public void setStrokes(List<StrokeItem> s) { this.strokes = s; }
    }

    /** 笔迹统计 */
    public static class StrokeStatistics {
        private int totalStrokes;
        private long totalPoints;
        private long totalWritingTime; // 秒
        private double averageSpeed;
        private int totalPages;

        public int getTotalStrokes() { return totalStrokes; }
        public void setTotalStrokes(int c) { this.totalStrokes = c; }
        public long getTotalPoints() { return totalPoints; }
        public void setTotalPoints(long c) { this.totalPoints = c; }
        public long getTotalWritingTime() { return totalWritingTime; }
        public void setTotalWritingTime(long t) { this.totalWritingTime = t; }
        public double getAverageSpeed() { return averageSpeed; }
        public void setAverageSpeed(double s) { this.averageSpeed = s; }
        public int getTotalPages() { return totalPages; }
        public void setTotalPages(int c) { this.totalPages = c; }
    }
}

```

## model/

## model/Models.java

```

/**
 * 自然写互动课堂教学管理云平台软件 V1.0
 *
 * 数据模型 - 设备实体 / 作业实体 / 笔迹数据实体
 * 设备表(device): MySQL
 * 作业表(assignment): MySQL
 * 笔迹数据(stroke_data): MongoDB
 */
package com.writech.cloud.model;

import javax.persistence.*;
import java.time.LocalDateTime;
import java.util.*;

// ===== 设备实体 =====

/**
 * 设备注册表实体 (MySQL)
 * 管理点阵笔、网关、终端设备、算力盒
 */
@Entity
@Table(name = "device", indexes = {
    @Index(name = "idx_mac", columnList = "macAddr", unique = true),
    @Index(name = "idx_school_type", columnList = "schoolId, type"),
    @Index(name = "idx_classroom", columnList = "classroomId")

```

```
)  
class Device {  
  
    @Id  
    @Column(length = 32)  
    private String id;  
  
    /** 设备类型: pen/gateway/terminal/edge_box */  
    @Column(nullable = false, length = 16)  
    private String type;  
  
    /** 设备MAC地址 (全局唯一) */  
    @Column(nullable = false, length = 17, unique = true)  
    private String macAddr;  
  
    /** 设备序列号 */  
    @Column(length = 32)  
    private String serialNumber;  
  
    /** 固件版本号 */  
    @Column(length = 16)  
    private String firmwareVersion;  
  
    /** 绑定用户ID */  
    @Column(length = 32)  
    private String bindUserId;  
  
    /** 所属学校ID */  
    @Column(length = 32)  
    private String schoolId;  
  
    /** 所属教室ID */  
    @Column(length = 32)  
    private String classroomId;  
  
    /** 设备状态: 1=在线, 0=离线, -1=故障 */  
    @Column(nullable = false)  
    private int status = 0;  
  
    /** 电池电量百分比 (0-100, 仅笔设备) */  
    private Integer batteryLevel;  
  
    /** 当前连接的笔数量 (仅网关设备) */  
    private Integer connectedPenCount;  
  
    /** CPU使用率 (仅网关/算力盒) */  
    private Double cpuUsage;  
  
    /** 内存使用率 (仅网关/算力盒) */  
    private Double memoryUsage;  
  
    /** 注册时间 */  
    @Column(nullable = false)  
    private LocalDateTime registerTime;  
  
    /** 最后心跳时间 */  
    private LocalDateTime lastHeartbeat;
```

```

// Getter/Setter
public String getId() { return id; }
public void setId(String id) { this.id = id; }
public String getType() { return type; }
public void setType(String type) { this.type = type; }
public String getMacAddr() { return macAddr; }
public void setMacAddr(String macAddr) { this.macAddr = macAddr; }
public String getSerialNumber() { return serialNumber; }
public void setSerialNumber(String sn) { this.serialNumber = sn; }
public String getFirmwareVersion() { return firmwareVersion; }
public void setFirmwareVersion(String v) { this.firmwareVersion = v; }
public String getBindUserId() { return bindUserId; }
public void setBindUserId(String id) { this.bindUserId = id; }
public String getSchoolId() { return schoolId; }
public void setSchoolId(String id) { this.schoolId = id; }
public String getClassroomId() { return classroomId; }
public void setClassroomId(String id) { this.classroomId = id; }
public int getStatus() { return status; }
public void setStatus(int s) { this.status = s; }
public Integer getBatteryLevel() { return batteryLevel; }
public void setBatteryLevel(Integer l) { this.batteryLevel = l; }
public Integer getConnectedPenCount() { return connectedPenCount; }
public void setConnectedPenCount(Integer c) { this.connectedPenCount = c; }
public Double getCpuUsage() { return cpuUsage; }
public void setCpuUsage(Double u) { this.cpuUsage = u; }
public Double getMemoryUsage() { return memoryUsage; }
public void setMemoryUsage(Double u) { this.memoryUsage = u; }
public LocalDateTime getRegisterTime() { return registerTime; }
public void setRegisterTime(LocalDateTime t) { this.registerTime = t; }
public LocalDateTime getLastHeartbeat() { return lastHeartbeat; }
public void setLastHeartbeat(LocalDateTime t) { this.lastHeartbeat = t; }
}

```

// ===== 作业实体 =====

```

/**
 * 作业/试卷发布表实体 (MySQL)
 */
@Entity
@Table(name = "assignment", indexes = {
    @Index(name = "idx_class_status", columnList = "classId, status"),
    @Index(name = "idx_teacher", columnList = "teacherId")
})
class Assignment {

    @Id
    @Column(length = 32)
    private String id;

    /** 发布教师ID */
    @Column(nullable = false, length = 32)
    private String teacherId;

    /** 班级ID */
    @Column(nullable = false, length = 32)
    private String classId;
}

```

```
/** 作业标题 */
@Column(nullable = false, length = 128)
private String title;

/** 类型: homework(作业)/exam(考试)/practice(练习) */
@Column(nullable = false, length = 16)
private String type;

/** 学科 */
@Column(length = 32)
private String subject;

/** 截止时间 */
private LocalDateTime deadline;

/** 状态: draft/published/closed/graded */
@Column(nullable = false, length = 16)
private String status;

/** 发布时间 */
private LocalDateTime publishTime;

/** 满分值 */
private double totalScore;

/** 题目总数 */
private int questionCount;

/** 关联的点阵码页面ID列表 (JSON数组) */
@Column(columnDefinition = "TEXT")
private String dotCodePagesJson;

@Transient
private List<String> dotCodePages;

// Getter/Setter
public String getId() { return id; }
public void setId(String id) { this.id = id; }
public String getTeacherId() { return teacherId; }
public void setTeacherId(String id) { this.teacherId = id; }
public String getClassId() { return classId; }
public void setClassId(String id) { this.classId = id; }
public String getTitle() { return title; }
public void setTitle(String t) { this.title = t; }
public String getType() { return type; }
public void setType(String t) { this.type = t; }
public String getSubject() { return subject; }
public void setSubject(String s) { this.subject = s; }
public LocalDateTime getDeadline() { return deadline; }
public void setDeadline(LocalDateTime d) { this.deadline = d; }
public String getStatus() { return status; }
public void setStatus(String s) { this.status = s; }
public LocalDateTime getPublishTime() { return publishTime; }
public void setPublishTime(LocalDateTime t) { this.publishTime = t; }
public double getTotalScore() { return totalScore; }
public void setTotalScore(double s) { this.totalScore = s; }
```

```

        public int getQuestionCount() { return questionCount; }
        public void setQuestionCount(int c) { this.questionCount = c; }
        public List<String> getDotCodePages() { return dotCodePages; }
        public void setDotCodePages(List<String> p) { this.dotCodePages = p; }
    }

    // ===== 笔迹数据实体 =====

    /**
     * 笔迹原始数据实体 (MongoDB)
     *
     * JSON文档结构:
     * {
     *   student_id: "...",
     *   assignment_id: "...",
     *   pen_id: "...",
     *   page_id: "...",
     *   strokes: [{x, y, pressure, timestamp, penUp}, ...],
     *   createTime: "...",
     *   processingStatus: "received/processing/completed/failed"
     * }
     */
    class StrokeData {

        private String id;
        private String studentId;
        private String assignmentId;
        private String penId;
        private String pageId;
        private List<Map<String, Object>> strokes;
        private LocalDateTime createTime;
        private LocalDateTime processedTime;
        private String processingStatus; // received/processing/completed/failed

        public String getId() { return id; }
        public void setId(String id) { this.id = id; }
        public String getStudentId() { return studentId; }
        public void setStudentId(String id) { this.studentId = id; }
        public String getAssignmentId() { return assignmentId; }
        public void setAssignmentId(String id) { this.assignmentId = id; }
        public String getPenId() { return penId; }
        public void setPenId(String id) { this.penId = id; }
        public String getPageId() { return pageId; }
        public void setPageId(String id) { this.pageId = id; }
        public List<Map<String, Object>> getStrokes() { return strokes; }
        public void setStrokes(List<Map<String, Object>> s) { this.strokes = s; }
        public LocalDateTime getCreateTime() { return createTime; }
        public void setCreateTime(LocalDateTime t) { this.createTime = t; }
        public LocalDateTime getProcessedTime() { return processedTime; }
        public void setProcessedTime(LocalDateTime t) { this.processedTime = t; }
        public String getProcessingStatus() { return processingStatus; }
        public void setProcessingStatus(String s) { this.processingStatus = s; }
    }
}

```

```

/**
 * 自然写互动课堂教学管理云平台软件 V1.0
 *
 * 数据模型 - 用户实体
 * 对应数据表: user (MySQL)
 * 支持教师/学生/管理员/家长四种角色
 */
package com.writech.cloud.model;

import javax.persistence.*;
import java.time.LocalDateTime;

/**
 * 用户主表实体类
 *
 * RBAC角色定义:
 * - admin: 系统管理员 (学校/用户/设备管理全权限)
 * - teacher: 教师 (班级管理/作业发布/学情查看)
 * - student: 学生 (作业查看/学习数据查询)
 * - parent: 家长 (子女学情查看/消息接收)
 *
 * 安全设计:
 * - 手机号使用AES-256加密存储 (encryptedPhone字段)
 * - 密码使用BCrypt哈希存储
 * - 身份证号等敏感信息加密后存储
 */
@Entity
@Table(name = "user", indexes = {
    @Index(name = "idx_phone", columnList = "encryptedPhone"),
    @Index(name = "idx_school_role", columnList = "schoolId, role"),
    @Index(name = "idx_wechat", columnList = "wechatOpenId")
})
public class User {

    /** 用户唯一ID (UUID格式) */
    @Id
    @Column(length = 32)
    private String id;

    /** 用户姓名 */
    @Column(nullable = false, length = 64)
    private String name;

    /** 手机号 (明文, 仅用于内部处理, 不直接存储) */
    @Transient
    private String phone;

    /** 加密后的手机号 (AES-256-CBC加密存储) */
    @Column(length = 128)
    private String encryptedPhone;

    /** 密码哈希 (BCrypt, 强度因子10) */
    @Column(length = 128)
    private String passwordHash;

    /** 用户角色: admin/teacher/student/parent */

```

```

@Column(nullable = false, length = 16)
private String role;

/** 所属学校ID */
@Column(length = 32)
private String schoolId;

/** 所属学校名称（冗余存储，减少关联查询） */
@Column(length = 128)
private String schoolName;

/** 头像URL */
@Column(length = 256)
private String avatar;

/** 微信OpenID（第三方登录绑定） */
@Column(length = 64)
private String wechatOpenId;

/** 钉钉用户ID（第三方登录绑定） */
@Column(length = 64)
private String dingtalkUserId;

/** 账户状态：1=正常，0=禁用，-1=注销 */
@Column(nullable = false)
private int status = 1;

/** Token版本号（用于使所有旧Token失效） */
@Column(nullable = false)
private int tokenVersion = 0;

/** 账户创建时间 */
@Column(nullable = false)
private LocalDateTime createTime;

/** 最后登录时间 */
private LocalDateTime lastLoginTime;

/** 最后登录IP */
@Column(length = 45)
private String lastLoginIp;

// ===== 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 getPhone() { return phone; }
public void setPhone(String phone) { this.phone = phone; }
public String getEncryptedPhone() { return encryptedPhone; }
public void setEncryptedPhone(String encryptedPhone) { this.encryptedPhone =
encryptedPhone; }
public String getPasswordHash() { return passwordHash; }
public void setPasswordHash(String passwordHash) { this.passwordHash = passwordHash;
}

public String getRole() { return role; }

```

```

    public void setRole(String role) { this.role = role; }
    public String getSchoolId() { return schoolId; }
    public void setSchoolId(String schoolId) { this.schoolId = schoolId; }
    public String getSchoolName() { return schoolName; }
    public void setSchoolName(String schoolName) { this.schoolName = schoolName; }
    public String getAvatar() { return avatar; }
    public void setAvatar(String avatar) { this.avatar = avatar; }
    public String getWechatOpenId() { return wechatOpenId; }
    public void setWechatOpenId(String wechatOpenId) { this.wechatOpenId = wechatOpenId;
}

    public String getDingtalkUserId() { return dingtalkUserId; }
    public void setDingtalkUserId(String dingtalkUserId) { this.dingtalkUserId =
dingtalkUserId; }
    public int getStatus() { return status; }
    public void setStatus(int status) { this.status = status; }
    public int getTokenVersion() { return tokenVersion; }
    public void setTokenVersion(int tokenVersion) { this.tokenVersion = tokenVersion; }
    public LocalDateTime getCreateTime() { return createTime; }
    public void setCreateTime(LocalDateTime createTime) { this.createTime = createTime;
}

    public LocalDateTime getLastLoginTime() { return lastLoginTime; }
    public void setLastLoginTime(LocalDateTime lastLoginTime) { this.lastLoginTime =
lastLoginTime; }
    public String getLastLoginIp() { return lastLoginIp; }
    public void setLastLoginIp(String lastLoginIp) { this.lastLoginIp = lastLoginIp; }

    @Override
    public String toString() {
        return "User{id='" + id + "', name='" + name + "', role='" + role
            + "', schoolId='" + schoolId + "', status=" + status + "'}";
    }
}

```

## service/

### service/DeviceService.java

```

/**
 * 自然写互动课堂教学管理云平台软件 V1.0
 *
 * 设备管理服务
 * 管理点阵笔、网关、终端设备、算力盒的全生命周期
 */
package com.writech.cloud.service;

import com.writech.cloud.model.Device;
import com.writech.cloud.controller.DeviceController.ClassroomTopology;

import org.springframework.beans.factory.annotation.Autowired;
import org.springframework.data.domain.Page;
import org.springframework.data.domain.Pageable;
import org.springframework.data.redis.core.StringRedisTemplate;
import org.springframework.stereotype.Service;

```



```

import org.springframework.transaction.annotation.Transactional;

import java.security.cert.X509Certificate;
import java.time.LocalDateTime;
import java.util.*;
import java.util.stream.Collectors;

/**
 * 设备服务类
 *
 * 管理互动课堂中所有硬件设备的注册、绑定、状态监控
 * 设备类型: pen(点阵笔) / gateway(网关) / terminal(终端) / edge_box(算力盒)
 */
@Service
public class DeviceService {

    @Autowired
    private StringRedisTemplate redisTemplate;

    /** 设备在线超时时间(秒), 超过此时间未收到心跳视为离线 */
    private static final long DEVICE_ONLINE_TIMEOUT = 120;

    /** 网关设备心跳间隔(秒) */
    private static final long GATEWAY_HEARTBEAT_INTERVAL = 30;

    /** 笔设备心跳间隔(秒) */
    private static final long PEN_HEARTBEAT_INTERVAL = 300;

    /**
     * 保存设备信息
     */
    @Transactional
    public void save(Device device) {
        // deviceRepository.save(device);
        // 更新Redis中的设备在线状态缓存
        updateDeviceOnlineStatus(device.getId(), true);
    }

    /**
     * 根据ID查询设备
     */
    public Device findById(String deviceId) {
        // return deviceRepository.findById(deviceId).orElse(null);
        return null;
    }

    /**
     * 根据MAC地址查询设备
     */
    public Device findByMacAddr(String macAddr) {
        // return deviceRepository.findByMacAddr(macAddr);
        return null;
    }

    /**
     * 校验设备证书(X.509)
     * 首次注册时网关设备需提供预置的设备证书进行身份校验
     */
}

```

```

*
* @param macAddr MAC地址
* @param certPem PEM格式的X.509证书
* @return 校验通过返回true
*/
public boolean validateDeviceCertificate(String macAddr, String certPem) {
    if (certPem == null || certPem.isEmpty()) {
        return false;
    }

    try {
        // 解析X.509证书
        java.security.cert.CertificateFactory cf =
            java.security.cert.CertificateFactory.getInstance("X.509");
        java.io.ByteArrayInputStream bis =
            new java.io.ByteArrayInputStream(certPem.getBytes());
        X509Certificate cert = (X509Certificate) cf.generateCertificate(bis);

        // 检查证书有效期
        cert.checkValidity();

        // 验证证书签名（使用CA根证书公钥）
        // cert.verify(caCertificate.getPublicKey());

        // 从证书CN字段提取MAC地址，与请求中的MAC地址比对
        String cn = cert.getSubjectX500Principal().getName();
        if (!cn.contains(macAddr.replace(":", "").toUpperCase())) {
            return false;
        }

        return true;
    } catch (Exception e) {
        return false;
    }
}

/**
 * 设备绑定
 * 将设备绑定至指定用户和教室
 */
@Transactional
public void bindDevice(String deviceId, String userId, String classroomId) {
    // deviceRepository.updateBinding(deviceId, userId, classroomId);
}

/**
 * 设备解绑
 */
@Transactional
public void unbindDevice(String deviceId) {
    // deviceRepository.clearBinding(deviceId);
}

/**
 * 分页查询设备列表
 * 支持按学校、教室、类型、状态多维度过滤
 */

```

```

public Page<Device> queryDevices(String schoolId, String classroomId,
                                String deviceType, Integer status,
                                Pageable pageable) {
    // return deviceRepository.queryByConditions(schoolId, classroomId,
    //      deviceType, status, pageable);
    return null;
}

/**
 * 更新设备心跳
 * 心跳数据写入MySQL并更新Redis在线状态缓存
 */
public void updateHeartbeat(Device device) {
    // deviceRepository.updateHeartbeat(device.getId(),
    //      device.getLastHeartbeat(), device.getBatteryLevel(),
    //      device.getConnectedPenCount(), device.getCpuUsage(),
    //      device.getMemoryUsage());

    // 更新Redis在线状态（设置过期时间为心跳超时时间）
    updateDeviceOnlineStatus(device.getId(), true);
}

/**
 * 构建教室设备拓扑
 * 查询教室内所有设备，按类型分组并建立连接关系
 *
 * @param classroomId 教室ID
 * @return 拓扑结构（网关/算力盒/终端/笔）
 */
public ClassroomTopology buildClassroomTopology(String classroomId) {
    // 查询教室下所有设备
    // List<Device> devices = deviceRepository.findByClassroomId(classroomId);
    List<Device> devices = new ArrayList<>();

    ClassroomTopology topology = new ClassroomTopology();
    topology.setClassroomId(classroomId);

    // 按设备类型分组
    Map<String, List<Device>> grouped = devices.stream()
        .collect(Collectors.groupingBy(Device::getType));

    topology.setGateways(grouped.getDefault("gateway", new ArrayList<>()));
    topology.setEdgeBoxes(grouped.getDefault("edge_box", new ArrayList<>()));
    topology.setTerminals(grouped.getDefault("terminal", new ArrayList<>()));
    topology.setPens(grouped.getDefault("pen", new ArrayList<>()));
    topology.setTotalDeviceCount(devices.size());

    return topology;
}

/**
 * 批量检查设备在线状态
 * 通过Redis缓存快速判断设备是否在线
 */
public Map<String, Boolean> checkOnlineStatus(List<String> deviceIds) {
    Map<String, Boolean> result = new HashMap<>();
    for (String deviceId : deviceIds) {

```

```

        String key = "writech:device:online:" + deviceId;
        result.put(deviceId, Boolean.TRUE.equals(redisTemplate.hasKey(key)));
    }
    return result;
}

/**
 * 发送远程指令至设备
 * 通过MQTT向指定设备下发控制指令（重启/配置更新/OTA等）
 */
public void sendCommand(String deviceId, String command, Map<String, Object> params)
{
    // 构建MQTT消息
    Map<String, Object> message = new HashMap<>();
    message.put("command", command);
    message.put("params", params);
    message.put("timestamp", System.currentTimeMillis());

    // 根据设备类型确定Topic
    Device device = findById(deviceId);
    if (device == null) return;

    String topic;
    switch (device.getType()) {
        case "gateway":
            topic = "gateway/" + deviceId + "/command";
            break;
        case "edge_box":
            topic = "edgebox/" + deviceId + "/command";
            break;
        default:
            topic = "device/" + deviceId + "/command";
    }

    // mqttTemplate.convertAndSend(topic, message);
}

/**
 * 统计学校设备概况
 */
public DeviceOverview getSchoolDeviceOverview(String schoolId) {
    DeviceOverview overview = new DeviceOverview();
    // 各类型设备数量统计
    // overview.setTotalPens(deviceRepository.countBySchoolAndType(schoolId,
"pen"));
    // overview.setTotalGateways(deviceRepository.countBySchoolAndType(schoolId,
"gateway"));
    // overview.setOnlinePens(countOnlineDevices(schoolId, "pen"));
    // overview.setOnlineGateways(countOnlineDevices(schoolId, "gateway"));
    return overview;
}

// ===== 内部方法 =====

/** 更新Redis中设备在线状态 */
private void updateDeviceOnlineStatus(String deviceId, boolean online) {
    String key = "writech:device:online:" + deviceId;

```

```

        if (online) {
            redisTemplate.opsForValue().set(key, "1",
                DEVICE_ONLINE_TIMEOUT, java.util.concurrent.TimeUnit.SECONDS);
        } else {
            redisTemplate.delete(key);
        }
    }
}

// ===== 内部类 =====

/** 设备概况统计 */
public static class DeviceOverview {
    private int totalPens;
    private int totalGateways;
    private int totalEdgeBoxes;
    private int totalTerminals;
    private int onlinePens;
    private int onlineGateways;
    private int onlineEdgeBoxes;
    private double averageBatteryLevel;

    public int getTotalPens() { return totalPens; }
    public void setTotalPens(int c) { this.totalPens = c; }
    public int getTotalGateways() { return totalGateways; }
    public void setTotalGateways(int c) { this.totalGateways = c; }
    public int getTotalEdgeBoxes() { return totalEdgeBoxes; }
    public void setTotalEdgeBoxes(int c) { this.totalEdgeBoxes = c; }
    public int getTotalTerminals() { return totalTerminals; }
    public void setTotalTerminals(int c) { this.totalTerminals = c; }
    public int getOnlinePens() { return onlinePens; }
    public void setOnlinePens(int c) { this.onlinePens = c; }
    public int getOnlineGateways() { return onlineGateways; }
    public void setOnlineGateways(int c) { this.onlineGateways = c; }
    public int getOnlineEdgeBoxes() { return onlineEdgeBoxes; }
    public void setOnlineEdgeBoxes(int c) { this.onlineEdgeBoxes = c; }
    public double getAverageBatteryLevel() { return averageBatteryLevel; }
    public void setAverageBatteryLevel(double l) { this.averageBatteryLevel = l; }
}
}

```

## service/MessageService.java

```

/**
 * 自然写互动课堂教学管理云平台软件 V1.0
 *
 * 消息推送服务
 * 基于 WebSocket 实现多终端实时消息推送
 * 支持新作业通知、批改完成通知、课堂互动指令等
 */
package com.writech.cloud.service;

import org.springframework.beans.factory.annotation.Autowired;
import org.springframework.data.redis.core.StringRedisTemplate;
import org.springframework.stereotype.Service;

```

```

import org.springframework.web.socket.*;
import org.springframework.web.socket.handler.TextWebSocketHandler;
import org.springframework.web.socket.config.annotation.*;

import java.io.IOException;
import java.time.LocalDateTime;
import java.util.*;
import java.util.concurrent.ConcurrentHashMap;

/**
 * 消息服务类
 *
 * WebSocket实时消息通道: /ws/v1/notify
 *
 * 消息类型:
 * - ASSIGNMENT_NEW: 新作业通知
 * - ASSIGNMENT_GRADED: 批改完成通知
 * - STROKE_REALTIME: 实时笔迹数据推送
 * - CLASSROOM_INTERACTION: 课堂互动指令
 * - SYSTEM_NOTIFICATION: 系统公告
 */
@Service
public class MessageService extends TextWebSocketHandler implements WebSocketConfigurer
{

    @Autowired
    private StringRedisTemplate redisTemplate;

    /** 在线用户WebSocket会话映射 (userId → session列表, 支持多终端同时在线) */
    private final ConcurrentHashMap<String, List<WebSocketSession>> userSessions =
        new ConcurrentHashMap<>();

    /** 教室频道会话映射 (classroomId → session列表) */
    private final ConcurrentHashMap<String, List<WebSocketSession>> classroomChannels =
        new ConcurrentHashMap<>();

    /**
     * WebSocket端点注册
     */
    @Override
    public void registerWebSocketHandlers(WebSocketHandlerRegistry registry) {
        registry.addHandler(this, "/ws/v1/notify")
            .setAllowedOrigins("*");
    }

    /**
     * WebSocket连接建立
     * 从Token中解析用户ID, 注册到在线会话映射
     */
    @Override
    public void afterConnectionEstablished(WebSocketSession session) throws Exception {
        String userId = extractUserIdFromSession(session);
        if (userId != null) {
            // 注册用户会话
            userSessions.computeIfAbsent(userId, k -> new ArrayList<>()).add(session);
            // 更新在线状态
            updateOnlineStatus(userId, true);
        }
    }
}

```

```

        // 推送离线期间的未读消息
        pushOfflineMessages(userId, session);
    }
}

/**
 * WebSocket消息接收
 * 处理客户端发送的消息（心跳、课堂互动指令等）
 */
@Override
protected void handleTextMessage(WebSocketSession session, TextMessage message)
    throws Exception {
    String payload = message.getPayload();
    Map<String, Object> msg = parseMessage(payload);

    String type = (String) msg.get("type");
    if (type == null) return;

    switch (type) {
        case "HEARTBEAT":
            // 回复心跳
            session.sendMessage(new TextMessage("{\"type\":\"HEARTBEAT_ACK\"}"));
            break;
        case "JOIN_CLASSROOM":
            // 加入教室频道（课堂互动场景）
            String classroomId = (String) msg.get("classroomId");
            joinClassroomChannel(classroomId, session);
            break;
        case "LEAVE_CLASSROOM":
            // 离开教室频道
            String leaveClassroom = (String) msg.get("classroomId");
            leaveClassroomChannel(leaveClassroom, session);
            break;
        case "CLASSROOM_COMMAND":
            // 教师发送课堂控制指令（广播至教室内所有终端）
            broadcastToClassroom(msg);
            break;
        default:
            break;
    }
}

/**
 * WebSocket连接断开
 */
@Override
public void afterConnectionClosed(WebSocketSession session, CloseStatus status)
    throws Exception {
    String userId = extractUserIdFromSession(session);
    if (userId != null) {
        // 移除会话
        List<WebSocketSession> sessions = userSessions.get(userId);
        if (sessions != null) {
            sessions.remove(session);
            if (sessions.isEmpty()) {
                userSessions.remove(userId);
                updateOnlineStatus(userId, false);
            }
        }
    }
}

```

```

    }
    }
}
// 从教室频道移除
classroomChannels.values().forEach(list -> list.remove(session));
}

/**
 * 向指定用户推送消息
 * 支持多终端同时推送（手机/Pad/PC同时在线时都能收到）
 *
 * @param userId 目标用户ID
 * @param messageType 消息类型
 * @param data 消息数据
 */
public void pushToUser(String userId, String messageType, Map<String, Object> data)
{
    Map<String, Object> message = new HashMap<>();
    message.put("type", messageType);
    message.put("data", data);
    message.put("timestamp", System.currentTimeMillis());

    String json = toJson(message);
    List<WebSocketSession> sessions = userSessions.get(userId);

    if (sessions != null && !sessions.isEmpty()) {
        // 在线推送
        for (WebSocketSession session : sessions) {
            try {
                if (session.isOpen()) {
                    session.sendMessage(new TextMessage(json));
                }
            } catch (IOException e) {
                // 发送失败，记录日志
            }
        }
    } else {
        // 离线存储（用户上线后推送）
        storeOfflineMessage(userId, json);
    }
}

/**
 * 向班级所有学生推送消息
 *
 * @param classId 班级ID
 * @param messageType 消息类型
 * @param data 消息数据
 */
public void pushToClass(String classId, String messageType, Map<String, Object>
data) {
    // 查询班级学生列表
    // List<String> studentIds = classService.getStudentIds(classId);
    List<String> studentIds = new ArrayList<>();
    for (String studentId : studentIds) {
        pushToUser(studentId, messageType, data);
    }
}

```



```

    }

    /**
     * 向教室频道广播消息
     * 用于课堂互动场景，将消息推送至教室内所有终端（黑板/PC/电视/Pad）
     */
    public void broadcastToClassroom(Map<String, Object> message) {
        String classroomId = (String) message.get("classroomId");
        if (classroomId == null) return;

        String json = toJson(message);
        List<WebSocketSession> sessions = classroomChannels.get(classroomId);
        if (sessions != null) {
            for (WebSocketSession session : sessions) {
                try {
                    if (session.isOpen()) {
                        session.sendMessage(new TextMessage(json));
                    }
                } catch (IOException e) {
                    // 发送失败处理
                }
            }
        }
    }

    /**
     * 推送作业发布通知
     */
    public void pushAssignmentNotification(String classId, String title, String
assignmentId) {
        Map<String, Object> data = new HashMap<>();
        data.put("assignmentId", assignmentId);
        data.put("title", title);
        data.put("message", "教师发布了新作业: " + title);
        pushToClass(classId, "ASSIGNMENT_NEW", data);
    }

    /**
     * 推送批改完成通知
     */
    public void pushGradingNotification(String studentId, String assignmentTitle,
double score) {
        Map<String, Object> data = new HashMap<>();
        data.put("title", assignmentTitle);
        data.put("score", score);
        data.put("message", "作业\""+ assignmentTitle + "\"批改完成, 得分: " + score);
        pushToUser(studentId, "ASSIGNMENT_GRADED", data);
    }

    /**
     * 推送实时笔迹数据至教室大屏
     * 低延迟推送，用于黑板/电视大屏实时展示学生书写过程
     */
    public void pushRealtimeStroke(String classroomId, String studentId,
List<Map<String, Object>> strokePoints) {
        Map<String, Object> data = new HashMap<>();
        data.put("studentId", studentId);
    }

```

```

        data.put("points", strokePoints);

        Map<String, Object> message = new HashMap<>();
        message.put("type", "STROKE_REALTIME");
        message.put("classroomId", classroomId);
        message.put("data", data);

        broadcastToClassroom(message);
    }

    // ===== 内部方法 =====

    /** 加入教室频道 */
    private void joinClassroomChannel(String classroomId, WebSocketSession session) {
        classroomChannels.computeIfAbsent(classroomId, k -> new ArrayList<>
()).add(session);
    }

    /** 离开教室频道 */
    private void leaveClassroomChannel(String classroomId, WebSocketSession session) {
        List<WebSocketSession> sessions = classroomChannels.get(classroomId);
        if (sessions != null) {
            sessions.remove(session);
        }
    }

    /** 从WebSocket会话中提取用户ID */
    private String extractUserIdFromSession(WebSocketSession session) {
        // 从URL参数或握手头中的Token解析用户ID
        String query = session.getUri() != null ? session.getUri().getQuery() : null;
        if (query != null && query.contains("token=")) {
            // 解析Token获取userId
            return "extracted_user_id";
        }
        return null;
    }

    /** 更新用户在线状态 */
    private void updateOnlineStatus(String userId, boolean online) {
        String key = "writech:user:online:" + userId;
        if (online) {
            redisTemplate.opsForValue().set(key, "1");
        } else {
            redisTemplate.delete(key);
        }
    }

    /** 存储离线消息 */
    private void storeOfflineMessage(String userId, String message) {
        String key = "writech:offline:msg:" + userId;
        redisTemplate.opsForList().rightPush(key, message);
        // 最多保留100条离线消息
        redisTemplate.opsForList().trim(key, -100, -1);
    }

    /** 推送离线期间积累的未读消息 */
    private void pushOfflineMessages(String userId, WebSocketSession session)

```

```

        throws IOException {
            String key = "writech:offline:msg:" + userId;
            List<String> messages = redisTemplate.opsForList().range(key, 0, -1);
            if (messages != null) {
                for (String msg : messages) {
                    session.sendMessage(new TextMessage(msg));
                }
                redisTemplate.delete(key);
            }
        }
    }

    /** JSON序列化（简化版本） */
    private String toJson(Map<String, Object> map) {
        StringBuilder sb = new StringBuilder("{");
        boolean first = true;
        for (Map.Entry<String, Object> entry : map.entrySet()) {
            if (!first) sb.append(",");
            sb.append("\"").append(entry.getKey()).append("\":");
            Object value = entry.getValue();
            if (value instanceof String) {
                sb.append("\"").append(value).append("\"");
            } else {
                sb.append(value);
            }
            first = false;
        }
        sb.append("}");
        return sb.toString();
    }

    /** JSON解析（简化版本） */
    private Map<String, Object> parseMessage(String json) {
        return new HashMap<>();
    }

    /**
     * 获取在线用户统计
     */
    public Map<String, Integer> getOnlineStats() {
        Map<String, Integer> stats = new HashMap<>();
        stats.put("totalOnlineUsers", userSessions.size());
        stats.put("totalSessions", userSessions.values().stream()
            .mapToInt(List::size).sum());
        stats.put("activeClassrooms", classroomChannels.size());
        return stats;
    }
}

```

## service/StrokeService.java

```

/**
 * 自然写互动课堂教学管理云平台软件 V1.0
 *
 * 笔迹数据处理服务

```

```

    * 负责笔迹数据的Kafka消费、存储、AI引擎调度
    */
package com.writech.cloud.service;

import com.writech.cloud.model.StrokeData;

import org.springframework.beans.factory.annotation.Autowired;
import org.springframework.data.mongodb.core.MongoTemplate;
import org.springframework.data.mongodb.core.query.Criteria;
import org.springframework.data.mongodb.core.query.Query;
import org.springframework.kafka.annotation.KafkaListener;
import org.springframework.kafka.core.KafkaTemplate;
import org.springframework.stereotype.Service;

import java.time.LocalDateTime;
import java.util.*;
import java.util.concurrent.*;
import java.util.stream.Collectors;

/**
 * 笔迹数据服务
 *
 * 数据流处理管道:
 * 1. 网关/算力盒通过MQTT上报笔迹数据到云平台
 * 2. 云平台接收服务将数据推入Kafka消息队列
 * 3. 本服务作为Kafka消费者接收并处理数据
 * 4. 原始笔迹数据存入MongoDB（高写入吞吐量）
 * 5. 触发AI引擎异步识别（OCR/数学/笔顺）
 * 6. 识别结果回写MongoDB，推送至各终端
 */
@Service
public class StrokeService {

    @Autowired
    private MongoTemplate mongoTemplate;

    @Autowired
    private KafkaTemplate<String, String> kafkaTemplate;

    /** AI引擎调用线程池 */
    private final ExecutorService aiExecutor = Executors.newFixedThreadPool(16);

    /** AI引擎服务地址 */
    private static final String AI_ENGINE_URL = "http://ai-engine-service:8001";

    /** 笔迹数据MongoDB集合名 */
    private static final String STROKE_COLLECTION = "stroke_data";

    /** 识别结果MongoDB集合名 */
    private static final String RESULT_COLLECTION = "recognition_result";

    /**
     * Kafka消费者：接收笔迹数据
     * 监听 writtech-stroke-topic 主题，批量消费笔迹数据
     *
     * @param message JSON格式的笔迹数据
     */

```

```

@KafkaListener(topics = "writech-stroke-topic", groupId = "stroke-consumer-group")
public void consumeStrokeData(String message) {
    try {
        // 解析笔迹数据JSON
        StrokeData strokeData = parseStrokeData(message);
        if (strokeData == null) return;

        // 数据预处理（坐标校验、时间戳排序、去重）
        preprocessStrokeData(strokeData);

        // 写入MongoDB存储
        saveToMongoDB(strokeData);

        // 判断是否需要触发AI识别
        if (shouldTriggerRecognition(strokeData)) {
            // 异步调用AI引擎
            submitRecognitionTask(strokeData);
        }

    } catch (Exception e) {
        // 处理失败的消息发送到死信队列
        kafkaTemplate.send("writech-stroke-dlq", message);
    }
}

/**
 * 保存笔迹数据到MongoDB
 * 使用批量写入提升性能，每批最多500条
 */
public void saveToMongoDB(StrokeData strokeData) {
    strokeData.setCreateTime(LocalDateTime.now());
    strokeData.setProcessingStatus("received");
    mongoTemplate.save(strokeData, STROKE_COLLECTION);
}

/**
 * 批量保存笔迹数据
 * 用于网关批量上传场景，提升写入吞吐量
 */
public void batchSave(List<StrokeData> strokeDataList) {
    if (strokeDataList == null || strokeDataList.isEmpty()) return;

    LocalDateTime now = LocalDateTime.now();
    for (StrokeData data : strokeDataList) {
        data.setCreateTime(now);
        data.setProcessingStatus("received");
    }

    // MongoDB批量插入
    mongoTemplate.insertAll(strokeDataList);
}

/**
 * 查询学生笔迹数据
 *
 * @param studentId 学生ID
 * @param assignmentId 作业ID（可选）

```

```

    * @param startTime 开始时间 (可选)
    * @param endTime 结束时间 (可选)
    * @return 笔迹数据列表
    */
    public List<StrokeData> queryStrokes(String studentId, String assignmentId,
                                         LocalDateTime startTime, LocalDateTime
endTime) {
        Query query = new Query();
        query.addCriteria(Criteria.where("studentId").is(studentId));

        if (assignmentId != null) {
            query.addCriteria(Criteria.where("assignmentId").is(assignmentId));
        }
        if (startTime != null && endTime != null) {
            query.addCriteria(Criteria.where("timestamp")
                                .gte(startTime).lte(endTime));
        }

        // 按时间戳排序 (回放场景需要)
        query.with(org.springframework.data.domain.Sort.by(
            org.springframework.data.domain.Sort.Direction.ASC, "timestamp"));

        return mongoTemplate.find(query, StrokeData.class, STROKE_COLLECTION);
    }

    /**
     * 提交AI识别任务
     * 将笔迹数据异步发送至AI引擎进行识别
     */
    private void submitRecognitionTask(StrokeData strokeData) {
        aiExecutor.submit(() -> {
            try {
                // 根据作业题目类型选择识别方式
                String recognitionType = determineRecognitionType(strokeData);

                // 调用AI引擎REST API
                Map<String, Object> requestBody = new HashMap<>();
                requestBody.put("strokeId", strokeData.getId());
                requestBody.put("studentId", strokeData.getStudentId());
                requestBody.put("strokes", strokeData.getStrokes());
                requestBody.put("type", recognitionType);

                // String apiUrl = AI_ENGINE_URL + "/api/v1/ocr/recognize";
                // RestTemplate restTemplate = new RestTemplate();
                // ResponseEntity<String> response = restTemplate.postForEntity(
                //     apiUrl, requestBody, String.class);

                // 保存识别结果
                // saveRecognitionResult(strokeData.getId(), response.getBody());

                // 更新笔迹数据处理状态
                updateProcessingStatus(strokeData.getId(), "completed");

            } catch (Exception e) {
                updateProcessingStatus(strokeData.getId(), "failed");
            }
        });
    }

```

```

}

/**
 * 笔迹数据预处理
 * - 坐标范围校验（过滤异常值）
 * - 时间戳排序
 * - 重复数据去重
 * - 坐标归一化（适配不同纸面规格）
 */
private void preprocessStrokeData(StrokeData strokeData) {
    if (strokeData.getStrokes() == null) return;

    List<Map<String, Object>> processed = strokeData.getStrokes().stream()
        // 过滤无效坐标点
        .filter(point -> {
            int x = ((Number) point.getDefault("x", -1)).intValue();
            int y = ((Number) point.getDefault("y", -1)).intValue();
            return x >= 0 && x <= 65535 && y >= 0 && y <= 65535;
        })
        // 按时间戳排序
        .sorted((a, b) -> {
            long ta = ((Number) a.getDefault("timestamp", 0L)).longValue();
            long tb = ((Number) b.getDefault("timestamp", 0L)).longValue();
            return Long.compare(ta, tb);
        })
        .collect(Collectors.toList());

    // 去重（相同时间戳的重复点）
    List<Map<String, Object>> deduplicated = new ArrayList<>();
    long lastTimestamp = -1;
    for (Map<String, Object> point : processed) {
        long ts = ((Number) point.getDefault("timestamp", 0L)).longValue();
        if (ts != lastTimestamp) {
            deduplicated.add(point);
            lastTimestamp = ts;
        }
    }

    strokeData.setStrokes(deduplicated);
}

/**
 * 判断是否需要触发AI识别
 * - 抬笔事件（笔画结束）触发单字识别
 * - 作业提交事件触发整页识别
 * - 超过5秒无新数据触发段落识别
 */
private boolean shouldTriggerRecognition(StrokeData strokeData) {
    // 如果关联了作业ID，则需要识别
    if (strokeData.getAssignmentId() != null) {
        return true;
    }
    // 检查是否有抬笔标记
    if (strokeData.getStrokes() != null) {
        return strokeData.getStrokes().stream()
            .anyMatch(p -> Boolean.TRUE.equals(p.get("penUp")));
    }
}

```

```

        return false;
    }

    /** 确定识别类型 */
    private String determineRecognitionType(StrokeData strokeData) {
        // 根据作业题目类型确定: ocr/math/stroke_order/essay
        return "ocr";
    }

    /** 解析笔迹数据JSON */
    private StrokeData parseStrokeData(String json) {
        // JSON反序列化
        return null;
    }

    /** 更新处理状态 */
    private void updateProcessingStatus(String strokeId, String status) {
        Query query = new Query(Criteria.where("_id").is(strokeId));
        org.springframework.data.mongodb.core.query.Update update =
            new org.springframework.data.mongodb.core.query.Update();
        update.set("processingStatus", status);
        update.set("processedTime", LocalDateTime.now());
        mongoTemplate.updateFirst(query, update, STROKE_COLLECTION);
    }
}

```

## service/UserService.java

```

/**
 * 自然写互动课堂教学管理云平台软件 V1.0
 *
 * 用户与权限服务
 * 实现 RBAC 角色权限模型, 管理教师/学生/管理员/家长四级权限
 */
package com.writech.cloud.service;

import com.writech.cloud.model.User;

import org.springframework.beans.factory.annotation.Autowired;
import org.springframework.data.redis.core.StringRedisTemplate;
import org.springframework.security.crypto.bcrypt.BCryptPasswordEncoder;
import org.springframework.stereotype.Service;
import org.springframework.transaction.annotation.Transactional;

import java.time.LocalDateTime;
import java.util.*;
import java.util.concurrent.TimeUnit;

/**
 * 用户服务类
 *
 * 提供用户管理、身份验证、权限控制、Token管理等核心功能
 * RBAC权限模型: 管理员 > 教师 > 学生/家长
 * - 管理员: 系统全局管理 (学校/用户/设备管理)

```



```

* - 教师：班级管理、作业发布批改、学情查看
* - 学生：作业查看、学习数据查询
* - 家长：子女学情查看、消息接收
*/
@Service
public class UserService {

    @Autowired
    private StringRedisTemplate redisTemplate;

    /** 密码加密器（BCrypt算法，强度因子10） */
    private final BCryptPasswordEncoder passwordEncoder = new BCryptPasswordEncoder(10);

    /** Token黑名单前缀（存储在Redis中） */
    private static final String TOKEN_BLACKLIST_PREFIX = "writech:token:blacklist:";

    /** 短信验证码前缀 */
    private static final String SMS_CODE_PREFIX = "writech:sms:code:";

    /** 验证码有效期（秒） */
    private static final long SMS_CODE_EXPIRE = 300;

    /** 验证码发送间隔（秒） */
    private static final long SMS_CODE_INTERVAL = 60;

    /**
     * 手机号+密码验证登录
     *
     * @param phone 手机号
     * @param password 明文密码
     * @return 验证通过返回用户对象，失败返回null
     */
    public User verifyByPassword(String phone, String password) {
        if (phone == null || password == null) {
            return null;
        }

        // 查询用户（手机号AES解密后匹配）
        User user = findByPhone(phone);
        if (user == null) {
            return null;
        }

        // BCrypt密码比对
        if (passwordEncoder.matches(password, user.getPasswordHash())) {
            return user;
        }

        // 登录失败计数（防暴力破解，5次失败后锁定30分钟）
        incrementLoginFailCount(user.getId());
        return null;
    }

    /**
     * 手机号+短信验证码验证登录
     */
    public User verifyBySmsCode(String phone, String smsCode) {

```

```

        if (phone == null || smsCode == null) {
            return null;
        }

        // 从Redis获取验证码
        String key = SMS_CODE_PREFIX + phone;
        String storedCode = redisTemplate.opsForValue().get(key);

        if (storedCode == null || !storedCode.equals(smsCode)) {
            return null;
        }

        // 验证码匹配成功，删除已使用的验证码
        redisTemplate.delete(key);

        // 查找或自动注册用户
        User user = findByPhone(phone);
        if (user == null) {
            // 首次登录自动创建账户
            user = autoRegister(phone);
        }

        return user;
    }

    /**
     * 微信授权登录验证
     */
    public User verifyByWechat(String wechatCode) {
        if (wechatCode == null) return null;

        // 调用微信开放平台API获取用户openId
        String openId = exchangeWechatOpenId(wechatCode);
        if (openId == null) return null;

        // 查找绑定的用户
        User user = findByWechatOpenId(openId);
        return user;
    }

    /**
     * 钉钉授权登录验证
     */
    public User verifyByDingtalk(String dingtalkCode) {
        if (dingtalkCode == null) return null;
        String userId = exchangeDingtalkUserId(dingtalkCode);
        if (userId == null) return null;
        return findByDingtalkUserId(userId);
    }

    /**
     * 发送短信验证码
     *
     * @param phone 手机号
     * @throws RuntimeException 发送频率过高时抛出异常
     */
    public void sendSmsVerificationCode(String phone) {

```

```

        // 检查发送频率 (60秒内不可重复发送)
        String intervalKey = SMS_CODE_PREFIX + "interval:" + phone;
        if (Boolean.TRUE.equals(redisTemplate.hasKey(intervalKey))) {
            throw new RuntimeException("验证码发送过于频繁, 请60秒后重试");
        }

        // 生成6位随机验证码
        String code = String.format("%06d", new Random().nextInt(1000000));

        // 存入Redis (5分钟有效期)
        String codeKey = SMS_CODE_PREFIX + phone;
        redisTemplate.opsForValue().set(codeKey, code, SMS_CODE_EXPIRE,
            TimeUnit.SECONDS);

        // 设置发送间隔标记 (60秒)
        redisTemplate.opsForValue().set(intervalKey, "1", SMS_CODE_INTERVAL,
            TimeUnit.SECONDS);

        // 调用短信服务发送验证码
        sendSms(phone, code);
    }

    /**
     * 查询用户信息
     */
    public User findById(String userId) {
        // 先查Redis缓存
        // User cachedUser = getCachedUser(userId);
        // if (cachedUser != null) return cachedUser;
        // 查数据库
        // User user = userRepository.findById(userId).orElse(null);
        // if (user != null) cacheUser(user);
        return null;
    }

    /**
     * 根据手机号查询用户
     * 手机号在数据库中AES-256加密存储, 查询时需加密后匹配
     */
    public User findByPhone(String phone) {
        String encryptedPhone = encryptField(phone);
        // return userRepository.findByEncryptedPhone(encryptedPhone);
        return null;
    }

    /**
     * 更新用户登录信息
     */
    public void updateLoginInfo(String userId, LocalDateTime loginTime, String loginIp)
    {
        // userRepository.updateLoginInfo(userId, loginTime, loginIp);
    }

    /**
     * 验证密码
     */
    public boolean verifyPassword(String userId, String password) {

```

```

        User user = findById(userId);
        if (user == null) return false;
        return passwordEncoder.matches(password, user.getPasswordHash());
    }

    /**
     * 更新密码
     * 密码使用BCrypt加密后存储, 强度因子10
     */
    @Transactional
    public void updatePassword(String userId, String newPassword) {
        // 密码强度校验 (最少8位, 包含大小写字母和数字)
        if (!isStrongPassword(newPassword)) {
            throw new RuntimeException("密码强度不足, 需包含大小写字母和数字, 不少于8位");
        }

        String passwordHash = passwordEncoder.encode(newPassword);
        // userRepository.updatePassword(userId, passwordHash);
    }

    /**
     * 将Token加入黑名单 (使其立即失效)
     * 黑名单存储在Redis中, 有效期与Token过期时间一致
     */
    public void invalidateToken(String token) {
        String key = TOKEN_BLACKLIST_PREFIX + token;
        redisTemplate.opsForValue().set(key, "1", 7200, TimeUnit.SECONDS);
    }

    /**
     * 使用户所有Token失效 (强制重新登录)
     */
    public void invalidateAllTokens(String userId) {
        // 更新用户tokenVersion字段, 旧版本Token将在校验时失效
        // userRepository.incrementTokenVersion(userId);
    }

    /**
     * 检查Token是否在黑名单中
     */
    public boolean isTokenBlacklisted(String token) {
        String key = TOKEN_BLACKLIST_PREFIX + token;
        return Boolean.TRUE.equals(redisTemplate.hasKey(key));
    }

    /**
     * 创建用户
     * 管理员创建教师/学生/家长账户
     */
    @Transactional
    public User createUser(CreateUserRequest request) {
        // 检查手机号唯一性
        if (request.getPhone() != null && findByPhone(request.getPhone()) != null) {
            throw new RuntimeException("手机号已被注册");
        }

        User user = new User();
    }

```

```

        user.setId(UUID.randomUUID().toString().replace("-", ""));
        user.setName(request.getName());
        user.setPhone(request.getPhone());
        user.setRole(request.getRole());
        user.setSchoolId(request.getSchoolId());
        user.setSchoolName(request.getSchoolName());
        user.setStatus(1);
        user.setCreateTime(LocalDateTime.now());

        // 加密手机号存储
        if (request.getPhone() != null) {
            user.setEncryptedPhone(encryptField(request.getPhone()));
        }

        // 设置初始密码
        if (request.getPassword() != null) {
            user.setPasswordHash(passwordEncoder.encode(request.getPassword()));
        }

        // userRepository.save(user);
        return user;
    }

    /**
     * 查询学校下的用户列表
     * 按角色过滤（教师/学生/家长）
     */
    public List<User> findBySchoolAndRole(String schoolId, String role) {
        // return userRepository.findBySchoolIdAndRole(schoolId, role);
        return new ArrayList<>();
    }

    // ===== 内部方法 =====

    /** 自动注册用户（首次短信登录） */
    private User autoRegister(String phone) {
        User user = new User();
        user.setId(UUID.randomUUID().toString().replace("-", ""));
        user.setPhone(phone);
        user.setEncryptedPhone(encryptField(phone));
        user.setRole("parent"); // 默认家长角色
        user.setStatus(1);
        user.setCreateTime(LocalDateTime.now());
        return user;
    }

    /** 登录失败计数（防暴力破解） */
    private void incrementLoginFailCount(String userId) {
        String key = "writech:login:fail:" + userId;
        Long count = redisTemplate.opsForValue().increment(key);
        if (count != null && count == 1) {
            redisTemplate.expire(key, 1800, TimeUnit.SECONDS); // 30分钟窗口
        }
        if (count != null && count >= 5) {
            // 锁定账户30分钟
            String lockKey = "writech:login:lock:" + userId;
            redisTemplate.opsForValue().set(lockKey, "1", 1800, TimeUnit.SECONDS);
        }
    }

```

```

    }
}

/** AES-256加密字段（手机号、身份信息敏感数据） */
private String encryptField(String plainText) {
    // 使用AES-256-CBC模式加密
    // Cipher cipher = Cipher.getInstance("AES/CBC/PKCS5Padding");
    // 实际实现使用配置的密钥
    return Base64.getEncoder().encodeToString(plainText.getBytes());
}

/** AES-256解密字段 */
private String decryptField(String cipherText) {
    return new String(Base64.getDecoder().decode(cipherText));
}

/** 密码强度校验 */
private boolean isStrongPassword(String password) {
    if (password == null || password.length() < 8) return false;
    boolean hasUpper = false, hasLower = false, hasDigit = false;
    for (char c : password.toCharArray()) {
        if (Character.isUpperCase(c)) hasUpper = true;
        if (Character.isLowerCase(c)) hasLower = true;
        if (Character.isDigit(c)) hasDigit = true;
    }
    return hasUpper && hasLower && hasDigit;
}

/** 微信OpenId获取（模拟） */
private String exchangeWechatOpenId(String code) {
    // 调用 https://api.weixin.qq.com/sns/oauth2/access_token
    return null;
}

/** 钉钉UserId获取（模拟） */
private String exchangeDingtalkUserId(String code) {
    return null;
}

private User findByWechatOpenId(String openId) { return null; }
private User findByDingtalkUserId(String userId) { return null; }
private void sendSms(String phone, String code) { /* 调用短信服务商API */ }

// ===== 请求 DTO =====

public static class CreateUserRequest {
    private String name;
    private String phone;
    private String password;
    private String role;
    private String schoolId;
    private String schoolName;

    public String getName() { return name; }
    public void setName(String n) { this.name = n; }
    public String getPhone() { return phone; }
    public void setPhone(String p) { this.phone = p; }
}

```

```
    public String getPassword() { return password; }  
    public void setPassword(String p) { this.password = p; }  
    public String getRole() { return role; }  
    public void setRole(String r) { this.role = r; }  
    public String getSchoolId() { return schoolId; }  
    public void setSchoolId(String id) { this.schoolId = id; }  
    public String getSchoolName() { return schoolName; }  
    public void setSchoolName(String n) { this.schoolName = n; }  
}  
}
```