AI Demo

API 接口文档

版本 **v2.12.6**

生成日期 2026-07-31

生产 Base URL https://ai.maono.com

本文档为离线交付材料,在线版以 `https://ai.maono.com/api-docs` 为准

AI Demo API 文档 v2.12.6

目录

API 接口文档

> **生产 Base URL:** `https://ai.maono.com` (必须 HTTPS,不要用 IP 直连) **测试 Base URL:** `http://8.163.67.66` (无 HTTPS,仅供联调)
>
> 版本: v2.12.6
>
> ⚠ **生产环境不要用 `http://116.205.186.240/...` 直连**:
>
> - 生产 nginx 强制 80 → 443 跳转(`/api/health` 探活除外),用 HTTP 请求会收到 `301 Moved Permanently` + `text/html` 响应,RestTemplate / OkHttp 等 Java 客户端默认不跟随 301 且无法解析 text/html → 表现为 `UnknownContentTypeException` 或类似错误。
> - IP 直连即便走 HTTPS,SSL 证书是签给 `*.maono.com` 的,IP 不匹配会 SSL 校验失败。
> - **永远用 `https://ai.maono.com` 域名访问。**

通用说明

响应格式

所有 HTTP JSON 接口使用统一的响应信封:

```
{
  "code": 0,
  "message": "ok",
  "data": { ... }
}
```

错误响应（业务错误）:

```
{
  "code": 40001,
  "message": "错误描述"
}
```

错误响应（参数错误 / 资源不存在）：

```
{
  "detail": "错误描述"
}
```

业务追踪 trace_id（v2.4.2+）

所有业务接口均支持传入可选的 `trace_id` 字段，用于**关联同一次业务操作涉及的多个接口调用**。

典型场景：甲方一次"录音处理"操作需要依次调用 ASR 转写 → 智能总结 → 智能分章，三个接口各自推送独立的 webhook 事件。如果不带 `trace_id`，甲方服务端收到 `asr.completed` + `summarize.completed` + `chapters.completed` 三个事件后，只能靠 `client_reference_id`（用户级）关联——**同一用户并发提交两段录音时无法区分**。

用法：在每个接口调用时传入同一个 `trace_id`，所有相关的 webhook 事件 payload 中都会原样返回该值。

```
// 甲方生成一个业务操作 ID
trace_id = "recording_20260417_meeting_abc"

POST /api/v1/asr/submit     { url: "...", trace_id: "recording_20260417_meeting_abc" }
POST /api/v1/llm/summarize  { text: "...", trace_id: "recording_20260417_meeting_abc" }
POST /api/v1/llm/chapters   { utterances: [...], trace_id: "recording_20260417_meeting_abc" }

→ webhook asr.completed       { trace_id: "recording_20260417_meeting_abc", ... }
→ webhook summarize.completed { trace_id: "recording_20260417_meeting_abc", ... }
→ webhook chapters.completed  { trace_id: "recording_20260417_meeting_abc", ... }
```

生成建议：

- 格式不限，只要同一次业务操作内保持一致即可
- 推荐使用有业务含义的字符串（如 `recording_{日期}_{会议ID}`），便于人工排查
- 也可以用 UUID（如 `550e8400-e29b-41d4-a716-446655440000`），适合纯自动化场景
- 长度建议不超过 128 字符

规则：

- 不传或传空字符串时，webhook payload 中 `trace_id` 为 `null`（服务端不自动生成）

- 服务端**不校验唯一性**——甲方自行保证同一次操作用同一个值、不同操作用不同值
- `trace_id` 和 `client_reference_id` 互补：前者是**操作级**（甲方主动传），后者是**用户级**（从 JWT 自动提取）

鉴权

通过环境变量 `AUTH_ENABLED=true` 启用鉴权（默认关闭）。启用后，系统有**三级入站鉴权 + 一级出站签名**：

1. 公开接口 — 无需鉴权

以下路径不需要任何认证：

- 鉴权入口：`/api/v1/auth/exchange`
- Ticket 调试工具：`/api/v1/auth/inspect-ticket`（只读校验 Ticket，不消费 nonce）
- 健康检查：`/api/health`
- 总结模版元数据：`/api/v1/llm/summary-templates`（只读模版列表，供客户端登录前渲染下拉选择）
- Webhook 调试接收器：`/api/v1/debug/webhook-receiver`（服务端自身 dispatcher 回调进来，必须公开）
- 静态资源：`/static/*`
- API 文档：`/docs`（Swagger UI）、`/api-docs`（接口文档）

2. 用户级鉴权（JWT）— 业务接口

适用于终端用户/客户端调用的业务接口（ASR、LLM）。

HTTP 接口 — 通过 Header 传递：

```
Authorization: Bearer <jwt_token>
```

WebSocket 接口 — 通过 query 参数传递（浏览器 WebSocket API 不支持自定义 Header）：

```
ws://host/api/v1/asr/stream?token=<jwt_token>
```

> 如果客户端支持自定义 Header（如硬件端 SDK），也可通过 Header 传递，服务端优先读取 Header。

JWT 通过 Ticket 换取，详见下方 鉴权 章节。

JWT 鉴权错误响应：

HTTP 状态码	code	含义
401	40101	缺少 Authorization header
401	40102	JWT 无效（签名错误、格式错误）
401	40103	JWT 已过期

WebSocket 鉴权失败 — 服务端会**先完成握手再发 close frame**,保证客户端能明确拿到失败原因(而非神秘的 HTTP 403):

1. 先完成 WebSocket 握手(返回 101 Switching Protocols)
2. 发送一条 JSON 消息: `{"type": "auth_error", "reason": "JWT 已过期"}`
3. 主动 close,close code 为 `4001`,reason 含错误描述

客户端应监听 `onclose` 检查 `event.code === 4001`,若命中则**提示用户重新鉴权**(JWT 过期的常见触发场景:本地缓存的老 JWT / 长时间未活动的 Tab)。

```
ws.onclose = (evt) => {
  if (evt.code === 4001) {
    // 清掉本地缓存的 JWT,引导用户重新鉴权
    localStorage.removeItem("jwt_token");
    alert("鉴权失败:" + evt.reason);
  }
};
```

3. 系统级鉴权（API Secret）— 管理接口

适用于甲方服务端调用的管理接口（Webhook 注册/删除）。

```
X-Api-Secret: <api_secret>
```

API Secret 预先线下交换给甲方运维，长期有效。

API Secret 鉴权错误响应：

HTTP 状态码	code	含义
401	40101	缺少 X-Api-Secret header
403	40301	API Secret 无效

鉴权范围汇总

接口	鉴权方式	说明
公开接口		
`POST /api/v1/auth/exchange`	无	用于 Ticket 换取 JWT
`POST /api/v1/auth/inspect-ticket`	无	只读校验 Ticket 内容（调试工具）
`GET /api/health`	无	健康检查
调试接口（调试台使用，甲方无需调用）		
`POST /api/v1/debug/test-ticket`	**API Secret**	生成测试 Ticket（v2.5.4 起需 `X-Api-Secret`，防滥用签发）
`GET /api/v1/debug/webhook-logs`	**API Secret**	查看 Webhook 推送日志（v2.5.4 起需 `X-Api-Secret`）
`DELETE /api/v1/debug/webhook-logs`	**API Secret**	清空 Webhook 推送日志（v2.5.4 起需 `X-Api-Secret`）
`POST /api/v1/debug/webhook-receiver`	无	接收 Webhook 推送（服务端自身 dispatcher 回调，保持公开）
业务接口（JWT）		
`POST /api/v1/asr/submit`	JWT	提交录音文件转写任务
`GET /api/v1/asr/tasks/{id}`	JWT	查询转写任务状态
`WS /api/v1/asr/stream`	JWT (query)	实时流式转写
`POST /api/v1/llm/summarize`	JWT	智能总结
`POST /api/v1/llm/translate`	JWT	智能翻译
`POST /api/v1/llm/chapters`	JWT	提交智能分章任务（v2.4.1 起异步，立即返回 task_id）
`GET /api/v1/llm/chapters/tasks/{id}`	JWT	查询分章任务状态和结果
`GET /api/v1/llm/summary-templates`	无	列出总结模版（公开元数据）
管理接口（API Secret）		
		Webhook 管理

接口	鉴权方式	说明
`POST/GET/DELETE /api/v1/webhooks`	API Secret	
`POST /api/v1/admin/webhooks/reset`	API Secret	批量重置所有 Webhook（危险操作，v2.3.7+）
`POST /api/v1/admin/reload-templates`	API Secret	热重载总结模版
`GET/PUT/POST/DELETE /api/v1/admin/templates/{lang}/{id}`	API Secret	总结模版 CRUD（自动热重载）
`GET/PUT /api/v1/admin/sensitive-words`	API Secret	ASR 敏感词过滤配置
`POST /api/v1/admin/sensitive-words/reload`	API Secret	热重载敏感词配置
`GET /api/v1/admin/auth-config`	API Secret	调试：查询 Ticket 签发参数（见 § 鉴权 → 查询 Ticket 签发参数）

4. 出站签名 — Webhook 推送

上述三级都是**入站**（甲方 → 我们）。我们主动推送给甲方的 webhook 回调是**出站**方向,请求头里带 `X-Webhook-Signature: sha256=<hmac>`,**甲方 receiver 端需做 HMAC-SHA256 验签**以确认消息真实来自本服务端(而非伪造)。

签名算法、Secret 优先级、3 种语言的完整 receiver 代码、接收端安全清单详见 § 6.7 签名验证与安全约定。

鉴权

鉴权流程

```
客户端 → 甲方服务器（登录）→ 甲方生成 Ticket → 返回客户端
客户端 → POST /api/v1/auth/exchange {ticket} → 返回 JWT
客户端 → 后续请求携带 JWT 访问业务接口
```

Ticket 由甲方服务端使用共享密钥（HMAC-SHA256）签名生成，AI 服务端本地验签，无需回调。

Ticket 格式

```
ticket = base64url(payload) + "." + base64url(signature)
```

Payload 结构：

```
{
  "app_id": "maono",
  "user_id": "user_12345",
  "device_id": "device_abc",
  "exp": 1775200300,
  "nonce": "a1b2c3d4e5f6g7h8i9j0k1l2m3n4o5p6"
}
```

字段	类型	必填	说明
app_id	string	**是**	预分配的应用标识
user_id	string	**是**	用户唯一标识
device_id	string	否	设备标识
exp	int	**是**	过期时间戳（秒），建议当前时间 + 300
nonce	string	**是**	32 字符随机串，防重放

Ticket 换取 JWT

POST /api/v1/auth/exchange
Content-Type: application/json

入参

```
{
  "ticket": "eyJhcHBfaWQiOiJtYW9ubyIsInVzZXJfaWQiOiJ0ZXN0X3VzZXIi...signature"
}
```

字段	类型	必填	说明
ticket	string	是	甲方签发的 Ticket

出参（成功）

```
{
  "code": 0,
  "message": "ok",
  "data": {
    "token": "eyJhbGciOiJIUzI1NiIsInR5cCI6IkpXVCJ9...",
    "expires_in": 7200
  }
}
```

字段	类型	说明
data.token	string	JWT 令牌
data.expires_in	int	有效期（秒），默认 7200（2 小时）

出参（失败）

HTTP 401

```
{
  "detail": {
    "code": "invalid_ticket",
    "message": "签名验证失败"
  }
}
```

code	说明
invalid_ticket	Ticket 格式错误或签名无效
ticket_expired	Ticket 已过期
nonce_reused	Ticket 已被使用（防重放）

Demo — 客户端换取 JWT

```
# 用 Ticket 换 JWT
TOKEN=$(curl -s -X POST http://8.163.67.66/api/v1/auth/exchange \
  -H "Content-Type: application/json" \
  -d '{"ticket": "YOUR_TICKET"}' | jq -r '.data.token')

# 用 JWT 调用业务接口
curl http://8.163.67.66/api/v1/asr/tasks/some-task-id \
  -H "Authorization: Bearer $TOKEN"
```

```
// 客户端用 Ticket 换 JWT
const resp = await fetch("/api/v1/auth/exchange", {
  method: "POST",
  headers: {"Content-Type": "application/json"},
  body: JSON.stringify({ticket: "YOUR_TICKET"})
});
const {data: {token}} = await resp.json();

// 后续请求带 JWT
const headers = {"Authorization": `Bearer ${token}`};
```

```
import requests

resp = requests.post("http://8.163.67.66/api/v1/auth/exchange",
    json={"ticket": "YOUR_TICKET"})
token = resp.json()["data"]["token"]
headers = {"Authorization": f"Bearer {token}"}
```

Demo — 甲方服务端生成 Ticket

```java
import javax.crypto.Mac;
import javax.crypto.spec.SecretKeySpec;
import java.util.Base64;
import java.util.UUID;

public class TicketGenerator {
    private static final String SHARED_SECRET = "your-shared-secret";

    public static String generate(String appId, String userId) throws Exception {
        long exp = System.currentTimeMillis() / 1000 + 300;
        String nonce = UUID.randomUUID().toString().replace("-", "");
        String payload = String.format(
            "{\"app_id\":\"%s\",\"user_id\":\"%s\",\"exp\":%d,\"nonce\":\"%s\"}",
            appId, userId, exp, nonce);
        String encodedPayload = Base64.getUrlEncoder()
            .withoutPadding().encodeToString(payload.getBytes());
        Mac mac = Mac.getInstance("HmacSHA256");
        mac.init(new SecretKeySpec(SHARED_SECRET.getBytes(), "HmacSHA256"));
        byte[] sig = mac.doFinal(encodedPayload.getBytes());
        String encodedSig = Base64.getUrlEncoder()
            .withoutPadding().encodeToString(sig);
        return encodedPayload + "." + encodedSig;
    }
}
```

```python
import hmac, hashlib, base64, json, time, uuid

SHARED_SECRET = "your-shared-secret"

def generate_ticket(app_id, user_id):
    payload = json.dumps({"app_id": app_id, "user_id": user_id,
        "exp": int(time.time()) + 300,
        "nonce": uuid.uuid4().hex}, separators=(",", ":"))
    payload_b64 = base64.urlsafe_b64encode(payload.encode()).rstrip(b"=").decode()
    sig = hmac.new(SHARED_SECRET.encode(), payload_b64.encode(), hashlib.sha256).digest()
    sig_b64 = base64.urlsafe_b64encode(sig).rstrip(b"=").decode()
    return f"{payload_b64}.{sig_b64}"
```

验证 Ticket（调试工具）

用于甲方服务端在开发阶段验证自己生成的 Ticket 是否正确。此接口**不消费 nonce**，不会影响后续 `/api/v1/auth/exchange` 换取 JWT。

```
POST /api/v1/auth/inspect-ticket
Content-Type: application/json

{"ticket": "eyJhcHBfaWQiOiJtYW9ubyJ9.HMAC_SIGNATURE"}
```

出参


```
{
  "code": 0,
  "message": "ok",
  "data": {
    "valid": true,
    "payload": {
      "app_id": "maono",
      "user_id": "u_12345",
      "device_id": "d_001",
      "exp": 1712800000,
      "nonce": "a1b2c3d4e5f6"
    },
    "checks": {
      "signature": true,
      "app_id": true,
      "expired": false,
      "nonce_used": false
    },
    "error_code": null,
    "message": null
  }
}
```


- `valid`：整体是否有效（四项 check 全部通过）
- `checks.signature`：签名是否与 `AUTH_SHARED_SECRET` 匹配
- `checks.app_id`：`app_id` 是否等于服务端配置
- `checks.expired`：是否已过期（true = 已过期）
- `checks.nonce_used`：nonce 是否已被消费（true = 已使用过）
- `error_code` / `message`：首个失败项的原因；验证成功时为 `null`

> **注意：** 此接口返回 HTTP 200，是否有效看 `data.valid`，错误信息在 `data.error_code`。

生成测试 Ticket（仅调试台使用）

```
POST /api/v1/debug/test-ticket
Content-Type: multipart/form-data
X-Api-Secret: <API_SECRET>
```

> **注意：** 此接口服务端会用生产 `AUTH_SHARED_SECRET` 签一个真实可用的 Ticket， v2.5.4 起必须带 `X-Api-Secret` header(防任意人白嫖签发)。甲方正常流程不需要调用—— Ticket 应由甲方自己的 Java 服务端签发(参见前文 "Ticket 签发")。

入参

参数	类型	必填	默认值	说明
user_id	string	否	`test_user`	模拟用户 ID
device_id	string	否	`""`	模拟设备 ID

出参

```
{
  "code": 0,
  "message": "ok",
  "data": {
    "ticket": "eyJhcHBfaWQiOiJtYW9ubyI..."
  }
}
```

查询 Ticket 签发参数（甲方自查密钥）

用于甲方服务端自查 **`app_id`** 和 **`AUTH_SHARED_SECRET`** 是否与我们服务端一致——避免出现 "Ticket 签名验证失败" 时搞不清是哪一侧的密钥错了。需 `X-Api-Secret` 鉴权。

```
GET /api/v1/admin/auth-config
X-Api-Secret: <api_secret>
```

出参


```
{
  "code": 0,
  "message": "ok",
  "data": {
    "app_id": "maono",
    "auth_shared_secret": "the-full-plaintext-secret",
    "auth_shared_secret_preview": "the-****cret",
    "auth_shared_secret_length": 32,
    "auth_shared_secret_fingerprint": "e3b0c44298fc1c149afbf4c8996fb92427ae41e4649b934ca495991b7852b855",
    "note": "app_id 和 auth_shared_secret 用于甲方签发 Ticket。JWT_SECRET 不返回（纯服务端内部使用）。"
  }
}
```


字段	说明
`app_id`	签 Ticket 时 payload 里的 `app_id` 字段必须等于此值
`auth_shared_secret`	HMAC-SHA256 签名用的完整明文 secret（**直接与本地值对比即可**）
`auth_shared_secret_preview`	前 4 位 + `****` + 后 4 位，便于日志/截图中目测
`auth_shared_secret_length`	明文长度（字节数），长度不一致通常说明复制粘贴时带入了尾部换行或空格
`auth_shared_secret_fingerprint`	可选：`SHA-256(secret)` 的 hex。当不想把明文 secret 贴到聊天/日志对比时，可用指纹替代（双方各自算各自的指纹交换即可）

Demo

```
curl http://8.163.67.66/api/v1/admin/auth-config \
  -H "X-Api-Secret: <api_secret>"
```

```java
// 最直接的做法：拉取响应，直接 String.equals 对比明文 secret
HttpClient client = HttpClient.newHttpClient();
HttpRequest req = HttpRequest.newBuilder()
    .uri(URI.create("http://8.163.67.66/api/v1/admin/auth-config"))
    .header("X-Api-Secret", System.getenv("API_SECRET"))
    .GET().build();
String body = client.send(req, HttpResponse.BodyHandlers.ofString()).body();
// 用任意 JSON 库解析（Jackson / Gson / Fastjson 均可）
String serverSecret = JSON.parseObject(body).getJSONObject("data").getString("auth_shared_
            secret");

String mySecret = System.getenv("AUTH_SHARED_SECRET");
if (mySecret.equals(serverSecret)) {
    System.out.println("✓ secret 匹配");
} else {
    System.out.println("  不匹配:"
        + " 本地长度=" + mySecret.length()
        + " 服务端长度=" + serverSecret.length());
}
```

```python
import os, requests

resp = requests.get("http://8.163.67.66/api/v1/admin/auth-config",
    headers={"X-Api-Secret": os.environ["API_SECRET"]}).json()

my_secret = os.environ["AUTH_SHARED_SECRET"]
if my_secret == resp["data"]["auth_shared_secret"]:
    print("✓ secret 匹配")
else:
    print(f"  不匹配: 本地长度={len(my_secret)} 服务端长度={resp['data']['auth_shared_secret_length
            ']}")
```

> **长度对不上时的常见坑**：复制粘贴时带入了尾部换行（ `\n` ）、空格、零宽空格（ `\u200b` ）、UTF-8 BOM（ `\ufeff` ）等不可见字符，肉眼看起来一致但实际不同。用 `repr()` 或 `bytes()` 打出 secret 末尾几个字符检查其 ASCII 码即可定位。

---

1. 录音文件转写

传入音频文件的**公网可访问 URL**，创建异步转写任务。URL 由甲方自行上传到对象存储（阿里云 OSS / 腾讯云 COS / 七牛等）或 CDN 后获得。

> **注意（v2.4.0 起）**：服务端不再提供文件上传接口。音频存储方案由甲方自管，本服务只负责接收 URL 并下载转写。URL 需要支持 HTTP HEAD + GET（短期有效的预签名 URL 也支持）。

> **(v2.5.12+)** 受上游豆包账号 QPS 上限保护,任务在我方 Celery 内排队等令牌后再 submit。日常低峰期等待 < 100ms 无感知;50 并发瞬时 burst 时排队约 3 秒;极端高峰累计排队超 120 秒时任务会标 `failed` + `error.code=50001` ,error.message 含 `RateLimitTimeout` ,甲方可重试。该机制只针对录音文件转写,流式 ASR / 总结 / 翻译 / 分章不受影响。

1.1 提交转写任务

```
POST /api/v1/asr/submit
Content-Type: multipart/form-data
Authorization: Bearer <jwt_token>
```

入参

参数	类型	必填	默认值	说明
url	string	**是**	-	音频文件的公网可访问 URL
language	string	否	`zh-CN`	语言代码，见附录 B
speaker_info	bool	否	`false`	开启声纹分人(基于音色聚类区分不同说话人),10 人以内效果较好。音量/远近差异大时效果会下降。utterances 返回 `speaker` 字段。**可跟 `enable_channel_split` 同时启用**(v2.5.35+):并存时变成"每通道内独立声纹分人",适合 L/R 各有多人轮流说的场景
channel	int	否	`1`	音频声道数。`1` =单声道 (mono) / `2` =双声道 (stereo)。非 1/2 的值会被拒绝（HTTP 400）
sample_rate	int	否	`16000`	**(v2.5.14+)** 音频采样率（Hz），按豆包大模型录音文件识别官方约定下发到 audio.rate 字段。范围 `[8000, 192000]`，超出会被拒绝（HTTP 400）。常见值：电话语音 `8000`，主流 ASR `16000`（默认），音乐级 `44100` / `48000`
enable_channel_split	bool	否	`false`	**(v2.5.35+)** 双声道独立识别。stereo 录音(如 Maono 双麦)**必须启用**,否则豆包把 L/R 混合识别,音量小的通道会整段丢失(实测一通道 13 秒内容默认参数下完全没识别出来)。启用要求 `channel=2` 。**可跟 `speaker_info` 同时启用**(实测豆包共存正常):同时启用时 speaker 变成"每通道内独立声纹分人",适合 L/R 多人轮流说场景;对 L/R 各一人场景 speaker 无额外价值。结果 utterances 带 `channel_id` 字段(`1` =L `2` =R)
hotwords	string	否	`null`	**(v2.5.35+)** 热词列表,中英文逗号分隔(`闪克,机器学习,人工智能`)。豆包支持最多 5000 个词,通过 `corpus.context` 透传给豆包做识别概率加权(boost)。**命中由豆包模型决定**,服务端不做兜底替换 — 文件转写是单 pass

参数	类型	必填	默认值	说明
				出最终结果,没有"草稿/二遍"中间状态,无法做流式 ASR(v2.5.34)那样的草稿门控
trace_id	string	否	`null`	业务追踪 ID，原样透传到 webhook payload。用于关联同一次业务操作的多个步骤（如 ASR → 总结 → 分章）

URL 要求： 必须为公网可访问的音频文件地址（如 OSS 签名 URL），格式需为豆包 ASR 支持的格式（wav, mp3, ogg 等）

channel 使用建议：默认 `1`（mono）适配绝大多数手机/麦克风录音；双声道素材（多轨播客、会议双麦）请显式传 `2`，让豆包按 stereo 处理，避免隐式降混丢信息。**双麦独立录音(L/R 各一个人)场景必须配合 `enable_channel_split=true` 使用**，否则豆包默认混合识别会丢一个通道。

sample_rate 使用建议：默认 16000 适配主流 ASR 场景（包括手机录音重采样后的标准格式）。如果上传的音频确实是其他采样率（如 48k 母带、44.1k 音乐），按音频实际采样率显式传，与豆包文档"rate 字段表明音频采样率"语义一致。

双声道独立识别 (`enable_channel_split=true`) 使用场景:

- ☒**Maono / 双麦录音设备**:L 通道一个人 R 通道另一个人,可能同时说话 → 按通道天然分人;每通道各 1 人时无需额外开 `speaker_info`
- ☒**采访 / 播客分轨**:主持人 / 嘉宾各占一轨录音 → 客户端按 channel_id 渲染发言人
- ☒**多人会议双麦录音**(每通道多人轮流说) → **同时启用 `speaker_info=true`**,会在每个通道内再做声纹分人
- ☒**手机单麦录音**:即使硬件输出 stereo,L/R 实际是同一信号复制,启用 channel_split 不会出错但浪费一通道识别费用
- ☒**混音后的成品音频**:L/R 都包含所有人声 → 单独走 `speaker_info=true` 走声纹分离即可,不要开 channel_split(L/R 内容相同会得到两份重复转写)

`speaker_info` vs `enable_channel_split` 选哪个:

- **声纹分人(`speaker_info`)**:任何音频都能用,基于音色识别;**同时说话分不开**;音量/远近差异大时不准

- **通道分人(`enable_channel_split`)**:仅 stereo 输入有效,基于物理通道分离;**同时说话也分得开**,准确度由录音硬件保证;但要求每个通道里只有一个人(或通道内人少)
- **两者并存**:取通道分人的物理精度 + 声纹分人补每通道内多人场景,实测豆包共存正常

出参

```
{
  "code": 0,
  "message": "ok",
  "data": {
    "task_id": "asr_a1b2c3d4e5f6",
    "status": "pending",
    "created_at": "2026-04-06T04:00:00Z"
  }
}
```

字段	类型	说明
data.task_id	string	任务唯一 ID，用于查询结果
data.status	string	初始状态，固定为 `pending`
data.created_at	string	创建时间（UTC ISO 8601）

Demo

cURL:

```
curl -X POST http://8.163.67.66/api/v1/asr/submit \
  -H "Authorization: Bearer <jwt_token>" \
  -F "url=https://your-oss.com/audio/meeting.wav" \
  -F "language=zh-CN" \
  -F "speaker_info=true" \
  -F "channel=1" \
  -F "hotwords=闪克,机器学习,人工智能" \
  -F "trace_id=recording_20260417_meeting_abc"
```

```
const form = new FormData();
form.append("url", "https://your-oss.com/audio/meeting.wav");
form.append("language", "zh-CN");
form.append("speaker_info", "true");
form.append("channel", "1");
form.append("hotwords", "闪克,机器学习,人工智能");          // 可选,v2.5.35+
form.append("trace_id", "recording_20260417_meeting_abc");  // 可选

const resp = await fetch("/api/v1/asr/submit", {
  method: "POST",
  headers: {"Authorization": "Bearer <jwt_token>"},
  body: form
});
const {data: {task_id}} = await resp.json();
```

```
import requests

headers = {"Authorization": "Bearer <jwt_token>"}
resp = requests.post("http://8.163.67.66/api/v1/asr/submit",
    data={
        "url": "https://your-oss.com/audio/meeting.wav",
        "language": "zh-CN",
        "hotwords": "闪克,机器学习,人工智能",       # 可选,v2.5.35+
        "trace_id": "recording_20260417_meeting_abc",  # 可选
    },
    headers=headers)
task_id = resp.json()["data"]["task_id"]
```

1.2 查询转写结果

根据 task_id 查询转写任务的状态和结果。建议每 3-5 秒轮询一次。

> **敏感词过滤：** 服务端默认启用系统敏感词库，返回结果的 `text` 和 `utterances[].text` 中的敏感词可能被替换为 `*`（或清空）。运营可在管理接口 §8 敏感词管理 中调整规则。客户端**无需传任何参数**，过滤逻辑对前端透明。

请求

```
GET /api/v1/asr/tasks/{task_id}
Authorization: Bearer <jwt_token>
```

入参

参数	位置	类型	说明
task_id	URL Path	string	提交时返回的任务 ID

出参

状态：pending（排队中）


```
{
  "code": 0,
  "message": "ok",
  "data": {
    "task_id": "asr_a1b2c3d4e5f6",
    "status": "pending",
    "progress": 0,
    "created_at": "2026-04-06T04:00:00Z",
    "completed_at": null,
    "result": null,
    "error": null,
    "usage": null
  }
}
```


状态：processing（处理中）


```
{
  "code": 0,
  "message": "ok",
  "data": {
    "task_id": "asr_a1b2c3d4e5f6",
    "status": "processing",
    "progress": 40,
    "created_at": "2026-04-06T04:00:00Z",
    "completed_at": null,
    "result": null,
    "error": null,
    "usage": null
  }
}
```

状态：completed（完成）


```
{
  "code": 0,
  "message": "ok",
  "data": {
    "task_id": "asr_a1b2c3d4e5f6",
    "status": "completed",
    "progress": 100,
    "created_at": "2026-04-06T04:00:00Z",
    "completed_at": "2026-04-06T04:01:23Z",
    "result": {
      "text": "[00:00:01.080] 你好，今天我们讨论一下项目进度。\n[00:00:05.200] 好的，我先汇报一下
              开发情况。",
      "utterances": [
        {
          "start_time": 1080,
          "end_time": 4500,
          "text": "你好，今天我们讨论一下项目进度。",
          "speaker": null
        },
        {
          "start_time": 5200,
          "end_time": 8900,
          "text": "好的，我先汇报一下开发情况。",
          "speaker": null
        }
      ]
    },
    "error": null,
    "usage": {
      "duration_ms": 36000,
      "duration_sec": 36.0,
      "provider": "volcengine",
      "model": "bigmodel"
    }
  }
}
```


状态：failed — 音频未识别到语音(v2.5.11+ 错误码 50002)

如果音频未识别到有效语音(纯背景音/音量过低/远离麦克风/声道错配等),任务标 `failed` + `error.code=50002` ,error.message 带豆包原文 + 排查清单。

```
{
  "code": 0,
  "message": "ok",
  "data": {
    "task_id": "asr_14a6d2ccc8c2",
    "status": "failed",
    "progress": 100,
    "created_at": "2026-04-30T07:51:00Z",
    "completed_at": "2026-04-30T07:51:06Z",
    "result": null,
    "error": {
      "code": 50002,
      "message": "音频未识别到有效语音内容(豆包 ASR 状态码 20000003)。豆包原文:\"[Normal
          silence audio] Handle response: no valid speech in audio\"。提交参数:channel=1。常见
          原因排查:(1) 声道不匹配 — stereo 音频必须传 channel=2;(2) 有效语音段太短或前导静默太
          长 — 建议客户端 VAD trim + loudnorm 归一化到 -16 LUFS 后再上传;(3) 音频确实静默/低
          音量/远场;(4) 非人声(纯音乐、噪音)。"
    },
    "usage": {
      "duration_ms": 6310,
      "duration_sec": 6.3,
      "provider": "volcengine",
      "model": "bigmodel"
    }
  }
}
```


甲方接入建议:

1. 用 `error.code === 50002` 单独识别"音频无语音"场景,UI 提示"未识别到语音,请重试"而不是和服务异常一起爆红
2. **客户端 submit 前用 ffprobe / file 命令探测音频真实声道数**: `channels=2` 时显式传 `channel=2` ,否则豆包按 mono 解码 stereo 会乱码识别失败 — 这是**最常见的"假静默"根因**
3. 50002 携带 `usage.duration_sec` (任务跑完只是没识别到内容),可按需用于额度统计

状态：failed — 服务异常


```
{
  "code": 0,
  "message": "ok",
  "data": {
    "task_id": "asr_a1b2c3d4e5f6",
    "status": "failed",
    "progress": 20,
    "created_at": "2026-04-06T04:00:00Z",
    "completed_at": "2026-04-06T04:00:15Z",
    "result": null,
    "error": {
      "code": 50001,
      "message": "ASR 服务超时"
    },
    "usage": null
  }
}
```

字段说明

字段	类型	说明
data.status	string	`pending` / `processing` / `completed` / `failed`
data.progress	int	处理进度 0-100
data.result.text	string	带时间戳的完整转写文本
data.result.utterances	array	分句列表
data.result.utterances[].start_time	int	开始时间（毫秒）
data.result.utterances[].end_time	int	结束时间（毫秒）
data.result.utterances[].text	string	句子文本
data.result.utterances[].speaker	string/null	发言人 ID（开启 `speaker_info=true` 时返回声纹分人 ID;未开启时为 `null` ）。同时开启 `enable_channel_split=true` 时,该 ID 是"通道内"独立编号(每个通道各自从 1 开始)
data.result.utterances[].channel_id	int/null	**(v2.5.35+)** 声道 ID（开启 `enable_channel_split=true` 时返回）。`1` =L (Left) / `2` =R (Right);未启用 channel_split 时为 `null`
data.result.utterances[].volume	float/null	**分句音量**，单位分贝（典型范围 -60 ~ 0 dB）；识别引擎未给出时为 `null`
data.result.utterances[].emotion	string/null	**情绪标签**：`angry` / `happy` / `neutral` / `sad` / `surprise` ；识别引擎未给出时为 `null`
data.usage.duration_ms	int	音频时长（毫秒）
data.usage.duration_sec	float	音频时长（秒）
data.usage.provider	string	服务商（volcengine）
data.usage.model	string	模型（bigmodel）
data.error.code	int	错误码
data.error.message	string	错误描述

> 任务结果保留 **1 小时**后自动清理。超时查询返回 HTTP 404。

Demo（提交 + 轮询完整流程）

```bash
# 1. 提交
TASK_ID=$(curl -s -X POST http://8.163.67.66/api/v1/asr/submit \
  -H "Authorization: Bearer <jwt_token>" \
  -F "url=https://your-oss.com/audio/meeting.wav" \
  -F "language=zh-CN" \
  -F "hotwords=闪克,机器学习,人工智能" | jq -r '.data.task_id')

# 2. 轮询
while true; do
  RESULT=$(curl -s http://8.163.67.66/api/v1/asr/tasks/$TASK_ID \
    -H "Authorization: Bearer <jwt_token>")
  STATUS=$(echo $RESULT | jq -r '.data.status')
  [ "$STATUS" = "completed" ] && echo $RESULT | jq '.data.result.text' && break
  [ "$STATUS" = "failed" ] && echo "Failed" && break
  sleep 3
done
```

```javascript
const headers = {"Authorization": "Bearer <jwt_token>"};

// 1. 提交
const form = new FormData();
form.append("url", "https://your-oss.com/audio/meeting.wav");
form.append("language", "zh-CN");
form.append("hotwords", "闪克,机器学习,人工智能");  // 可选,v2.5.35+
const {data: {task_id}} = await (await fetch("/api/v1/asr/submit", {
  method: "POST", headers, body: form
})).json();

// 2. 轮询
while (true) {
  await new Promise(r => setTimeout(r, 3000));
  const {data: task} = await (await fetch(`/api/v1/asr/tasks/${task_id}`, {headers})).json();
  if (task.status === "completed") { console.log(task.result.text); break; }
  if (task.status === "failed") { console.error(task.error.message); break; }
  console.log(`处理中... ${task.progress}%`);
}
```

```python
import requests, time

headers = {"Authorization": "Bearer <jwt_token>"}

# 1. 提交
resp = requests.post("http://8.163.67.66/api/v1/asr/submit",
    data={
        "url": "https://your-oss.com/audio/meeting.wav",
        "language": "zh-CN",
        "hotwords": "闪克,机器学习,人工智能",  # 可选,v2.5.35+
    },
    headers=headers)
task_id = resp.json()["data"]["task_id"]

# 2. 轮询
while True:
    time.sleep(3)
    task = requests.get(f"http://8.163.67.66/api/v1/asr/tasks/{task_id}",
        headers=headers).json()["data"]
    if task["status"] == "completed":
        print("转写结果:", task["result"]["text"])
        break
    elif task["status"] == "failed":
        print("失败:", task["error"]["message"])
        break
    print(f"处理中... {task['progress']}%")
```

---

2. 实时流式转写 (WebSocket)

通过 WebSocket 实时发送音频流，边说边出文字。适用于实时会议、直播字幕等场景。

> **敏感词过滤：** 服务端默认启用系统敏感词库， `result` 消息中 `text` 和 `utterances[].text` 的敏感词可能被替换为 `*` （或清空）。和录音文件转写共用一套配置，运营可在管理接口 §8 敏感词管理 中调整规则。客户端**无需传任何参数**，过滤逻辑对前端透明。

> ⚠ **`language` 参数在本接口中不生效（豆包服务端约束）：**
>
> 火山引擎官方文档《大模型流式语音识别 API》中 `request.language` 字段明确说明：
>
> > 仅流式输入模式（ `bigmodel_nostream` ）支持此参数，**二遍识别不支持**。
>
> 本服务固定使用 `bigmodel_async` （二遍识别）端点，目的是获得三个能力：
>
> 1. `utterances[].definite=true` 在句末正确翻转（甲方明确要求）
> 2. `utterances[].volume` / `utterances[].emotion` 字段返回（v2.3.1+）
> 3. 会话结束后的精确时长 `duration_sec` 统计
>
> 这些特性都绑定在二遍识别流水线上，与 `language` 参数**互斥**——豆包服务端内部约束，不是可配置的 bug。
>
> **如需强制识别特定语言**（英文 / 日文 / 韩文等），请使用录音文件转写接口 `POST /api/v1/asr/submit` ，该接口走 `/api/v3/auc/bigmodel/submit` 端点，完整支持 25 种语言指定（见附录 B: 支持的语言代码）。
>
> 流式转写场景下豆包默认即支持**中英文自动识别 + 上海话 / 闽南语 / 四川话 / 陕西话 / 粤语**等方言，不传 `language` 通常够用。

连接地址

```
WebSocket ws://8.163.67.66/api/v1/asr/stream?token=<jwt_token>&trace_id=<可选>&hotwords=<可选,英文逗号分隔>
```

参数	必填	位置	说明
token	是（启用鉴权时）	query	JWT 令牌
trace_id	否	query	业务追踪 ID，原样透传到 `asr.stream.completed` / `asr.stream.failed` webhook payload。**实时转写场景特别需要：** 由于本接口不支持 `language` 参数（详见上方说明），甲方若要做"流式中文转写 → 翻译为英文"，必须先收到 `asr.stream.completed` 事件拿到 `text`，再调 `POST /api/v1/llm/translate`。两次调用都带上同一个 `trace_id`，甲方服务端才能把"翻译结果 webhook"和原始转写会话精确关联（仅靠 `client_reference_id` 无法区分同一用户并发的多段录音）。详见文档开头「业务追踪 trace_id」章节。
hotwords	否	query	热词列表（英文逗号分隔），作为本会话的 fallback 热词。`start` 动作中显式传 `hotwords` 字段会**覆盖**这里的 fallback。建议优先在 `start` 动作里传，结构化数组更清晰。

会话生命周期

一个完整的流式转写会话分为 4 个阶段，必须按顺序执行：

```
Phase 1 — 建立连接
  客户端 → WebSocket ws://host/api/v1/asr/stream?token=<jwt>
  服务端 → {"type": "connected", "session_id": "stream_xxx"}

Phase 2 — 开始转写
  客户端 → {"action": "start"}
  服务端 → {"type": "started"}

Phase 3 — 音频流 + 心跳（交替进行）
  客户端 → 二进制帧（PCM 音频，建议每 200ms 一个）
  客户端 → {"action": "ping"}（穿插发送，建议每 15-30 秒）
  服务端 → {"type": "result", ...}（识别结果）
  服务端 → {"type": "pong", ...}（心跳回应）

Phase 4 — 结束
  客户端 → {"action": "stop"}
  服务端 → {"type": "usage", ...}（用量统计）
  连接关闭
```

Phase 1 — 建立连接

建立 WebSocket 连接后，服务端返回 session 信息：

```
{"type": "connected", "session_id": "asr_stream_a1b2c3d4e5f6"}
```

此时 ASR 引擎尚未启动，客户端需发送 start 动作才开始转写。

鉴权失败的处理(JWT 过期 / 无效 / 缺失 token):

服务端**先完成握手再发 close frame** —— 客户端能通过 `onclose` 的 code/reason 精确判断原因,而不是一个含糊的 HTTP 403:

```
服务端 → (握手成功 101 Switching Protocols)
    {"type": "auth_error", "reason": "JWT 已过期"}
    close code=4001, reason="JWT 已过期"
```

close code	含义	推荐客户端动作
`4001`	JWT 缺失 / 无效 / 已过期	**清掉本地缓存的 JWT**,引导用户重新鉴权换 token
其他 4000+	业务层异常(详见 Phase 3 错误列表)	按 reason 重试或向用户展示

典型场景:JWT 的 `JWT_EXPIRE_MINUTES` 默认 120 分钟(即 2 小时),调试台 / 移动 App 若缓存了过期 token 再连 WS,会命中 4001。客户端务必在 `onclose` 里处理这种情况,不能简单 reconnect:

```
ws.onclose = (evt) => {
  if (evt.code === 4001) {
    localStorage.removeItem("jwt_token");
    // 跳到鉴权页重新换 token,然后再 new WebSocket(...)
    return redirectToAuth();
  }
  // 其他情况才走 reconnect 或展示 reason
};
```

Phase 2 — 开始转写

发送 `start` 动作启动 ASR 引擎，可选携带参数：

```
{"action": "start"}
```

带参数的 start：

```
{
  "action": "start",
  "hotwords": ["Maono", "闪克", "PD200X"],
  "language": "zh-CN",
  "speaker_info": true,
  "result_type": "full"
}
```

字段	类型	必填	说明
action	string	**是**	固定为 `"start"`
hotwords	string[]	否	热词列表，提升专有名词识别率，最多 100 tokens
language	string	否	⚠ **本接口不生效**（豆包 `bigmodel_async` 端点约束，详见 §2 顶部说明）。不传即按豆包默认识别（中英文 + 常见中文方言自动判别）。需强制语言请走文件转写接口
speaker_info	bool	否	是否开启发言人分离，默认 `false`
result_type	string	否	结果返回模式：`"full"`（默认）或 `"single"`，详见下方说明

> `trace_id` **不在这里传** —— 它是会话级参数，在 Phase 1 连接 URL 上以 query 参数形式一次性传入，整个会话沿用。 `start` 动作只承载本次转写的运行时配置（hotwords / speaker_info / result_type 等）。

服务端确认后返回：

```
{"type": "started", "result_type": "full"}
```

> 收到 `started` 后才能开始发送音频数据。 `result_type` 字段回传实际使用的结果模式。

Phase 3 — 音频流 + 心跳

发送音频

以二进制帧发送 PCM 音频数据：

参数	要求
采样率	16kHz
位深	16bit
声道	单声道 (mono)
格式	PCM（原始二进制帧）
分包大小	建议 200ms（6400 字节）

心跳

穿插在音频帧之间发送（建议每 15-30 秒）：

```
{"action": "ping"}
```

服务端回应：

```
{"type": "pong", "token_valid": true, "token_expires_in": 3580}
```

字段	类型	说明
token_valid	bool	JWT 是否仍然有效
token_expires_in	int	JWT 剩余有效期（秒），过期为 0

> 收到 `token_valid: false` 时应结束当前会话，重新换取 JWT 后新建连接。

识别结果

中间结果（definite=false）：

```
{
  "type": "result",
  "text": "你好今天我们",
  "utterances": [
    {"text": "你好今天我们", "start_time": 1080, "end_time": 3200, "definite": false, "speaker_id":
        null}
  ],
  "definite": false
}
```


分句定稿（definite=true）：


```
{
  "type": "result",
  "text": "你好，今天我们讨论一下项目进度。明天",
  "utterances": [
    {"text": "你好，今天我们讨论一下项目进度。", "start_time": 1080, "end_time": 4500, "definite": tr
        ue, "speaker_id": 0},
    {"text": "明天", "start_time": 4800, "end_time": 5200, "definite": false, "speaker_id": 1}
  ],
  "definite": true
}
```


result 消息顶层字段：

字段	类型	说明
type	string	固定为 `"result"`
text	string	所有分句拼接后的完整文本
utterances	array	分句数组，见下表
definite	bool	本帧中是否包含已定稿分句（ `any(u.definite)` ）
is_last	bool	是否为**会话最后一帧**。发 `stop` 后二遍识别完成时 = `true` ，正常录音中为 `false`

utterances[] 字段说明：

字段	类型	说明
text	string	识别文本
start_time	int	起始时间（毫秒）
end_time	int	结束时间（毫秒）
definite	bool	是否已定稿
speaker_id	int/null	发言人 ID（开启 `speaker_info` 时返回，未开启时为 `null`）
volume	float/null	**分句音量**，单位分贝（典型范围 -60 ~ 0 dB）；识别引擎未给出时为 `null`
emotion	string/null	**情绪标签**：`angry` / `happy` / `neutral` / `sad` / `surprise`；识别引擎未给出时为 `null`

definite 字段说明：

- `definite: false` — 中间结果，随时被后续覆盖，客户端应**替换**显示
- `definite: true` — 该分句已定稿，不会再改变，客户端应**锁定**显示

is_last 字段说明：

- `is_last: false`（默认，录音中所有帧）— 会话仍在继续
- `is_last: true` — 本帧是服务端完成**二遍识别**后的最终 result 帧，所有 utterances 均为 `definite: true`。客户端收到此标志可立即定稿展示，不必等后续 `usage` 事件
- 顶层 `definite` 为便捷字段，任一 utterance 定稿即为 true

result_type 模式说明

通过 `start` 动作的 `result_type` 参数选择结果返回模式：

模式	utterances 内容	客户端处理方式	适用场景
`full`（默认）	每帧返回**所有历史分句**	每次用最新 utterances **全量重建**显示	通用场景，实现简单
`single`	每帧仅返回**当前句子**	客户端自行**累加**已确认（definite=true）的句子	长时间录音，减少传输数据量

> **full 模式注意：** 随着录音时间增长，每帧数据量会线性增长（包含所有历史句子）。
>
> **single 模式注意：** 客户端需自行维护已确认句子列表。会话结束时，最后一句未确认的草稿可能丢失，建议在收到 `usage` 事件时将最后的草稿也纳入结果。

Phase 4 — 结束

发送 stop 结束录音：

```
{"action": "stop"}
```

发完 `stop` 后，服务端会触发**二遍识别**对整段音频做 offline 复盘，产出更准确的最终结果。客户端应**继续监听**消息直到收到结束信号，**不要立即关闭 WebSocket**。

典型消息时序（stop 之后）：

```
客户端 → {"action":"stop"}
          （服务端二遍识别中，通常 < 3 秒，长录音可能更久）
服务端 → {"type":"result", "is_last": true, "utterances":[ ... 全部 definite ... ]}
服务端 → {"type":"usage", "duration_sec": 15.0, ...}
       （服务端主动关闭连接）
```

两个结束信号

信号	含义	建议处理
`result` 帧带 `is_last: true`	二遍识别完成，所有文本已定稿	立即用该帧 utterances 做最终显示
`usage` 事件	服务端关闭前推送的用量统计	记账、展示时长等

推荐实现：**收到 `is_last: true` 或 `usage` 任一信号即可认为会话结束**。为防网络异常，客户端可设一个兜底超时（建议 30 秒）。

usage 字段

```
{
  "type": "usage",
  "duration_ms": 15000,
  "duration_sec": 15.0,
  "provider": "volcengine",
  "model": "bigmodel"
}
```

错误：

```
{
  "type": "error",
  "code": 45000151,
  "message": "[Invalid audio format] ..."
}
```

音频格式要求

参数	值	说明
采样率	16000 Hz	16kHz
位深	16 bit	有符号整数
声道	1	单声道
编码	PCM	原始 PCM，无压缩

Demo

> **full 模式**：每帧 utterances 包含所有历史句子，客户端直接全量重建显示。
>
> **single 模式**：每帧仅返回当前句子，客户端自行累加已确认句子。适合长时间录音。

JavaScript（浏览器）：

```
const ws = new WebSocket("ws://8.163.67.66/api/v1/asr/stream?token=<jwt>");
ws.binaryType = "arraybuffer";

let audioCtx, processor, micStream;

// JWT 过期 / 无效时服务端发 close code=4001,此处清掉缓存并引导重新鉴权
ws.onclose = (evt) => {
  if (evt.code === 4001) {
    localStorage.removeItem("jwt_token");
    alert("JWT 已过期,请重新获取:" + evt.reason);
    // 这里跳回登录/鉴权流程重新换 token,不要直接 reconnect
    return;
  }
  // 其他关闭原因按需处理
};

ws.onmessage = (e) => {
  const msg = JSON.parse(e.data);

  if (msg.type === "connected") {
    console.log("Session:", msg.session_id);
    ws.send(JSON.stringify({
      action: "start",
      result_type: "full",          // 全量模式
      speaker_info: true,           // 开启发言人分离
      hotwords: ["Maono", "闪克", "PD200X"]
    }));
  }

  else if (msg.type === "started") {
    console.log("ASR 已启动，模式:", msg.result_type);
    startMic();
    setInterval(() => ws.send(JSON.stringify({action: "ping"})), 20000);
  }

  // full 模式：每帧用最新 utterances 全量重建显示
  else if (msg.type === "result") {
    const display = document.getElementById("result");
    display.innerHTML = "";
    for (const u of msg.utterances || []) {
      const div = document.createElement("div");
      if (u.speaker_id != null) {
        const badge = document.createElement("span");
        badge.textContent = `S${u.speaker_id + 1}`;
        badge.style.cssText = "font-size:11px;font-weight:700;padding:1px 5px;border-radius:
4px;margin-right:6px;background:#dbeafe;color:#1d4ed8";
        div.appendChild(badge);
      }
      div.appendChild(document.createTextNode(u.text + (u.definite ? "" : " ...")));
      if (!u.definite) div.style.cssText = "opacity:0.55;font-style:italic";
      display.appendChild(div);
    }
```

```
  }

  else if (msg.type === "pong") {
    if (!msg.token_valid) {
      console.warn("Token 已过期，需重新换取 JWT");
      stop();
    }
  }

  else if (msg.type === "usage") {
    console.log(`时长: ${msg.duration_sec}s`);
  }
};

// 采集麦克风 → 降采样 16kHz PCM → 发送
async function startMic() {
  micStream = await navigator.mediaDevices.getUserMedia({ audio: true });
  audioCtx = new AudioContext({ sampleRate: 48000 });
  const source = audioCtx.createMediaStreamSource(micStream);
  processor = audioCtx.createScriptProcessor(4096, 1, 1);
  const ratio = audioCtx.sampleRate / 16000;

  processor.onaudioprocess = (e) => {
    const input = e.inputBuffer.getChannelData(0);
    const len = Math.floor(input.length / ratio);
    const pcm = new Int16Array(len);
    for (let i = 0; i < len; i++) {
      const s = Math.max(-1, Math.min(1, input[Math.floor(i * ratio)]));
      pcm[i] = s < 0 ? s * 0x8000 : s * 0x7FFF;
    }
    ws.send(pcm.buffer);
  };
  source.connect(processor);
  processor.connect(audioCtx.destination);
}

function stop() {
  ws.send(JSON.stringify({ action: "stop" }));
  micStream?.getTracks().forEach(t => t.stop());
  audioCtx?.close();
}
```

```
const ws = new WebSocket("ws://8.163.67.66/api/v1/asr/stream?token=<jwt>");
ws.binaryType = "arraybuffer";

const confirmedLines = [];  // 已确认的句子列表

// JWT 过期处理(同 full 模式)
ws.onclose = (evt) => {
  if (evt.code === 4001) {
    localStorage.removeItem("jwt_token");
    alert("JWT 已过期,请重新获取:" + evt.reason);
    return;
  }
};

ws.onmessage = (e) => {
  const msg = JSON.parse(e.data);

  if (msg.type === "connected") {
    ws.send(JSON.stringify({
      action: "start",
      result_type: "single",        // 增量模式
      speaker_info: true
    }));
  }

  else if (msg.type === "started") {
    console.log("ASR 已启动，模式:", msg.result_type);
    startMic();
    setInterval(() => ws.send(JSON.stringify({action: "ping"})), 20000);
  }

  // single 模式：累加已确认句子，草稿单独渲染
  else if (msg.type === "result") {
    for (const u of msg.utterances || []) {
      if (u.definite && u.text) {
        confirmedLines.push({ text: u.text, speakerId: u.speaker_id });
      }
    }
    const draft = (msg.utterances || []).find(u => !u.definite);

    const display = document.getElementById("result");
    display.innerHTML = "";
    for (const line of confirmedLines) {
      const div = document.createElement("div");
      if (line.speakerId != null) div.textContent = `[S${line.speakerId + 1}] `;
      div.textContent += line.text;
      display.appendChild(div);
    }
    if (draft?.text) {
      const div = document.createElement("div");
      div.style.cssText = "opacity:0.55;font-style:italic";
      div.textContent = draft.text + " ...";
```

```
      display.appendChild(div);
    }
  }

  // 会话结束，收尾
  else if (msg.type === "usage") {
    console.log(`时长: ${msg.duration_sec}s`);
    const finalText = confirmedLines.map(l => l.text).join("\n");
    console.log("最终结果:\n" + finalText);
  }
};

// startMic() / stop() 同 full 模式，此处省略
```

Python（硬件端 / 服务端调用）：

```
import asyncio, json, websockets
from websockets.exceptions import ConnectionClosed

class JWTExpiredError(Exception):
    """服务端发来的 WS close code=4001,需要调用方重新换 JWT。"""

async def stream_asr(audio_file: str, token: str):
    uri = f"ws://8.163.67.66/api/v1/asr/stream?token={token}"

    try:
        async with websockets.connect(uri) as ws:
            msg = json.loads(await ws.recv())
            # 鉴权失败时服务端先发 auth_error 再 close,客户端这里能收到
            if msg.get("type") == "auth_error":
                raise JWTExpiredError(msg.get("reason", "JWT 过期"))
            assert msg["type"] == "connected"
            print(f"Session: {msg['session_id']}")

            await ws.send(json.dumps({
                "action": "start",
                "result_type": "full",
                "speaker_info": True,
                "hotwords": ["Maono", "PD200X"],
                "language": "zh-CN"
            }))
            msg = json.loads(await ws.recv())
            assert msg["type"] == "started"

            CHUNK = 6400  # 200ms @ 16kHz 16bit mono
            with open(audio_file, "rb") as f:
                while chunk := f.read(CHUNK):
                    await ws.send(chunk)
                    await asyncio.sleep(0.2)
                    try:
                        resp = json.loads(await asyncio.wait_for(ws.recv(), 0.01))
                        if resp["type"] == "result":
                            # full 模式：每帧全量，直接打印所有 utterances
                            for u in resp.get("utterances", []):
                                tag = "✓" if u.get("definite") else "..."
                                speaker = f"[S{u['speaker_id'] + 1}] " if u.get("speaker_id") is not None else ""
                                print(f"  [{tag}] {speaker}{u['text']}")
                    except asyncio.TimeoutError:
                        pass

            await ws.send(json.dumps({"action": "stop"}))
            async for raw in ws:
                msg = json.loads(raw)
                if msg["type"] == "usage":
                    print(f"时长: {msg['duration_sec']}s")
                    break
    except ConnectionClosed as e:
        # 兜底:服务端也可能不发 auth_error 直接 close 4001
```

```
        if e.code == 4001:
            raise JWTExpiredError(e.reason or "JWT 过期")
        raise

# 上层调用侧处理 JWT 过期:捕获后重新换 token 再重试
try:
    asyncio.run(stream_asr("audio.pcm", "<jwt_token>"))
except JWTExpiredError as e:
    print(f"需重新换 token: {e}")
    # new_token = refresh_jwt()
    # asyncio.run(stream_asr("audio.pcm", new_token))
```

```
import asyncio, json, websockets

async def stream_asr(audio_file: str, token: str):
    uri = f"ws://8.163.67.66/api/v1/asr/stream?token={token}"
    confirmed = []

    async with websockets.connect(uri) as ws:
        msg = json.loads(await ws.recv())
        assert msg["type"] == "connected"

        await ws.send(json.dumps({
            "action": "start",
            "result_type": "single",
            "speaker_info": True,
            "hotwords": ["Maono", "PD200X"],
            "language": "zh-CN"
        }))
        msg = json.loads(await ws.recv())
        assert msg["type"] == "started"

        CHUNK = 6400
        with open(audio_file, "rb") as f:
            while chunk := f.read(CHUNK):
                await ws.send(chunk)
                await asyncio.sleep(0.2)
                try:
                    resp = json.loads(await asyncio.wait_for(ws.recv(), 0.01))
                    if resp["type"] == "result":
                        # single 模式：只累加 definite 句子
                        for u in resp.get("utterances", []):
                            if u.get("definite") and u.get("text"):
                                speaker = f"[S{u['speaker_id'] + 1}] " if u.get("speaker_id") is not None else ""
                                confirmed.append(f"{speaker}{u['text']}")
                                print(f"  [✓] {confirmed[-1]}")
                        draft = next((u for u in resp.get("utterances", []) if not u.get("definite")), None)
                        if draft and draft.get("text"):
                            print(f"  [...] {draft['text']}")
                except asyncio.TimeoutError:
                    pass

        await ws.send(json.dumps({"action": "stop"}))
        async for raw in ws:
            msg = json.loads(raw)
            if msg["type"] == "usage":
                print(f"时长: {msg['duration_sec']}s")
                print(f"最终结果 ({len(confirmed)} 句):")
                for line in confirmed:
                    print(f"  {line}")
                break

asyncio.run(stream_asr("audio.pcm", "<jwt_token>"))
```

3. 智能总结

对文本进行结构化总结。内置多套模版（通用场景、单人播客、多人访谈），客户端通过 `template` 参数选择；后续会逐步补充更多语言和场景模版，模版 ID 稳定不变。

请求

```
POST /api/v1/llm/summarize
Content-Type: multipart/form-data
Authorization: Bearer <jwt_token>
```

入参

参数	类型	必填	默认值	说明
text	string	与 `utterances` 二选一	-	待总结文本（与 `utterances` 二选一传任意一个，两者都传时 `utterances` 优先）
utterances	string (JSON)	与 `text` 二选一	-	可选：JSON 字符串，结构同 §5 智能分章 入参（`[{start_time, end_time, text, speaker?, volume?, emotion?}]`）。**当前 LLM 仅消费 text，speaker/volume/emotion 仅作结构兼容透传，不影响输出**——便于甲方把转写结果原样喂入
template	string	否	`general`	总结模版 ID，见 3.2 模版列表；未知 ID 返回 400
trace_id	string	否	`null`	业务追踪 ID，原样透传到 webhook payload

内置模版一览：

ID	名称	适用场景
`general`	通用场景总结	文章、报告、邮件、笔记、会议、对话等任意文本（默认）
`podcast_solo`	单人播客创作	单人讲述、知识分享、独立创作类播客（含金句/运营素材/拟题）
`podcast_interview`	多人对话/访谈播客	访谈、圆桌、对谈（含观点交锋/嘉宾立场/后期剪辑备忘）

> 模版列表可通过 `GET /api/v1/llm/summary-templates` 动态获取，推荐客户端拉取后渲染下拉框，以便未来新增模版时无需改前端。

出参

统一信封 JSON 一次性响应(v2.5.7 起新增 `title` / `abstract` / `summary` 三字段,方便直接入库/渲染):

```
{
  "code": 0,
  "message": "ok",
  "data": {
    "task_id": "sum_abc123def456",
    "template": "general",
    "title": "Q2 产品评审会议",
    "abstract": "本次会议决定 Q2 优先做自动翻译,分章次之,转写速度最后。",
    "summary": "###  重点脉络\n- 用户调研:70% 希望翻译...\n###  价值沉淀\n...\n###  结论与行
               动指引\n...",
    "usage": {
      "provider": "dashscope",
      "model": "qwen3.6-plus",
      "input_tokens": 120,
      "output_tokens": 380
    }
  }
}
```

三字段含义(v2.5.7+):

字段	类型	说明
`title`	string	一句话标题(≤20 字),适合作为录音条目名直接入库
`abstract`	string	一句话摘要(50-80 字),列表预览用
`summary`	string	详细结构化 Markdown,包含重点脉络/价值沉淀/行动指引等完整章节

> **v2.7.0 起删 `result` 兼容字段**:之前 `result` 是 `# {title}\n\n> {abstract}\n\n{summary}` 服务端拼好的整块 Markdown,作为老集成的兜底。v2.7.0 起客户端按需自行用三字段拼回(3 行 JS 即可),不再依赖服务端兼容字段。

输出语言行为(v2.10.0 三层防护): 服务端在调用 LLM 前自动检测输入文本的源语言(`detect_source_lang` 复用翻译接口同款逻辑),通过三层防护确保 `title` / `abstract` / `summary` 三个字段以及 `summary` 内所有 markdown 章节标题、列表项、关键词、原声金句**全部使用输入文本的主体语言**。

三层防护(从主到兜底):

1. **主修复 — 运行时翻译模板**(日韩场景):服务端用 `deepseek-v4-flash` 把甲方模板字符串翻译为目标语言版,LLM 看到的就是目标语言模板,模型层就不会出错字符。LRU 缓存 256 条,首次 ~10s cold start,后续命中 0 开销。
2. **第二层 — 服务端追加段**:强制输出语言指令 append 到模板末尾,保证英 / 法 / 西等其他语言场景也稳定匹配输入。
3. **第三层 — 字符替换兜底**:LLM 偶发输出简体字独有 codepoint 时替换为日语 JIS 字形(实测 v2.10.0 后几乎不再触发,但留着以防意外)。

- 之前(≤v2.8.x):靠 prompt 头部"输出语言规则"的弱约束,qwen-plus 概率性忽略 — 实测全英文 BBC 播稿偶发输出整段中文摘要(2026-05-26 甲方反馈 case)
- v2.9.0:服务端追加段(第二层) + 字符替换兜底(第三层),修复了语言一致性但靠字符表兜底
- v2.10.0:增加运行时翻译模板(第一层主修复),模型层就 0 错字符,不依赖字符兜底。日韩章节翻译更地道(`価値の抽出` vs 之前的 `価値沈殿`)。
- 中文输入零影响,行为跟之前一致
- 服务端 prompt 跟甲方业务模板**物理分离**,模板内容字符级零修改
- 性能影响:ja/ko 首次请求 ~22s(含 ~10s 模板翻译 cold start),后续缓存命中等同 v2.9.0(~12s)。中 / 英 / 法 / 其他语言无变化。

- 可通过 env `SUMMARY_RUNTIME_TRANSLATE_TEMPLATE=false` 一键回退到 v2.9.0 两层防护行为

输出结构: 由所选 `template` 决定。每套模版都是 Markdown 格式,字段固定、层次清晰,可以直接渲染到 UI 或做二次解析。

以下是各模版的典型输出片段（仅展示一级结构，实际输出包含更多细节）：

```
### 一句话概要
### 重点脉络
### 价值沉淀
### 结论与行动指引
** 关键词**
```

```
### 本期主旨
### 内容脉络与干货
### 高光时刻（可直接用于宣发）
### 运营素材库
** 检索标签**
```

```
### 对谈概览
### 话题流转与交锋拆解
### 嘉宾/参与者立场速览
### 名场面预警（后期剪辑重点）
### ✂ 后期制作备忘录
** 检索标签**
```

Demo

```
curl -X POST http://8.163.67.66/api/v1/llm/summarize \
  -H "Authorization: Bearer <jwt_token>" \
  -F "text=今天开会讨论了三个议题..." \
  -F "template=general" \
  -F "trace_id=recording_20260417_meeting_abc"
```

```
const form = new FormData();
form.append("text", "今天开会讨论了三个议题...");
form.append("template", "general");   // general / podcast_solo / podcast_interview
form.append("trace_id", "recording_20260417_meeting_abc");  // 可选

const resp = await fetch("http://8.163.67.66/api/v1/llm/summarize", {
  method: "POST",
  headers: { "Authorization": "Bearer <jwt_token>" },
  body: form,
});
const result = await resp.json();
if (result.code === 0) {
  const d = result.data;
  console.log(d.title, d.abstract, d.summary);  // 三字段独立
}
```

```
import requests

resp = requests.post(
    "http://8.163.67.66/api/v1/llm/summarize",
    headers={"Authorization": "Bearer <jwt_token>"},
    data={"text": "今天开会讨论了三个议题...", "template": "general"},
)
result = resp.json()
if result["code"] == 0:
    d = result["data"]
    print(d["title"], d["abstract"], d["summary"])  # 三字段独立
```

> 模版切换：把上述任一示例的 `template=general` 改成 `podcast_solo` 或 `podcast_interview` 即可——其余参数不变。

3.1 从转写结果衔接到总结

总结接口最常见的上游就是录音文件 / 实时流式转写。两边的产物都可以喂给总结，但**字段对应关系略有差异**，这一节给出标准接法，避免甲方踩坑。

选 `text` 还是 `utterances`？

入参	适用场景	优劣
`text`（拼好的纯文本）	不关心发言人 / 时间戳 / 情绪，只要拿一段总结	最省事，**但丢失结构信息**——未来若我们升级"按发言人归纳观点"等能力，纯文本场景拿不到
`utterances`（结构化数组）	想保留发言人 / 时间戳 / 情绪以便未来扩展	推荐用法，**当前 LLM 仅消费 text 字段**，其余字段是"兼容透传"——客户端代码升级零成本

> 两者**二选一**。两者都传时 `utterances` 优先生效。

A. 录音文件转写 → 总结

文件转写的 `utterances[]` 字段名与总结接口**完全一致**，可以原样透传。

完整示例（utterances 路径，保留结构信息）：

```
// 1. 查询转写任务结果
const taskRes = await fetch(`http://8.163.67.66/api/v1/asr/tasks/${taskId}`, {
  headers: { "Authorization": `Bearer ${jwt}` },
});
const utterances = (await taskRes.json()).data.result.utterances;

// 2. 喂给总结接口
const form = new FormData();
form.append("utterances", JSON.stringify(utterances));  // ← JSON 字符串
form.append("template", "podcast_interview");

const sumRes = await fetch("http://8.163.67.66/api/v1/llm/summarize", {
  method: "POST",
  headers: { "Authorization": `Bearer ${jwt}` },
  body: form,
});
const summary = await sumRes.json();
console.log(summary.data.result);
```

```
import json, requests

# 1. 拿到文件转写结果
asr = requests.get(f"http://8.163.67.66/api/v1/asr/tasks/{task_id}",
                   headers={"Authorization": f"Bearer {jwt}"}).json()
utterances = asr["data"]["result"]["utterances"]

# 2. 原样透传到总结
summary = requests.post(
    "http://8.163.67.66/api/v1/llm/summarize",
    headers={"Authorization": f"Bearer {jwt}"},
    data={
        "utterances": json.dumps(utterances, ensure_ascii=False),
        "template": "podcast_interview",
    },
).json()
print(summary["data"]["result"])
```

最简路径（拼成纯文本传 text）：

```
const text = utterances.map(u => u.text).filter(Boolean).join("\n");
const form = new FormData();
form.append("text", text);
await fetch("http://8.163.67.66/api/v1/llm/summarize", {
  method: "POST",
  headers: { "Authorization": `Bearer ${jwt}` },
  body: form,
});
```

```
text = "\n".join(u["text"] for u in utterances if u["text"])
summary = requests.post(
    "http://8.163.67.66/api/v1/llm/summarize",
    headers={"Authorization": f"Bearer {jwt}"},
    data={"text": text},
).json()
```

B. 实时流式转写 → 总结

流式转写的 `utterances[]` 与文件转写有两点差异，客户端必须做适配：

差异	流式	总结接口能认	客户端需要做
发言人字段名	`speaker_id`	`speaker`	**重命名**
`definite` 字段	有（true/false）	无	客户端自己**只取 `definite=true` 的句子**
草稿态文本	会随会话变化	-	累积时**忽略 `definite=false`**，避免重复

完整示例（WS 累积 + 字段重命名 + 提交）：

```
const ws = new WebSocket(`ws://8.163.67.66/api/v1/asr/stream?token=${jwt}`);
ws.binaryType = "arraybuffer";

const finalUtts = [];

ws.onmessage = async (e) => {
  const msg = JSON.parse(e.data);

  if (msg.type === "connected") {
    ws.send(JSON.stringify({ action: "start", result_type: "full" }));
    // ... 后续按 §2 Phase 3 持续 send 二进制 PCM 帧 ...
  }

  if (msg.type === "result") {
    for (const u of msg.utterances || []) {
      if (!u.definite) continue;          // 跳过草稿态，避免重复
      finalUtts.push({
        start_time: u.start_time,
        end_time: u.end_time,
        text: u.text,
        speaker: u.speaker_id,            // ⚠ 关键：speaker_id → speaker 重命名
        volume: u.volume,
        emotion: u.emotion,
      });
    }
  }

  if (msg.type === "usage") {             // 会话结束信号 → 提交总结
    await submitSummarize(finalUtts);
    ws.close();
  }
};

async function submitSummarize(utts) {
  const form = new FormData();
  form.append("utterances", JSON.stringify(utts));
  form.append("template", "general");

  const res = await fetch("http://8.163.67.66/api/v1/llm/summarize", {
    method: "POST",
    headers: { "Authorization": `Bearer ${jwt}` },
    body: form,
  });
  const summary = await res.json();
  console.log(summary.data.result);
}
```

```
import json, asyncio, websockets, requests

final_utts = []

async def collect():
    async with websockets.connect(f"ws://8.163.67.66/api/v1/asr/stream?token={jwt}") as ws:
        await ws.send(json.dumps({"action": "start", "result_type": "full"}))
        # ... 发送音频 ...
        async for raw in ws:
            msg = json.loads(raw)
            if msg.get("type") != "result":
                if msg.get("type") == "usage": break
                continue
            for u in msg.get("utterances", []):
                if not u.get("definite"): continue
                final_utts.append({
                    "start_time": u["start_time"],
                    "end_time": u["end_time"],
                    "text": u["text"],
                    "speaker": u.get("speaker_id"),    # ← 改名
                    "volume": u.get("volume"),
                    "emotion": u.get("emotion"),
                })

asyncio.run(collect())

# 提交总结
summary = requests.post(
    "http://8.163.67.66/api/v1/llm/summarize",
    headers={"Authorization": f"Bearer {jwt}"},
    data={
        "utterances": json.dumps(final_utts, ensure_ascii=False),
        "template": "general",
    },
).json()
```

最简路径（拼成纯文本）：

```
const text = finalUtts.map(u => u.text).filter(Boolean).join("");
// form.append("text", text); 其余同上方文件转写示例的最简路径
```

```
text = "".join(u["text"] for u in final_utts if u["text"])
# 提交方式与文件转写的纯文本路径相同
```

字段对照表

字段	文件转写 `result.utterances[i]`	流式 `result` 帧的 `utterances[i]`	总结/分章接口期望
`start_time` (ms)	☒	☒	☒
`end_time` (ms)	☒	☒	☒
`text`	☒	☒	☒
发言人	`speaker` (string/null)	`speaker_id` (string/null)	**`speaker`**
`volume` (dB)	☒	☒	☒
`emotion`	☒	☒	☒
`definite`	—	☒	会被忽略（不报错）

> **唯一需要客户端转换的字段是 `speaker_id` → `speaker`。** 其他字段完全对齐，可以无脑透传。

长度上限

总结接口本身不做截断，但 Qwen 模型有约 128K tokens 输入上限。**超长会议**（2 小时以上的转写文本）建议甲方在客户端**先按章节切分**，每段单独总结后再二次合并；或者使用 § 5 智能分章接口先分章、再每章独立总结。

如果文本超出 Qwen 上限，服务端会推 `summarize.failed` 事件（错误码 `51001`），HTTP 返回 502。

3.2 总结模版列表

查询当前服务支持的所有总结模版及其可用语言。建议客户端启动时拉取一次，缓存后渲染下拉选择，新增模版或新增语言时**无需改客户端代码**。

无需鉴权。该接口只返回模版元数据（ID/名称/描述/支持语言），不涉及用户数据，客户端在用户登录前也可以调用。

请求：

```
GET /api/v1/llm/summary-templates
```

响应：


```
{
  "code": 0,
  "message": "ok",
  "data": {
    "templates": [
      {
        "id": "general",
        "name": "通用场景总结",
        "description": "适用于任意文本的结构化总结，自动提炼核心信息与行动指引",
        "applicable": "文章、报告、邮件、笔记、会议、对话等任意文本",
        "available_langs": ["zh"]
      },
      {
        "id": "podcast_solo",
        "name": "单人播客创作",
        "description": "为单人独白类播客提供创作型总结，含金句提取、运营素材、拟题参考",
        "applicable": "单人讲述、知识分享、独立创作类播客",
        "available_langs": ["zh"]
      },
      {
        "id": "podcast_interview",
        "name": "多人对话/访谈播客",
        "description": "为多人对话或访谈类播客提供总结，含观点交锋、嘉宾立场、后期剪辑备忘",
        "applicable": "访谈、圆桌、对谈、多人讨论类播客",
        "available_langs": ["zh"]
      }
    ],
    "default_template": "general",
    "default_lang": "zh"
  }
}
```


字段说明：

字段	说明
`templates[].id`	模版 ID，用于 `POST /summarize` 的 `template` 参数
`templates[].name`	模版中文名，用于前端展示
`templates[].description`	模版用途简介
`templates[].applicable`	推荐的适用场景
`templates[].available_langs`	该模版当前支持的语言数组（未来扩展多语言时会增长）
`default_template`	默认模版 ID，与 `POST /summarize` 的默认值一致
`default_lang`	默认语言，与 `POST /summarize` 的默认值一致

> 模版的**管理操作**（热重载、CRUD、创建新模版等）见 § 7 模版管理。

---

4. 智能翻译

将文本翻译为指定语言，保留时间戳和发言人标注。

请求

```
POST /api/v1/llm/translate
Content-Type: multipart/form-data
Authorization: Bearer <jwt_token>
```

入参

参数	类型	必填	默认值	说明
text	string	**是**	-	待翻译文本
target_lang	string	否	`en`	目标语言代码
trace_id	string	否	`null`	业务追踪 ID，原样透传到 webhook payload

目标语言代码：

代码	语言	代码	语言
zh	中文	en	英文
ja	日语	ko	韩语
fr	法语	de	德语
es	西班牙语	ru	俄语
th	泰语	id	印尼语

出参

统一信封 JSON 一次性响应。v2.5.27 起 LLM 直接输出 JSON,服务端解析后填 `title` / `abstract` / `translation` 三字段独立(不再用 markdown 头部拆分):

```
{
  "code": 0,
  "message": "ok",
  "data": {
    "task_id": "trans_abc123def456",
    "target_lang": "en",
    "title": "Q2 Product Plan Review",
    "abstract": "The team prioritized auto-translate for Q2.",
    "translation": "Today we will discuss the project progress.",
    "usage": {
      "provider": "dashscope",
      "model": "deepseek-v4-flash",
      "input_tokens": 138,
      "output_tokens": 24
    },
    "was_translated": true,
    "source_lang_detected": "中文",
    "detect_method": "charset"
  }
}
```


字段含义(均为目标语言,与翻译方向一致):

字段	类型	说明
`title`	string	LLM 对内容生成的一句话标题(≤20 字),v2.11.2 起服务端 prompt 加 forced_lang 强制段,确保 100% 用 `target_lang` 生成(之前 ds-flash 在中文输入下偶发出中文 title)
`abstract`	string	LLM 对内容生成的一句话摘要(10-80 字),v2.11.2 同款修复确保 100% 用 `target_lang` 生成
`translation`	string	**正文翻译结果**(v2.8.0 起不含 `# title\n> abstract\n` 头部 — 跟 `title` / `abstract` 字段语义重复, 客户端拼显示会重复两次)。保留正文内的所有 markdown 结构(章节标题 / 时间戳 / 发言人标注 / 列表 / blockquote)。**客户端拼完整 markdown 用 `# {title}\n\n> {abstract}\n\n{translation}` 三字段拼接。**
`was_translated`	bool	**v2.6.0+**。是否真的执行了跨语翻译。`false` 表示源语言已经等于 `target_lang` ,服务端走 self-translate 短路 — `translation` 字段等于原文剥头后内容(若原文是带 `# title\n> abstract\n` 的总结 markdown, 头部同样被剥, 跟跨语场景口径一致),只额外生成了 title/abstract。
`source_lang_detected`	string	**v2.6.0+**。服务端检测到的源语言(中文/英文/日语/韩语/法语/西班牙语/葡萄牙语/越南语/俄语/德语/印尼语/泰语/意大利语),或 `"unknown"` 。配合 `was_translated` 让客户端能解释"为什么翻译没改"。
`detect_method`	string	**v2.6.1+**。检测源语言用的路径: `"charset"` (Unicode 字符块强信号)/ `"langdetect"` (本地 n-gram)/ `"llm"` (ds-flash 兜底)/ `"fallback"` (LLM 失败降级)。统计 LLM 兜底占比看成本结构用。

> **v2.7.0 起删 `result` 兼容字段**:之前 `result` 是 `# {title}\n\n> {abstract}\n\n{translation}` 服务端拼好的整块 Markdown。客户端按需自行用三字段拼回(3 行 JS 即可),不再依赖服务端兼容字段。

⚠ **BREAKING(v2.8.0)**: `translation` 字段不再包含 `# title\n> abstract\n` markdown 头部, 只含正文翻译。

- 旧版(≤ v2.7.x): `translation` 是 body 调用整段翻译,客户端送总结 markdown 时头部 `# title\n> abstract\n` 也被翻译并跟在前面 — 跟独立的 `title` / `abstract` 字段语义重复,客户端拼显示会重复两次。
- 新版(v2.8.0+):服务端在 `services/llm.py` 用正则剥掉头部, `translation` 只保留正文(从 `### 章节` 或正文段开始)。Fallback safe:头部格式不符合预期(裸文本翻译 / 缺 title 或 abstract / LLM 输出抖动)时原样返回,不会破坏数据。

- **迁移指引**:如果之前客户端直接拿 `translation` 整段渲染,现在会缺标题和摘要。改成读 `title` / `abstract` 字段,自行拼 `# {title}\n\n> {abstract}\n\n{translation}` (3 行 JS 即可)。如果之前已经分别取了三字段,本次升级零影响。

关键变更(v2.5.27+):LLM 直接输出 JSON, `translation` 永远是完整翻译结果,即使原文很短也有内容(不再像 v2.5.7~v2.5.26 时期出现"标题+摘要无正文" → `translation` 为空字符串的边界 bug)。

关键变更(v2.6.0+):服务端在入口加了源语言检测 + 语言×长度路由分流。

- **self-translate 短路**:若客户端把已经是 `target_lang` 的文本传进来(典型场景:转写中文音频后 `target_lang=zh` 又调一次翻译),服务端不再花 ~30s 跑无意义的 body 翻译调用。直接返回原文 + LLM 生成的 title/abstract,响应里 `was_translated=false` 、 `source_lang_detected` 等于 `target_lang` 。客户端要展示"无需翻译"提示就读 `was_translated` 。
- **跨语场景路由**:跨语翻译会按"语言难度 × 文本长度 × 韩语俚语词典"三维选模型。易语言(中/英/西/法/葡)短文本走 ds-flash(~10s, ~0.5x 单价), 难语言长文本 / 易语言超长 / 韩语短文本含俚语 走 ds-pro。客户端无感知, 但实际 `usage.model` 字段会反映本次实际用的模型。
- **webhook payload 加 3 个业务字段(v2.6.1+)**: `translate.completed` 事件 data 加 `was_translated` / `source_lang_detected` / `detect_method` ,跟 HTTP response 对齐口径(避免甲方客户端通道跟服务端通道两个数据源对账)。其他诊断字段(`detect_tokens` / `model_chosen` / `dict_hit` / `length_bucket`)仍只走 audit log,不进 webhook。 `usage.input_tokens` / `output_tokens` 在 LLM 检测兜底场景下涵盖检测调用消耗,跟 DashScope 账单一致。详见下方 webhook 章节字段表。

强制输出(v2.5.8 起延续):**不论原文长短**(哪怕只有一个单词),3 字段都有非空内容。即使 LLM 偶发不规范输出(多余文本/缺字段),服务端兜底逻辑确保 `translation` 至少回退到 LLM 原始输出,接口不崩。

Demo

```
curl -X POST http://8.163.67.66/api/v1/llm/translate \
  -H "Authorization: Bearer <jwt_token>" \
  -F "text=你好，今天我们讨论一下项目进度。" \
  -F "target_lang=en" \
  -F "trace_id=recording_20260417_meeting_abc"
```

```javascript
const form = new FormData();
form.append("text", "你好，今天我们讨论一下项目进度。");
form.append("target_lang", "en");
form.append("trace_id", "recording_20260417_meeting_abc"); // 可选

const resp = await fetch("http://8.163.67.66/api/v1/llm/translate", {
  method: "POST",
  headers: {"Authorization": "Bearer <jwt_token>"},
  body: form
});
const result = await resp.json();
if (result.code === 0) {
  console.log(result.data.translation);
}
```

```python
import requests

resp = requests.post("http://8.163.67.66/api/v1/llm/translate",
    data={
        "text": "你好，今天我们讨论一下项目进度。",
        "target_lang": "en",
        "trace_id": "recording_20260417_meeting_abc",  # 可选
    },
    headers={"Authorization": "Bearer <jwt_token>"})

result = resp.json()
if result["code"] == 0:
    print(result["data"]["translation"])
```

---

5. 智能分章

根据带时间戳的转录片段（utterances），自动按话题切分出 3-10 个章节，每个章节含起止时间、标题和一句话摘要。常用于长播客/访谈/录播的目录索引、视频分段跳转。

> **v2.4.1 行为变更**：分章接口改为**异步任务模式**（与文件转写一致）。`POST` 提交后立即返回 `task_id`，后台 Celery worker 执行 LLM 分章，完成后通过 webhook 推送或轮询查询获取结果。**旧版同步响应（直接返回 chapters）已移除。**

5.1 提交分章任务

```
POST /api/v1/llm/chapters
Content-Type: application/json
Authorization: Bearer <jwt_token>
```

入参

Body 为 JSON 对象：

字段	类型	必填	说明
utterances	array	**是**	转录片段数组，不可为空。**直接使用 § 1.2 查询转写结果 返回的 `data.result.utterances`**
trace_id	string	否	业务追踪 ID，原样透传到 webhook payload

单条 utterance 结构（与录音转写结果中的 utterance 格式完全一致）：

字段	类型	必填	说明
start_time	int	**是**	片段起始时间（毫秒）
end_time	int	**是**	片段结束时间（毫秒）
text	string	**是**	该片段文本
speaker	string	否	发言人 ID（可选透传，当前 LLM 不消费）
volume	float	否	分句音量（可选透传）
emotion	string	否	情绪标签（可选透传）

> ☒**典型用法：** 转写完成后，把 `data.result.utterances` 原样作为本接口的 `utterances` 字段传入，无需裁剪或重命名。

出参

立即返回（不等待 LLM 完成）：

```
{
  "code": 0,
  "message": "ok",
  "data": {
    "task_id": "chap_8f7a3c9d1e2b",
    "status": "pending",
    "created_at": "2026-04-17T04:00:00Z"
  }
}
```

字段	说明
`data.task_id`	分章任务 ID，用于轮询查询
`data.status`	初始状态，固定为 `pending`
`data.created_at`	创建时间（UTC ISO 8601）

5.2 查询分章结果

```
GET /api/v1/llm/chapters/tasks/{task_id}
Authorization: Bearer <jwt_token>
```

出参

状态：pending / processing


```
{
  "code": 0,
  "message": "ok",
  "data": {
    "task_id": "chap_8f7a3c9d1e2b",
    "status": "processing",
    "created_at": "2026-04-17T04:00:00Z",
    "completed_at": null,
    "result": null,
    "error": null,
    "usage": null
  }
}
```

状态：completed


```
{
  "code": 0,
  "message": "ok",
  "data": {
    "task_id": "chap_8f7a3c9d1e2b",
    "status": "completed",
    "created_at": "2026-04-17T04:00:00Z",
    "completed_at": "2026-04-17T04:00:18Z",
    "result": {
      "chapters": [
        {
          "start_time": 0,
          "end_time": 62000,
          "title": "开场与嘉宾介绍",
          "summary": "主播介绍本期主题与嘉宾背景，概述本集讨论的三个核心问题。"
        },
        {
          "start_time": 62000,
          "end_time": 318000,
          "title": "AI 编程范式的转变",
          "summary": "讨论 LLM 辅助编码对工程实践的影响，以及团队协作的新挑战。"
        }
      ]
    },
    "usage": {
      "provider": "dashscope",
      "model": "qwen3.6-plus",
      "input_tokens": 1240,
      "output_tokens": 380
    },
    "error": null
  }
}
```

状态：failed


```
{
  "code": 0,
  "message": "ok",
  "data": {
    "task_id": "chap_8f7a3c9d1e2b",
    "status": "failed",
    "created_at": "2026-04-17T04:00:00Z",
    "completed_at": "2026-04-17T04:00:05Z",
    "result": null,
    "usage": null,
    "error": {
      "code": 53001,
      "message": "LLM 返回的章节 JSON 格式异常"
    }
  }
}
```


输出约束（completed 时）：

- 章节数自动控制在 3-10 个之间
- 章节按时间顺序排列，**前一章的 `end_time` 等于下一章的 `start_time`**（无缝衔接）
- 第一章 `start_time` 为第一条 utterance 的起始时间，最后一章 `end_time` 为最后一条 utterance 的结束时间
- 标题约 10-20 字符，摘要 1-2 句话
- 任务状态在 Redis 中保留 1 小时（与文件转写一致），过期后查询返回 404

错误码

HTTP	说明
400	`utterances` 为空（提交阶段）
404	任务不存在或已过期（查询阶段）

任务失败时的 `error.code`：

code	说明
53001	LLM 返回的 JSON 格式异常（偶发，建议重试）
53002	LLM 调用失败（网络超时、上游错误等）

Demo

```
# 1. 提交
curl -X POST http://8.163.67.66/api/v1/llm/chapters \
  -H "Authorization: Bearer <jwt_token>" \
  -H "Content-Type: application/json" \
  -d '{
    "trace_id": "recording_20260417_meeting_abc",
    "utterances": [
      {"start_time": 0,     "end_time": 3200,  "text": "大家好，欢迎收听本期节目。"},
      {"start_time": 3200,  "end_time": 12000, "text": "今天我们请到了老王，聊聊 AI 编程。"},
      {"start_time": 12000, "end_time": 28000, "text": "老王你好，先简单介绍一下自己吧。"}
    ]
  }'
# 返回: {"data": {"task_id": "chap_xxx", "status": "pending", ...}}

# 2. 轮询
curl http://8.163.67.66/api/v1/llm/chapters/tasks/chap_xxx \
  -H "Authorization: Bearer <jwt_token>"
# 返回: status=processing / completed / failed
```

```
import time, requests

API = "http://8.163.67.66"
HEADERS = {"Authorization": "Bearer <jwt_token>", "Content-Type": "application/json"}

# 1. 提交（trace_id 可选，用于 webhook 串联）
resp = requests.post(f"{API}/api/v1/llm/chapters",
    json={
        "utterances": asr_result["utterances"],
        "trace_id": "recording_20260417_meeting_abc",
    },
    headers=HEADERS)
task_id = resp.json()["data"]["task_id"]

# 2. 轮询
while True:
    time.sleep(2)
    r = requests.get(f"{API}/api/v1/llm/chapters/tasks/{task_id}", headers=HEADERS).json()
    if r["data"]["status"] == "completed":
        for ch in r["data"]["result"]["chapters"]:
            print(f"[{ch['start_time']}ms] {ch['title']} — {ch['summary']}")
        break
    elif r["data"]["status"] == "failed":
        print("失败:", r["data"]["error"])
        break
```

推荐链路

```
POST /api/v1/asr/submit → 拿到 asr_task_id
GET  /api/v1/asr/tasks/{asr_task_id} → 拿到 result.utterances
POST /api/v1/llm/chapters (utterances) → 拿到 chap_task_id（立即返回）
GET  /api/v1/llm/chapters/tasks/{chap_task_id} → 轮询拿到 result.chapters
  │
  └── 或监听 chapters.completed webhook 事件，收到通知后再调 GET 查询拿完整 chapters
```

> **注意：** 与旧版（v2.4.0 及之前）不同，分章结果**不再在提交响应中直接返回**——需要通过 GET 查询接口或 webhook 获取。Webhook 推送中**不包含完整的 chapters 数组**（节省带宽），甲方需拿 `task_id` 调查询接口获取 `data.result.chapters`。

---

6. Webhook 管理

注册全局 Webhook，当转写任务完成/失败时自动推送通知到指定 URL。

6.1 注册 Webhook

```
POST /api/v1/webhooks
Content-Type: application/json
X-Api-Secret: <api_secret>
```

入参

```
{
  "url": "https://your-server.com/webhook/callback",
  "events": ["asr.completed", "asr.failed"],
  "secret": "your-signing-secret"
}
```

字段	类型	必填	默认值	说明
url	string	**是**	-	回调地址（HTTPS 推荐）
events	string[]	否	`["asr.completed", "asr.failed"]`	订阅的事件类型，支持 `["*"]` 通配符（见下方"全订阅"说明）
secret	string	否	`""`	专用签名密钥。**留空时自动用 `AUTH_SHARED_SECRET` 签名**（v2.5.5 起）；传了就用传入的 secret。生产推荐显式传一个专用值和 `AUTH_SHARED_SECRET` 隔离（单点密钥泄露影响面更小），详见 § 6.7。

支持的事件类型：

事件	触发时机
`asr.completed`	文件转写任务完成
`asr.failed`	文件转写任务失败
`asr.stream.completed`	实时流式转写会话结束
`asr.stream.failed`	实时流式转写会话异常中断
`summarize.completed`	智能总结完成
`summarize.failed`	智能总结失败
`translate.completed`	智能翻译完成
`translate.failed`	智能翻译失败
`chapters.completed`	智能分章完成
`chapters.failed`	智能分章失败

关于 `*.failed` 事件的推送边界：

- 只有 JWT 鉴权通过后的阶段失败才推送（JWT 解不出来 → 无 `client_reference_id` → 推送无意义）
- 同步接口（summarize / translate / chapters）失败本身已通过 HTTP 4xx/5xx 直接返回给调用方客户端；`*.failed` 事件是**为甲方服务端审计 / 风控 / 告警设计**的独立通道，与 HTTP 错误并行存在
- Pydantic 请求体自动校验失败（HTTP 422）不触发推送，因为它拦截在业务逻辑之前

全订阅（通配符）：

若希望订阅全部事件，`events` 传 `["*"]` 即可（约定与 Stripe / GitHub / PayPal 一致）：

```
{ "url": "https://your-server.com/webhook", "events": ["*"] }
```

服务端会在注册时把 `*` **展开**为当前支持的全量事件列表并持久化，响应里 `data.events` 返回展开后的显式列表，便于甲方自查实际订阅范围。

> **注意：** 展开发生在**注册时**。未来我们新增事件类型（如 `asr.stream.failed`）时，老的通配符注册不会自动跟进，如需订阅新事件请重新注册或更新。推送 payload 里始终带 `event` 字段区分具体事件，下游分支逻辑不受通配符影响。

重复注册检测（Shopify 风格严格去重）：

同一个 URL 若已被某个 active webhook 订阅了本次请求中的**任一事件**，注册**不会成功**，返回 HTTP 409 Conflict：

```
HTTP/1.1 409 Conflict
{
  "detail": {
    "code": 40901,
    "message": "URL https://your-server.com/webhook 已被 webhook wh_abc123456789 订阅事件
               ['asr.completed', 'asr.failed']，请先 DELETE 再重注册",
    "existing_webhook_id": "wh_abc123456789",
    "overlapping_events": ["asr.completed", "asr.failed"],
    "url": "https://your-server.com/webhook"
  }
}
```

设计原理： 允许重复注册会导致同一条事件被推送多次，甲方服务端会出现重复记账 / 重复告警。强制去重从根源消除风险。

甲方侧推荐流程：

1. **首次注册**：直接 POST，正常 200 返回。即使不传 `secret`，推送也会用 `AUTH_SHARED_SECRET` 自动签名
2. **需要更新订阅范围 / 轮换专用 secret**：先 `GET /api/v1/webhooks` 或 `DELETE /api/v1/webhooks/{existing_webhook_id}` 清掉旧的，再 POST 新的（注册时的 `secret` 是写入字段，不支持原地 PATCH）
3. **CI / 自动化脚本遇到 409**：根据 `detail.existing_webhook_id` 调 DELETE 后重试——属于可预期流程，不是错误

出参

```
{
  "code": 0,
  "message": "ok",
  "data": {
    "webhook_id": "wh_a1b2c3d4e5f6",
    "url": "https://your-server.com/webhook/callback",
    "events": ["asr.completed", "asr.failed"],
    "created_at": "2026-04-06T12:00:00Z",
    "active": true
  }
}
```

Demo

cURL：

```
curl -X POST http://8.163.67.66/api/v1/webhooks \
  -H "Content-Type: application/json" \
  -H "X-Api-Secret: your-api-secret" \
  -d '{"url":"https://your-server.com/callback","events":
              ["asr.completed","asr.failed"],"secret":"signing-secret"}'
```

```
HttpClient client = HttpClient.newHttpClient();
String body = "{\"url\":\"https://your-server.com/callback\","
    + "\"events\":[\"asr.completed\",\"asr.failed\"],"
    + "\"secret\":\"signing-secret\"}";
HttpRequest request = HttpRequest.newBuilder()
    .uri(URI.create("http://8.163.67.66/api/v1/webhooks"))
    .header("Content-Type", "application/json")
    .header("X-Api-Secret", "your-api-secret")
    .POST(HttpRequest.BodyPublishers.ofString(body))
    .build();
HttpResponse<String> resp = client.send(request, HttpResponse.BodyHandlers.ofString());
```

```
import requests

resp = requests.post("http://8.163.67.66/api/v1/webhooks",
    json={"url": "https://your-server.com/callback",
          "events": ["asr.completed", "asr.failed"],
          "secret": "signing-secret"},
    headers={"X-Api-Secret": "your-api-secret"})
webhook_id = resp.json()["data"]["webhook_id"]
```

6.2 列出所有 Webhook

```
GET /api/v1/webhooks
X-Api-Secret: <api_secret>
```

出参


```
{
  "code": 0,
  "message": "ok",
  "data": [
    {
      "webhook_id": "wh_a1b2c3d4e5f6",
      "url": "https://your-server.com/webhook/callback",
      "events": ["asr.completed", "asr.failed"],
      "created_at": "2026-04-06T12:00:00Z",
      "active": true
    }
  ]
}
```

6.3 查询单个 Webhook

```
GET /api/v1/webhooks/{webhook_id}
X-Api-Secret: <api_secret>
```

出参


```
{
  "code": 0,
  "message": "ok",
  "data": {
    "webhook_id": "wh_a1b2c3d4e5f6",
    "url": "https://your-server.com/webhook/callback",
    "events": ["asr.completed", "asr.failed"],
    "created_at": "2026-04-06T12:00:00Z",
    "active": true
  }
}
```

6.4 删除 Webhook

```
DELETE /api/v1/webhooks/{webhook_id}
X-Api-Secret: <api_secret>
```

出参

```
{
  "code": 0,
  "message": "ok"
}
```

6.5 批量重置所有 Webhook（危险操作，v2.3.7+）

```
POST /api/v1/admin/webhooks/reset
X-Api-Secret: <api_secret>
Content-Type: application/json
```

> **路径刻意放在 `/api/v1/admin/` 前缀下**（而非 `/api/v1/webhooks/*`），强调这是**运维级动作**——甲方业务代码不应该日常调用此接口。鉴权、错误码、响应信封保持与其他 admin 接口一致。

用途

一次性清空服务端所有已注册的 webhook。调用成功后，所有事件（`asr.*` / `summarize.*` / `translate.*` / `chapters.*`）的推送通道都会失效，直到甲方重新注册。

典型使用场景：

- 环境切换：从测试回调地址整体切到生产
- 回归/压测后清理遗留订阅
- 怀疑订阅列表被污染后一键重置

不要用于：

- 换 URL / 改事件订阅 → 用 DELETE + 重注册（HTTP 409 会直接给出 `existing_webhook_id`）
- 业务代码里自动循环重置——这不是日常 CRUD 接口

入参

字段	类型	必填	说明
confirm	boolean	☒	必须显式传 `true`。传 `false` 或缺失 → HTTP 400，防止空 body 误触发

请求示例：

```
{ "confirm": true }
```

出参

```
{
  "code": 0,
  "message": "ok",
  "data": {
    "deleted_count": 3,
    "deleted": [
      {
        "webhook_id": "wh_a1b2c3d4e5f6",
        "url": "https://your-server.com/webhook/callback",
        "events": ["asr.completed", "asr.failed"],
        "created_at": "2026-04-06T12:00:00Z",
        "active": true
      },
      {
        "webhook_id": "wh_b7c8d9e0f1a2",
        "url": "https://your-server.com/summarize",
        "events": ["summarize.completed"],
        "created_at": "2026-04-06T12:05:00Z",
        "active": true
      }
    ]
  }
}
```

字段	说明
`data.deleted_count`	实际被清空的 webhook 数量。二次调用时会返回 `0`（不抛错，但调用方应该意识到这是 no-op）
`data.deleted`	被清空的完整快照，按 `webhook_id` 排序，不含 `secret` 字段。**务必保存**，以便需要时重建订阅

错误响应

缺 confirm / confirm 非 true → HTTP 400：

```
{ "detail": "请求被拒绝：重置操作必须传 `confirm: true`" }
```

Demo

```
curl -X POST http://8.163.67.66/api/v1/admin/webhooks/reset \
  -H "Content-Type: application/json" \
  -H "X-Api-Secret: your_api_secret" \
  -d '{"confirm": true}'
```

```
import requests

resp = requests.post(
    "http://8.163.67.66/api/v1/admin/webhooks/reset",
    headers={
        "X-Api-Secret": "your_api_secret",
        "Content-Type": "application/json",
    },
    json={"confirm": True},
)
data = resp.json()["data"]
print(f"已清空 {data['deleted_count']} 个")
# data['deleted'] 是完整快照（URL + 事件列表，不含 secret），按需落库以便重建
```

安全约定

- 服务端以 **WARNING** 级别记录每次调用，日志含 `deleted_count`、调用方 IP、被清理的 `webhook_id` 列表
- 接口在 `/api/v1/admin/` 下，与其他 admin 能力（模版 CRUD、敏感词配置）共享同一把 `X-Api-Secret`
- 该接口**不支持**按事件或按 URL 过滤——这是"全清"语义的刻意选择。若需要按条件批量操作，请在 `GET /api/v1/webhooks` 结果上过滤后逐个 `DELETE`

6.6 Webhook 推送格式

当事件触发时，服务端会向注册的 URL 发送 HTTP POST 请求。

```
POST {your_webhook_url}
Content-Type: application/json
X-Webhook-Signature: sha256=xxxxxxxx
```

所有推送都带 HMAC-SHA256 签名，验证方式和安全约定详见 § 6.7 签名验证与安全约定。

所有 `*.completed` 事件的 `data` 包含 `usage` 字段，透传上游服务商的资源消耗数据。甲方可据此按合同价格自行计费。

asr.completed — 文件转写完成


```json
{
  "event": "asr.completed",
  "timestamp": "2026-04-06T04:01:23Z",
  "data": {
    "task_id": "asr_a1b2c3d4e5f6",
    "client_reference_id": "user_12345",
    "trace_id": "recording_20260417_meeting_abc",
    "result": {
      "text": "[00:00:01.080] 你好...",
      "utterances": [
        {"start_time": 1080, "end_time": 4500, "text": "你好...", "speaker": null}
      ]
    },
    "usage": {
      "provider": "volcengine",
      "model": "bigmodel",
      "duration_ms": 36000,
      "duration_sec": 36.0
    }
  }
}
```

asr.failed — 文件转写失败


```
{
  "event": "asr.failed",
  "timestamp": "2026-04-06T04:00:15Z",
  "data": {
    "task_id": "asr_a1b2c3d4e5f6",
    "client_reference_id": "user_12345",
    "trace_id": "recording_20260417_meeting_abc",
    "error": {
      "code": 50001,
      "message": "ASR 服务超时"
    }
  }
}
```


> 失败事件不含 `usage` 字段。

asr.stream.completed — 流式转写会话结束

流式转写的实时结果通过 WebSocket 返回，Webhook 仅在会话结束时推送一次用量摘要，不含转写文本。


```
{
  "event": "asr.stream.completed",
  "timestamp": "2026-04-06T04:05:00Z",
  "data": {
    "task_id": "asr_stream_a1b2c3d4e5f6",
    "user_id": "user_12345",
    "client_reference_id": "user_12345",
    "trace_id": "recording_20260417_meeting_abc",
    "usage": {
      "provider": "volcengine",
      "model": "bigmodel",
      "duration_ms": 15000,
      "duration_sec": 15.0
    }
  }
}
```

字段	类型	说明
task_id	string	任务 ID，格式：`asr_xxx`（离线转写）、`asr_stream_xxx`（流式转写）、`sum_xxx`（总结）、`trans_xxx`（翻译）、`chap_xxx`（分章）
client_reference_id	string/null	用户标识（自动从 JWT 的 `sub` 字段提取，即 Ticket 中的 `user_id`）
trace_id	string/null	业务追踪 ID（甲方入参传什么就原样带回；未传时为 `null`）
usage	object	用量统计

summarize.completed — 智能总结完成


```
{
  "event": "summarize.completed",
  "timestamp": "2026-04-06T04:10:00Z",
  "data": {
    "task_id": "sum_a1b2c3d4e5f6",
    "client_reference_id": "user_12345",
    "trace_id": "recording_20260417_meeting_abc",
    "input_length": 2048,
    "result_length": 356,
    "template": "podcast_solo",
    "usage": {
      "provider": "dashscope",
      "model": "qwen3.6-plus",
      "input_tokens": 1520,
      "output_tokens": 280
    }
  }
}
```


`template` 是客户端请求时传入的值。输出语言由 LLM 根据输入文本自动判定（中文音频/文本出中文摘要，英文出英文摘要，无需手动指定）。

translate.completed — 智能翻译完成


```
{
  "event": "translate.completed",
  "timestamp": "2026-05-22T04:12:00Z",
  "data": {
    "task_id": "trans_a1b2c3d4e5f6",
    "client_reference_id": "user_12345",
    "trace_id": "recording_20260417_meeting_abc",
    "input_length": 1024,
    "target_lang": "en",
    "result_length": 890,
    "usage": {
      "provider": "dashscope",
      "model": "deepseek-v4-flash",
      "input_tokens": 800,
      "output_tokens": 650
    },
    "was_translated": true,
    "source_lang_detected": "中文",
    "detect_method": "charset"
  }
}
```


v2.6.1+ 三个新字段(老接口没有, 客户端做 null check):

字段	类型	说明
`was_translated`	bool	服务端是否真的执行了跨语翻译。`false` = 源语言已经等于 `target_lang`, 服务端走 self-translate 短路只跑 meta 调用, `usage` 仅含 meta 消耗
`source_lang_detected`	string	服务端检测到的源语言(`"中文"` / `"英文"` / `"日语"` / `"韩语"` / `"法语"` / `"西班牙语"` / `"葡萄牙语"` / `"越南语"` / `"俄语"` / `"德语"` / `"印尼语"` / `"泰语"` / `"意大利语"` / `"unknown"`)。配合 `was_translated` 让甲方能解释 self-translate 短路的原因
`detect_method`	string	服务端检测语言用的路径: `"charset"` (Unicode 字符块强信号,中/韩/日 短文本免费秒判)/ `"langdetect"` (本地 n-gram,长文本免费判)/ `"llm"` (ds-flash 兜底,短拉丁词必用,~180 token + ~4s) / `"fallback"` (LLM 失败降级)。甲方拿这个统计 LLM 兜底占比 → 看翻译成本结构

对账要点: `usage.input_tokens` / `output_tokens` 在 `detect_method="llm"` 场景下**包含 detect 调用消耗**(+~180 in / +~4 out),跟 DashScope 账单口径一致。`detect_method="charset"` / `"langdetect"` 场景 detect 不消耗 token, `usage` 仅含 meta+body。

self-translate 短路特殊性: `was_translated=false` 时 `usage.model` 反映实际跑的 meta 模型(默认 `deepseek-v4-flash`), `usage.input_tokens` 只含 meta 消耗(典型 ~80 tokens),没有 body 调用消耗。

输出层兜底重译(v2.12.1+):服务端翻译完成后会扫输出有无源语言残留,命中则对残留段落做一次精准重译以保证"目标语言里不出现源语言字符"。这次兜底是**按需触发**(绝大多数请求不触发),触发时它消耗的 token **会累加进** `usage.input_tokens` / `output_tokens` —— 跟主翻译一起计入本次调用成本,因此同样长度的输入偶尔会比预期略高,属正常质量保障开销,跟 DashScope 账单口径一致。是否触发是服务端内部诊断信息,不在 webhook payload 暴露。

chapters.completed — 智能分章完成


```
{
  "event": "chapters.completed",
  "timestamp": "2026-04-06T04:18:00Z",
  "data": {
    "task_id": "chap_8f7a3c9d1e2b",
    "client_reference_id": "user_12345",
    "trace_id": "recording_20260417_meeting_abc",
    "utterance_count": 312,
    "chapter_count": 6,
    "usage": {
      "provider": "dashscope",
      "model": "qwen3.6-plus",
      "input_tokens": 1240,
      "output_tokens": 380
    }
  }
}
```


> **注意：** 为节省带宽，Webhook 推送中**不包含完整的 chapters 数组**，甲方需要拿到 `task_id` 通知后，从 HTTP 响应同步获得 `data.chapters` 内容；此事件主要用于异步记账和审计。

asr.stream.failed — 流式转写会话异常中断


```
{
  "event": "asr.stream.failed",
  "timestamp": "2026-04-14T08:33:21Z",
  "data": {
    "task_id": "asr_stream_a1b2c3d4e5f6",
    "client_reference_id": "user_12345",
    "trace_id": "recording_20260417_meeting_abc",
    "phase": "streaming",
    "error": {
      "code": 54002,
      "message": "豆包返回错误 code=1001: ..."
    }
  }
}
```


字段	取值	说明
phase	`connecting`	建立豆包连接时失败（如账号额度耗尽、鉴权失败）
phase	`streaming`	识别过程中豆包返回错误、WebSocket 异常断开
phase	`finalizing`	收尾（二遍识别）阶段超时（>30s）

> 与 `asr.stream.completed` 在同一会话中**互斥**：服务端保证同一个 `task_id` 最多只推一个收尾事件。

summarize.failed — 智能总结失败


```
{
  "event": "summarize.failed",
  "timestamp": "2026-04-14T08:33:21Z",
  "data": {
    "client_reference_id": "user_12345",
    "trace_id": "recording_20260417_meeting_abc",
    "template_id": "podcast_solo",
    "error": {
      "code": 51001,
      "message": "LLM 调用失败：上游超时"
    }
  }
}
```

translate.failed — 智能翻译失败


```
{
  "event": "translate.failed",
  "timestamp": "2026-04-14T08:33:21Z",
  "data": {
    "client_reference_id": "user_12345",
    "trace_id": "recording_20260417_meeting_abc",
    "target_language": "en",
    "error": {
      "code": 52001,
      "message": "LLM 调用失败：Qwen 返回 503"
    }
  }
}
```


chapters.failed — 智能分章失败


```
{
  "event": "chapters.failed",
  "timestamp": "2026-04-14T08:33:21Z",
  "data": {
    "client_reference_id": "user_12345",
    "trace_id": "recording_20260417_meeting_abc",
    "utterance_count": 234,
    "error": {
      "code": 53001,
      "message": "LLM 返回结构不合法"
    }
  }
}
```


错误码速查

同步/流式 `*.failed` 事件的 `error.code` 统一 5 位整数，首位区分性质、第二位区分接口：

前缀	含义
`4xxxx`	业务/参数错误（调用方可通过修改请求自行修复）
`5xxxx`	系统/上游错误（建议客户端重试）
`41xxx` `51xxx`	summarize
`42xxx` `52xxx`	translate
`43xxx` `53xxx`	chapters
`44xxx` `54xxx`	asr.stream

> 当前版本仅使用粗粒度码（如 `51001` = summarize 上游错误），后续按需细化时**向下兼容**——新码只会是现有前缀段内的扩充，不会突然出现新前缀。

6.7 签名验证与安全约定

所有 webhook 推送都会带 HMAC-SHA256 签名,供接收方验证消息确实来自本服务端(而非伪造)。v2.5.5 起签名为**默认行为**,无需注册时额外配置就能启用。

签名头格式

```
X-Webhook-Signature: sha256=<64位十六进制摘要>
```

HMAC 算法固定 SHA-256,输出 hex 小写,前缀 `sha256=` 用于未来兼容多算法。

使用的 Secret(优先级)

1. **注册 webhook 时传的 `secret`** (§6.1 `POST /api/v1/webhooks` 的 body 字段) — 优先使用,适合希望和 `AUTH_SHARED_SECRET` 隔离的生产环境
2. **`AUTH_SHARED_SECRET`** (即甲方签发 Ticket 用的共享密钥) — 未传 secret 时的 fallback,甲方本就持有此密钥,无需额外交付

> **生产建议:** 显式给每个 webhook 注册一个专用 `secret`,与 `AUTH_SHARED_SECRET` 隔离。单点密钥泄露时影响面更小。

⚠ 必须用 raw body 计算 HMAC

服务端发送时 body 是 `json.dumps(payload, ensure_ascii=False)` 的原始字节串。**接收方必须用同一份字节流**算 HMAC —— 反序列化后再 stringify 得到的 body 会因空格、字段顺序、非 ASCII 字符编码等差异导致字节级不一致,HMAC 一定对不上。

主流框架里"拿原始字节"的正确方式:

语言/框架	正确姿势
Spring Boot	`@RequestBody byte[] rawBody`
FastAPI	`await request.body()`
Express	`app.use(express.raw({ type: 'application/json' }))` + `req.body` (Buffer)
Flask	`request.get_data(as_text=False)` (bytes)
Gin (Go)	`c.GetRawData()`

Receiver 实现示例

```
package com.example.webhook;

import org.apache.commons.codec.digest.HmacUtils;
import org.springframework.http.HttpStatus;
import org.springframework.http.ResponseEntity;
import org.springframework.web.bind.annotation.*;
import java.nio.charset.StandardCharsets;
import java.security.MessageDigest;

@RestController
public class WebhookController {
    // 推荐用配置注入,不要硬编码
    private static final String SECRET = System.getenv("AUTH_SHARED_SECRET");

    @PostMapping("/webhook/maono")
    public ResponseEntity<String> receive(
            @RequestBody byte[] rawBody,   // 必须用 byte[] 拿原始字节
            @RequestHeader(value = "X-Webhook-Signature", required = false) String sigHeader) {

        if (sigHeader == null || !sigHeader.startsWith("sha256=")) {
            return ResponseEntity.status(HttpStatus.UNAUTHORIZED).body("missing signature");
        }

        String expected = HmacUtils.hmacSha256Hex(
                SECRET.getBytes(StandardCharsets.UTF_8), rawBody);
        String actual = sigHeader.substring("sha256=".length());

        // 常数时间比较防 timing attack — 不要直接 equals/==
        if (!MessageDigest.isEqual(expected.getBytes(), actual.getBytes())) {
            return ResponseEntity.status(HttpStatus.UNAUTHORIZED).body("invalid signature");
        }

        // 到这里签名已验证,可以安全解析 JSON 做业务处理
        String payload = new String(rawBody, StandardCharsets.UTF_8);
        // ... JSON.parse + 业务逻辑 ...

        return ResponseEntity.ok("ok");
    }
}
```

```
import hmac, hashlib, os, json
from fastapi import FastAPI, Request, HTTPException, Header

app = FastAPI()
SECRET = os.getenv("AUTH_SHARED_SECRET", "").encode()

@app.post("/webhook/maono")
async def receive(
    request: Request,
    x_webhook_signature: str | None = Header(default=None),
):
    raw_body = await request.body()   # 必须拿 bytes,不是 request.json()

    if not x_webhook_signature or not x_webhook_signature.startswith("sha256="):
        raise HTTPException(401, "missing signature")

    expected = hmac.new(SECRET, raw_body, hashlib.sha256).hexdigest()
    actual = x_webhook_signature.removeprefix("sha256=")

    # 常数时间比较防 timing attack
    if not hmac.compare_digest(expected, actual):
        raise HTTPException(401, "invalid signature")

    # 签名通过,可以安全解析
    payload = json.loads(raw_body)
    # ... 业务逻辑(按 payload["event"] 分发) ...
    return {"ok": True}
```

```
const express = require('express');
const crypto = require('crypto');

const app = express();
const SECRET = process.env.AUTH_SHARED_SECRET;

// 必须用 express.raw 保留原始 body,不能用 express.json()
app.post('/webhook/maono',
  express.raw({ type: 'application/json' }),
  (req, res) => {
    const sigHeader = req.get('X-Webhook-Signature') || '';
    if (!sigHeader.startsWith('sha256=')) {
      return res.status(401).send('missing signature');
    }

    const expected = crypto.createHmac('sha256', SECRET)
      .update(req.body)  // req.body 是 Buffer(原始字节)
      .digest('hex');
    const actual = sigHeader.slice('sha256='.length);

    // 常数时间比较防 timing attack
    if (!crypto.timingSafeEqual(Buffer.from(expected), Buffer.from(actual))) {
      return res.status(401).send('invalid signature');
    }

    const payload = JSON.parse(req.body.toString());
    // ... 业务逻辑 ...
    res.json({ ok: true });
  }
);
```

调试台自带验证工具

不方便跑本地 receiver 代码时,可用调试台「Webhook」Tab → ☒签名验证工具:

- 「开启调试接收器」收到的每条 webhook 右上角有「☒验证」按钮,自动填入 body + signature,手动补 secret 即验
- 手动输入场景:前端算出的 HMAC 和 header 不一致时会**自动尝试** Python `json.dumps(ensure_ascii=False)` 格式重算,匹配时把 body 框修正为正确格式
- 全程在浏览器本地 Web Crypto API 计算,secret 不出设备、不联网

重试机制

- 推送失败(非 2xx 响应或超时)自动重试 **3 次**
- 重试间隔:10s → 30s → 60s
- 接收方返回 HTTP **200** 表示确认收到

- 3 次仍失败 → 放弃 + 在审计日志(`logs/errors/YYYY-MM-DD.jsonl`)留完整上下文供排障

接收端安全清单

- ☒先验签,**签名通过再解析业务数据**(不要先 `json.parse` 再验签 — payload 本身可能是恶意构造)
- ☒常数时间比较(`MessageDigest.isEqual` / `hmac.compare_digest` / `crypto.timingSafeEqual`) — 不用 `==` 或 `.equals`
- ☒Secret 从环境变量或密钥管理服务读,不硬编码
- ☒幂等处理:重试机制意味着同一事件可能收到多次,按 `data.task_id` 或 `data.trace_id` 去重
- ☒**不要**把 secret 写进日志
- ☒**不要**用接收到的 header 里的 secret 反过来做其他用途(这里从来没有 secret header,只有 signature)

版本说明

版本	行为
v2.5.5 起	默认签名(即使注册时未传 secret,自动用 `AUTH_SHARED_SECRET`)。**对已注册的 webhook 也立即生效,无需重新注册。**
v2.5.5 之前	仅当注册时提供 secret 才签名。未传 secret 的 webhook 推送不带 `X-Webhook-Signature` ,无法验证来源。

---

7. 模版管理

管理总结 `templates/summary/` 目录下的模版文件。所有写操作完成后自动热重载，**秒级生效、不影响在线请求**。需要 `X-Api-Secret` 鉴权。

> 客户端只读场景（渲染下拉框）请用 §3.2 总结模版列表（公开接口）。

7.1 热重载模版目录

运维或产品直接修改 `templates/summary/` 目录下的 Markdown 文件后，调用此接口可立即生效，无需重启服务。

```
POST /api/v1/admin/reload-templates
X-Api-Secret: <api_secret>
```

响应：

```
{
  "code": 0,
  "message": "ok",
  "data": {
    "loaded": 3,
    "templates": [ /* 同 §3.1 的 templates 结构 */ ]
  }
}
```

7.2 模版 CRUD

通过 API 读取、新建、更新、删除模版文件。所有写操作完成后自动热重载，无需额外调用 §7.1 接口。调试台的「模版管理」Tab 即基于此组接口实现。

`{lang}` 和 `{id}` 只允许小写字母、数字、下划线（防路径穿越）。

读取模版

```
GET /api/v1/admin/templates/{lang}/{id}
X-Api-Secret: <api_secret>
```

响应：

```
{
  "code": 0,
  "message": "ok",
  "data": {
    "lang": "zh",
    "id": "podcast_solo",
    "content": "---\nid: podcast_solo\nname: 单人播客创作\n..."
  }
}
```


更新模版

```
PUT /api/v1/admin/templates/{lang}/{id}
X-Api-Secret: <api_secret>
Content-Type: application/json

{"content": "---\nid: podcast_solo\n..."}
```

模版不存在时返回 404（新建请用 POST）。内容必须以 `---` 开头（front matter 校验）。成功后返回同 §7.1 的热重载响应。

新建模版

```
POST /api/v1/admin/templates/{lang}/{id}
X-Api-Secret: <api_secret>
Content-Type: application/json

{"content": "---\nid: my_tpl\nname: 我的模版\n...\n---\n\nsystem prompt..."}
```

模版已存在时返回 409（更新请用 PUT）。`{lang}` 目录不存在时自动创建。

删除模版

```
DELETE /api/v1/admin/templates/{lang}/{id}
X-Api-Secret: <api_secret>
```

模版不存在时返回 404。删除后自动热重载。

Demo

```
# 读取模版
curl http://8.163.67.66/api/v1/admin/templates/zh/podcast_solo \
  -H "X-Api-Secret: <api_secret>"

# 更新模版
curl -X PUT http://8.163.67.66/api/v1/admin/templates/zh/podcast_solo \
  -H "X-Api-Secret: <api_secret>" \
  -H "Content-Type: application/json" \
  -d '{"content": "---\nid: podcast_solo\nname: 单人播客创作\n...\n---\n\nsystem prompt..."}'

# 手动热重载（不改文件）
curl -X POST http://8.163.67.66/api/v1/admin/reload-templates \
  -H "X-Api-Secret: <api_secret>"
```

---

8. 敏感词管理

统一配置录音文件转写 + 实时流式转写的**敏感词过滤规则**。三项均为空时不下发该参数，火山引擎按默认行为处理（展示原文）。均需要 `X-Api-Secret` 鉴权。

8.1 读取当前配置

```
GET /api/v1/admin/sensitive-words
X-Api-Secret: <api_secret>
```

出参

```
{
  "code": 0,
  "message": "ok",
  "data": {
    "system_reserved_filter": true,
    "filter_with_empty": [],
    "filter_with_signed": []
  }
}
```

字段	类型	说明
system_reserved_filter	bool	是否启用系统敏感词库（替换为 `*`，默认包含限制级词汇）
filter_with_empty	string[]	自定义敏感词，命中后**替换为空**
filter_with_signed	string[]	自定义敏感词，命中后**替换为 `*`**

8.2 更新配置（自动热重载）

```
PUT /api/v1/admin/sensitive-words
X-Api-Secret: <api_secret>
Content-Type: application/json

{
  "system_reserved_filter": true,
  "filter_with_empty": ["词1", "词2"],
  "filter_with_signed": ["词3"]
}
```

服务端会自动去重、去空白、排序后写回 `sensitive_words.json`，并即时热重载（流式和文件转写都秒级生效，无需重启）。

出参


```
{
  "code": 0,
  "message": "ok",
  "data": {
    "applied": {
      "system_reserved_filter": true,
      "filter_with_empty": ["词1", "词2"],
      "filter_with_signed": ["词3"]
    },
    "downstream_payload": {
      "system_reserved_filter": true,
      "filter_with_empty": ["词1", "词2"],
      "filter_with_signed": ["词3"]
    }
  }
}
```


- `applied`：实际落地到配置文件的内容（已去重/去空）
- `downstream_payload`：下发给火山引擎的实际对象；三项全为空时为 `null`，表示不下发该参数

8.3 仅触发热重载

```
POST /api/v1/admin/sensitive-words/reload
X-Api-Secret: <api_secret>
```

用于手动修改 `sensitive_words.json` 文件后触发流式转写侧的缓存刷新（文件转写每次任务都会 fresh 读盘，无需手动 reload）。

Demo

```
# 读取当前配置
curl http://8.163.67.66/api/v1/admin/sensitive-words \
  -H "X-Api-Secret: <api_secret>"

# 更新配置（自动热重载）
curl -X PUT http://8.163.67.66/api/v1/admin/sensitive-words \
  -H "X-Api-Secret: <api_secret>" \
  -H "Content-Type: application/json" \
  -d '{
    "system_reserved_filter": true,
    "filter_with_empty": ["敏感词A"],
    "filter_with_signed": ["敏感词B"]
  }'

# 仅触发热重载（手改 sensitive_words.json 文件后使用）
curl -X POST http://8.163.67.66/api/v1/admin/sensitive-words/reload \
  -H "X-Api-Secret: <api_secret>"
```

```
// 读取
const cfg = await fetch("/api/v1/admin/sensitive-words", {
  headers: { "X-Api-Secret": "<api_secret>" }
}).then(r => r.json());

// 更新 + 热重载
const resp = await fetch("/api/v1/admin/sensitive-words", {
  method: "PUT",
  headers: {
    "X-Api-Secret": "<api_secret>",
    "Content-Type": "application/json"
  },
  body: JSON.stringify({
    system_reserved_filter: true,
    filter_with_empty: ["敏感词A"],
    filter_with_signed: ["敏感词B"]
  })
});
const { data } = await resp.json();
console.log("已应用:", data.applied);
console.log("下发给引擎:", data.downstream_payload);  // null 表示不下发 = 展示原文
```

```python
import requests

HEADERS = {"X-Api-Secret": "<api_secret>"}

# 读取
cfg = requests.get("http://8.163.67.66/api/v1/admin/sensitive-words",
    headers=HEADERS).json()

# 更新 + 热重载
resp = requests.put("http://8.163.67.66/api/v1/admin/sensitive-words",
    headers={**HEADERS, "Content-Type": "application/json"},
    json={
        "system_reserved_filter": True,
        "filter_with_empty": ["敏感词A"],
        "filter_with_signed": ["敏感词B"],
    })
print(resp.json()["data"]["applied"])

# 仅热重载
requests.post("http://8.163.67.66/api/v1/admin/sensitive-words/reload",
    headers=HEADERS)
```

9. 健康检查

```
GET /api/health
```

出参

```
{
  "status": "ok",
  "dashscope_configured": true,
  "dashscope_preview": "sk-e5f...",
  "volcengine_configured": true
}
```

字段	说明
dashscope_configured	DashScope API Key 是否已配置
dashscope_preview	API Key 前 6 位预览（脱敏）
volcengine_configured	火山引擎 Access Key 是否已配置

附录 A: 错误码

HTTP 状态码

状态码	说明
200	成功
400	请求参数错误
404	资源不存在（任务过期 / Webhook 不存在）
409	资源冲突（Webhook 重复注册，detail 中 `code=40901` 含 `existing_webhook_id` ）
500	服务端内部错误

鉴权错误码

用户级（JWT）— HTTP 401：

code	说明
40101	缺少 Authorization header 或 X-Api-Secret
40102	JWT 签名无效
40103	JWT 已过期
invalid_ticket	Ticket 格式错误或签名无效
ticket_expired	Ticket 已过期
nonce_reused	Ticket 已被使用

系统级（API Secret）— HTTP 403：

code	说明
40301	API Secret 无效

业务错误码（task.error.code）

错误码	说明
50001	ASR 服务异常（豆包接口错误 / 超时）
50002	**(v2.5.11+)** ASR 未识别到有效语音(豆包 20000003) — 静默/低音量/声道错配/非人声

豆包 ASR 错误码（WebSocket type=error）

错误码	说明
45000081	等待音频包超时
45000151	音频格式无效

附录 B: 支持的语言代码

文件转写 `language` 参数支持以下语言代码（豆包大模型 ASR）：

代码	语言	代码	语言
zh-CN	中文	es-MX	西班牙语
en-US	英文	fr-FR	法语
ru-RU	俄语	ko-KR	韩语
id-ID	印尼语	th-TH	泰语
de-DE	德语	ja-JP	日语