TTSmate V1 API接口文档

1 概述

1.1 文档说明

本文档详细描述TTSmate V1系统中各个模块的API接口规范，包括内部模块接口和外部服务接口，为开发人员提供完整的接口调用指南。

1.2 接口分类

内部模块接口：各功能模块间的接口定义

外部服务接口：与TTS服务器、DeepSeek AI等外部服务的接口

系统集成接口：与Windows系统和第三方音频软件的接口

1.3 接口规范

所有接口采用异步设计，支持错误处理和超时控制

使用Result类型进行错误传播

支持配置化的重试机制

提供详细的错误信息和状态码

2 TTS客户端模块接口

2.1 TTSClient结构体

```rust

pub struct TTSClient {

base\_url: String,

client: reqwest::Client,

config: TTSConfig,

cache: Arc<Mutex<TTSCache>>,

}

pub struct TTSConfig {

pub timeout: Duration,

pub retry\_count: u32,

pub cache\_enabled: bool,

pub default\_voice: String,

pub audio\_format: AudioFormat,

}

pub struct TTSCache {

max\_size: usize,

entries: HashMap<String, CacheEntry>,

}

```

2.2 核心接口方法

2.2.1 语音合成接口

```rust

impl TTSClient {

/// 文本转语音合成

///

/// # 参数

/// \* `text` - 要合成的文本内容

/// \* `options` - 合成选项（可选）

///

/// # 返回值

/// \* `Ok(AudioData)` - 成功时返回音频数据

/// \* `Err(TTSError)` - 失败时返回错误信息

pub async fn synthesize(

&self,

text: &str,

options: Option<SynthesizeOptions>

) -> Result<AudioData, TTSError>;

/// 批量文本合成

pub async fn synthesize\_batch(

&self,

texts: Vec<&str>,

options: Option<SynthesizeOptions>

) -> Result<Vec<AudioData>, TTSError>;

}

pub struct SynthesizeOptions {

pub voice: Option<String>,

pub speed: Option<f32>,

pub pitch: Option<f32>,

pub volume: Option<f32>,

pub format: Option<AudioFormat>,

}

pub struct AudioData {

pub data: Vec<u8>,

pub format: AudioFormat,

pub duration: Duration,

pub sample\_rate: u32,

}

```

2.2.2 语音配置接口

```rust

impl TTSClient {

/// 获取可用语音列表

pub async fn get\_voices(&self) -> Result<Vec<Voice>, TTSError>;

/// 设置默认语音

pub fn set\_default\_voice(&mut self, voice: &str);

/// 获取服务器状态

pub async fn get\_server\_status(&self) -> Result<ServerStatus, TTSError>;

/// 测试连接

pub async fn test\_connection(&self) -> Result<bool, TTSError>;

}

pub struct Voice {

pub id: String,

pub name: String,

pub language: String,

pub gender: Gender,

pub sample\_rate: u32,

}

pub enum Gender {

Male,

Female,

Neutral,

}

pub struct ServerStatus {

pub online: bool,

pub version: String,

pub load: f32,

pub available\_voices: usize,

}

```

2.3 错误处理

```rust

#[derive(Debug, thiserror::Error)]

pub enum TTSError {

#[error("网络连接错误: {0}")]

NetworkError(#[from] reqwest::Error),

#[error("服务器错误: {status\_code} - {message}")]

ServerError { status\_code: u16, message: String },

#[error("音频格式错误: {0}")]

AudioFormatError(String),

#[error("配置错误: {0}")]

ConfigError(String),

#[error("缓存错误: {0}")]

CacheError(String),

#[error("超时错误")]

TimeoutError,

}

```

3 AI文案生成模块接口

3.1 AIContentGenerator结构体

```rust

pub struct AIContentGenerator {

api\_key: String,

client: reqwest::Client,

config: AIConfig,

template\_manager: TemplateManager,

history: ContentHistory,

}

pub struct AIConfig {

pub base\_url: String,

pub model: String,

pub max\_tokens: u32,

pub temperature: f32,

pub timeout: Duration,

}

```

3.2 内容生成接口

```rust

impl AIContentGenerator {

/// 生成文案内容

///

/// # 参数

/// \* `prompt` - 生成提示词

/// \* `content\_type` - 内容类型

/// \* `options` - 生成选项

///

/// # 返回值

/// \* `Ok(GeneratedContent)` - 成功时返回生成的内容

/// \* `Err(AIError)` - 失败时返回错误信息

pub async fn generate\_content(

&self,

prompt: &str,

content\_type: ContentType,

options: Option<GenerationOptions>

) -> Result<GeneratedContent, AIError>;

/// 优化现有文案

pub async fn optimize\_content(

&self,

content: &str,

optimization\_type: OptimizationType

) -> Result<String, AIError>;

/// 批量生成文案

pub async fn generate\_batch(

&self,

prompts: Vec<GenerationRequest>

) -> Result<Vec<GeneratedContent>, AIError>;

}

pub enum ContentType {

Chat, // 聊天对话

Meeting, // 会议发言

GameNarration, // 游戏旁白

Announcement, // 公告通知

Custom(String), // 自定义类型

}

pub struct GenerationOptions {

pub max\_length: Option<u32>,

pub style: Option<String>,

pub tone: Option<Tone>,

pub language: Option<String>,

}

pub enum Tone {

Formal, // 正式

Casual, // 随意

Humorous, // 幽默

Serious, // 严肃

Friendly, // 友好

}

pub struct GeneratedContent {

pub id: String,

pub content: String,

pub content\_type: ContentType,

pub created\_at: DateTime<Utc>,

pub metadata: ContentMetadata,

}

pub struct ContentMetadata {

pub word\_count: u32,

pub estimated\_duration: Duration,

pub quality\_score: f32,

pub tags: Vec<String>,

}

```

3.3 模板管理接口

```rust

impl AIContentGenerator {

/// 保存内容模板

pub fn save\_template(&mut self, template: ContentTemplate) -> Result<(), AIError>;

/// 获取模板列表

pub fn get\_templates(&self, category: Option<&str>) -> Vec<ContentTemplate>;

/// 删除模板

pub fn delete\_template(&mut self, template\_id: &str) -> Result<(), AIError>;

/// 使用模板生成内容

pub async fn generate\_from\_template(

&self,

template\_id: &str,

variables: HashMap<String, String>

) -> Result<GeneratedContent, AIError>;

}

pub struct ContentTemplate {

pub id: String,

pub name: String,

pub category: String,

pub template: String,

pub variables: Vec<TemplateVariable>,

pub created\_at: DateTime<Utc>,

}

pub struct TemplateVariable {

pub name: String,

pub description: String,

pub default\_value: Option<String>,

pub required: bool,

}

```

4 音效板模块接口

4.1 SoundBoard结构体

```rust

pub struct SoundBoard {

sounds: HashMap<String, SoundEffect>,

categories: HashMap<String, SoundCategory>,

keybindings: HashMap<KeyCode, String>,

mixer: AudioMixer,

config: SoundBoardConfig,

}

pub struct SoundBoardConfig {

pub master\_volume: f32,

pub fade\_duration: Duration,

pub max\_concurrent\_sounds: usize,

pub audio\_format: AudioFormat,

}

```

4.2 音效管理接口

```rust

impl SoundBoard {

/// 添加音效文件

///

/// # 参数

/// \* `file\_path` - 音效文件路径

/// \* `name` - 音效名称

/// \* `category` - 音效分类

///

/// # 返回值

/// \* `Ok(String)` - 成功时返回音效ID

/// \* `Err(SoundBoardError)` - 失败时返回错误信息

pub fn add\_sound(

&mut self,

file\_path: &Path,

name: &str,

category: &str

) -> Result<String, SoundBoardError>;

/// 播放音效

pub async fn play\_sound(&self, sound\_id: &str) -> Result<(), SoundBoardError>;

/// 停止音效

pub fn stop\_sound(&self, sound\_id: &str) -> Result<(), SoundBoardError>;

/// 停止所有音效

pub fn stop\_all\_sounds(&self);

/// 获取音效列表

pub fn get\_sounds(&self, category: Option<&str>) -> Vec<&SoundEffect>;

/// 删除音效

pub fn remove\_sound(&mut self, sound\_id: &str) -> Result<(), SoundBoardError>;

}

pub struct SoundEffect {

pub id: String,

pub name: String,

pub file\_path: PathBuf,

pub category: String,

pub volume: f32,

pub duration: Duration,

pub format: AudioFormat,

pub created\_at: DateTime<Utc>,

}

pub struct SoundCategory {

pub id: String,

pub name: String,

pub color: String,

pub icon: Option<String>,

}

```

4.3 快捷键管理接口

```rust

impl SoundBoard {

/// 绑定快捷键

pub fn bind\_key(&mut self, key: KeyCode, sound\_id: &str) -> Result<(), SoundBoardError>;

/// 解除快捷键绑定

pub fn unbind\_key(&mut self, key: KeyCode) -> Result<(), SoundBoardError>;

/// 获取快捷键绑定

pub fn get\_keybindings(&self) -> &HashMap<KeyCode, String>;

/// 处理按键事件

pub async fn handle\_key\_event(&self, key: KeyCode) -> Result<(), SoundBoardError>;

}

#[derive(Debug, Clone, Copy, PartialEq, Eq, Hash)]

pub enum KeyCode {

F1, F2, F3, F4, F5, F6, F7, F8, F9, F10, F11, F12,

Num1, Num2, Num3, Num4, Num5, Num6, Num7, Num8, Num9, Num0,

A, B, C, D, E, F, G, H, I, J, K, L, M, N, O, P, Q, R, S, T, U, V, W, X, Y, Z,

Space, Enter, Escape, Tab, Backspace,

// 组合键

CtrlA, CtrlB, CtrlC, // ... 其他组合键

}

```

5 虚拟声卡集成模块接口

5.1 AudioRouter结构体

```rust

pub struct AudioRouter {

input\_devices: Vec<AudioDevice>,

output\_devices: Vec<AudioDevice>,

virtual\_devices: Vec<VirtualDevice>,

mixer: AudioMixer,

config: AudioConfig,

}

pub struct AudioConfig {

pub sample\_rate: u32,

pub buffer\_size: u32,

pub channels: u16,

pub bit\_depth: u16,

pub latency: Duration,

}

```

5.2 音频设备管理接口

```rust

impl AudioRouter {

/// 枚举音频设备

pub fn enumerate\_devices(&mut self) -> Result<(), AudioError>;

/// 获取输入设备列表

pub fn get\_input\_devices(&self) -> &[AudioDevice];

/// 获取输出设备列表

pub fn get\_output\_devices(&self) -> &[AudioDevice];

/// 设置默认输入设备

pub fn set\_default\_input(&mut self, device\_id: &str) -> Result<(), AudioError>;

/// 设置默认输出设备

pub fn set\_default\_output(&mut self, device\_id: &str) -> Result<(), AudioError>;

/// 创建虚拟音频设备

pub fn create\_virtual\_device(

&mut self,

name: &str,

device\_type: VirtualDeviceType

) -> Result<String, AudioError>;

}

pub struct AudioDevice {

pub id: String,

pub name: String,

pub device\_type: DeviceType,

pub channels: u16,

pub sample\_rate: u32,

pub is\_default: bool,

pub is\_available: bool,

}

pub enum DeviceType {

Input,

Output,

InputOutput,

}

pub enum VirtualDeviceType {

Cable, // 虚拟音频线缆

Mixer, // 虚拟混音器

Loopback, // 回环设备

}

```

5.3 音频流处理接口

```rust

impl AudioRouter {

/// 创建音频流

pub fn create\_stream(

&self,

input\_device: &str,

output\_device: &str,

config: StreamConfig

) -> Result<AudioStream, AudioError>;

/// 启动音频流

pub fn start\_stream(&self, stream\_id: &str) -> Result<(), AudioError>;

/// 停止音频流

pub fn stop\_stream(&self, stream\_id: &str) -> Result<(), AudioError>;

/// 设置音频流音量

pub fn set\_stream\_volume(&self, stream\_id: &str, volume: f32) -> Result<(), AudioError>;

/// 混合多个音频流

pub fn mix\_streams(&self, stream\_ids: Vec<&str>) -> Result<String, AudioError>;

}

pub struct StreamConfig {

pub sample\_rate: u32,

pub channels: u16,

pub buffer\_size: u32,

pub volume: f32,

pub effects: Vec<AudioEffect>,

}

pub struct AudioStream {

pub id: String,

pub input\_device: String,

pub output\_device: String,

pub config: StreamConfig,

pub status: StreamStatus,

}

pub enum StreamStatus {

Created,

Running,

Paused,

Stopped,

Error(String),

}

```

6 配置管理模块接口

6.1 ConfigManager结构体

```rust

pub struct ConfigManager {

config\_path: PathBuf,

config: AppConfig,

watchers: Vec<ConfigWatcher>,

}

pub struct AppConfig {

pub tts: TTSConfig,

pub ai: AIConfig,

pub audio: AudioConfig,

pub ui: UIConfig,

pub keybindings: KeyBindingConfig,

}

```

6.2 配置操作接口

```rust

impl ConfigManager {

/// 加载配置文件

pub fn load\_config(&mut self) -> Result<(), ConfigError>;

/// 保存配置文件

pub fn save\_config(&self) -> Result<(), ConfigError>;

/// 获取配置项

pub fn get<T>(&self, key: &str) -> Result<T, ConfigError>

where

T: serde::de::DeserializeOwned;

/// 设置配置项

pub fn set<T>(&mut self, key: &str, value: T) -> Result<(), ConfigError>

where

T: serde::Serialize;

/// 重置为默认配置

pub fn reset\_to\_default(&mut self) -> Result<(), ConfigError>;

/// 导出配置

pub fn export\_config(&self, path: &Path) -> Result<(), ConfigError>;

/// 导入配置

pub fn import\_config(&mut self, path: &Path) -> Result<(), ConfigError>;

}

```

7 外部服务接口

7.1 TTS服务器API接口

基于Gradio API的接口调用：

TTS服务器使用Gradio框架，支持三种调用方式：cURL、JavaScript和Python。

推荐使用Python gradio\_client库进行调用。

主要API端点：

1. 语音选择API (/on\_voice\_change)：

```python

from gradio\_client import Client

client = Client("http://192.168.11.153:8080/")

result = client.predict(

vocie\_selection="Default", # 可选: Default, Timbre1-9

api\_name="/on\_voice\_change"

)

# 返回: 音频种子数值 (float)

```

2. 音频种子生成API (/generate\_seed)：

```python

from gradio\_client import Client

client = Client("http://192.168.11.153:8080/")

result = client.predict(

api\_name="/generate\_seed"

)

# 返回: 音频种子数值 (float)

```

3. 文本种子生成API (/generate\_seed\_1)：

```python

from gradio\_client import Client

client = Client("http://192.168.11.153:8080/")

result = client.predict(

api\_name="/generate\_seed\_1"

)

# 返回: 文本种子数值 (float)

```

4. 音频种子变更API (/on\_audio\_seed\_change)：

```python

from gradio\_client import Client

client = Client("http://192.168.11.153:8080/")

result = client.predict(

audio\_seed\_input=2, # 音频种子数值

api\_name="/on\_audio\_seed\_change"

)

# 返回: 说话人嵌入字符串 (str)

```

5. 样本音频上传API (/on\_upload\_sample\_audio)：

```python

from gradio\_client import Client, handle\_file

client = Client("http://192.168.11.153:8080/")

result = client.predict(

sample\_audio\_input=handle\_file('path/to/audio.wav'),

api\_name="/on\_upload\_sample\_audio"

)

# 返回: 处理结果字符串 (str)

```

6. DVAE系数配置API (/reload\_chat)：

```python

from gradio\_client import Client

client = Client("http://192.168.11.153:8080/")

result = client.predict(

coef="Hello!!", # DVAE系数字符串

api\_name="/reload\_chat"

)

# 返回: 配置结果字符串 (str)

```

注意：当前文档显示的是配置和预处理API，主要的文本转语音生成API可能需要进一步确认。

7.2 DeepSeek API接口

基于HTTPS协议的API调用：

```

POST https://api.deepseek.com/v1/chat/completions

Authorization: Bearer YOUR\_API\_KEY

Content-Type: application/json

{

"model": "deepseek-chat",

"messages": [

{

"role": "user",

"content": "生成一段游戏旁白"

}

],

"max\_tokens": 1000,

"temperature": 0.7

}

Response:

{

"choices": [

{

"message": {

"content": "生成的文案内容"

}

}

]

}

```

8 错误处理规范

8.1 统一错误类型

```rust

#[derive(Debug, thiserror::Error)]

pub enum AppError {

#[error("TTS错误: {0}")]

TTS(#[from] TTSError),

#[error("AI服务错误: {0}")]

AI(#[from] AIError),

#[error("音频错误: {0}")]

Audio(#[from] AudioError),

#[error("配置错误: {0}")]

Config(#[from] ConfigError),

#[error("网络错误: {0}")]

Network(#[from] reqwest::Error),

#[error("IO错误: {0}")]

IO(#[from] std::io::Error),

}

```

8.2 错误处理策略

- 所有异步操作使用Result类型

- 提供详细的错误信息和上下文

- 支持错误链追踪

- 实现自动重试机制

- 提供用户友好的错误提示

9 接口使用示例

9.1 TTS语音合成示例

```rust

use ttsmate::tts::TTSClient;

#[tokio::main]

async fn main() -> Result<(), Box<dyn std::error::Error>> {

let mut client = TTSClient::new("http://192.168.11.153:8080")?;

let audio = client.synthesize("你好，世界！", None).await?;

// 播放音频

client.play\_audio(&audio).await?;

Ok(())

}

```

9.2 AI文案生成示例

```rust

use ttsmate::ai::AIContentGenerator;

#[tokio::main]

async fn main() -> Result<(), Box<dyn std::error::Error>> {

let generator = AIContentGenerator::new("your\_api\_key")?;

let content = generator.generate\_content(

"为狼人杀游戏生成开场白",

ContentType::GameNarration,

None

).await?;

println!("生成的内容: {}", content.content);

Ok(())

}

```

9.3 音效板使用示例

```rust

use ttsmate::soundboard::SoundBoard;

#[tokio::main]

async fn main() -> Result<(), Box<dyn std::error::Error>> {

let mut soundboard = SoundBoard::new()?;

// 添加音效

let sound\_id = soundboard.add\_sound(

Path::new("sounds/applause.wav"),

"掌声",

"反应"

)?;

// 绑定快捷键

soundboard.bind\_key(KeyCode::F1, &sound\_id)?;

// 播放音效

soundboard.play\_sound(&sound\_id).await?;

Ok(())

}

```

10 版本控制和兼容性

10.1 API版本管理

- 使用语义化版本控制

- 向后兼容性保证

- 废弃API的迁移指南

- 版本升级通知机制

10.2 接口变更记录

- 详细的变更日志

- 破坏性变更警告

- 迁移工具和脚本

- 兼容性测试套件

TTSmate API 使用说明

本文档概述了 TTSmate 应用程序所依赖的外部 API。

一、 DeepSeek API (用于 AI 文本生成)

功能说明:

此 API 接收用户提供的系统提示和主题，生成一段相关的文本。

端点 (Endpoint):

POST https://api.deepseek.com/chat/completions

认证方式:

Bearer Token。需要将 DeepSeek API Key 填入 HTTP 请求的 Authorization 头中。

请求格式 (JSON):

model: "deepseek-chat"

messages: 一个包含两个消息对象的数组

第一个对象:

role: "system"

content: 系统提示，用于指导 AI 的行为（例如："你是一个为TTS语音合成生成文本的助手..."）。

第二个对象:

role: "user"

content: 用户输入的具体话题或提示词。

响应格式 (JSON):

响应体包含一个 choices 数组，其中第一个元素的 message.content 字段即为 AI 生成的文本。

二、 百度语音合成 (TTS) API

此功能分为两步：获取访问令牌和请求语音合成。

步骤 1: 获取 Access Token

功能说明:

使用 API Key 和 Secret Key 获取一个临时的访问令牌 (Access Token)，用于后续的 API 调用认证。

端点 (Endpoint):

POST https://aip.baidubce.com/oauth/2.0/token

请求格式 (x-www-form-urlencoded):

grant\_type: "client\_credentials"

client\_id: 你的百度 API Key。

client\_secret: 你的百度 Secret Key。

响应格式 (JSON):

响应体中的 access\_token 字段即为所需令牌。

步骤 2: 文本转语音

功能说明:

将文本转换为 MP3 格式的音频数据。

端点 (Endpoint):

POST https://tsn.baidu.com/text2audio

请求格式 (x-www-form-urlencoded):

tex: 需要被转换为语音的文本。

tok: 从步骤 1 中获取的 Access Token。

cuid: 客户端唯一标识符，可为任意字符串 (例如: "ttsmate\_rust\_client")。

ctp: 客户端类型，固定为 "1"。

lan: 语言，固定为 "zh" (中文)。

spd: 语速，一个 0 到 15 之间的整数。

pit: 音调，一个 0 到 15 之间的整数。

vol: 音量，一个 0 到 15 之间的整数。

per: 发音人代码，一个代表不同声音的整数 (例如: 5118)。

aue: 音频编码格式，固定为 "3"，代表 MP3 格式。

响应格式:

成功时，响应体为原始的 MP3 音频二进制数据。

失败时，响应体为一个包含错误信息的 JSON 对象。