AI角色扮演网站 - 系统架构设计文档

1. 系统概览

1.1 项目简介

AI角色扮演网站是一个基于Flask的Web应用，提供用户与AI角色的实时聊天和语音通话功能。系统采用分层架构模式，支持多种AI服务提供商，具备完整的角色管理和WebSocket实时通信能力。

1.2 核心功能

- **角色对话系统**: 支持与多个预设AI角色进行个性化对话
- **实时语音通话**: 基于WebSocket的低延迟语音交互
- **角色管理**: 支持自定义角色的创建、编辑和删除
- **多模态交互**: 文字输入、语音识别、语音合成
- **流式响应**: 实时流式显示AI回复内容

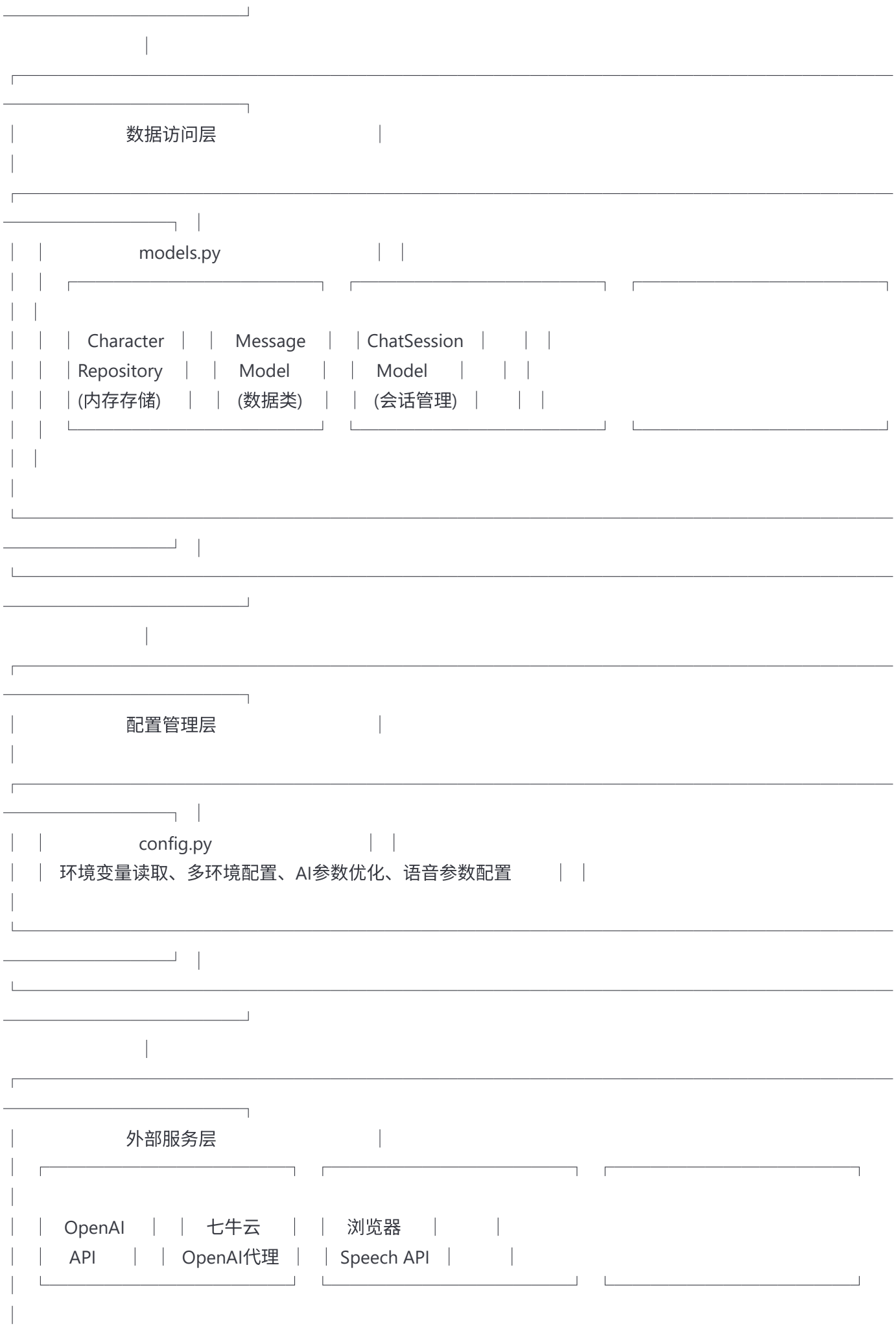
1.3 技术栈

- **后端框架**: Flask 2.3.3
- **实时通信**: Flask-SocketIO 5.3.5
- **异步处理**: Eventlet 0.33.3
- **AI接口**: OpenAI兼容API
- **前端技术**: HTML5模板内嵌JavaScript、WebSocket、Web Speech API
- **模板引擎**: Jinja2
- **配置管理**: python-dotenv

2. 系统架构

2.1 整体架构图


WebSocket + Web Speech API + DOM操作
HTTP/WebSocket
Web应用层
app.py
Flask路由管理
SocketIO事件
错误处理中文支持
角色ID生成器 + 中文拼音映射
业务逻辑层
services.py
ChatService
聊天管理
流式响应
AIService
AI调用
多API支持
VoiceService
语音处理
浏览器TTS


数据访问层
models.py
Character Repository (内存存储)
Message Model (数据类)
ChatSession Model (会话管理)
配置管理层
config.py
环境变量读取、多环境配置、AI参数优化、语音参数配置
外部服务层
OpenAI API
七牛云 OpenAI代理
浏览器 Speech API

2.2 架构特点

- **HTML模板集成**: 前端逻辑通过HTML模板中的`<script>`标签实现
- **分层架构**: 清晰的职责分离，自上而下的依赖关系
- **内存数据**: 使用内存存储，适合小规模应用快速部署
- **中文支持**: 特别优化的中文角色名处理和拼音转换

3. 核心模块设计

3.1 Web应用层 (app.py)

主要职责:

- Flask应用初始化和配置
- HTTP路由管理
- WebSocket事件处理
- 中文角色ID生成
- 全局错误处理

核心路由组件:

```python
# 页面路由
@app.route('/')                          # 主页(角色列表)
@app.route('/character/<character_id>')  # 角色详情页
@app.route('/chat/<character_id>')       # 聊天页面
@app.route('/admin/characters')          # 角色管理后台

# API路由
@app.route('/api/characters', methods=['GET', 'POST'])       # 角色CRUD
@app.route('/api/characters/<id>', methods=['PUT', 'DELETE']) # 角色更新/删除
@app.route('/api/chat/send', methods=['POST'])               # 发送消息
@app.route('/api/chat/stream', methods=['POST'])             # 流式聊天
@app.route('/api/voice/*')                                   # 语音相关API

# 表单路由
@app.route('/api/characters/create', methods=['POST'])       # 表单创建角色
@app.route('/api/characters/<id>/delete', methods=['POST'])  # 表单删除角色
```

WebSocket事件处理:

```python
@socketio.on('start_voice_call')    # 开始语音通话
@socketio.on('voice_stream')        # 语音流数据处理
@socketio.on('end_voice_call')      # 结束语音通话
@socketio.on('interrupt_ai_response') # 中断AI响应
```

特色功能:

- **中文角色ID生成器**: 支持中文角色名自动转换为有效ID
- **拼音映射表**: 内置常见中文人名的拼音对照
- **智能ID冲突处理**: 自动处理重复ID的命名冲突

3.2 业务逻辑层 (services.py)

3.2.1 ChatService (聊天服务)

```python
class ChatService:
    def start_chat_session(user_id, character_id) -> ChatSession
    def send_message(session_id, message, is_voice_call=False) -> str
    def send_message_stream(session_id, message, is_voice_call=False) -> Generator
    def get_chat_history(session_id) -> List[Message]
```

设计特点:

- 支持语音通话和文字聊天的参数差异化
- 流式响应生成，提升用户体验
- 会话状态管理和上下文维护

3.2.2 AIService (AI服务)

```python
class AIService:
    def generate_response(character, message, history, is_voice_call) -> str
    def generate_response_stream(character, message, history, is_voice_call) -> Generator
    def _build_messages(character, message, history, is_voice_call) -> List[Dict]
```

核心功能:

- **多API支持**: OpenAI、七牛云代理、自定义API
- **参数优化**: 区分语音通话和文字聊天的AI参数
- **流式处理**: 支持Server-Sent Events流式响应

- **错误处理**: 完整的重试机制和降级策略

语音通话优化:

```python
# 语音通话专用参数
VOICE_CALL_TEMPERATURE=0.6      # 更稳定的创造性
VOICE_CALL_MAX_TOKENS=150       # 限制回复长度
VOICE_CALL_FREQUENCY_PENALTY=0.3 # 减少重复内容
```

3.2.3 VoiceService (语音服务)

```python
class VoiceService:
    def text_to_speech(text, character) -> Dict
    def get_voice_settings_for_character(character) -> Dict
    def _get_browser_voice_config(character) -> Dict
```

设计特点:

- 基于浏览器Web Speech API
- 角色化语音配置(性别、年龄、语言)
- 多语言支持映射

3.3 数据访问层 (models.py)

3.3.1 数据模型设计

```python
```

```python
@dataclass
class Character:
    id: str                    # 角色唯一标识
    name: str                  # 角色名称
    personality: str           # 性格描述
    background: str            # 背景故事
    voice_config: Dict[str, Any]    # 语音配置
    temperature_modifier: float     # AI参数调节

@dataclass
class Message:
    sender_type: str           # 'user' or 'character'
    content: str               # 消息内容
    timestamp: datetime        # 时间戳
    metadata: Dict[str, Any]   # 元数据

@dataclass
class ChatSession:
    user_id: str               # 用户ID
    character_id: str          # 角色ID
    messages: List[Message]    # 消息列表
```

3.3.2 数据仓库模式

```python
class CharacterRepository:
    def get_all() -> List[Character]
    def get_by_id(character_id) -> Optional[Character]
    def add_character(character) -> bool
    def delete_character(character_id) -> bool
    def search(query, category) -> List[Character]

class ChatRepository:
    def create_session(user_id, character_id) -> ChatSession
    def get_session(session_id) -> Optional[ChatSession]
    def get_user_sessions(user_id) -> List[ChatSession]
```

预设角色数据:

- 8个精心设计的经典角色
- 完整的人设、技能和对话示例
- 多样化的类别覆盖(魔幻、推理、历史、科学等)

3.4 配置管理层 (config.py)

3.4.1 配置层次结构

```python
class Config:                # 基础配置
class DevelopmentConfig(Config): # 开发环境
class ProductionConfig(Config):  # 生产环境
class TestingConfig(Config):     # 测试环境
```

3.4.2 核心配置项

```python
# AI服务配置
AI_MODEL_PROVIDER = openai
AI_API_KEY = sk-xxx
AI_API_BASE = https://api.openai.com/v1
AI_MODEL = gpt-3.5-turbo

# 性能参数
DEFAULT_TEMPERATURE = 0.7       # 标准对话创造性
DEFAULT_MAX_TOKENS = 1000       # 标准回复长度
VOICE_CALL_TEMPERATURE = 0.6    # 语音通话创造性
VOICE_CALL_MAX_TOKENS = 150     # 语音回复长度限制

# 系统限制
MAX_CONVERSATION_LENGTH = 50    # 最大对话轮数
MAX_MESSAGE_LENGTH = 2000       # 单条消息长度限制
RESPONSE_TIMEOUT = 30           # API超时时间
```

4. 前端架构 (HTML模板)

4.1 模板结构

```
templates/
├── base.html                # 基础布局模板
├── index.html               # 主页(角色选择)
├── character.html           # 角色详情页
├── chat.html                # 聊天界面
├── character_management.html    # 角色管理后台
├── about.html               # 关于页面
├── error.html               # 错误页面
├── 404.html                 # 404页面
└── 500.html                 # 500页面
```

4.2 前端技术实现

4.2.1 JavaScript功能集成

所有JavaScript代码内嵌在HTML模板的`<script>`标签中：

```html
<script>
// WebSocket连接管理
const socket = io();

// 语音识别功能
if ('webkitSpeechRecognition' in window) {
    const recognition = new webkitSpeechRecognition();
    // 语音识别配置和事件处理
}

// 语音合成功能
function speakText(text, voiceConfig) {
    const utterance = new SpeechSynthesisUtterance(text);
    // 语音合成配置和播放
}

// 聊天功能
function sendMessage(message) {
    // 消息发送和处理逻辑
}

// 语音通话功能
function startVoiceCall() {
    socket.emit('start_voice_call', {
        session_id: sessionId,
        character_id: characterId
    });
}
</script>
```

4.2.2 CSS样式集成

所有样式定义内嵌在HTML模板的`<style>`标签中：

```html
```

```
<style>
/* 聊天界面样式 */
.chat-container { /* 聊天容器布局 */ }
.message-bubble { /* 消息气泡样式 */ }
.voice-controls { /* 语音控制按钮 */ }

/* 响应式设计 */
@media (max-width: 768px) {
    /* 移动端适配 */
}

/* 动画效果 */
@keyframes typing {
    /* 打字效果动画 */
}
</style>
```

4.3 模板继承结构

python

```
# base.html - 基础模板
{% block title %}{% endblock %}
{% block head %}{% endblock %}
{% block content %}{% endblock %}

# 子模板继承示例
{% extends "base.html" %}
{% block title %}聊天 - {{ character.name }}{% endblock %}
{% block content %}
<!-- 具体页面内容 -->
{% endblock %}
```

5. 数据流设计

5.1 文字聊天流程

```
用户输入 → HTML表单/AJAX → Flask路由 → ChatService
→ AIService → OpenAI API → 响应处理 → JSON返回 → 前端更新
```

5.2 语音通话流程

```
语音输入 → Web Speech API → WebSocket发送 → SocketIO处理
→ ChatService → AIService → 流式响应 → WebSocket推送
```

→ 前端接收 → TTS合成 → 语音播放

5.3 角色管理流程

表单提交 → Flask路由 → CharacterRepository → 内存更新
→ 文件上传处理 → 重定向返回 → 页面刷新

6. 安全和性能设计

6.1 安全措施

- **输入验证**: 消息长度和内容格式验证
- **XSS防护**: Jinja2模板自动转义
- **CSRF保护**: Flask内置CSRF令牌
- **API密钥保护**: 环境变量隔离存储

6.2 性能优化

- **流式响应**: 减少用户等待时间
- **WebSocket复用**: 降低连接开销
- **内存存储**: 快速数据访问
- **错误降级**: 备用响应机制

6.3 可扩展性设计

- **模块化架构**: 各层独立可替换
- **配置驱动**: 支持多环境部署
- **API抽象**: 支持多个AI服务提供商
- **模板继承**: 便于界面定制和扩展

7. 部署架构

7.1 开发环境

Python app.py → Flask开发服务器 → 单进程运行

7.2 生产环境

Gunicorn → 多Worker进程 → Eventlet异步处理 → 负载均衡

7.3 容器化部署

Docker容器 → 环境隔离 → 快速部署 → 水平扩展