WebSocket API 接口文档

目录

连接建立

连接端点

WebSocket 服务提供以下连接端点：

1. **主服务端点**：`/api/mainservice`
   - 用于所有主要业务功能（聊天、查询、API密钥管理等）
2. **语音交互端点**：`/api/speechinteractive`
   - 用于语音识别和语音合成功能
   - 与主服务端点使用相同的处理逻辑

连接步骤

1. **建立 WebSocket 连接**

```
const ws = new
WebSocket('ws://101.43.119.131:20718/api/mainservice');
```

2. **连接参数**
   - 协议：`ws://` 或 `wss://`（推荐使用 WSS 进行 TLS 加密）
   - 无需额外的查询参数或请求头
   - 服务器会自动升级 HTTP 连接为 WebSocket
3. **连接限制**
   - 读缓冲区：4096 字节

- 写缓冲区：4096 字节
- 读取超时：70 秒（自动延长）
- 心跳间隔：60 秒

认证流程

认证消息（明文）

在建立连接后，客户端必须首先发送认证消息。认证消息**必须使用明文**（不加密），因为此时尚未建立加密通道。

请求格式：

```
{
  "type": "auth",
  "data": {
    "token": "your_jwt_token_here"
  }
}
```

响应格式：

成功：

```
{
  "type": "auth_success",
  "data": {
    "status": "success",
    "message": "认证成功",
    "user_id": "user_123"
  }
}
```

失败：

```
{
  "type": "auth_failed",
  "data": {
    "status": "error",
    "message": "认证失败：token无效"
  }
}
```

认证状态

- **未认证状态**：连接建立后，在收到有效的 `auth` 消息之前，连接处于未认证状态
- **已认证状态**：收到 `auth_success` 响应后，连接进入已认证状态
- **未认证限制**：未认证状态下，除 `auth`、`ping`、`speech_interaction_start`、`speech_interaction_end` 外的所有消息都会被拒绝

加密机制

加密概述

WebSocket 通信采用**混合加密策略**：

- **传输层加密**：推荐使用 WSS（WebSocket Secure）进行 TLS 加密
- **应用层加密**：使用 RSA 非对称加密对敏感数据进行端到端加密

加密类型分类

1. 客户端到服务器（Client → Server）

单边加密（必须加密的消息类型）

以下消息类型**必须加密**，使用服务器的公钥进行加密：

- `chat` - 聊天消息
- `query` - 查询请求（如查询 token 余额）
- `api_key_manage` - API 密钥管理
- `check_version` - 版本检查
- `check_db_version` - 数据库版本检查
- `user_token` - 用户 token 相关
- `logout` - 登出请求
- `deactivation` - 账户注销
- `modify_username` - 修改用户名
- `modify_nickname` - 修改昵称
- `modify_mobile` - 修改手机号
- `modify_email` - 修改邮箱

加密格式：

```
{
  "ciphertext": "16进制编码的加密数据"
}
```

加密流程：

1. 客户端构造原始 JSON 消息
2. 使用服务器的 RSA 公钥加密整个 JSON 字符串
3. 将加密后的数据转换为 16 进制字符串
4. 包装在 `ciphertext` 字段中发送

明文消息（允许不加密的消息类型）

以下消息类型**允许明文**传输：

- `auth` - 认证消息（必须在认证前发送，无法加密）
- `ping` - 心跳消息（低敏感度）
- `speech_interaction_start` - 语音交互开始（控制消息）
- `speech_interaction_end` - 语音交互结束（控制消息）

明文格式：

```
{
  "type": "auth",
  "data": {
    "token": "your_token"
  }
}
```

2. 服务器到客户端（Server → Client）

条件加密（根据客户端公钥）

服务器响应采用**条件加密**策略：

- **有客户端公钥**：使用客户端的 RSA 公钥加密响应
- **无客户端公钥**：发送明文响应

加密响应格式：

```
{
  "ciphertext": "16进制编码的加密数据"
}
```

明文响应格式：

```
{
  "type": "response_type",
  "data": {
    // 响应数据
  }
}
```

客户端公钥提供方式：

客户端在请求的 `data` 字段中提供 `userPublicKeyHex`：

```
{
  "type": "query",
  "data": {
    "operation": "check_tokens",
    "userPublicKeyHex": "16进制编码的公钥字符串"
  }
}
```

公钥格式支持：

- 16 进制编码的 PEM 格式（推荐）
- PEM 格式（直接，也支持）

3. 端对端加密（End-to-End Encryption）

当前实现：不支持真正的端对端加密

- 服务器可以解密所有客户端发送的加密消息
- 服务器可以看到所有业务数据
- 加密主要用于：
  - 保护传输过程中的数据（配合 TLS）
  - 防止中间人攻击
  - 满足合规要求

加密流程：

```
客户端 → [RSA加密] → 服务器 → [解密] → 服务器处理 → [RSA加密] → 客户端
```

加密算法

- **算法**：RSA 非对称加密
- **密钥长度**：根据服务器配置（通常 2048 位或更高）
- **编码方式**：16 进制字符串

- **加密服务**：`digitalsingularity/backend/common/security/asymmetricencryption`

加密失败处理

- **客户端加密失败**：消息将被拒绝，服务器返回错误
- **服务器加密失败**：降级为明文发送（记录警告日志）
- **解密失败**：返回错误响应，要求客户端重新发送

消息格式

文本消息（JSON）

所有文本消息使用 JSON 格式：

```
{
  "type": "message_type",
  "data": {
    // 消息数据
  }
}
```

二进制消息

用于传输音频数据：

- **消息类型**：`websocket.BinaryMessage`
- **内容**：原始音频字节流
- **用途**：语音识别时传输音频数据
- **要求**：必须先发送 `speech_interaction_start` 消息建立语音会话

消息类型

认证相关

auth

- **方向**：客户端 → 服务器
- **加密**：明文（必须）
- **认证要求**：无需认证
- **描述**：用户认证

logout

- **方向**：客户端 → 服务器
- **加密**：必须加密
- **认证要求**：需要认证
- **描述**：用户登出，连接将关闭

心跳相关

ping

- **方向**：客户端 → 服务器
- **加密**：明文（允许）
- **认证要求**：建议认证后使用
- **描述**：客户端心跳

pong

- **方向**：服务器 → 客户端
- **加密**：明文
- **描述**：服务器心跳响应

查询相关

query

- **方向**：客户端 → 服务器
- **加密**：必须加密
- **认证要求**：需要认证
- **描述**：通用查询接口

支持的 **operation**：

- `check_tokens` - 查询用户 token 余额

请求示例：

```
{
  "type": "query",
  "data": {
    "operation": "check_tokens",
    "userPublicKeyHex": "16进制编码的公钥字符串"
  }
}
```

响应示例（加密）：

```
{
  "type": "check_tokens_response",
  "data": {
    "ciphertext": "encrypted_data"
  }
}
```

响应示例（明文）：

```
{
  "type": "check_tokens_response",
  "status": true,
  "message": "查询成功",
  "data": {
    "balance": 1000,
    "gifted_tokens": 500,
    "owned_tokens": 500,
    "has_enough": true
  }
}
```

聊天相关

chat

- **方向**：客户端 → 服务器
- **加密**：必须加密
- **认证要求**：需要认证
- **描述**：AI 聊天消息

请求格式：

```
{
  "type": "chat",
  "data": {
    "model": "deepseek-chat",
    "messages": [
      {
        "role": "user",
        "content": "Hello"
      }
    ],
    "stream": true,
    "userPublicKeyHex": "16进制编码的公钥字符串",
    "enable_tts": false,
    "voice_gender": "female"
```

```
    }
}
```

响应类型：

1. **chat_started** - 处理开始
2. **session_id** - 会话 ID
3. **chat_chunk** - 流式数据块
4. **chat_think** - 思考内容（MCP 调用时）
5. **chat_complete** - 完整响应（非流式）
6. **chat_done** - 处理完成
7. **chat_error** - 错误消息

API 密钥管理

api_key_manage

- **方向**：客户端 → 服务器
- **加密**：必须加密
- **认证要求**：需要认证
- **描述**：API 密钥管理

支持的操作：

- `list_api_keys` - 列出所有 API 密钥
- `create_api_key` - 创建新的 API 密钥
- `delete_api_key` - 删除 API 密钥
- `update_api_key_status` - 更新 API 密钥状态

版本检查

check_version

- **方向**：客户端 → 服务器
- **加密**：必须加密
- **认证要求**：需要认证
- **描述**：检查客户端版本

check_db_version

- **方向**：客户端 → 服务器
- **加密**：必须加密
- **认证要求**：需要认证
- **描述**：检查数据库版本

语音交互

speech_interaction_start

- **方向**：客户端 → 服务器
- **加密**：明文（允许）
- **认证要求**：需要认证
- **描述**：开始语音交互会话

请求格式：

```
{
  "type": "speech_interaction_start",
  "data": {
    "model": "claude-3-7-sonnet-20250219"
  }
}
```

speech_interaction_end

- **方向**：客户端 → 服务器
- **加密**：明文（允许）
- **认证要求**：需要认证
- **描述**：结束语音交互会话

音频数据传输：

- 在 `speech_interaction_start` 后，客户端可以发送二进制消息传输音频数据
- 服务器会实时返回识别结果

识别结果消息类型：

- `sentence_state` - 句子状态（开始/结束）
- `partial` - 部分识别结果（实时更新）
- `final` - 最终识别结果
- `error` - 识别错误

语音合成消息类型：

- `speech_synthesis_audio` - 音频数据块（Base64 编码）
- `speech_synthesis_complete` - 合成完成
- `speech_synthesis_error` - 合成错误

心跳机制

服务器心跳

- **间隔**：60 秒
- **类型**：WebSocket Ping 帧
- **响应**：客户端自动回复 Pong 帧
- **超时**：70 秒无响应则断开连接

客户端心跳

- **消息类型**：`ping`
- **响应类型**：`pong`
- **格式**：

```
{
  "type": "ping"
}
```

响应：

```
{
  "type": "pong",
  "data": {
    "timestamp": "1234567890"
  }
}
```

错误处理

错误响应格式

```
{
  "type": "error",
  "data": {
    "status": "error",
```

```
    "message": "错误描述"
  }
}
```

常见错误

1. 认证失败

```
{
  "type": "auth_failed",
  "data": {
    "status": "error",
    "message": "认证失败: token无效"
  }
}
```

2. 未认证访问

```
{
  "type": "error",
  "data": {
    "status": "error",
    "message": "请先进行认证"
  }
}
```

3. 消息解密失败

```
{
  "type": "error",
  "data": {
    "status": "error",
    "message": "消息解密失败: ..."
  }
}
```

4. 非语音消息必须加密

```
{
  "type": "error",
  "data": {
    "status": "error",
    "message": "非语音消息必须加密"
  }
}
```

加密传输总结表

客户端 → 服务器

消息类型	加密要求	说明
`auth`	❌ 明文	必须在认证前发送
`ping`	❌ 明文	心跳消息
`speech_interaction_start`	❌ 明文	语音会话控制
`speech_interaction_end`	❌ 明文	语音会话控制
`chat`	✅ 必须加密	聊天消息
`query`	✅ 必须加密	查询请求
`api_key_manage`	✅ 必须加密	API 密钥管理
`check_version`	✅ 必须加密	版本检查
`check_db_version`	✅ 必须加密	数据库版本检查
`logout`	✅ 必须加密	登出
`deactivation`	✅ 必须加密	账户注销
`modify_*`	✅ 必须加密	用户信息修改
二进制音频数据	❌ 不加密	音频流

服务器 → 客户端

响应类型	加密方式	说明
所有响应	🔀 条件加密	如果客户端提供公钥则加密，否则明文
心跳响应	❌ 明文	`pong` 消息
语音识别结果	❌ 明文	实时识别结果
语音合成音频	❌ 明文	Base64 编码的音频数据

加密类型说明

- ✅ **必须加密**：消息必须使用服务器的公钥加密
- ❌ **明文**：消息以明文形式传输
- 🔀 **条件加密**：根据客户端是否提供公钥决定是否加密

安全建议

1. **使用 WSS**：生产环境必须使用 `wss://` 进行 TLS 加密
2. **保护私钥**：服务器私钥必须妥善保管，不得泄露
3. **定期轮换**：建议定期轮换加密密钥
4. **验证证书**：客户端应验证服务器 TLS 证书
5. **限制来源**：生产环境应限制 WebSocket 连接的来源
6. **监控异常**：监控加密失败、解密失败等异常情况

示例代码

Python 客户端示例

```python
import asyncio
import json
import websockets
from cryptography.hazmat.primitives import serialization
from cryptography.hazmat.primitives.asymmetric import rsa, padding
from cryptography.hazmat.primitives import hashes
from cryptography.hazmat.backends import default_backend

# WebSocket 服务器地址
WS_URL = "ws://101.43.119.131:20718/api/mainservice"
# 或使用语音交互端点
# WS_URL = "ws://101.43.119.131:20718/api/speechinteractive"

# JWT Token
JWT_TOKEN = "your_jwt_token_here"

# RSA 密钥（需要从服务器获取公钥，客户端使用私钥）
# 注意：服务器公钥是16进制字符串，需要先解码为PEM格式
SERVER_PUBLIC_KEY_HEX = None  # 需要从服务器获取（16进制字符串）
SERVER_PUBLIC_KEY_PEM = None  # 解码后的PEM格式公钥
CLIENT_PRIVATE_KEY = None  # 客户端私钥（PEM格式）

def hex_to_pem(hex_string: str) -> str:
    """将16进制字符串转换为PEM格式"""
```

```
    # 将16进制字符串转换为字节
    pem_bytes = bytes.fromhex(hex_string)
    # 转换为字符串（PEM格式）
    return pem_bytes.decode('utf-8')

def rsa_encrypt(data: str, public_key_pem: str) -> str:
    """使用 RSA 公钥加密数据，返回16进制字符串"""
    # 加载公钥
    public_key = serialization.load_pem_public_key(
        public_key_pem.encode(),
        backend=default_backend()
    )

    # 加密数据（RSA 有长度限制，需要分段加密或使用混合加密）
    # 这里简化处理，实际使用时可能需要处理长数据
    encrypted = public_key.encrypt(
        data.encode('utf-8'),
        padding.OAEP(
            mgf=padding.MGF1(algorithm=hashes.SHA256()),
            algorithm=hashes.SHA256(),
            label=None
        )
    )

    # 转换为16进制字符串
    return encrypted.hex()

def rsa_decrypt(ciphertext_hex: str, private_key_pem: str) -> str:
    """使用 RSA 私钥解密数据，输入为16进制字符串"""
    # 加载私钥
    private_key = serialization.load_pem_private_key(
        private_key_pem.encode(),
        password=None,
        backend=default_backend()
    )

    # 将16进制字符串转换为字节
    encrypted_data = bytes.fromhex(ciphertext_hex)

    # 解密数据
    decrypted = private_key.decrypt(
        encrypted_data,
        padding.OAEP(
            mgf=padding.MGF1(algorithm=hashes.SHA256()),
            algorithm=hashes.SHA256(),
            label=None
        )
    )

    return decrypted.decode('utf-8')
```

```python
async def send_encrypted_message(websocket, message_type: str, data:
dict, server_public_key: str):
    """发送加密消息"""
    message = {
        "type": message_type,
        "data": data
    }

    # 将消息转换为 JSON 字符串
    message_json = json.dumps(message, ensure_ascii=False)

    # 加密消息
    encrypted = rsa_encrypt(message_json, server_public_key)

    # 发送加密后的消息
    encrypted_message = {
        "ciphertext": encrypted
    }
    await websocket.send(json.dumps(encrypted_message))

async def handle_message(message_str: str, client_private_key: str =
None):
    """处理接收到的消息"""
    try:
        message = json.loads(message_str)

        # 检查是否为加密消息
        if "ciphertext" in message:
            if client_private_key:
                # 解密响应
                decrypted = rsa_decrypt(message["ciphertext"],
client_private_key)
                data = json.loads(decrypted)
                print(f"收到加密响应: {data}")

                # 根据消息类型处理
                handle_message_type(data)
            else:
                print("收到加密消息，但未提供客户端私钥，无法解密")
        else:
            # 明文响应
            print(f"收到明文响应: {message}")
            handle_message_type(message)

    except Exception as e:
        print(f"处理消息失败: {e}")

def handle_message_type(message: dict):
    """根据消息类型处理不同的响应"""
```

```
    msg_type = message.get("type")

    if msg_type == "auth_success":
        print("认证成功")
        data = message.get("data", {})
        user_id = data.get("user_id")
        print(f"用户 ID: {user_id}")

    elif msg_type == "auth_failed":
        print("认证失败")
        data = message.get("data", {})
        error_msg = data.get("message", "未知错误")
        print(f"错误信息: {error_msg}")

    elif msg_type == "chat_chunk":
        # 处理流式响应块
        chunk = message.get("chunk", "")
        print(f"收到文本块: {chunk}")

    elif msg_type == "chat_complete":
        # 处理完整响应
        content = message.get("content", "")
        print(f"收到完整响应: {content}")

    elif msg_type == "chat_done":
        print("聊天会话完成")

    elif msg_type == "chat_error":
        error_msg = message.get("message", "未知错误")
        print(f"聊天错误: {error_msg}")

    elif msg_type == "pong":
        # 心跳响应
        print("收到心跳响应")

    else:
        print(f"未知消息类型: {msg_type}")

async def main():
    """主函数"""
    try:
        # 建立 WebSocket 连接
        async with websockets.connect(WS_URL) as websocket:
            print("WebSocket 连接已建立")

            # 发送认证消息（明文）
            auth_message = {
                "type": "auth",
                "data": {
                    "token": JWT_TOKEN
```

```
                }
            }
            await websocket.send(json.dumps(auth_message))
            print("已发送认证消息")

            # 接收认证响应
            auth_response = await websocket.recv()
            await handle_message(auth_response)

            # 如果认证成功，可以发送其他消息
            # 注意：发送 chat 等消息需要使用加密

            # 示例：发送聊天消息（需要先获取服务器公钥）
            # chat_data = {
            #     "model": "deepseek-chat",
            #     "messages": [
            #         {
            #             "role": "user",
            #             "content": "Hello"
            #         }
            #     ],
            #     "stream": True,
            #     "enable_tts": False,
            #     "voice_gender": "female"
            # }
            # if SERVER_PUBLIC_KEY_PEM:
            #     await send_encrypted_message(websocket, "chat",
chat_data, SERVER_PUBLIC_KEY_PEM)

            # 持续接收消息
            async for message in websocket:
                await handle_message(message, CLIENT_PRIVATE_KEY)

    except Exception as e:
        print(f"连接错误: {e}")

# 运行示例
if __name__ == "__main__":
    # 注意：需要先获取服务器公钥和配置客户端私钥
    # 1. 从 HTTP API 获取服务器公钥（16进制格式）
    #    POST http://101.43.119.131:20717/api/getServerPublicKey
    #    响应：{"status": "success", "serverPublicKey": "16进制字符串"}
    # 2. 将16进制字符串转换为PEM格式
    #    SERVER_PUBLIC_KEY_HEX = "从API获取的16进制字符串"
    #    SERVER_PUBLIC_KEY_PEM = hex_to_pem(SERVER_PUBLIC_KEY_HEX)
    # 3. 配置客户端私钥（PEM格式）
    #    CLIENT_PRIVATE_KEY = "客户端私钥（PEM 格式）"

    asyncio.run(main())
```

依赖安装：

```
pip install websockets cryptography
```

更新日志

- **2025-11-18**：初始版本文档
  - 定义连接建立流程
  - 说明加密机制
  - 列出所有消息类型

联系方式

如有问题或建议，请联系**坤泽开发团队**。