软件全称：盛果智能应用软件
软件简称：盛果
版 本 号：V1.0

说明：本文档包含软件前30页和后30页源代码，每页50行。
前30页为软件前端代码，后30页为软件后端代码。

第一部分：前端源代码（前30页）

```
// ============================================
// 文件: my-uniapp-vue3/src/main.ts
// ============================================
import { createSSRApp } from 'vue';
import { createPinia } from 'pinia';
import App from './App.vue';
import { useUserStore } from './store/user';
// #ifdef H5
import VConsole from 'vconsole';
// #endif
export function createApp() {
  const app = createSSRApp(App);
  const pinia = createPinia();
  app.use(pinia);
  // 初始化用户状态
  const userStore = useUserStore();
  userStore.initUser();
  // #ifdef H5
  // vConsole 开启，用于调试微信支付问题
  new VConsole();
  // #endif
  return {
    app,
    pinia,
  };
}

// ============================================
// 文件: my-uniapp-vue3/src/App.vue
// ============================================
<script setup lang="ts">
import { onLaunch, onShow, onHide } from '@dcloudio/uni-app';
import { createPinia } from 'pinia';
import { useUserStore } from './store/user';
import { initDebug } from './utils/debug';
import { initAnalytics } from './utils/analytics';
```

```
import MiniPlayer from './components/MiniPlayer.vue';
import CreditInsufficientModal from './components/CreditInsufficientModal.vue';
// 初始化全局调试工具
initDebug();
// ==================== 微信环境检测 ====================
function isInWechatBrowser(): boolean {
  // #ifdef H5
  const ua = navigator.userAgent.toLowerCase();
  return ua.includes('micromessenger');
  // #endif
  return false;
}
// ==================== 微信 OAuth 回调处理 ====================
async function handleWechatOAuthCallback() {
  // 只在 H5 环境处理
  // #ifdef H5
  const searchParams = new URLSearchParams(window.location.search);
  const code = searchParams.get('code');
  const state = searchParams.get('state');
  console.log('[App] handleWechatOAuthCallback 触发', { code: !!code, fullUrl: window.location.href });
  if (!code) {
    console.log('[App] 无 code 参数，跳过 OAuth 处理');
    return; // 不是 OAuth 回调，跳过
  }
  console.log('[App] 检测到微信 OAuth 回调，处理 code...');
  uni.showLoading({ title: '授权中...' });
  try {
    const { post } = await import('./utils/request');
    const result = await post<{ openid: string }>('/payment/wechat/openid', { code });
    if (result.openid) {
      console.log('[App] OpenID 获取成功');
      // 先尝试根据 openid 自动登录
      try {
        const userStore = useUserStore();
        await userStore.loginByOpenid(result.openid);
        console.log('[App] OpenID 已绑定，自动登录成功');
        // 清除 URL 参数，停留在当前页
        searchParams.delete('code');
        searchParams.delete('state');
        const newSearch = searchParams.toString();
        const newUrl = window.location.origin + window.location.pathname
          + (newSearch ? '?' + newSearch : '') + window.location.hash;
        window.history.replaceState({}, '', newUrl);
        uni.hideLoading();
        uni.showToast({ title: '自动登录成功', icon: 'success' });
        return;
      } catch (e: any) {
        if (e?.message?.includes('未绑定') || e?.code === -1) {
          // 自动登录失败（未绑定账号），继续正常流程
          console.log('[App] OpenID 未绑定账号，将存储待登录');
        } else {
          console.error('[App] 自动登录异常:', e);
        }
      }
      // 未绑定账号或自动登录失败，存储 openid 等待手机号登录绑定
      localStorage.setItem('wx_openid', result.openid);
      //    标记刚从 OAuth 回来，页面组件可据此自动触发支付
      try { sessionStorage.setItem('wx_oauth_just_done', '1'); } catch (_) {}
```

```
      // 恢复之前保存的页面上下文
      let returnUrl: string | null = null;
      try {
        returnUrl = sessionStorage.getItem('wx_oauth_return');
        sessionStorage.removeItem('wx_oauth_return');
      } catch (_) {}
      if (returnUrl) {
        // 跳回原始页面（保留完整 hash 路由）
        // 用 location.replace 避免产生多余的历史记录
        window.location.replace(returnUrl);
      } else {
        // 没有上下文，清除 URL 参数后停留在当前页
        searchParams.delete('code');
        searchParams.delete('state');
        const newSearch = searchParams.toString();
        const newUrl = window.location.origin + window.location.pathname

// ============================================
// 文件: my-uniapp-vue3/src/config.ts
// ============================================
/**
 * 全局配置
 */
import { getApiBaseUrl } from './utils/config'
export const BASE_URL = getApiBaseUrl()

// ============================================
// 文件: my-uniapp-vue3/src/store/user.ts
// ============================================
import { defineStore } from 'pinia';
import { ref, computed } from 'vue';
import { getToken, setToken, removeToken, getUserInfo, setUserInfo, removeUserInfo } from '../utils/storage';
import { get, post } from '../utils/request';
import type { UserInfo, MemberStatus } from '../types';
export const useUserStore = defineStore('user', () => {
  // 状态
  const token = ref<string | null>(getToken());
  const userInfo = ref<UserInfo | null>(getUserInfo<UserInfo>());
  const memberStatus = ref<MemberStatus | null>(null);
  // 计算属性
  const isLoggedIn = computed(() => !!token.value && !!userInfo.value);
  const isMember = computed(() => memberStatus.value?.isValid && memberStatus.value.level > 0);
  // 初始化用户状态
  function initUser() {
    // 检查是否有保存的登录状态
    const savedToken = getToken();
    const savedUserInfo = getUserInfo<UserInfo>();
    // 如果有有效的登录状态，使用它
    if (savedToken && savedUserInfo && savedToken !== 'test-token-for-dev') {
      token.value = savedToken;
      userInfo.value = savedUserInfo;
      fetchMemberStatus();
    }
    // 否则不设置默认用户，等待用户主动登录
  }
  // 登录
  async function login(phone: string, code: string) {
    const inviteCode = uni.getStorageSync('invite_code') || '';
```

```
    // 读取微信 openid（从公众号进入时已存储在 localStorage）
    const wxOpenid = (typeof localStorage !== 'undefined' && localStorage.getItem('wx_openid')) || '';
    const result = await post<{
      token: string;
      user: UserInfo;
    }>('/auth/login', { phone, code, inviteCode, openid: wxOpenid });
    token.value = result.token;
    userInfo.value = result.user;
    setToken(result.token);
    setUserInfo(result.user);
    // 清除已使用的邀请码
    if (inviteCode) {
      uni.removeStorageSync('invite_code');
    }
    // 绑定成功后清除 localStorage 中的 openid（避免后续登录重复携带）
    if (wxOpenid && typeof localStorage !== 'undefined') {
      localStorage.removeItem('wx_openid');
    }
    await fetchMemberStatus();
    return result;
  }
  // 发送验证码
  async function sendCode(phone: string) {
    return post('/auth/send-code', { phone });
  }
  // 获取用户信息
  async function fetchUserInfo() {
    const info = await get<UserInfo>('/auth/user-info');
    userInfo.value = info;
    setUserInfo(info);
    return info;
  }
  // 获取会员状态
  async function fetchMemberStatus() {
    try {
      const status = await get<MemberStatus>('/member/status');
      memberStatus.value = status;
      return status;
    } catch {
      memberStatus.value = null;
      return null;
    }
  }
  // 登出
  function logout() {
    token.value = null;
    userInfo.value = null;
    memberStatus.value = null;
    removeToken();
    removeUserInfo();
  }

// ============================================
// 文件: my-uniapp-vue3/src/store/audio.ts
// ============================================
import { defineStore } from 'pinia';
import { ref, computed } from 'vue';
import { get, post, getFullUrl } from '../utils/request';
```

```
import type { AudioItem, Voice, VoiceParams } from '../types';
export const useAudioStore = defineStore('audio', () => {
  // 状态
  const voices = ref<Voice[]>([]);
  const currentAudio = ref<AudioItem | null>(null);
  const playlist = ref<AudioItem[]>([]);
  const currentIndex = ref(-1);
  const isPlaying = ref(false);
  const currentTime = ref(0);
  const duration = ref(0);
  const playRate = ref(1);
  // 播放模式: 'sequence' | 'loop' | 'single' | 'random'
  const playMode = ref<'sequence' | 'loop' | 'single' | 'random'>('sequence');
  // 音频上下文 - 双模式：BackgroundAudioManager（App后台播放）+ InnerAudioContext（H5降级）
  let audioContext: UniApp.InnerAudioContext | null = null;
  let bgAudioManager: any = null;
  // H5 环境不支持 BackgroundAudioManager，使用 InnerAudioContext
  const useBackgroundAudio: boolean = (function() {
    // #ifdef H5
    return false;
    // #endif
    // #ifndef H5
    return true;
    // #endif
  })();
  // 计算属性
  const hasPlaylist = computed(() => playlist.value.length > 0);
  const hasNext = computed(() => currentIndex.value < playlist.value.length - 1);
  const hasPrev = computed(() => currentIndex.value > 0);
  // 获取当前活跃的音频上下文（背景模式优先）
  function getActiveContext() {
    if (useBackgroundAudio && bgAudioManager) return bgAudioManager;
    return audioContext;
  }
  // 初始化音频上下文
  function initAudioContext() {
    if (useBackgroundAudio) {
      if (!bgAudioManager) {
        try {
          bgAudioManager = uni.getBackgroundAudioManager();
          setupBackgroundAudioEvents();
          console.log('[Audio] 使用 BackgroundAudioManager 支持后台播放');
        } catch (e) {
          console.warn('[Audio] BackgroundAudioManager 不可用，降级到 InnerAudioContext');
          initInnerAudioContext();
        }
      }
      return;
    }
    initInnerAudioContext();
  }
  function initInnerAudioContext() {
    if (audioContext) return;
    audioContext = uni.createInnerAudioContext();
    audioContext.onPlay(() => {
      isPlaying.value = true;
    });
    audioContext.onPause(() => {
```

```
    isPlaying.value = false;
  });
  audioContext.onEnded(() => {
    isPlaying.value = false;
    if (hasNext.value) {
      playNext();
    } else {
      handlePlayMode();
    }
  });
  audioContext.onTimeUpdate(() => {
    currentTime.value = audioContext?.currentTime || 0;
    const d = audioContext?.duration || 0;
    if (d > 0 && d < 3600) {
      if (duration.value === 0 || duration.value > 3600 || Math.abs(duration.value - d) < 1) {
        duration.value = d;
      }
    }
  });

// ═══════════════════════════════════════
// 文件: my-uniapp-vue3/src/store/notification.ts
// ═══════════════════════════════════════
import { defineStore } from 'pinia';
import { ref, computed } from 'vue';
export interface Notification {
  id: string;
  type: 'audio_complete' | 'video_complete' | 'content_complete' | 'error';
  title: string;
  message: string;
  bookId?: string | number;
  timestamp: number;
  read: boolean;
}
const STORAGE_KEY = 'app_notifications';
function loadFromStorage(): Notification[] {
  try {
    const data = uni.getStorageSync(STORAGE_KEY);
    return data ? JSON.parse(data) : [];
  } catch {
    return [];
  }
}
function saveToStorage(notifications: Notification[]) {
  try {
    uni.setStorageSync(STORAGE_KEY, JSON.stringify(notifications));
  } catch (e) {
    console.error('[Notification] 保存失败:', e);
  }
}
export const useNotificationStore = defineStore('notification', () => {
  const notifications = ref<Notification[]>(loadFromStorage());
  const unreadCount = computed(() => notifications.value.filter(n => !n.read).length);
  function add(notif: Omit<Notification, 'id' | 'timestamp' | 'read'>) {
    const id = `notif_${Date.now()}_${Math.random().toString(36).substr(2, 6)}`;
    const notification: Notification = {
      ...notif,
      id,
```

```
      timestamp: Date.now(),
      read: false,
    };
    notifications.value.unshift(notification);
    // 最多保留 50 条
    if (notifications.value.length > 50) {
      notifications.value = notifications.value.slice(0, 50);
    }
    saveToStorage(notifications.value);
  }
  function markRead(id: string) {
    const n = notifications.value.find(item => item.id === id);
    if (n) {
      n.read = true;
      saveToStorage(notifications.value);
    }
  }
  function markAllRead() {
    notifications.value.forEach(n => { n.read = true; });
    saveToStorage(notifications.value);
  }
  function remove(id: string) {
    notifications.value = notifications.value.filter(n => n.id !== id);
    saveToStorage(notifications.value);
  }
  function clear() {
    notifications.value = [];
    saveToStorage(notifications.value);
  }
  return {
    notifications,
    unreadCount,
    add,
    markRead,
    markAllRead,
    remove,
    clear,
  };
});

// ═══════════════════════════════════════
// 文件: my-uniapp-vue3/src/pages/index/index.vue
// ═══════════════════════════════════════
<template>
  <view class="page">
    <!-- 搜索栏 -->
    <view class="search-bar" @click="goToSearch">
      <text class="search-icon">  </text>
      <text class="search-placeholder">搜索专辑...</text>
      <!-- 通知铃铛 -->
      <view class="notif-bell" @click.stop="showNotifications">
        <text class="bell-icon">  </text>
        <view v-if="notifStore.unreadCount > 0" class="bell-badge">{{ notifStore.unreadCount > 99 ? '99+' : notifStore.unreadCount }}</view>
      </view>
    </view>
    <!-- 骨架屏加载状态 -->
    <scroll-view scroll-y class="album-list" @scrolltolower="loadMore" v-if="loading && bookList.length === 0" :scroll-top="scrollTop">
      <!-- 最近收听骨架屏 -->
```

```
  <view v-if="userStore.isLoggedIn" class="recent-section">
    <view class="skeleton-section-title"></view>
    <scroll-view scroll-x class="recent-scroll" enable-flex>
      <view v-for="i in 4" :key="i" class="recent-card skeleton-recent-card">
        <view class="skeleton-recent-cover"></view>
        <view class="skeleton-recent-title"></view>
        <view class="skeleton-recent-progress"></view>
      </view>
    </scroll-view>
  </view>
  <view class="skeleton-grid">
    <view v-for="i in 6" :key="i" class="skeleton-card">
      <view class="skeleton-cover"></view>
      <view class="skeleton-info">
        <view class="skeleton-title"></view>
        <view class="skeleton-meta"></view>
      </view>
    </view>
  </view>
</scroll-view>
<!-- 项目列表 -->
<scroll-view scroll-y class="album-list" @scrolltolower="loadMore" v-else :scroll-top="scrollTop">
  <!-- 最近收听 -->
  <view v-if="userStore.isLoggedIn && recentAudios.length > 0" class="recent-section">
    <text class="section-title">最近收听</text>
    <scroll-view scroll-x class="recent-scroll" enable-flex>
      <view
        v-for="item in recentAudios"
        :key="item.id"
        class="recent-card"
        @click="goToPlayer(item)"
      >
        <view class="recent-cover" :style="{ background: getRecentGradient(item.id) }">
          <image v-if="item.coverUrl" :src="item.coverUrl" mode="aspectFill" class="cover-img" />
          <text v-else class="cover-icon">  </text>
          <view class="recent-progress-bar">
            <view class="progress-fill" :style="{ width: item.progress + '%' }"></view>
          </view>
        </view>
        <text class="recent-title">{{ item.title }}</text>
        <text class="recent-progress">{{ formatProgress(item.progress) }}</text>
      </view>
    </scroll-view>
  </view>
  <!-- 冷启动引导（新用户无内容时） -->
  <view v-if="bookList.length === 0 && !loading && recentAudios.length === 0" class="onboarding">
    <view class="onboarding-card">
      <text class="onboarding-icon">  </text>
      <text class="onboarding-title">欢迎来到有声书 AI</text>
      <text class="onboarding-desc">AI 为你转换专属有声书，从文字到配音全自动完成</text>
      <button class="onboarding-btn" @click="quickCreateBook">
          创建我的第一本有声书
      </button>
    </view>
    <view class="template-section">
      <text class="template-title">   快速模板</text>
      <view class="template-list">
        <view
```

```
            v-for="tpl in quickTemplates"
            :key="tpl.title"
            class="template-card"
            @click="useQuickTemplate(tpl)"
          >
            <text class="template-icon">{{ tpl.icon }}</text>
            <text class="template-name">{{ tpl.title }}</text>

// ═══════════════════════════════════════
// 文件: my-uniapp-vue3/src/pages/create/index.vue
// ═══════════════════════════════════════
<template>
  <view class="page">
    <!-- 主内容区：只有文本输入 -->
    <view class="main-content">
      <!-- 文本输入 -->
      <view class="input-area">
        <textarea
          v-model="text"
          class="text-input"
          placeholder="请输入要转换的文本内容..."
          :maxlength="2000"
          auto-height
        />
        <!-- 字数统计 -->
        <text class="word-count" :class="{ warning: text.length > 1800, error: text.length >= 2000 }">
          {{ text.length }}/2000
        </text>
        <!-- 已上传文档信息 -->
        <view v-if="uploadedDoc.fileName && !uploading" class="doc-info-bar">
          <text class="doc-format-badge">{{ uploadedDoc.format?.toUpperCase() || 'DOC' }}</text>
          <text class="doc-filename">{{ uploadedDoc.fileName }}</text>
          <text class="doc-meta">{{ uploadedDoc.wordCount }} 字</text>
          <text v-if="uploadedDoc.truncated" class="doc-truncated-hint">  已截取</text>
          <text class="doc-remove-btn" @click="clearDoc">  </text>
        </view>
        <!-- 上传中 -->
        <view v-if="uploading" class="doc-uploading-bar">
          <text>正在解析文档，请稍候...</text>
        </view>
        <!-- 工具按钮行 -->
        <view class="tool-row">
          <view class="tool-btn" @click="handleUploadDoc">
            <text>{{ uploadedDoc.fileName ? '  重新上传' : '  上传文档' }}</text>
          </view>
          <view class="tool-btn" @click="goToAIGenerate">
            <text>  AI 生成</text>
          </view>
          <view class="tool-btn" @click="pasteText">
            <text>  粘贴</text>
          </view>
          <view class="tool-btn" @click="clearText">
            <text>  清空</text>
          </view>
        </view>
        <!-- 专辑选择（必填） -->
        <view class="album-selector" @click="showAlbumListPanel = true">
          <text class="album-selector-label">专辑</text>
```

```
        <text class="album-selector-value" :class="{ placeholder: !selectedAlbum }">
          {{ selectedAlbum ? selectedAlbum.title : '请选择专辑（必填）' }}
        </text>
        <text class="album-selector-arrow">▼</text>
      </view>
    </view>
  </view>
  <!-- 底部固定区：Tab 工具栏 -->
  <view class="bottom-bar">
    <!-- 主操作行：生成按钮 + 快捷信息 -->
    <view class="bottom-action-row">
      <view class="quick-info">
        <text class="quick-info-text">  {{ currentVoiceName || '默认音色' }}</text>
      </view>
      <view class="generate-btn-tab" :class="{ disabled: !canGenerate }" @click="handleGenerate">
        <text class="generate-btn-tab-text">{{ generating ? '提交中...' : '开始生成' }}</text>
      </view>
    </view>
    <!-- 图标 Tab 栏（等宽分布，图标+文字居中） -->
    <view class="tab-toolbar">
      <view class="tab-item" :class="{ active: showVoicePanel }" @click="showVoicePanel = true">
        <text class="tab-icon">  </text>
        <text class="tab-text">音色</text>
      </view>
      <view class="tab-item" @click="showParamsPanel = true">
        <text class="tab-icon">    </text>
        <text class="tab-text">参数</text>
      </view>
    </view>
  </view>
  <!-- 音色面板（暗色主题） -->
  <view v-if="showVoicePanel" class="bottom-panel-overlay" @click="showVoicePanel = false">
    <view class="bottom-panel voice-panel-dark" @click.stop>

// ═══════════════════════════════════════
// 文件: my-uniapp-vue3/src/pages/audio-project/create.vue
// ═══════════════════════════════════════
<template>
  <view class="page">
    <!-- 顶部导航栏 -->
    <view class="nav-bar">
      <view class="nav-content">
        <view class="nav-left" @click="goBack">
          <text class="back-icon">←</text>
        </view>
        <text class="page-title">{{ isEditMode ? '编辑项目' : '创建有声作品' }}</text>
      </view>
    </view>
    <!-- 主内容区 -->
    <view class="main-content">
      <!-- ===== 区域 1：描述你想要的音频作品 ===== -->
      <view class="card hero-card">
        <view class="section-label">
          <text class="label-text">描述你想要的有声书</text>
          <text class="label-required">*</text>
        </view>
        <text class="section-hint">用自然语言描述主题、风格和读者，越具体效果越好</text>
        <!-- 示例提示词 -->
```

```
  <view class="examples-row">
    <text class="examples-label">试试：</text>
    <view class="example-chips">
      <text
        v-for="ex in examplePrompts"
        :key="ex"
        class="example-chip"
        @click="newBook.description = ex"
      >{{ ex }}</text>
    </view>
  </view>
  <!-- 文本域 -->
  <view class="textarea-wrapper">
    <textarea
      v-model="newBook.description"
      class="form-textarea"
      :class="{ 'textarea-error': attemptedCreate && !newBook.description.trim() }"
      placeholder="用一句话描述你想要的有声作品，AI 会自动分析并推荐最佳配置..."
      :maxlength="2000"
      @blur="detectScaleFromDesc"
    />
    <view class="textarea-footer">
      <text class="char-count">{{ newBook.description.length }}/2000</text>
      <text v-if="newBook.description.trim()" class="clear-btn" @click="newBook.description = ''">清空</text>
    </view>
  </view>
  <!-- AI 智能推荐按钮 -->
  <view v-if="!isEditMode" class="ai-btn-row">
    <button
      class="btn-ai-recommend"
      :class="{ 'btn-ai-loading': isRecommending }"
      :disabled="isRecommending || !newBook.description.trim()"
      @click="requestSmartRecommend"
    >
    <text>{{ isRecommending ? 'AI 分析中...' : 'AI 智能推荐配置' }}</text>
    </button>
    <text class="ai-btn-hint">AI 会分析描述并自动填充下方所有配置</text>
  </view>
</view>
<!-- ===== 区域 2：AI 推荐结果（有配置时显示） ===== -->
<view v-if="!isEditMode && hasAnyConfig" class="card config-summary-card">
  <view class="section-label">
    <text class="label-text">当前配置</text>
    <text v-if="recommendSource === 'ai'" class="config-source-tag">AI 推荐</text>
    <text v-else-if="recommendSource === 'template'" class="config-source-tag template-tag">模板</text>
  </view>
  <view class="config-tags">
    <view v-if="newBook.bookScale" class="config-tag scale-tag" @click="scrollToSection('scale')">
      <text class="config-tag-label">规模</text>
      <text class="config-tag-value">{{ getScaleShortLabel(newBook.bookScale) }}</text>
      <text class="config-tag-edit">修改</text>
    </view>
    <view v-if="newBook.targetAudience" class="config-tag" @click="scrollToSection('audience')">
      <text class="config-tag-value">{{ getAudienceLabel(newBook.targetAudience) }}</text>
      <text class="config-tag-edit">修改</text>
    </view>
    <view v-if="newBook.knowledgeLevel" class="config-tag" @click="scrollToSection('knowledge')">
      <text class="config-tag-value">{{ getKnowledgeLevelLabel(newBook.knowledgeLevel) }}</text>
```

```
          <text class="config-tag-edit">修改</text>
// ========================================
// 文件: my-uniapp-vue3/src/pages/audio-project/detail.vue
// ========================================
<template>
  <view class="page">
    <!-- 顶部导航栏 -->
    <view class="nav-bar">
      <view class="nav-content">
        <view class="nav-left" @click="goBack">
          <text class="back-icon">←</text>
        </view>
        <text class="page-title">{{ isTitleGenerating ? 'AI 分析中...' : (currentBook?.title || '项目详情') }}</text>
        <view class="nav-right">
          <text v-if="canEdit" class="nav-edit-btn" @click="goToEdit">编辑</text>
        </view>
      </view>
    </view>
    <!-- 主内容区 -->
    <view class="main-content">
      <!-- 项目信息 -->
      <view class="card book-info">
        <view class="book-title-row">
          <text class="book-title-large">{{ isTitleGenerating ? '  AI 正在生成标题...' : currentBook?.title }}</text>
          <GenerationStatusBadge :status="currentBook?.genStage || 'draft'"></GenerationStatusBadge>
        </view>
        <text v-if="currentBook?.subtitle" class="book-subtitle">{{ currentBook.subtitle }}</text>
        <text class="book-desc">{{ currentBook?.description }}</text>
        <view class="book-meta-row">
          <text>章节：{{ currentBook?.totalChapters }} 章</text>
          <text>预估：{{ currentBook?.estimatedWords || 0 }} 字</text>
        </view>
      </view>
      <!-- 进度显示 -->
      <view v-if="currentBook" class="card progress-card">
        <text class="card-title">  生成进度</text>
        <view class="progress-display">
          <text class="progress-text" :class="{ 'progress-failed': isGenerationFailed }">{{ isGenerationFailed ? '失败' : currentBook.progress }}
%</text>
          <text class="progress-detail">
            {{ completedChapters }}/{{ currentBook.totalChapters }} 章
          </text>
        </view>
        <view class="progress-bar-large">
          <view class="progress-fill" :class="{ 'progress-fill-failed': isGenerationFailed }" :style="{ width: (isGenerationFailed ? 0 : currentB
ook.progress) + '%' }"></view>
        </view>
        <!-- 失败状态 -->
        <view v-if="isGenerationFailed" class="failed-tip">
          <text class="failed-icon">  </text>
          <text class="failed-text">生成失败</text>
          <text v-if="currentBook.error" class="failed-error">{{ currentBook.error }}</text>
          <text class="failed-hint">请检查网络或稍后重试</text>
          <button class="retry-btn" @click="handleRetryGenerate">  重新生成</button>
        </view>
        <!-- 实时生成状态面板 -->
        <view v-if="isCurrentlyGenerating && !isGenerationFailed" class="generation-status">
```

```
          <view class="status-header">
            <text class="status-icon">  </text>
            <text class="status-title">正在生成中...【时间较长，请耐心等待，可关闭页面，后台会继续生成】</text>
          </view>
          <view class="status-stage">
            <text class="stage-label">当前阶段：</text>
            <text class="stage-value">{{ getCurrentStage() }}</text>
          </view>
          <view v-if="currentGeneratingChapter" class="current-chapter-info">
            <text class="chapter-label">正在处理：</text>
            <text class="chapter-name">第{{ currentGeneratingChapter.number }}{{ currentGeneratingChapter.levelLabel }} {{ currentGeneratingChapt
er.title }}</text>
          </view>
          <view v-if="currentBook.error" class="status-error">
            <text class="error-icon">    </text>
            <text class="error-text">{{ currentBook.error }}</text>
          </view>
          <view v-if="currentBook.error && currentBook.error.includes('自动恢复')" class="status-recovery">
            <text class="recovery-icon">  </text>
            <text class="recovery-text">系统正在尝试自动恢复...</text>
          </view>
          <view class="status-tip">
            <text class="tip-text">   生成过程可能需要几分钟，请耐心等待</text>
          </view>
        </view>
        <!-- 实时配额显示 -->
        <view v-if="isCurrentlyGenerating" class="quota-monitor">
          <text class="quota-monitor-title">   额度监控</text>
          <view class="quota-monitor-row">
            <text>已用：{{ usedAudioMinutes }}分钟</text>

// ============================================
// 文件: my-uniapp-vue3/src/pages/audio-project/interactive.vue
// ============================================
<template>
  <view class="page">
    <!-- 顶部导航栏 -->
    <view class="nav-bar">
      <view class="nav-content">
        <view class="nav-left" @click="goBack">
          <text class="back-icon">←</text>
        </view>
        <text class="page-title">交互式创建</text>
      </view>
      <!-- 模式切换独立行 -->
      <view class="mode-bar">
        <text class="mode-switch" @click="switchToSimple">切换到一键模式</text>
      </view>
    </view>
    <!-- 步骤指示器 -->
    <view class="steps-bar">
      <view v-for="(step, i) in steps" :key="i" :class="['step-dot', currentStep >= i ? 'active' : '', currentStep > i ? 'done' : '']">
        <view class="dot-num">{{ currentStep > i ? '  ' : step.num }}</view>
        <text class="dot-label">{{ step.label }}</text>
      </view>
      <view class="step-line" :style="{ width: (currentStep / (steps.length - 1) * 100) + '%' }"></view>
    </view>
    <!-- 主内容区 -->
```

```
<view class="main-content">
  <!-- ============ Step 1: 信息输入 ============ -->
  <view v-if="currentStep === 0" class="card">
    <view class="card-header">
      <text class="card-title">  第一步：基本信息</text>
    </view>
    <view class="form-item">
      <text class="form-label">书名 *</text>
      <input v-model="form.title" class="form-input" placeholder="例如：《时间是什么》" />
    </view>
    <view class="form-item">
      <text class="form-label">内容描述 *</text>
      <textarea v-model="form.description" class="form-textarea" placeholder="描述这本书的内容、主题、写作目的..." :maxlength="500" />
      <view class="ai-recommend-row">
        <button class="btn-ai-recommend" :disabled="!form.description.trim() || recommendLoading" @click="doSmartRecommend">
          {{ recommendLoading ? 'AI分析中...' : '  AI智能推荐' }}
        </button>
        <text class="ai-recommend-hint">根据描述自动填充目标人群和项目类型</text>
      </view>
    </view>
    <view class="form-item">
      <text class="form-label">项目规模 *</text>
      <view class="scale-picker">
        <view
          v-for="scale in bookScales"
          :key="scale.value"
          :class="['scale-chip', form.bookScale === scale.value ? 'active' : '']"
          @click="form.bookScale = scale.value"
        >
          <text class="scale-label">{{ scale.label }}</text>
          <text class="scale-desc">{{ scale.chapters }} • {{ scale.audio }}</text>
        </view>
      </view>
    </view>
    <!-- 目标人群 -->
    <view class="form-item">
      <view class="label-with-badge">
        <text class="form-label">目标人群</text>
        <view class="required-badge">核心</view>
      </view>
      <text class="form-hint-small">指定读者，AI会据此调整语言风格和深度</text>
      <view class="chip-group" style="margin-top: 8rpx;">
        <view
          v-for="a in audiences"
          :key="a.value"
          :class="['chip', form.targetAudience === a.value ? 'active' : '']"
          @click="form.targetAudience = a.value"
        >
          <text class="chip-icon">{{ a.icon }}</text>
          <text class="chip-text">{{ a.label }}</text>
          <text class="chip-desc" style="font-size: 20rpx; margin-left: 4rpx;">{{ a.desc }}</text>
        </view>
      </view>
    </view>
    <view class="form-item">
      <text class="form-label">项目类型</text>

// ========================================
```

```
// 文件: my-uniapp-vue3/src/pages/audio-project/chapter-detail.vue
// ============================================
<template>
  <view class="page">
    <!-- 顶部导航栏 -->
    <view class="nav-bar">
      <view class="nav-content">
        <view class="nav-left" @click="goBack">
          <text class="back-icon">←</text>
        </view>
        <text class="page-title">{{ chapterTitle || '章节详情' }}</text>
        <view class="nav-right"></view>
      </view>
    </view>
    <!-- 章节内容 -->
    <scroll-view
      class="chapter-content"
      scroll-y="true"
      :scroll-top="scrollTop"
      :style="{ transform: `translateX(${swipeOffset.value}px)`, transition: swipeTransition.value }"
      @touchstart="onTouchStart"
      @touchmove="onTouchMove"
      @touchend="onTouchEnd"
    >
      <!-- 章节头部信息 -->
      <view class="chapter-header">
    <text class="chapter-num-badge">
      第{{ chapterParentNumber || chapterNumber }}{{ chapterLevel === 2 ? '章 第' + chapterNumber + '节' : '章' }}
    </text>
    <text class="chapter-title-large">{{ chapterTitle }}</text>
        <view class="chapter-meta">
          <text class="word-count">{{ chapterWordCount }}字</text>
          <GenerationStatusBadge v-if="chapterGenStage" :status="chapterGenStage" />
          <view v-if="chapterAudioUrl" class="media-badge audio-badge">
            <text>  音频</text>
          </view>
          <view v-if="chapterVideoUrl" class="media-badge video-badge">
            <text>  视频</text>
          </view>
          <view v-if="isLeafNode" class="edit-toggle-btn" @click="toggleEditMode">
            <text>{{ isEditMode ? '  完成' : '  编辑' }}</text>
          </view>
          <view v-if="isLeafNode && chapterGenStage !== 'idle' && chapterGenStage !== 'content_generating'" class="regenerate-btn" @click="handle
RegenerateContent">
            <text>{{ regeneratingContent ? '生成中...' : '  重新生成' }}</text>
          </view>
        </view>
      </view>
      <!-- 章节内容（level=3小节 或 level=1短文章节） -->
      <!-- 加载中 -->
      <view v-if="isLoading && !chapterContent" class="chapter-body">
        <view class="level-hint-large">
          <text class="hint-icon-large">  </text>
          <text class="hint-text-large">正在加载章节内容...</text>
        </view>
      </view>
      <!-- 加载失败 -->
      <view v-else-if="loadError" class="chapter-body">
```

```
      <view class="level-hint-large">
        <text class="hint-icon-large">  </text>
        <text class="hint-text-large">加载章节失败</text>
        <text class="hint-subtext">{{ loadError }}</text>
        <button class="retry-btn" @click="retryLoad">重试</button>
      </view>
    </view>
    <view v-else-if="isLeafNode || (chapterLevel === 1 && chapterContent)" class="chapter-body">
      <!-- 短文类章标识 -->
      <view v-if="chapterLevel === 1 && isLeafNode" class="article-badge">
        <text>   短文正文</text>
      </view>
      <!-- 编辑模式 -->
      <view v-if="isEditMode" class="edit-mode">
        <textarea
          v-model="editContent"
          class="content-editor"
          placeholder="编辑章节内容..."
          :maxlength="-1"
        />
        <view class="edit-actions">
          <button class="edit-btn cancel" @click="cancelEdit">取消</button>
          <button class="edit-btn save" @click="saveContent" :disabled="savingContent">
            {{ savingContent ? '保存中...' : '保存' }}
          </button>


// ═══════════════════════════════════════
// 文件: my-uniapp-vue3/src/pages/player/index.vue
// ═══════════════════════════════════════
<template>
  <view class="page">
    <!-- 顶部导航 -->
    <view class="nav-bar">
      <view class="nav-btn" @click="goBack">
        <text class="nav-icon">  </text>
      </view>
      <view class="nav-center">
        <text class="nav-title">{{ audio?.title || '正在播放' }}</text>
        <view v-if="sleepTimer" class="sleep-timer-badge" @click="showSleepTimerPicker = true">
          <text class="sleep-timer-text">{{ formatSleepTimer(sleepTimer.remaining) }}</text>
        </view>
      </view>
      <view class="nav-actions">
        <view class="nav-btn" @click="showSleepTimerPicker = true">
          <text class="nav-icon">  </text>
        </view>
        <view class="nav-btn" @click="showLyrics = !showLyrics">
          <text class="nav-icon">{{ showLyrics ? '  ' : '  ' }}</text>
        </view>
      </view>
    </view>
    <!-- 无音频提示 -->
    <view v-if="!audio?.audioUrl" class="empty-state">
      <text class="empty-icon">  </text>
      <text class="empty-title">暂无音频</text>
      <text class="empty-desc">该章节还没有生成音频</text>
      <button class="empty-btn" @click="goBack">返回</button>
    </view>
```

```
    <!-- 正常播放界面 -->
    <template v-else>
    <!-- 封面和标题 -->
    <view class="cover-section" v-show="showLyrics">
      <view class="cover">
        <text class="cover-icon">  </text>
      </view>
      <text class="title">{{ audio?.title || '未知标题' }}</text>
    </view>
    <!-- 文本预览（歌词同步） -->
    <scroll-view
      v-show="showLyrics"
      class="text-preview"
      scroll-y
      :style="{ maxHeight: '550rpx' }"
      :scroll-into-view="scrollIntoViewId"
      scroll-with-animation
    >
      <view class="lyrics-container">
        <view
          v-for="(line, index) in lyrics"
          :key="index"
          :id="'lyric-' + index"
          class="lyric-line"
          :class="{ active: index === currentLyricIndex }"
        >
          <text class="lyric-text">{{ line.text }}</text>
        </view>
      </view>
    </scroll-view>
    <!-- 进度条 -->
    <view class="progress-section">
      <text class="time">{{ formatDuration(currentTime) }}</text>
      <slider
        :value="currentTime"
        :max="duration || 100"
        @change="handleSeek"
        activeColor="#4F46E5"
        backgroundColor="#e5e7eb"
        block-size="16"
        class="progress-slider"
      />
      <text class="time">{{ formatDuration(duration) }}</text>
    </view>
    <!-- 播放控制 -->
    <view class="controls">
      <!-- 左侧按钮：播放速度、下载 -->
      <view class="controls-side">
        <view class="control-btn" @click="showRatePicker = true">
          <text class="control-text">{{ playRate }}x</text>
        </view>

// ============================================
// 文件: my-uniapp-vue3/src/pages/member/index.vue
// ============================================
<template>
  <view class="page">
    <view class="nav-bar">
```

```
  <view class="nav-btn" @click="goBack">
    <text class="nav-icon">←</text>
  </view>
  <text class="nav-title">订阅套餐</text>
  <view class="nav-btn" />
</view>
<view class="balance-section" v-if="isLoggedIn">
  <view class="balance-info">
    <text class="balance-label">剩余积分</text>
    <text class="balance-value">{{ formatNumber(tokenBalance.remainingTokens) }}</text>
    <text class="balance-unit"> / {{ tokenBalance.isUnlimited ? '无限' : formatNumber(tokenBalance.totalTokens) }}</text>
  </view>
  <view class="balance-bar">
    <view class="balance-progress" :style="{ width: progressWidth + '%' }"></view>
  </view>
</view>
<view class="plans-section">
  <text class="section-title">选择您的套餐</text>
  <view v-if="loading" class="loading-container">
    <text class="loading-text">加载中...</text>
  </view>
  <view v-else class="plans-list">
    <view
      v-for="plan in plans"
      :key="plan.id"
      class="plan-card"
      :class="{ active: selectedPlanId === plan.id, free: plan.level === 0 }"
      @click="selectedPlanId = plan.id"
    >
      <view v-if="plan.level === 0" class="free-badge">免费</view>
      <view class="plan-header">
        <text class="plan-name">{{ plan.name }}</text>
        <view class="plan-pricing">
          <text class="price-symbol">¥</text>
          <text class="price-value">{{ plan.priceMonthly }}</text>
          <text class="price-unit">/月</text>
        </view>
      </view>
      <text class="plan-description">{{ plan.description }}</text>
      <view class="plan-features">
        <view v-for="(feature, idx) in plan.features" :key="idx" class="feature-item">
          <text class="feature-icon">  </text>
          <text class="feature-text">{{ feature }}</text>
        </view>
      </view>
      <view class="select-btn" :class="{ selected: selectedPlanId === plan.id }">
        <text>{{ selectedPlanId === plan.id ? '   已选择' : '选择此套餐' }}</text>
      </view>
    </view>
  </view>
</view>
<view class="bottom-section">
  <view class="total-info">
    <text class="total-label">应付金额</text>
    <text class="total-value">¥{{ selectedPlan?.priceMonthly || 0 }}</text>
  </view>
  <button class="subscribe-btn" @click="handleSubscribe" :disabled="!selectedPlan || selectedPlan.level === 0">
    <text>{{ selectedPlan?.level === 0 ? '免费套餐' : '立即订阅' }}</text>
```

```
      </button>
    </view>
  </view>
</template>
<script setup lang="ts">
import { ref, computed, onMounted } from 'vue';
import { useUserStore } from '../../store/user';
import { get, post } from '../../utils/request';
import { formatNumber } from '../../utils/format';
const userStore = useUserStore();
const plans = ref<any[]>([]);
const selectedPlanId = ref<number>(17);
const loading = ref(false);
const tokenBalance = ref({ totalTokens: 0, usedTokens: 0, remainingTokens: 0, isUnlimited: false });
const selectedPlan = computed(() => plans.value.find(p => p.id === selectedPlanId.value));
const isLoggedIn = computed(() => userStore.isLoggedIn);
const progressWidth = computed(() => {
  if (tokenBalance.value.isUnlimited) return 100;
  if (tokenBalance.value.totalTokens === 0) return 0;

// ═══════════════════════════════════════
// 文件: my-uniapp-vue3/src/pages/mine/index.vue
// ═══════════════════════════════════════
<template>
  <view class="page">
    <!-- 用户信息卡片 -->
    <view class="user-card">
      <view class="user-info">
        <image v-if="userStore.userInfo?.avatar" :src="userStore.userInfo.avatar" class="avatar" mode="aspectFill" />
        <view v-else class="avatar-placeholder">
          <text class="avatar-text">  </text>
        </view>
        <view class="user-text">
          <text class="nickname">{{ userStore.userInfo?.nickname || '未登录' }}</text>
          <view class="member-badge" :class="memberClass">
            <text class="member-text">{{ memberText }}</text>
          </view>
        </view>
      </view>
      <!-- 积分余额（两行展示） -->
      <view class="credits-section" v-if="tokenBalance">
        <view class="credit-row">
          <view class="credit-item">
            <text class="credit-label">   订阅积分</text>
            <text class="credit-value">{{ formatNumber(tokenBalance.subscriptionRemaining ?? tokenBalance.remainingTokens) }}</text>
            <text class="credit-sub" v-if="tokenBalance.subscriptionTokens">/ {{ formatNumber(tokenBalance.subscriptionTokens) }} 总量</text>
          </view>
          <view class="credit-divider" />
          <view class="credit-item">
            <text class="credit-label">   积分包</text>
            <text class="credit-value">{{ formatNumber(tokenBalance.packRemaining ?? 0) }}</text>
            <text class="credit-sub" v-if="tokenBalance.packTokens > 0">/ {{ formatNumber(tokenBalance.packTokens) }} 总量</text>
            <text class="credit-sub" v-else>暂无</text>
          </view>
        </view>
        <!-- 进度条 -->
        <view class="credit-bar-row">
          <view class="credit-bar" :style="{ flex: tokenBalance.subscriptionTokens || 1 }">
```

```
          <view class="credit-bar-fill sub" :style="{ width: subscriptionPercent + '%' }"></view>
        </view>
        <view class="credit-bar-gap" />
        <view class="credit-bar" :style="{ flex: tokenBalance.packTokens || 1 }">
          <view class="credit-bar-fill pack" :style="{ width: packPercent + '%' }"></view>
        </view>
      </view>
    </view>
    <!-- 使用统计 -->
    <view class="stats-row" v-if="userStore.memberStatus">
      <view class="stat-item">
        <text class="stat-value">{{ userStore.memberStatus.quota.dailyRemaining }}</text>
        <text class="stat-label">今日次数</text>
      </view>
      <view class="stat-divider" />
      <view class="stat-item">
        <text class="stat-value">{{ userStore.memberStatus.quota.wordLimit === 0 ? '∞' : userStore.memberStatus.quota.wordLimit }}</text>
        <text class="stat-label">字数上限</text>
      </view>
    </view>
  </view>
  <!-- 今日概况 -->
  <view class="today-card" v-if="userStore.isLoggedIn">
    <view class="today-header">
      <text class="today-title">今日概况</text>
      <text class="today-date">{{ todayDate }}</text>
    </view>
    <view class="today-stats">
      <view class="today-stat-item">
        <text class="today-stat-value">{{ todayStats.audioCount }}</text>
        <text class="today-stat-label">生成音频</text>
      </view>
      <view class="today-stat-divider" />
      <view class="today-stat-item">
        <text class="today-stat-value">{{ todayStats.bookCount }}</text>
        <text class="today-stat-label">转换音频</text>
      </view>
      <view class="today-stat-divider" />
      <view class="today-stat-item">
        <text class="today-stat-value">{{ todayStats.usageQuota }}</text>
        <text class="today-stat-label">使用额度</text>
      </view>
    </view>
  </view>
  <!-- 最近的项目 -->

// ═══════════════════════════════════════
// 文件: my-uniapp-vue3/src/pages/search/index.vue
// ═══════════════════════════════════════
<template>
  <view class="page">
    <!-- 搜索栏 -->
    <view class="search-bar">
      <input
        v-model="searchQuery"
        class="search-input"
        placeholder="搜索音频..."
        @confirm="handleSearch"
```

```
      />
      <button class="search-btn" @click="handleSearch">搜索</button>
    </view>
    <!-- 骨架屏加载状态 -->
    <scroll-view scroll-y class="results" v-if="loading" :scroll-top="scrollTop">
      <SkeletonList layout="search-result" :count="5" />
    </scroll-view>
    <!-- 搜索结果 -->
    <scroll-view scroll-y class="results" v-else-if="searchResults.length > 0" :scroll-top="scrollTop">
      <view class="result-list">
        <view
          v-for="item in searchResults"
          :key="item.id"
          class="result-item"
          @click="playAudio(item)"
        >
          <view class="result-cover" :style="{ background: getCoverGradient(item.voiceId) }">
            <text class="cover-letter">{{ getTitleLetter(item.title) }}</text>
            <text class="cover-title-small">{{ item.title }}</text>
          </view>
          <view class="result-info">
            <text class="result-title" v-html="highlightText(item.title, searchQuery)"></text>
            <text class="result-desc" v-if="(item as any).description" v-html="highlightText((item as any).description.slice(0, 50) + '...', sear
chQuery)"></text>
            <text class="result-meta">{{ item.wordCount || 0 }}字</text>
          </view>
        </view>
      </view>
    </scroll-view>
    <!-- 无结果 -->
    <view v-else-if="hasSearched && searchResults.length === 0" class="no-result">
      <text class="no-result-icon">  </text>
      <text class="no-result-text">未找到相关音频</text>
    </view>
    <!-- 初始状态：搜索历史 + 热门推荐 -->
    <view v-else class="initial-state">
      <!-- 搜索历史 -->
      <view class="section" v-if="searchHistory.length > 0">
        <view class="section-header">
          <text class="section-title">搜索历史</text>
          <text class="clear-btn" @click="clearHistory">清空</text>
        </view>
        <view class="tag-list">
          <view
            v-for="(item, index) in searchHistory"
            :key="index"
            class="tag history-tag"
            @click="clickHistory(item.keyword)"
          >
            <text class="tag-text">{{ item.keyword }}</text>
            <text class="delete-icon" @click.stop="deleteHistoryItem(item.keyword)">×</text>
          </view>
        </view>
      </view>
      <!-- 热门推荐 -->
      <view class="section" v-if="hotSearches.length > 0">
        <view class="section-header">
          <text class="section-title">  热门推荐</text>
```

```
      </view>
      <view class="tag-list">
        <view
          v-for="(item, index) in hotSearches"
          :key="item.id"
          class="tag hot-tag"
          @click="clickHotSearch(item.keyword)"
        >
          <text class="tag-rank">{{ index + 1 }}</text>
          <text class="tag-text">{{ item.keyword }}</text>
        </view>
      </view>
    </view>
    <!-- 空状态提示 -->

// ============================================
// 文件: my-uniapp-vue3/src/pages/history/index.vue
// ============================================
<template>
  <view class="page">
    <!-- 顶部搜索栏 -->
    <view class="search-bar">
      <view class="search-inner">
        <text class="search-icon">  </text>
        <input
          class="search-input"
          v-model="searchKeyword"
          placeholder="搜索历史记录..."
          @confirm="handleSearch"
          @clear="handleClearSearch"
        />
        <text v-if="searchKeyword" class="clear-btn" @click="handleClearSearch">  </text>
      </view>
    </view>
    <!-- 编辑栏 -->
    <view class="edit-bar">
      <view v-if="!isEditing" class="edit-btn" @click="isEditing = true">
        <text>编辑</text>
      </view>
      <view v-else class="edit-actions">
        <view class="action-btn" @click="toggleSelectAll">
          <text>{{ isAllSelected ? '取消全选' : '全选' }}</text>
        </view>
        <view class="action-btn download" @click="handleBatchDownload" :disabled="selectedIds.length === 0">
          <text>下载 ({{ selectedIds.length }})</text>
        </view>
        <view class="action-btn delete" @click="handleBatchDelete" :disabled="selectedIds.length === 0">
          <text>删除 ({{ selectedIds.length }})</text>
        </view>
        <view class="action-btn cancel" @click="cancelEdit">
          <text>取消</text>
        </view>
      </view>
    </view>
    <!-- 标签栏 -->
    <view class="tab-bar">
      <scroll-view class="tab-scroll" scroll-x="true" :show-scrollbar="false">
        <view class="tab-list">
```

```
        <view
          v-for="tab in tabs"
          :key="tab.id"
          class="tab-item"
          :class="{ active: selectedTab === tab.id }"
          @click="selectTab(tab.id)"
        >
          <text>{{ tab.name }}</text>
        </view>
      </view>
    </scroll-view>
    <view class="group-toggle" @click="groupByBook = !groupByBook">
      <text class="group-toggle-text">{{ groupByBook ? '  列表' : '  聚合' }}</text>
    </view>
  </view>
  <!-- 音频列表 -->
  <scroll-view scroll-y class="audio-list" @scrolltolower="loadMore" :scroll-top="scrollTop">
    <!-- 骨架屏加载状态 -->
    <SkeletonList v-if="loading && audioList.length === 0" layout="history" :count="6" />
    <!-- 空状态 -->
    <view v-else-if="audioList.length === 0 && !loading" class="empty">
      <text class="empty-icon">  </text>
      <text class="empty-text">暂无历史记录</text>
      <text class="empty-hint">快去生成第一个音频吧</text>
    </view>
    <!-- 音频列表 -->
    <view v-else class="audio-grid">
      <!-- 分组模式 -->
      <view v-if="groupByBook" v-for="group in displayList" :key="group._id" class="group-card">
        <view class="group-header" @click="goToBookDetail(group.bookId)">
          <text class="group-title">{{ group.bookId ? '  项目 #' + group.bookId : '  其他音频' }}</text>
          <text class="group-count">{{ group.items.length }} 个音频 • {{ formatDuration(group.totalDuration) }}</text>
        </view>
        <view
          v-for="item in group.items"
          :key="item._id"
          class="audio-card group-item"
          :class="{ selected: selectedIds.includes(item._id) }"
          @click="isEditing ? toggleSelect(item._id) : playAudio(item)"
        >

// ============================================
// 文件: my-uniapp-vue3/src/pages/favorites/index.vue
// ============================================
<template>
  <view class="page">
    <!-- 顶部导航 -->
    <view class="nav-bar">
      <view class="nav-btn" @click="goBack">
        <text class="nav-icon">←</text>
      </view>
      <text class="nav-title">我的收藏</text>
      <view class="nav-btn" />
    </view>
    <!-- 加载状态 -->
    <view v-if="loading" class="loading">
      <text>加载中...</text>
    </view>
```

```
    <!-- 空状态 -->
    <view v-else-if="favorites.length === 0" class="empty">
      <text class="empty-icon">    </text>
      <text class="empty-text">暂无收藏</text>
      <text class="empty-hint">去听书页面收藏喜欢的音频吧</text>
    </view>
    <!-- 收藏列表 -->
    <view v-else class="list">
      <view
        v-for="item in favorites"
        :key="item.id"
        class="item"
        @click="playAudio(item.audio)"
      >
        <view class="item-cover">
          <text class="cover-icon">  </text>
        </view>
        <view class="item-info">
          <text class="item-title">{{ item.audio?.title || '未知标题' }}</text>
          <text class="item-meta">{{ item.audio?.wordCount || 0 }}字</text>
        </view>
        <view class="item-action" @click.stop="removeFavorite(item.audioId)">
          <text class="action-icon">    </text>
        </view>
      </view>
    </view>
  </view>
</template>
<script setup lang="ts">
import { ref, onMounted } from 'vue';
import { get, del } from '../../utils/request';
interface FavoriteItem {
  id: number;
  audioId: number;
  createdAt: string;
  audio?: {
    id: number;
    title: string;
    text: string;
    audioUrl: string;
    audioDuration: number;
    wordCount: number;
  };
}
const favorites = ref<FavoriteItem[]>([]);
const loading = ref(true);
onMounted(async () => {
  await fetchFavorites();
});
async function fetchFavorites() {
  loading.value = true;
  try {
    const result = await get<any>('/favorites');
    favorites.value = result || [];
  } catch (error) {
    console.error('获取收藏列表失败:', error);
  } finally {
    loading.value = false;
```

```
  }
}
async function removeFavorite(audioId: number) {
  try {
    await del(`/favorites/${audioId}`);
    uni.showToast({ title: '已取消收藏', icon: 'success' });
    // 从列表中移除
    favorites.value = favorites.value.filter(item => item.audioId !== audioId);

// ============================================
// 文件: my-uniapp-vue3/src/pages/login/index.vue
// ============================================
<template>
  <view class="page">
    <!-- 顶部背景 -->
    <view class="header">
      <text class="title">声工坊</text>
      <text class="subtitle">让文字变成声音</text>
    </view>
    <!-- 登录表单 -->
    <view class="form-card">
      <text class="form-title">手机号登录</text>
      <!-- 手机号输入 -->
      <view class="input-group">
        <text class="input-label">手机号</text>
        <input
          v-model="phone"
          type="number"
          class="input"
          placeholder="请输入手机号"
          maxlength="11"
        />
      </view>
      <!-- 验证码输入 -->
      <view class="input-group">
        <text class="input-label">验证码</text>
        <view class="code-row">
          <input
            v-model="code"
            type="number"
            class="input code-input"
            placeholder="请输入验证码"
            maxlength="6"
          />
          <button
            class="code-btn"
            :disabled="countdown > 0 || !isPhoneValid"
            @click="handleSendCode"
          >
            <text class="code-btn-text">{{ countdown > 0 ? `${countdown}s` : '获取验证码' }}</text>
          </button>
        </view>
      </view>
      <!-- 登录按钮 -->
      <button class="login-btn" :disabled="!canLogin" @click="handleLogin">
        <text class="login-btn-text">登录</text>
      </button>
      <!-- 提示 -->
```

```
      <text class="tip">登录即表示同意《用户协议》和《隐私政策》</text>
    </view>
  </view>
</template>
<script setup lang="ts">
import { ref, computed, onUnmounted } from 'vue';
import { useUserStore } from '../../store/user';
const userStore = useUserStore();
// 状态
const phone = ref('');
const code = ref('');
const countdown = ref(0);
const loading = ref(false);
let timer: number | null = null;
// 计算属性
const isPhoneValid = computed(() => /^1[3-9]\d{9}$/.test(phone.value));
// 开发环境免密登录：验证码可以为空
const canLogin = computed(() => isPhoneValid.value && !loading.value);
// 发送验证码
async function handleSendCode() {
  if (countdown.value > 0 || !isPhoneValid.value) return;
  try {
    loading.value = true;
    await userStore.sendCode(phone.value);
    uni.showToast({ title: '验证码已发送', icon: 'success' });
    // 开始倒计时
    countdown.value = 60;
    timer = setInterval(() => {
      countdown.value--;
      if (countdown.value <= 0 && timer) {
        clearInterval(timer);
        timer = null;
      }
    }, 1000) as unknown as number;

// ═══════════════════════════════════════
// 文件: my-uniapp-vue3/src/pages/settings/index.vue
// ═══════════════════════════════════════
<template>
  <view class="page">
    <!-- 顶部导航 -->
    <view class="nav-bar">
      <view class="nav-btn" @click="goBack">
        <text class="nav-icon">←</text>
      </view>
      <text class="nav-title">设置</text>
      <view class="nav-btn" />
    </view>
    <!-- 设置列表 -->
    <scroll-view scroll-y class="settings-list" :scroll-top="scrollTop">
      <!-- 播放设置 -->
      <view class="settings-section">
        <text class="section-title">播放</text>
        <!-- 默认音色 -->
        <view class="settings-item" @click="showVoicePicker = true">
          <text class="item-icon">  </text>
          <text class="item-text">默认音色</text>
          <view class="item-value">
```

```
      <text class="value-text">{{ getVoiceLabel(preferences.defaultVoiceId) }}</text>
      <text class="item-arrow">›</text>
    </view>
  </view>
  <!-- 默认音量 -->
  <view class="settings-item">
    <text class="item-icon">  </text>
    <text class="item-text">默认音量</text>
    <view class="item-value">
      <text class="value-text">{{ preferences.defaultVolume }}%</text>
      <slider
        class="volume-slider"
        :value="preferences.defaultVolume"
        @change="onVolumeChange"
        min="0"
        max="100"
        step="5"
        activeColor="#4F46E5"
        backgroundColor="#E5E7EB"
        block-size="20"
      />
    </view>
  </view>
  <!-- 自动播放下一首 -->
  <view class="settings-item">
    <text class="item-icon">    </text>
    <text class="item-text">自动播放下一首</text>
    <switch
      :checked="preferences.autoPlayNext"
      @change="onAutoPlayChange"
      activeColor="#4F46E5"
    />
  </view>
</view>
<!-- 主题设置 -->
<view class="settings-section">
  <text class="section-title">外观</text>
  <view class="settings-item" @click="showThemePicker = true">
    <text class="item-icon">  </text>
    <text class="item-text">主题</text>
    <view class="item-value">
      <text class="value-text">{{ themeLabels[theme] }}</text>
      <text class="item-arrow">›</text>
    </view>
  </view>
</view>
<!-- 下载设置 -->
<view class="settings-section">
  <text class="section-title">下载</text>
  <view class="settings-item">
    <text class="item-icon">  </text>
    <text class="item-text">仅WiFi下载</text>
    <switch
      :checked="preferences.wifiOnlyDownload"
      @change="onWifiOnlyChange"
      activeColor="#4F46E5"
    />
  </view>
```

```
    </view>
    <!-- 存储设置 -->

// ═══════════════════════════════════════
// 文件: my-uniapp-vue3/src/pages/billing/index.vue
// ═══════════════════════════════════════
<template>
  <view class="page">
    <view class="header">
      <text class="title">消费账单</text>
    </view>
    <!-- 汇总卡片 -->
    <view class="summary-card">
      <view class="summary-item">
        <text class="summary-value">¥{{ summary.totalCost }}</text>
        <text class="summary-label">总费用</text>
      </view>
      <view class="summary-divider" />
      <view class="summary-item">
        <text class="summary-value">{{ summary.totalCalls }}</text>
        <text class="summary-label">总调用次数</text>
      </view>
      <view class="summary-divider" />
      <view class="summary-item">
        <text class="summary-value">{{ formatNumber(summary.totalTokens) }}</text>
        <text class="summary-label">总积分消耗</text>
      </view>
    </view>
    <!-- 分类统计 -->
    <view class="type-stats" v-if="summary.byType.length">
      <view class="type-item" v-for="t in summary.byType" :key="t.type">
        <text class="type-name">{{ typeLabel(t.type) }}</text>
        <view class="type-data">
          <text class="type-count">{{ t.count }}次</text>
          <text class="type-cost">¥{{ t.cost.toFixed(4) }}</text>
        </view>
      </view>
    </view>
    <!-- 明细列表 -->
    <view class="section-title">调用明细</view>
    <view class="record-list">
      <view class="record-card" v-for="r in records" :key="r.id">
        <view class="record-top">
          <text :class="['call-badge', r.callType.startsWith('llm') ? 'badge-llm' : 'badge-tts']">
            {{ typeLabel(r.callType) }}
          </text>
          <text class="record-time">{{ formatTime(r.time) }}</text>
        </view>
        <view class="record-info">
          <text class="record-model">{{ r.model }}</text>
        </view>
        <view class="record-tokens" v-if="r.inputTokens || r.outputTokens">
          <text v-if="r.callType.startsWith('llm')">
            in: {{ r.inputTokens }}积分 / out: {{ r.outputTokens }}积分
          </text>
          <text v-else>
            字符: {{ r.textLen || r.outputTokens }}
          </text>
```

```
        </view>
        <view class="record-bottom">
          <text class="record-cost">¥{{ r.cost }}</text>
          <text class="record-dur" v-if="r.duration">{{ (r.duration / 1000).toFixed(1) }}s</text>
          <text :class="['record-status', r.success ? 'status-ok' : 'status-fail']">
            {{ r.success ? '  ' : '  ' }}
          </text>
        </view>
      </view>
    </view>
    <view class="empty" v-if="loading">
      <text>加载中...</text>
    </view>
    <view class="empty" v-else-if="!records.length">
      <text>暂无消费记录</text>
    </view>
  </view>
</template>
<script setup lang="ts">
import { ref, onMounted } from 'vue';
import { request } from '@/utils/request';
import { formatNumber } from '@/utils/format';
interface LogRecord {
  id: number; callType: string; provider: string; model: string;
  textLen: number; inputTokens: number; outputTokens: number;
  cost: number; duration: number; success: boolean; bookId: number; time: string;
}
const records = ref<LogRecord[]>([]);

// ═══════════════════════════════════════
// 文件: my-uniapp-vue3/src/pages/notifications/index.vue
// ═══════════════════════════════════════
<template>
  <view class="page">
    <view class="nav-bar">
      <view class="nav-btn" @click="goBack">
        <text class="nav-icon">←</text>
      </view>
      <text class="nav-title">消息通知</text>
      <view class="nav-btn" @click="markAllRead">
        <text class="read-all">全部已读</text>
      </view>
    </view>
    <scroll-view scroll-y class="content" :scroll-top="scrollTop">
      <view v-if="notifications.length === 0 && !loading" class="empty">
        <text class="empty-icon">  </text>
        <text class="empty-text">暂无通知</text>
      </view>
      <view v-else class="notification-list">
        <view
          v-for="item in notifications"
          :key="item.id"
          class="notification-item"
          :class="{ unread: !item.isRead }"
          @click="markAsRead(item)"
        >
          <view class="notification-icon" :class="getIconClass(item.type)">
            <text>{{ getIcon(item.type) }}</text>
```

```
          </view>
          <view class="notification-content">
            <text class="notification-title">{{ item.title }}</text>
            <text class="notification-desc">{{ item.content }}</text>
            <text class="notification-time">{{ formatTime(item.createdAt) }}</text>
          </view>
          <view v-if="!item.isRead" class="unread-dot"></view>
        </view>
      </view>
      <view v-if="loading" class="loading">
        <text>加载中...</text>
      </view>
    </scroll-view>
  </view>
</template>
<script setup lang="ts">
import { ref, onMounted } from 'vue';
import { get, put } from '../../utils/request';
// scroll-top 用于避免 scrollTop 错误
const scrollTop = ref(0);
const notifications = ref<any[]>([]);
const loading = ref(false);
function goBack() {
  const pages = getCurrentPages();
  if (pages.length > 1) {
    uni.navigateBack();
  } else {
    uni.switchTab({ url: '/pages/index/index' });
  }
}
function getIcon(type: string): string {
  const icons: Record<string, string> = {
    audio_completed: '  ',
    book_completed: '  ',
    video_completed: '  ',
    member_expiring: '    ',
    member_expired: '  ',
    daily_reminder: '  ',
  };
  return icons[type] || '  ';
}
function getIconClass(type: string): string {
  if (type.includes('completed')) return 'success';
  if (type.includes('expiring')) return 'warning';
  if (type.includes('expired')) return 'error';
  return 'info';
}
function formatTime(dateStr: string): string {
  const date = new Date(dateStr);
  const now = new Date();
  const diff = now.getTime() - date.getTime();
  const minutes = Math.floor(diff / (1000 * 60));
  const hours = Math.floor(diff / (1000 * 60 * 60));
  const days = Math.floor(diff / (1000 * 60 * 60 * 24));

// ============================================
// 文件: my-uniapp-vue3/src/pages/album/index.vue
// ============================================
```

```
    }
    // 检查是否有正在运行的批量生成任务
    const existingTask = runningTasks.get(bookId);
    if (existingTask) {
      ctx.status = 400;
      ctx.body = {
        code: 1,
        message: '项目已有批量转换任务在运行，请先取消后再试',
        data: { taskId: existingTask.taskId }
      };
      return;
    }
    // 创建批量生成任务
    const { taskId, orchestrator } = await createBatchGenerationTask(bookId, steps);
    // 存储任务信息
    runningTasks.set(bookId, { taskId, bookId });
    // 后台执行任务
    (async () => {
      try {
        const result = await orchestrator.execute();
        console.log(`[BatchGen][${taskId}] 任务完成:`, result);
        // 发送完成消息
        const { pushBatchGenerationProgress } = await import('../../services/websocket.service.js');
        pushBatchGenerationProgress(taskId, 'completed', 100);
      } catch (error: any) {
        console.error(`[BatchGen][${taskId}] 任务异常:`, error);

// ============================================
// 文件: server/src/modules/audio-project/audio-project.service.ts
// ============================================
/**
 * 批量转换编排服务
 * 负责按顺序执行生成步骤并推送进度
 */
import { bookStore } from './audio-project.store';
import { pushBatchGenerationProgress } from '../../services/websocket.service.js';
import { createVideoProjectFromBook, generateVideoForProject } from '../video-generator/video-generator.service';
import { mergeChapterAudios } from '../player/player.service';
import { prisma } from '../../models';
import { advanceChapter, regenerateChapter } from './stage-manager';
import { estimateBookWords, estimateAudioMinutesFromWords, atomicReserveQuota, releaseQuota, getUserMonthlyCost, AUDIO_BILLING_CONFIG } from '../
subscription/subscription.service';
// 步骤类型
export type GenerationStep = 'generate_content' | 'generate_audio' | 'merge_audio' | 'generate_video' | 'merge_video';
// 取消标志
const cancellationFlags = new Map<string, boolean>();
/**
 * 设置取消标志
 */
export function setCancellationFlag(taskId: string): void {
  cancellationFlags.set(taskId, true);
}
/**
 * 清除取消标志
 */
export function clearCancellationFlag(taskId: string): void {
  cancellationFlags.delete(taskId);
}
```

```
/**
 * 检查是否已取消
 */
export function isTaskCancelled(taskId: string): boolean {
  return cancellationFlags.get(taskId) === true;
}
/**
 * 批量生成编排器
 */
export class BatchGenerationOrchestrator {
  private taskId: string;
  private bookId: string;
  private steps: GenerationStep[];
  constructor(taskId: string, bookId: string, steps: GenerationStep[]) {
    this.taskId = taskId;
    this.bookId = bookId;
    this.steps = steps;
  }
  /**
   * 推送进度
   */
  private pushProgress(step: string, progress: number, message: string): void {
    pushBatchGenerationProgress(this.taskId, step, progress);
    console.log(`[BatchGen][${this.taskId}] ${step}: ${progress}% - ${message}`);
  }
  /**
   * 检查是否已取消
   */
  private checkCancellation(): void {
    if (isTaskCancelled(this.taskId)) {
      console.log(`[BatchGen][${this.taskId}] 任务已取消`);
      throw new Error('TASK_CANCELLED');
    }
  }
  /**
   * 执行所有步骤
   */
  async execute(): Promise<{ success: boolean; completedSteps: GenerationStep[]; failedStep?: string; error?: string }> {
    const completedSteps: GenerationStep[] = [];
    try {
      // 获取项目信息
      const book = await bookStore.getById(this.bookId);
      if (!book) {
        return { success: false, completedSteps, failedStep: 'init', error: '项目不存在' };
      }
      // 执行每个步骤
      for (let i = 0; i < this.steps.length; i++) {
        this.checkCancellation();
        const step = this.steps[i];
        const stepProgress = Math.round((i / this.steps.length) * 100);
        this.pushProgress(step, stepProgress, '开始执行');
        try {
          switch (step) {

// ============================================
// 文件: server/src/modules/audio-project/langgraph-controller.ts
// ============================================
/**
```

```
 * LangGraph 批量转换 - API 路由
 * 支持项目创建时的预估显示
 */
import Router from '@koa/router';
import { Context } from 'koa';
import { langGraphGenerator, getScaleConfig, resolveGenLevel, mapBookTypeToGenLevel } from './index';
import { bookStore, cancelAudioGeneration } from './audio-project.store';
import { prisma } from '../../models';
import { estimateBookWords, estimateAudioMinutesFromWords, checkBookGenerationQuota, checkAudioQuota, atomicReserveQuota, releaseQuota, AUDIO_BILLING_CONFIG } from '../subscription/subscription.service';
import { optionalAuth } from '../../middleware/auth';
import { getAllBookTypes, getDetectableTypes, getBookTypeConfig, BOOK_TYPE_CONFIG, DETECTABLE_TYPES } from './book-type-config';
import { callLLMWithMessages, callLLM, ChatMessage } from '../../services/llm';
import { cleanLlmShortText, extractJsonFromResponse } from '../../services/llm/response-cleaner';
import { runWithContext } from '../../services/llm-context';
import { createVideoProjectFromBook, generateVideoForProject } from '../video-generator/video-generator.service';
import { mergeChapterAudios } from '../player/player.service';
import { exec } from 'child_process';
import { promisify } from 'util';
import fs from 'fs/promises';
import path from 'path';
import { advanceChapter, regenerateChapter } from './stage-manager';
import { deepPlanBookNode } from './nodes/deep-plan.node';
import { richOutlineNode } from './nodes/rich-outline.node';
import { writeChaptersParallelNode } from './nodes/content.node';
import { continuityEditNode } from './nodes/continuity-edit.node';
import { GraphState } from './graph';
const execPromise = promisify(exec);
// 开发环境测试用户ID
const TEST_USER_ID = '1';
const router = new Router();
/**
 * GET /api/audio-project/langgraph/estimate
 * 获取项目规模预估信息
 */
router.get('/estimate', async (ctx: Context) => {
  const { scale, userId, bookType } = ctx.query as { scale?: string; userId?: string; bookType?: string };
  if (!scale) {
    ctx.status = 400;
    ctx.body = { code: 1, message: '请提供项目规模' };
    return;
  }
  const scaleConfig = getScaleConfig(scale);
  const totalWords = scaleConfig.totalWords;
  const audioMinutes = estimateAudioMinutesFromWords(totalWords);
  // 章节数
  const estimatedChapters = scaleConfig.isShortArticle ? 1 : scaleConfig.chapters;
  // 大纲层级（优先使用项目类型映射）
  const genLevel = bookType && bookType !== 'auto'
    ? mapBookTypeToGenLevel(bookType)
    : resolveGenLevel(scale);
  const genLevelDescriptions: Record<number, { label: string; desc: string }> = {
    1: { label: '仅章', desc: '无节和小节，适合短文、长篇文本' },
    2: { label: '章→节', desc: '有节无小节，适合科普、商业书' },
    3: { label: '章→节→小节', desc: '完整三级结构，适合教材、技术书' },
  };
  const result: any = {
    scale,
```

```
    scaleConfig,
    audioMinutes: {
      min: estimateAudioMinutesFromWords(Math.round(totalWords * 0.8)),
      max: estimateAudioMinutesFromWords(Math.round(totalWords * 1.2)),
      avg: audioMinutes
    },
    estimatedChapters,
    defaultGenLevel: genLevel,
    genLevelInfo: genLevelDescriptions[genLevel] || { label: '章→节', desc: '' },
  };
  // 如果提供了 userId，同时检查用户配额
  if (userId) {
    const quotaCheck = await checkBookGenerationQuota(parseInt(userId), scale);
    result.quotaCheck = quotaCheck;
  }
  ctx.body = {
    code: 0,
    message: 'success',
    data: result
  };
});
/**

// ═══════════════════════════════════════
// 文件: server/src/modules/audio-project/audio-project.store.ts
// ═══════════════════════════════════════
/**
 * 批量转换模块 - Prisma 数据库存储
 */
import crypto from 'crypto';
import { prisma } from '../../models';
import { Book, BookOutline, Chapter, ChapterGenStage, BookGenStage } from './audio-project.types';
import { Prisma } from '@prisma/client';
import { generateAudio } from '../tts/tts.service';
import { callLLMWithMessages, callLLMWithTools, ChatMessage } from '../../services/llm';
import { createBookTools } from '../../services/llm/tools';
import { SUBSECTION_CONTENT_SYSTEM_PROMPT } from './prompts/templates';
import { countWords } from './utils';
import { cleanThinkingText } from './utils/content-cleaner';
import { advanceChapter, regenerateChapter, safeTransitionChapter } from './stage-manager';
import { mergeChapterAudios } from '../player/player.service';
import { consumeAudioMinutes, canUseTtsProvider } from '../subscription/subscription.service';
import { runWithContext } from '../../services/llm-context';
/**
 * 取消项目所有章节的音频生成（将 pending/processing 任务标记为 cancelled，回退章节阶段）
 */
export async function cancelAudioGeneration(bookId: string): Promise<{ cancelledCount: number; rolledBackChapters: number[] }> {
  const chapters = await prisma.bookChapter.findMany({
    where: { bookId: Number(bookId) },
    select: { id: true, level: true },
  });
  if (chapters.length === 0) {
    return { cancelledCount: 0, rolledBackChapters: [] };
  }
  const maxLevel = Math.max(...chapters.map(c => c.level || 0));
  const leafChapterIds = chapters.filter(c => c.level === maxLevel).map(c => c.id);
  // 找出所有 pending/processing 状态的 TTS 任务
  const activeTasks = await prisma.ttsTask.findMany({
```

```
    where: {
      chapterId: { in: leafChapterIds },
      taskType: 'tts',
      status: { in: ['pending', 'processing'] },
    },
    select: { id: true, chapterId: true },
  });
  if (activeTasks.length === 0) {
    return { cancelledCount: 0, rolledBackChapters: [] };
  }
  const taskIds = activeTasks.map(t => t.id);
  const affectedChapterIds = [...new Set(activeTasks.map(t => t.chapterId))];
  // 批量标记任务为 cancelled
  await prisma.ttsTask.updateMany({
    where: { id: { in: taskIds } },
    data: { status: 'cancelled' },
  });
  // 回退受影响章节的 genStage 到 content_completed
  await prisma.bookChapter.updateMany({
    where: { id: { in: affectedChapterIds }, genStage: 'audio_generating' },
    data: { genStage: 'content_completed' },
  });
  return {
    cancelledCount: taskIds.length,
    rolledBackChapters: affectedChapterIds,
  };
}
// ============ 删除项目 ============
/**
 * 根据所有章节状态计算项目阶段
 * 项目阶段 = 所有章节中最低的阶段（最落后的章节决定了项目的进度）
 */
function computeBookGenStage(chapters: { genStage: string }[]): BookGenStage {
  if (chapters.length === 0) return 'draft';
  // 章节阶段顺序（索引越大越"后"）
  const stageOrder = ['idle', 'outline_completed', 'content_generating', 'content_completed', 'audio_generating', 'audio_completed', 'video_gener
ating', 'video_completed', 'failed'];
  // 找出最低阶段的索引
  let minIdx = stageOrder.length; // 默认最大
  for (const ch of chapters) {
    const idx = stageOrder.indexOf(ch.genStage);
    if (idx === -1) {
      console.warn(`[BookStore] 未知章节阶段: chapterId=${(ch as any).id}, genStage="${ch.genStage}"，跳过该章节`);
      continue;
    }
    if (idx < minIdx) {
      minIdx = idx;
    }
  }

// ============================================
// 文件: server/src/modules/audio-project/audio-project.types.ts
// ============================================
/**
 * 批量转换模块 - 类型定义
 * 定义项目、章节、任务的数据结构
 */
// ============ 核心类型 ============
```

```
/** 批量转换阶段 */
export type BookGenStage =
  | 'draft'
  | 'outlining'
  | 'outline_completed'
  | 'content_generating'
  | 'content_completed'
  | 'audio_generating'
  | 'audio_completed'
  | 'video_generating'
  | 'video_completed'
  | 'failed';
/** 章节生成阶段（唯一类型定义，stage-manager.ts 从此文件导入） */
export type ChapterGenStage =
  | 'idle'
  | 'outline_completed'
  | 'content_generating'
  | 'content_completed'
  | 'audio_generating'
  | 'audio_completed'
  | 'video_generating'
  | 'video_completed'
  | 'failed';
/** 项目基础信息 */
export interface BookBase {
  id: string;
  title: string;                    // 标题
  subtitle?: string;                // 副标题
  description: string;              // 项目描述/用户输入
  targetAudience: string;           // 目标受众
  style: string;                    // 文本风格
  bookScale: string;                // 项目规模：纯数字字符串，如 1000, 2000, 130000, 340000 等
  totalChapters: number;            // 总章节数
  estimatedWords: number;           // 预估总字数
  genStage?: BookGenStage;          // 线性阶段状态
  failedStage?: string;             // 失败阶段
  createdAt: Date;
  updatedAt: Date;
}
/** 项目完整信息 */
export interface Book extends BookBase {
  userId?: number;                  // 用户ID（用于配额检查）
  progress: number;                 // 处理进度 0-100
  isPublished: boolean;             // 是否已公开
  chapters: Chapter[];              // 章节列表
  outline?: BookOutline;            // 项目大纲
  metadata?: BookMetadata;          // 元数据
  error?: string;                   // 错误信息
  bookAnalysis?: string;            // AI 项目规划结果（planBookNode 输出）
}
/** 项目大纲 */
export interface BookOutline {
  bookType?: 'textbook' | 'novel' | 'popular' | 'essay';  // 文本类型（AI判断）
  mainTheme: string;                // 主题主线
  structureLogic: string;           // 结构逻辑
  chapters: OutlineChapter[];      // 章节大纲
}
/** 章节大纲（规划阶段）*/
```

```
export interface OutlineChapter {
  number: number;
  title: string;
  summary: string;                  // 章节概述
  keyPoints: string[];              // 核心知识点
  estimatedWords: number;           // 预估字数
  stories?: string[];               // 故事/案例
  sections?: {                      // 节（章下第一级）
    number: number;                 // 节序号
    title: string;
    summary?: string;               // 本节概述
    keyPoints?: string[];           // 本节要点
    estimatedWords?: number;        // 本节预估字数
    subsections?: {                 // 小节（节下第二级）
      number: number;               // 小节序号
      title: string;
      summary?: string;             // 本小节概述

// ═══════════════════════════════════════
// 文件: server/src/modules/audio-project/graph.ts
// ═══════════════════════════════════════
/**
 * LangGraph 状态定义和工作流
 */
import { Annotation, StateGraph, END } from '@langchain/langgraph';
// ============ 状态定义（借鉴 OpenMAIC Annotation 模式）============
/**
 * 进度 reducer：只增不减，防止中间步骤回退导致进度丢失
 */
const maxReducer = (prev: number, update: number) => Math.max(prev, update);
/**
 * 章节完成数 reducer：累加而非覆盖
 */
const appendReducer = <T>(prev: T[], update: T | T[] | undefined) => {
  if (!update) return prev;
  const items = Array.isArray(update) ? update : [update];
  return [...prev, ...items];
};
export const GraphState = Annotation.Root({
  bookId: Annotation<string>({
    reducer: (_prev, update) => update ?? _prev,
    default: () => '' as string,
  }),
  userId: Annotation<string | number>({
    reducer: (_prev, update) => update ?? _prev,
    default: () => '' as string,
  }),
  topic: Annotation<string>({
    reducer: (_prev, update) => update ?? _prev,
    default: () => '' as string,
  }),
  bookScale: Annotation<string>({
    reducer: (_prev, update) => update ?? _prev,
    default: () => '1000' as string,
  }),
  /**
   * 大纲层级：1=仅章，2=章→节，3=章→节→小节。
   * reducer: update ?? _prev — 仅当 update 非 null/undefined 时才覆盖旧值。
```

```
   * 0/false 被视为 falsy 会保留旧值；合法的 genLevel 值是 1/2/3，不受此限制。
   */
  genLevel: Annotation<number>({
    reducer: (_prev, update) => update ?? _prev,
    default: () => 2,
  }),
  /** 用户明确指定的大纲层级（undefined=未指定/自动，1/2/3=用户主动选择）。
   *  用于区分"用户主动选了2"和"系统默认2"，防止 AI 规划时误覆盖。 */
  userSpecifiedGenLevel: Annotation<number | undefined>({
    reducer: (_prev, update) => update ?? _prev,
    default: () => undefined as number | undefined,
  }),
  description: Annotation<string>({
    reducer: (_prev, update) => update ?? _prev,
    default: () => '' as string,
  }),
  /** AI 项目规划结果（planBookNode 输出） */
  bookPlan: Annotation<string | undefined>({
    reducer: (_prev, update) => update ?? _prev,
    default: () => undefined,
  }),
  /** 当前正在处理的章节号 */
  currentChapter: Annotation<number>({
    reducer: maxReducer,
    default: () => 0,
  }),
  /** 已成功完成的章节数（只增不减） */
  completedChapters: Annotation<number[]>({
    reducer: appendReducer,
    default: () => [] as number[],
  }),
  finished: Annotation<boolean>({
    reducer: (_prev, update) => update ?? _prev,
    default: () => false,
  }),
  error: Annotation<string | undefined>({
    reducer: (_prev, update) => update ?? _prev,
    default: () => undefined,
  }),
  /** 生成进度 0-100，只增不减 */
  progress: Annotation<number>({
    reducer: maxReducer,
    default: () => 0,

// ═══════════════════════════════════════
// 文件: server/src/modules/audio-project/nodes/content.node.ts
// ═══════════════════════════════════════
/**
 * 文本转换节点
 * 为所有叶节点（没有子节点的节点）进行文本分段处理
 */
import { GraphState } from '../graph';
import { bookStore } from '../audio-project.store';
import { prisma } from '../../../../models';
import { createBookTools } from '../../../../services/llm/tools';
import { checkQuotaForWords, markGenerationInterrupted, getGeneratedWordCount } from '../../../subscription/subscription.service';
import { PROGRESS, countWords } from '../utils';
import { advanceChapter, regenerateChapter } from '../stage-manager';
```

```
import { getBookWordLimit, getWordUpperLimit } from '../book-type-config';
import { AsyncPool } from '../utils/async-pool';
import { cleanThinkingText, truncateAtBoundary } from '../utils/content-cleaner';
import { safeTransitionChapter } from '../stage-manager';
import { BookConsistencyTracker } from '../utils/content-consistency';
import { withGlobalLLMConcurrency, getBookConcurrency } from '../utils/llm-concurrency';
/** 全局术语一致性跟踪器（单例，跨整本书的章节共享） */
const globalConsistencyTracker = new BookConsistencyTracker();
/**
 * 为被截断的内容补一个自然结尾句，
 * 避免截断后看起来像没写完。
 */
function appendNaturalClosure(content: string): string {
  if (!content || content.length < 50) return content;
  const trimmed = content.trimEnd();
  // 已有完整结尾标点的不补
  const naturalEndings = /[。！？.!?）』〞”’]\s*$/;
  if (naturalEndings.test(trimmed)) return trimmed;
  // 从内容末尾提取关键词，生成一个简短收尾句
  const lastSentences = trimmed.split(/[。！？.!?]/).filter(Boolean);
  const lastTopic = lastSentences.length > 0
    ? lastSentences[lastSentences.length - 1].substring(0, 30).trim()
    : '';
  if (lastTopic.length > 3) {
    return trimmed + `。以上就是关于${lastTopic}的讨论。`;
  }
  return trimmed + '。以上就是本章的主要内容。';
}
/**
 * 安全更新父章节状态：仅在父节点下所有子节点都已完成时，才推进父节点状态。
 * 替代原先的 updateMany 批量操作，避免失败子节点污染已完成父节点状态。
 */
async function safelyCompleteParentChapters(bookIdNum: number): Promise<void> {
  const parents = await prisma.bookChapter.findMany({
    where: { bookId: bookIdNum, level: { in: [1, 2] } },
    select: { id: true, genStage: true },
  });
  const allChildren = await prisma.bookChapter.findMany({
    where: { bookId: bookIdNum, parentId: { in: parents.map(p => p.id) } },
    select: { id: true, genStage: true, parentId: true },
  });
  // 按 parentId 分组
  const childrenByParent = new Map<number, typeof allChildren>();
  for (const child of allChildren) {
    if (!child.parentId) continue;
    if (!childrenByParent.has(child.parentId)) childrenByParent.set(child.parentId, []);
    childrenByParent.get(child.parentId)!.push(child);
  }
  for (const parent of parents) {
    const children = childrenByParent.get(parent.id) || [];
    if (children.length === 0) continue; // 无子节点（单层大纲），跳过
    const allChildrenDone = children.every(
      c => c.genStage === 'content_completed' || c.genStage === 'audio_generating'
        || c.genStage === 'audio_completed' || c.genStage === 'video_generating'
        || c.genStage === 'video_completed'
    );
    if (allChildrenDone && parent.genStage !== 'content_completed') {
      try {
```

```
        await safeTransitionChapter(parent.id, parent.genStage as any, 'content_completed');
      } catch (err: any) {
        console.warn(`[Content] 父节点 #${parent.id} 状态转移失败:`, err.message);
      }
    }
  }
}
/**
 * 文本转换节点
 * 为所有叶节点（没有子节点的节点）进行文本分段处理
 */

// ============================================
// 文件: server/src/modules/audio-project/nodes/plan.node.ts
// ============================================
/**
 * 项目规划节点（planBookNode）
 * 在文本转换之前，由 AI 对整篇文本做分析规划：
 *   - 分析文本类型、目标受众、内容深度
 *   - 决定分段层级（1/2/3层）
 *   - 规划文本结构和分段逻辑
 *   - 输出完整的分段方案
 *
 * 该节点的输出（planResult）会被存储到 GraphState，供后续节点参考。
 */
import { GraphState } from '../graph';
import { bookStore } from '../audio-project.store';
import { callLLMWithMessages, ChatMessage } from '../../../services/llm';
import { PROGRESS } from '../utils';
import { getScaleConfig } from '../book-type-config';
/**
 * 项目规划输出接口
 */
export interface BookPlan {
  /** 最终确定的大纲层级 */
  genLevel: number;
  /** 文本类型分析 */
  bookTypeAnalysis: string;
  /** 推荐的写作风格 */
  writingStyle: string;
  /** 结构逻辑说明 */
  structureLogic: string;
  /** 内容深度评估 */
  contentDepth: string;
  /** 目标读者分析 */
  targetAudienceAnalysis: string;
  /** 规划理由 */
  reasoning: string;
}
/**
 * 从描述中提取明确的字数要求
 * 如 "生成200字文本" → 200
 */
function extractWordCountFromDescription(description: string): number | null {
  const text = description.toLowerCase();
  // 匹配 "200字"、"500字" 等
  const match = text.match(/(\d+)\s*字/i);
  if (match) {
```

```
    const n = parseInt(match[1]);
    return isNaN(n) ? null : n;
  }
  return null;
}
/**
 * 构建规划提示词
 */
function buildPlanPrompt(
  title: string,
  description: string,
  bookScale: string
): string {
  const config = getScaleConfig(bookScale);
  const chapters = config?.chapters || 17;
  // 如果 description 里有明确字数要求，优先用它
  const descWordCount = extractWordCountFromDescription(description);
  const totalWords = descWordCount !== null
    ? descWordCount
    : (config?.totalWords || 130000);
  return `你对以下文本做全面分析规划。
## 文本信息
- 标题：${title}
- 字数规模：约${totalWords}字，约${chapters}章
- 描述：${description}
## 你的职责
对整篇文本做完整的结构分析。
## 分析维度
### 1. 文本类型分析
判断这篇文本属于以下哪种类型：
- 教材/学术类：系统教学、理论知识体系完整
- 技术教程类：有操作步骤、实践指南
- 文学类：叙事性、描述性内容
- 商业/经管类：管理、营销、投资、创业
- 科普/大众类：普及科学知识，通俗易懂
- 其他类型
### 2. 大纲层级决策

// ═══════════════════════════════════════
// 文件: server/src/modules/audio-project/nodes/outline.node.ts
// ═══════════════════════════════════════
/**
 * 大纲生成节点（集成容错机制）
 */
import { GraphState } from '../graph';
import { bookStore } from '../audio-project.store';
import { callLLMWithMessages } from '../../../services/llm';
import { parseOutline } from '../parsers/outline.parser';
import { buildOutlineMessages, SCALE_CHAPTER_RANGE } from '../prompts/builder';
import { PROGRESS } from '../utils';
import { callLLMWithRetry, executeNodeWithTimeout, FAULT_TOLERANCE_CONFIG } from '../fault-tolerance';
import { getChapterRange, calcChapters } from '../book-type-config';
export async function generateOutlineNode(state: typeof GraphState.State): Promise<Partial<typeof GraphState.State>> {
  console.log('[LangGraph] 生成大纲, bookId:', state.bookId, 'scale:', state.bookScale);
  // 读取前序规划节点的分析结果，用于指导大纲生成
  let bookPlan: any = null;
  if (state.bookPlan) {
    try {
```

```
    bookPlan = JSON.parse(state.bookPlan);
    console.log('[LangGraph] 使用规划结果指导大纲:', bookPlan.structureLogic, bookPlan.writingStyle);
  } catch {
    console.warn('[LangGraph] bookPlan 解析失败，忽略');
  }
}
try {
  // 使用容错包装器执行AI调用
  const response = await executeNodeWithTimeout(
    state.bookId,
    'generate_outline',
    async () => {
      return callLLMWithRetry(
        buildOutlineMessages(state.topic, state.bookScale, state.description, bookPlan),
        undefined,
        {
          bookId: state.bookId,
          nodeId: 'generate_outline',
          attempt: 0,
          maxAttempts: FAULT_TOLERANCE_CONFIG.aiRetry.maxRetries,
        }
      );
    },
    FAULT_TOLERANCE_CONFIG.nodeTimeout.generate_outline
  );
  const outline = parseOutline(response);
  if (!outline) throw new Error('大纲解析失败');
  // 校验章节数：允许 ±20% 浮动，超出才截断/补充
  const targetChapters = SCALE_CHAPTER_RANGE[state.bookScale as keyof typeof SCALE_CHAPTER_RANGE];
  if (targetChapters) {
    const { min: minAllowed, max: maxAllowed } = getChapterRange(targetChapters);
    const actual = outline.chapters.length;
    if (actual > maxAllowed) {
      console.warn(`[LangGraph] AI 返回章节数 ${actual} 超出上限 ${maxAllowed}，截断`);
      outline.chapters = outline.chapters.slice(0, maxAllowed);
    } else if (actual < minAllowed) {
      console.warn(`[LangGraph] AI 返回章节数 ${actual} 少于下限 ${minAllowed}，使用话题相关的默认章节补充`);
      const topicHint = state.topic?.substring(0, 30) || '';
      const perChapterWords = Math.round(parseInt(state.bookScale) / minAllowed);
      const gapTopics = [
        `概述与背景`,
        `核心概念`,
        `深入探究`,
        `应用与实践`,
        `总结与展望`,
      ];
      while (outline.chapters.length < minAllowed) {
        const idx = outline.chapters.length;
        const gapTitle = gapTopics[idx % gapTopics.length];
        outline.chapters.push({
          number: idx + 1,
          title: topicHint ? `第${idx + 1}章 ${topicHint} - ${gapTitle}` : `第${idx + 1}章 ${gapTitle}`,
          summary: `本章围绕"${topicHint || gapTitle}"展开，介绍${gapTitle}相关内容。`,
          keyPoints: [gapTitle, '关键知识点', '实践要点'],
          estimatedWords: perChapterWords,
        });
      }
    }
```

```
      // 在范围内（±20%）不做任何处理，让 AI 自己决定
    }
    // 短文类至少1个章节
    if (outline.chapters.length === 0) {
      console.warn('[LangGraph] AI 返回章节数为0，使用话题创建默认章节');

// ═══════════════════════════════════════
// 文件: server/src/modules/audio-project/nodes/quality-check.node.ts
// ═══════════════════════════════════════
/**
 * 质量校验节点（qualityCheckNode）
 *
 * 在正文内容并行生成完成后，对所有章节做四维质量评估：
 * - 通顺性（fluency）
 * - 逻辑性（logic）
 * - 是否跑题（relevance）
 * - 是否达标（completeness）
 *
 * 输出每个章节的评分和问题列表，决定是否进入重写环节。
 * 与 rewrite.node.ts 组成质量闭环：校验→不通过→重写→再校验
 */
import { GraphState } from '../graph';
import { bookStore } from '../audio-project.store';
import { callLLMWithMessages, ChatMessage } from '../../../services/llm';
import { callLLMWithRetry, executeNodeWithTimeout, FAULT_TOLERANCE_CONFIG } from '../fault-tolerance';
import { QUALITY_CHECK_SYSTEM_PROMPT } from '../prompts/templates';
import { PROGRESS, countWords } from '../utils';
import { prisma } from '../../../models';
// ============ 类型定义 ============
export interface QualityScore {
  fluency: number;      // 通顺 0-25
  logic: number;        // 逻辑 0-25
  relevance: number;    // 跑题 0-25
  completeness: number; // 达标 0-25
}
export interface QualityIssue {
  dimension: 'fluency' | 'logic' | 'relevance' | 'completeness';
  severity: 'high' | 'medium' | 'low';
  description: string;
  location: string;
  suggestion: string;
}
export interface FailedChapter {
  chapterNumber: number;
  chapterTitle: string;
  scores: QualityScore;
  totalScore: number;
  issues: QualityIssue[];
  rewriteInstructions: string;
}
export interface QualityCheckResult {
  overallScore: number;
  overallAssessment: string;
  passedChapters: number[];
  failedChapters: FailedChapter[];
}
// ============ 常量 ============
/** 合格分数线 */
```

```
const PASS_THRESHOLD = 80;
/** 每批最大发送字符数（避免 token 超限）*/
const MAX_BATCH_CHARS = 60000;
// ============ 核心逻辑 ============
/**
 * 构建单批质量校验消息
 */
function buildBatchQualityCheckMessages(
  bookTitle: string,
  bookTopic: string,
  batchChapters: Array<{ number: number; title: string; content: string; summary?: string; estimatedWords?: number }>
): ChatMessage[] {
  let chaptersText = '';
  for (const ch of batchChapters) {
    const header = `\n## 第${ch.number}章：${ch.title}\n> 概述：${ch.summary || '无'}\n> 预估字数：${ch.estimatedWords || 0} | 实际字数：${countWords(ch.content)}\n\n`;
    const content = ch.content || '（空）';
    chaptersText += header + content;
  }
  return [
    { role: 'system', content: QUALITY_CHECK_SYSTEM_PROMPT },
    {
      role: 'user',
      content: `书名：《${bookTitle}》
主题：${bookTopic}
以下是 ${batchChapters.length} 个章节的完整内容，请逐一评估质量：
${chaptersText}
请输出 JSON 格式的质量评估报告。`,
    },
  ];
}
/**

// ═══════════════════════════════════════════
// 文件: server/src/modules/tts/tts.service.ts
// ═══════════════════════════════════════════
import path from 'path';
import fs from 'fs';
import { v4 as uuidv4 } from 'uuid';
import { config } from '../../config';
import { prisma } from '../../models';
import { VoiceParams, Voice } from '../../types';
import { AudioMerger } from './audio-merger';
import { aiSummaryService } from './ai-summary.service';
import { storageService } from '../../services/storage.service';
import { getTtsRegistry, getAvailableTtsProvider, startTtsHealthCheck } from './provider.registry';
import { ITtsProvider } from './provider.interface';
import { CircuitBreakerOpenError } from '../../common/circuit-breaker';
import { ProviderNode } from '../../common/provider-registry';
import { ttsLogger } from './tts-logger';
import axios from 'axios';
// ============ 统一音色定义（10个固定音色，前端使用）============
// 前端使用统一 ID，后端根据 Provider 类型映射到真实音色
const UNIFIED_VOICES: Voice[] = [
  { id: 'voice_01', name: '温柔女声',  gender: 'female', description: '柔和温暖，适合情感故事' },
  { id: 'voice_02', name: '磁性男声',  gender: 'male',   description: '低沉有力，适合悬疑推理' },
  { id: 'voice_03', name: '活泼女声',  gender: 'female', description: '清新明亮，适合儿童故事' },
  { id: 'voice_04', name: '知性女声',  gender: 'female', description: '知性稳重，适合科普知识' },
```

```
  { id: 'voice_05', name: '阳光男声',  gender: 'male',   description: '阳光活力，适合校园青春' },
  { id: 'voice_06', name: '沧桑男声',  gender: 'male',   description: '成熟沧桑，适合历史军事' },
  { id: 'voice_07', name: '甜美女声',  gender: 'female', description: '甜美可爱，适合爱情都市' },
  { id: 'voice_08', name: '清朗男声',  gender: 'male',   description: '清朗干练，适合职场商战' },
  { id: 'voice_09', name: '亲切女声',  gender: 'female', description: '亲切自然，适合日常叙事' },
  { id: 'voice_10', name: '稚嫩童声',  gender: 'female', description: '稚嫩天真，适合童话寓言' },
];
// 统一音色 → 阿里云 真实音色映射
const ALIYUN_VOICE_MAP: Record<string, string> = {
  voice_01: 'longanyang',    voice_02: 'longsanshu_v3', voice_03: 'longhuhu_v3',
  voice_04: 'longyue_v3',    voice_05: 'longyichen_v3', voice_06: 'longlaobo_v3',
  voice_07: 'longmiao_v3',   voice_08: 'longshuo_v3',   voice_09: 'longwan_v3',
  voice_10: 'longhuhu_v3',
};
// 统一音色 → Edge-TTS 真实音色映射
const EDGE_VOICE_MAP: Record<string, string> = {
  voice_01: 'zh-CN-XiaoxiaoNeural',    // 温柔女声 → 晓晓
  voice_02: 'zh-CN-YunxiNeural',       // 磁性男声 → 云希
  voice_03: 'zh-CN-XiaoyiNeural',      // 活泼女声 → 晓依
  voice_04: 'zh-CN-YunyangNeural',     // 知性女声 → 云扬(新闻风格)
  voice_05: 'zh-CN-YunjianNeural',     // 阳光男声 → 云健
  voice_06: 'zh-CN-YunyangNeural',     // 沧桑男声 → 云扬
  voice_07: 'zh-CN-XiaoxiaoNeural',    // 甜美女声 → 晓晓
  voice_08: 'zh-CN-YunxiNeural',       // 清朗男声 → 云希
  voice_09: 'zh-CN-XiaoyiNeural',      // 亲切女声 → 晓依
  voice_10: 'zh-CN-XiaoshuangNeural',  // 稚嫩童声 → 晓双(童声)
};
function mapToProviderVoice(unifiedVoiceId: string, providerVendor: string): string {
  // Edge-TTS: 使用微软免费音色
  if (providerVendor === 'edge') {
    const mapped = EDGE_VOICE_MAP[unifiedVoiceId];
    if (mapped) return mapped;
    console.warn(`     [VoiceMap] Edge 未识别的音色ID: "${unifiedVoiceId}", 降级使用默认音色`);
    return 'zh-CN-XiaoxiaoNeural';
  }
  // 默认: 阿里云百炼 CosyVoice
  const mapped = ALIYUN_VOICE_MAP[unifiedVoiceId];
  if (mapped) return mapped;
  // 不在映射表中（如遗留的 'cherry' 等旧 MiniMax 音色）→ 用 CosyVoice 默认音色
  console.warn(`     [VoiceMap] 未识别的音色ID: "${unifiedVoiceId}", 降级使用默认音色 longyingling_v3`);
  return 'longyingling_v3';
}
// ============ Aliyun Instruct 情感/场景控制 ============
// 后期优化功能，暂不启用。启用时改为 true
const INSTRUCT_ENABLED = false;
interface EmotionScene {
  emotion: string;  // neutral | fearful | angry | sad | surprised | happy | disgusted
  scene: string;    // 闲聊互动 | 新闻播报 | 广告促销 | 比赛解说 | 一些儿童内容解说 | 语音导航 | 脱口秀表演
}
// 情感关键词库
const EMOTION_KEYWORDS: { emotion: string; keywords: string[] }[] = [
  { emotion: 'fearful',   keywords: ['恐怖', '可怕', '惊悚', '恐惧', '阴森', '黑暗', '鬼', '死亡', '谋杀', '悬疑', '危险', '深渊', '噩梦'] },
  { emotion: 'sad',       keywords: ['悲伤', '难过', '哭泣', '眼泪', '心痛', '遗憾', '孤独', '寂寞', '失落', '离别', '思念', '哀伤', '去世', '失去'] },
  { emotion: 'angry',     keywords: ['愤怒', '生气', '怒火', '仇恨', '战斗', '厮杀', '复仇', '战争', '侵略', '暴怒'] },
  { emotion: 'happy',     keywords: ['快乐', '开心', '幸福', '欢笑', '庆祝', '美好', '甜蜜', '温暖', '阳光', '喜悦', '高兴', '浪漫', '恋爱', '美好'] },
  { emotion: 'surprised', keywords: ['惊奇', '惊喜', '意外', '奇迹', '神奇', '魔法', '童话', '幻想', '奇妙'] },
```

```
  { emotion: 'disgusted', keywords: ['恶心', '厌恶', '肮脏', '丑陋', '卑鄙'] },
];

// ============================================
// 文件: server/src/modules/player/player.controller.ts
// ============================================
import Router from '@koa/router';
import { Context } from 'koa';
import * as PlayerService from './player.service';
import { BadRequestError } from '../../middleware/errorHandler';
import { optionalAuth } from '../../middleware/auth';
import { prisma } from '../../models';
// 测试用户ID（开发环境使用）
const TEST_USER_ID = '1';
const router = new Router();
// 获取播放进度列表
router.get('/progress', optionalAuth, async (ctx: Context) => {
  // 开发环境使用测试用户ID
  const userId = ctx.state.user?.userId || TEST_USER_ID;
  const { audioId } = ctx.query as { audioId?: string };
  const records = await PlayerService.getPlayProgress(
    userId,
    audioId ? parseInt(audioId) : undefined
  );
  ctx.body = {
    code: 0,
    message: 'success',
    data: records,
  };
});
// 保存播放进度
router.post('/progress', optionalAuth, async (ctx: Context) => {
  // 开发环境使用测试用户ID
  const userId = ctx.state.user?.userId || TEST_USER_ID;
  const body = ctx.request.body as {
    audioId: number;
    progress: number;
    duration: number;
  };
  const { audioId, progress, duration } = body;
  if (!audioId || typeof progress !== 'number' || typeof duration !== 'number') {
    throw new BadRequestError('参数错误');
  }
  const record = await PlayerService.savePlayProgress(userId, audioId, progress, duration);
  ctx.body = {
    code: 0,
    message: 'success',
    data: record,
  };
});
// 更新播放进度 (在 DELETE 路由之前定义)
router.put('/progress/:audioId', optionalAuth, async (ctx: Context) => {
  // 开发环境使用测试用户ID
  const userId = ctx.state.user?.userId || TEST_USER_ID;
  const audioId = parseInt(ctx.params.audioId as string);
  const body = ctx.request.body as { progress: number; duration?: number };
  const { progress, duration } = body;
  if (typeof progress !== 'number') {
```

```
    throw new BadRequestError('进度参数错误');
  }
  const record = await PlayerService.updatePlayProgress(userId, audioId, progress, duration);
  ctx.body = {
    code: 0,
    message: 'success',
    data: record,
  };
});
// 删除播放记录
router.delete('/progress/:audioId', optionalAuth, async (ctx: Context) => {
  // 开发环境使用测试用户ID
  const userId = ctx.state.user?.userId || TEST_USER_ID;
  const audioId = parseInt(ctx.params.audioId as string);
  await PlayerService.deletePlayRecord(userId, audioId);
  ctx.body = {
    code: 0,
    message: '删除成功',
  };
});
// 批量删除播放记录
router.delete('/progress/batch', optionalAuth, async (ctx: Context) => {
  const userId = ctx.state.user?.userId || TEST_USER_ID;
  const { audioIds } = ctx.request.body as { audioIds: number[] };
  if (!audioIds || !Array.isArray(audioIds)) {
    throw new BadRequestError('参数错误');
  }
  for (const audioId of audioIds) {

// ═══════════════════════════════════════
// 文件: server/src/modules/member/member.controller.ts
// ═══════════════════════════════════════
import Router from '@koa/router';
import { Context } from 'koa';
import * as MemberService from './member.service';
import { BadRequestError, NotFoundError } from '../../middleware/errorHandler';
import { authMiddleware } from '../../middleware/auth';
const router = new Router();
// 获取会员权益信息
router.get('/benefits', async (ctx: Context) => {
  const benefits = MemberService.getMemberBenefits();
  ctx.body = {
    code: 0,
    message: 'success',
    data: benefits,
  };
});
// 获取用户会员信息（兼容前端 /api/member/info）
router.get('/info', authMiddleware, async (ctx: Context) => {
  const userId = ctx.state.user.userId;
  const status = await MemberService.getMemberStatus(userId);
  ctx.body = {
    code: 0,
    message: 'success',
    data: status,
  };
});
// 获取用户会员状态
```

```
router.get('/status', authMiddleware, async (ctx: Context) => {
  const userId = ctx.state.user.userId;
  const status = await MemberService.getMemberStatus(userId);
  ctx.body = {
    code: 0,
    message: 'success',
    data: status,
  };
});
// 创建订单
router.post('/order', authMiddleware, async (ctx: Context) => {
  const userId = ctx.state.user.userId;
  const { productType } = ctx.request.body as { productType: 'monthly' | 'yearly' };
  if (!['monthly', 'yearly'].includes(productType)) {
    throw new BadRequestError('无效的产品类型');
  }
  const result = await MemberService.createOrder(userId, productType);
  ctx.body = {
    code: 0,
    message: '订单创建成功',
    data: result,
  };
});
// 模拟支付（仅开发环境）
router.post('/pay/mock', authMiddleware, async (ctx: Context) => {
  if (process.env.NODE_ENV === 'production') {
    throw new BadRequestError('生产环境不可用');
  }
  const userId = ctx.state.user.userId;
  const { orderNo } = ctx.request.body as { orderNo: string };
  if (!orderNo) {
    throw new BadRequestError('订单号不能为空');
  }
  const result = await MemberService.mockPaymentSuccess(orderNo, userId);
  ctx.body = {
    code: 0,
    message: '支付成功',
    data: result,
  };
});
// 获取订单列表
router.get('/orders', authMiddleware, async (ctx: Context) => {
  const userId = ctx.state.user.userId;
  const { page = 1, pageSize = 10 } = ctx.query as { page?: string; pageSize?: string };
  const result = await MemberService.getOrders(
    userId,
    Number(page) || 1,
    Number(pageSize) || 10
  );
  ctx.body = {
    code: 0,
    message: 'success',
    data: result,
  };

// ============================================
// 文件: server/src/modules/subscription/subscription.controller.ts
// ============================================
```

```
import Router from '@koa/router';
import { Context } from 'koa';
import * as SubscriptionService from './subscription.service';
import { BadRequestError } from '../../middleware/errorHandler';
import { authMiddleware, optionalAuth } from '../../middleware/auth';
import { prisma } from '../../models';
const router = new Router();
// 获取所有套餐列表
router.get('/plans', async (ctx: Context) => {
  const plans = await SubscriptionService.getPlans();
  ctx.body = {
    code: 0,
    message: 'success',
    data: { plans }
  };
});
// 获取单个套餐详情
router.get('/plans/:id', async (ctx: Context) => {
  const planId = parseInt(ctx.params.id);
  const plan = await SubscriptionService.getPlanById(planId);
  ctx.body = {
    code: 0,
    message: 'success',
    data: { plan }
  };
});
// 获取用户订阅信息
router.get('/subscription', authMiddleware, async (ctx: Context) => {
  const userId = parseInt(ctx.state.user.userId);
  const subscription = await SubscriptionService.getUserSubscription(userId);
  ctx.body = {
    code: 0,
    message: 'success',
    data: { subscription }
  };
});
// 获取用户Token余额
router.get('/balance', authMiddleware, async (ctx: Context) => {
  const userId = parseInt(ctx.state.user.userId);
  const balance = await SubscriptionService.getUserTokenBalance(userId);
  ctx.body = {
    code: 0,
    message: 'success',
    data: balance
  };
});
// 获取Token使用记录
router.get('/usage', authMiddleware, async (ctx: Context) => {
  const userId = parseInt(ctx.state.user.userId);
  const { page = '1', pageSize = '20' } = ctx.query as { page?: string; pageSize?: string };
  const result = await SubscriptionService.getTokenUsageList(
    userId,
    Number(page) || 1,
    Number(pageSize) || 20
  );
  ctx.body = {
    code: 0,
    message: 'success',
```

```
    data: result
  };
});
// 获取用户配额（兼容旧接口）
router.get('/quota', authMiddleware, async (ctx: Context) => {
  const userId = parseInt(ctx.state.user.userId);
  const quota = await SubscriptionService.getUserQuota(userId);
  ctx.body = {
    code: 0,
    message: 'success',
    data: quota
  };
});
// 检查配额
router.post('/check-quota', authMiddleware, async (ctx: Context) => {
  const userId = parseInt(ctx.state.user.userId);
  const { tokens } = ctx.request.body as { tokens: number };
  if (!tokens || tokens <= 0) {
    throw new BadRequestError('请提供正确的Token数量');
  }
  const result = await SubscriptionService.checkQuota(userId, tokens);
  ctx.body = {

// ═══════════════════════════════════════
// 文件: server/src/modules/payment/payment.controller.ts
// ═══════════════════════════════════════
import Router from '@koa/router';
import { Context } from 'koa';
import * as PaymentService from './payment.service';
import { BadRequestError } from '../../middleware/errorHandler';
import { authMiddleware } from '../../middleware/auth';
import { prisma } from '../../models';
import { safeParseInt } from '../../utils/safe-parse';
const router = new Router();
// 创建支付订单
router.post('/create', authMiddleware, async (ctx: Context) => {
  const userId = safeParseInt(ctx.state.user.userId);
  const { planId, paymentMethod, period = 'monthly', returnUrl } = ctx.request.body as {
    planId: number;
    paymentMethod: 'alipay' | 'wechat' | 'mock';
    period?: 'monthly' | 'yearly';
    returnUrl?: string;
  };
  if (!planId) {
    throw new BadRequestError('请选择套餐');
  }
  if (!['alipay', 'wechat', 'mock'].includes(paymentMethod)) {
    throw new BadRequestError('请选择支付方式');
  }
  const result = await PaymentService.createPaymentOrder(userId, planId, paymentMethod, period, returnUrl);
  ctx.body = {
    code: 0,
    message: '订单创建成功',
    data: result
  };
});
// 创建 Token 包支付订单
router.post('/token-packs/create', authMiddleware, async (ctx: Context) => {
```

```
  const userId = safeParseInt(ctx.state.user.userId);
  const { quantity, paymentMethod, returnUrl } = ctx.request.body as {
    quantity: number;
    paymentMethod: 'alipay' | 'wechat' | 'mock';
    returnUrl?: string;
  };
  if (!quantity || quantity < 1) {
    throw new BadRequestError('购买数量至少为1');
  }
  if (!['alipay', 'wechat', 'mock'].includes(paymentMethod)) {
    throw new BadRequestError('请选择支付方式');
  }
  const result = await PaymentService.createTokenPackOrder(userId, quantity, paymentMethod, returnUrl);
  ctx.body = {
    code: 0,
    message: '订单创建成功',
    data: result
  };
});
// 模拟支付（仅开发环境）
router.post('/mock', authMiddleware, async (ctx: Context) => {
  if (process.env.NODE_ENV === 'production') {
    throw new BadRequestError('生产环境不可用');
  }
  const userId = safeParseInt(ctx.state.user.userId);
  const { orderNo } = ctx.request.body as { orderNo: string };
  if (!orderNo) {
    throw new BadRequestError('订单号不能为空');
  }
  const result = await PaymentService.mockPaymentSuccess(orderNo, userId);
  ctx.body = {
    code: 0,
    message: result.message,
    data: result
  };
});
// 支付宝异步通知回调
router.post('/alipay/notify', async (ctx: Context) => {
  const params = ctx.request.body as Record<string, string>;
  console.log('[Alipay Notify] 收到异步通知:', params);
  // 验证签名
  const signVerified = PaymentService.verifyAlipaySign(params);
  if (!signVerified) {
    console.error('[Alipay Notify] 签名验证失败');
    ctx.status = 400;
    ctx.body = 'fail';
    return;
  }

// ═══════════════════════════════════════
// 文件: server/src/modules/favorites/favorites.controller.ts
// ═══════════════════════════════════════
import Router from '@koa/router';
import { Context } from 'koa';
import * as FavoritesService from './favorites.service';
import { BadRequestError } from '../../middleware/errorHandler';
import { optionalAuth } from '../../middleware/auth';
// 测试用户ID（开发环境使用）
```

```
const TEST_USER_ID = '1';
const router = new Router();
// 获取收藏列表（兼容前端 /api/favorites/list）
router.get('/list', optionalAuth, async (ctx: Context) => {
  // 开发环境使用测试用户ID
  const userId = ctx.state.user?.userId || TEST_USER_ID;
  const favorites = await FavoritesService.getFavorites(userId);
  ctx.body = {
    code: 0,
    message: 'success',
    data: favorites,
  };
});
// 获取收藏列表
router.get('/', optionalAuth, async (ctx: Context) => {
  // 开发环境使用测试用户ID
  const userId = ctx.state.user?.userId || TEST_USER_ID;
  const favorites = await FavoritesService.getFavorites(userId);
  ctx.body = {
    code: 0,
    message: 'success',
    data: favorites,
  };
});
// 添加收藏
router.post('/', optionalAuth, async (ctx: Context) => {
  // 开发环境使用测试用户ID
  const userId = ctx.state.user?.userId || TEST_USER_ID;
  const body = ctx.request.body as { audioId: number };
  const { audioId } = body;
  if (!audioId) {
    throw new BadRequestError('音频ID不能为空');
  }
  const favorite = await FavoritesService.addFavorite(userId, audioId);
  ctx.body = {
    code: 0,
    message: '收藏成功',
    data: favorite,
  };
});
// 取消收藏
router.delete('/:audioId', optionalAuth, async (ctx: Context) => {
  // 开发环境使用测试用户ID
  const userId = ctx.state.user?.userId || TEST_USER_ID;
  const audioId = parseInt(ctx.params.audioId as string);
  await FavoritesService.removeFavorite(userId, audioId);
  ctx.body = {
    code: 0,
    message: '已取消收藏',
  };
});
// 检查是否已收藏
router.get('/check/:audioId', optionalAuth, async (ctx: Context) => {
  // 开发环境使用测试用户ID
  const userId = ctx.state.user?.userId || TEST_USER_ID;
  const audioId = parseInt(ctx.params.audioId as string);
  const isFavorited = await FavoritesService.isFavorited(userId, audioId);
  ctx.body = {
```

```
    code: 0,
    message: 'success',
    data: { isFavorited },
  };
});
export default router;

// ============================================
// 文件: server/src/modules/history/history.controller.ts
// ============================================
import Router from '@koa/router';
import { Context } from 'koa';
import { optionalAuth } from '../../middleware/auth';
import { prisma } from '../../models';
const TEST_USER_ID = '1';
const router = new Router();
// 获取音频生成历史列表（兼容前端 /api/history/list）
router.get('/list', optionalAuth, async (ctx: Context) => {
  // 开发环境使用测试用户ID
  const userId = ctx.state.user?.userId || TEST_USER_ID;
  const page = parseInt(ctx.query.page as string) || 1;
  const pageSize = parseInt(ctx.query.pageSize as string) || 20;
  const startDate = ctx.query.startDate as string;
  const where: any = {};
  if (startDate) {
    where.createdAt = { gte: new Date(startDate) };
  }
  const [records, total] = await Promise.all([
    prisma.audioRecord.findMany({
      where,
      orderBy: { createdAt: 'desc' },
      skip: (page - 1) * pageSize,
      take: pageSize,
    }),
    prisma.audioRecord.count({ where }),
  ]);
  const list = records.map((r) => ({
    _id: r.audioId,
    id: r.audioId,
    title: r.title,
    text: r.text || '',
    summary: '',
    tags: [],
    audioUrl: r.audioUrl || '',
    audioDuration: r.audioDuration,
    audioSize: r.audioSize,
    wordCount: r.wordCount,
    voiceId: r.voiceId,
    voiceParams: r.voiceParams ? JSON.parse(r.voiceParams) : { speed: 1, pitch: 0, volume: 50 },
    status: r.status,
    isFavorite: false,
    bookId: r.bookId,
    createdAt: r.createdAt.toISOString(),
    updatedAt: r.updatedAt.toISOString(),
  }));
  ctx.body = {
    code: 0,
    message: 'success',
```

```
    data: {
      list,
      total,
      page,
      pageSize,
      totalPages: Math.ceil(total / pageSize),
    },
  };
});
// 获取音频生成历史列表
router.get('/', optionalAuth, async (ctx: Context) => {
  // 开发环境使用测试用户ID
  const userId = ctx.state.user?.userId || TEST_USER_ID;
  const page = parseInt(ctx.query.page as string) || 1;
  const pageSize = parseInt(ctx.query.pageSize as string) || 20;
  const startDate = ctx.query.startDate as string;
  const where: any = {};
  if (startDate) {
    where.createdAt = { gte: new Date(startDate) };
  }
  const [records, total] = await Promise.all([
    prisma.audioRecord.findMany({
      where,
      orderBy: { createdAt: 'desc' },
      skip: (page - 1) * pageSize,
      take: pageSize,
    }),
    prisma.audioRecord.count({ where }),
  ]);
  const list = records.map((r) => ({
    _id: r.audioId,
    id: r.audioId,

// ============================================
// 文件: server/src/modules/search/search.controller.ts
// ============================================
import Router from '@koa/router';
import { searchService } from './search.service';
const router = new Router();
/**
 * 搜索音频
 * GET /api/search?q=关键词
 */
router.get('/', async (ctx) => {
  try {
    const { q, limit } = ctx.query as { q?: string; limit?: string };
    if (!q || q.trim() === '') {
      ctx.body = {
        code: 0,
        message: 'success',
        data: [],
      };
      return;
    }
    const results = await searchService.searchAudio(q, parseInt(limit || '20'));
    ctx.body = {
      code: 0,
      message: 'success',
```

```
      data: results,
    };
  } catch (error: any) {
    ctx.body = {
      code: 400,
      message: error.message || '搜索失败',
    };
  }
});
/**
 * 获取热门搜索词
 * GET /api/search/hot
 */
router.get('/hot', async (ctx) => {
  try {
    const { limit } = ctx.query as { limit?: string };
    const hotSearches = await searchService.getHotSearches(parseInt(limit || '10'));
    ctx.body = {
      code: 0,
      message: 'success',
      data: hotSearches,
    };
  } catch (error: any) {
    ctx.body = {
      code: 400,
      message: error.message || '获取热门搜索失败',
    };
  }
});
/**
 * 获取搜索历史
 * GET /api/search/history?userId=1
 */
router.get('/history', async (ctx) => {
  try {
    const { userId, limit } = ctx.query as { userId?: string; limit?: string };
    const userIdNum = parseInt(userId || '0') || 0;
    const history = await searchService.getSearchHistory(userIdNum, parseInt(limit || '5'));
    ctx.body = {
      code: 0,
      message: 'success',
      data: history,
    };
  } catch (error: any) {
    ctx.body = {
      code: 400,
      message: error.message || '获取搜索历史失败',
    };
  }
});
/**
 * 保存搜索历史
 * POST /api/search/history
 */
router.post('/history', async (ctx) => {
  try {
    const { userId, keyword } = ctx.request.body as { userId?: number; keyword?: string };
    if (!keyword || keyword.trim() === '') {
```

```
// ═══════════════════════════════════════
// 文件: server/src/services/oss.service.ts
// ═══════════════════════════════════════
import OSS from 'ali-oss';
import path from 'path';
import fs from 'fs';
interface OSSConfig {
  region: string;
  accessKeyId: string;
  accessKeySecret: string;
  bucket: string;
  endpoint?: string;
}
class OSSService {
  private client: OSS;
  private bucket: string;
  private cdnDomain?: string;
  constructor() {
    const config: OSSConfig = {
      region: process.env.OSS_REGION || 'oss-cn-hangzhou',
      accessKeyId: process.env.OSS_ACCESS_KEY_ID || '',
      accessKeySecret: process.env.OSS_ACCESS_KEY_SECRET || '',
      bucket: process.env.OSS_BUCKET_NAME || '',
      endpoint: process.env.OSS_ENDPOINT || 'oss-cn-hangzhou.aliyuncs.com',
    };
    this.client = new OSS(config);
    this.bucket = config.bucket;
    this.cdnDomain = process.env.OSS_CDN_DOMAIN;
  }
  /**
   * 上传文件到 OSS
   * @param localPath 本地文件路径
   * @param objectKey OSS 对象键（路径）
   * @returns OSS 文件 URL
   */
  async uploadFile(localPath: string, objectKey: string): Promise<string> {
    try {
      // 确保 objectKey 格式正确
      const normalizedKey = objectKey.replace(/\\/g, '/').replace(/^\/+/, '');
      const result = await this.client.put(normalizedKey, localPath, {
        headers: {
          'Content-Type': this.getContentType(normalizedKey),
        },
      });
      console.log(`[OSS] 文件上传成功: ${normalizedKey}`);
      // 返回文件 URL
      return this.getFileUrl(normalizedKey);
    } catch (error) {
      console.error('[OSS] 文件上传失败:', error);
      throw new Error(`OSS 上传失败: ${(error as Error).message}`);
    }
  }
  /**
   * 上传 Buffer 到 OSS
   * @param buffer 文件 Buffer
   * @param objectKey OSS 对象键
   * @param contentType 内容类型
```

```
   * @returns OSS 文件 URL
   */
  async uploadBuffer(buffer: Buffer, objectKey: string, contentType?: string): Promise<string> {
    try {
      const normalizedKey = objectKey.replace(/\\/g, '/').replace(/^\/+/, '');
      const result = await this.client.put(normalizedKey, buffer, {
        headers: {
          'Content-Type': contentType || this.getContentType(normalizedKey),
        },
      });
      console.log(`[OSS] Buffer 上传成功: ${normalizedKey}`);
      return this.getFileUrl(normalizedKey);
    } catch (error) {
      console.error('[OSS] Buffer 上传失败:', error);
      throw new Error(`OSS Buffer 上传失败: ${(error as Error).message}`);
    }
  }
  /**
   * 上传音频文件
   * @param localPath 本地音频文件路径
   * @param audioId 音频 ID
   * @returns OSS 音频 URL
   */
  async uploadAudio(localPath: string, audioId: string): Promise<string> {
    const objectKey = `audio/${audioId}/${path.basename(localPath)}`;
    return this.uploadFile(localPath, objectKey);

// ═══════════════════════════════════════
// 文件: server/src/services/queue.service.ts
// ═══════════════════════════════════════
/**
 * 队列服务
 *
 * 【架构职责】
 * 队列只负责：排队 + 并发控制
 *
 * 【职责边界】
 * -    做：任务排队、并发限制、任务分发
 * -    不做：失败重试、超时管理、业务逻辑
 *
 * 【设计原则】
 * - 单一职责：队列只管排队，不管其他
 * - 失败重试 → 容错层（fault-tolerance.ts）
 * - 超时管理 → AI服务层（llm.service.ts）
 * - 业务逻辑 → 领域层（langgraph-generator.ts）
 */
import Queue from 'bull';
import { redisService } from './redis.service';
import { memoryQueue } from './memory-queue';
// 任务队列类型
export enum QueueType {
  AUDIO_GENERATION = 'audio:generation',
  VIDEO_GENERATION = 'video:generation',
  BOOK_GENERATION = 'book:generation',
  EMAIL_SEND = 'email:send',
}
// 任务状态
export enum TaskStatus {
```

```
  WAITING = 'waiting',
  ACTIVE = 'active',
  COMPLETED = 'completed',
  FAILED = 'failed',
  DELAYED = 'delayed',
}
// 任务数据接口
interface TaskData {
  userId?: number;
  [key: string]: any;
}
// 任务进度回调
export type ProgressCallback = (progress: number, data?: any) => void;
class QueueService {
  private queues: Map<string, Queue.Queue> = new Map();
  private progressCallbacks: Map<string, ProgressCallback> = new Map();
  private queueAvailable: boolean = true;
  constructor() {
    // 如果 Redis 不可用，记录警告但不阻止程序运行
    if (!redisService.isAvailable()) {
      console.warn('[Queue] Redis 不可用，任务队列将无法正常工作');
      this.queueAvailable = false;
    }
  }
  /**
   * 检查队列是否可用
   */
  isQueueAvailable(): boolean {
    return this.queueAvailable && redisService.isAvailable();
  }
  /**
   * 获取或创建队列
   * @param queueName 队列名称
   */
  private getQueue(queueName: string): Queue.Queue | null {
    if (!this.isQueueAvailable()) {
      return null;
    }
    if (!this.queues.has(queueName)) {
      try {
        const queue = new Queue(queueName, {
          redis: {
            host: process.env.REDIS_HOST || 'localhost',
            port: parseInt(process.env.REDIS_PORT || '6379'),
            password: process.env.REDIS_PASSWORD || undefined,
            db: parseInt(process.env.REDIS_DB || '0'),
            maxRetriesPerRequest: null,
          },
          defaultJobOptions: {
            removeOnComplete: 100,
            removeOnFail: 50,
          },

// ============================================
// 文件: server/src/services/websocket.service.ts
// ============================================
/**
 * WebSocket 服务
```

```
 * 用于推送音频/视频生成完成事件
 */
import { Server as HttpServer } from 'http';
import WebSocket, { WebSocketServer } from 'ws';
const wss = new WebSocketServer({ noServer: true });
// 客户端连接管理
const clients = new Map<string, WebSocket>();
// ============ 客户端管理 ============
/**
 * 注册客户端连接
 */
export function addClient(clientId: string, ws: WebSocket) {
  clients.set(clientId, ws);
  console.log(`[WS] 客户端连接: ${clientId}, 当前在线: ${clients.size}`);
}
/**
 * 移除客户端连接
 */
export function removeClient(clientId: string) {
  clients.delete(clientId);
  console.log(`[WS] 客户端断开: ${clientId}, 当前在线: ${clients.size}`);
}
/**
 * 通过 clientId 发送消息
 */
export function sendToClient(clientId: string, event: string, data: any): boolean {
  const ws = clients.get(clientId);
  if (!ws || ws.readyState !== WebSocket.OPEN) {
    return false;
  }
  try {
    ws.send(JSON.stringify({ event, data }));
    return true;
  } catch (error) {
    console.error(`[WS] 发送消息失败: ${clientId}`, error);
    return false;
  }
}
/**
 * 广播消息到所有客户端
 */
export function broadcast(event: string, data: any) {
  const message = JSON.stringify({ event, data });
  clients.forEach((ws, clientId) => {
    if (ws.readyState === WebSocket.OPEN) {
      try {
        ws.send(message);
      } catch (error) {
        console.error(`[WS] 广播失败: ${clientId}`, error);
      }
    }
  });
}
// ============ 事件推送 ============
/**
 * 推送音频生成完成事件
 */
export function pushAudioGenerationComplete(bookId: string, chapterId: number, status: 'completed' | 'failed') {
```

```
  const event = 'audio_generation_complete';
  const data = { bookId, chapterId, status };
  console.log(`[WS] 推送 ${event}:`, data);
  // 广播给所有客户端（前端可根据 bookId 过滤）
  broadcast(event, data);
}
/**
 * 推送视频生成完成事件
 */
export function pushVideoGenerationComplete(bookId: string, chapterId: number, status: 'completed' | 'failed') {
  const event = 'video_generation_complete';
  const data = { bookId, chapterId, status };
  console.log(`[WS] 推送 ${event}:`, data);
  broadcast(event, data);
}
/**
 * 推送批量生成进度
 */
export function pushBatchGenerationProgress(taskId: string, step: string, progress: number) {
  const event = 'batch_generation_progress';


# 统计信息

# 前端文件数: 30
# 后端文件数: 30
# 文档总行数: 4751
// ============ 系统常量定义 ============
/** 音频格式常量 */
const AUDIO_FORMATS = {
  MP3: 'mp3',
  WAV: 'wav',
  OGG: 'ogg',
  M4A: 'm4a',
  AAC: 'aac',
} as const;
export type AudioFormat = typeof AUDIO_FORMATS[keyof typeof AUDIO_FORMATS];
/** 生成唯一任务ID */
export function generateTaskId(prefix: string = 'task'): string {
  const ts = Date.now().toString(36);
  const rand = Math.random().toString(36).substring(2, 8);
  return `${prefix}_${ts}_${rand}`;
}
/** 系统默认配置 */
export const DEFAULT_CONFIG = {
  maxTextLength: 5000,
  maxFileSize: 10 * 1024 * 1024, // 10MB
  supportedLanguages: ['zh-CN', 'en-US', 'ja-JP', 'ko-KR'],
  apiTimeout: 30000,
  retryLimit: 3,
} as const;
```