

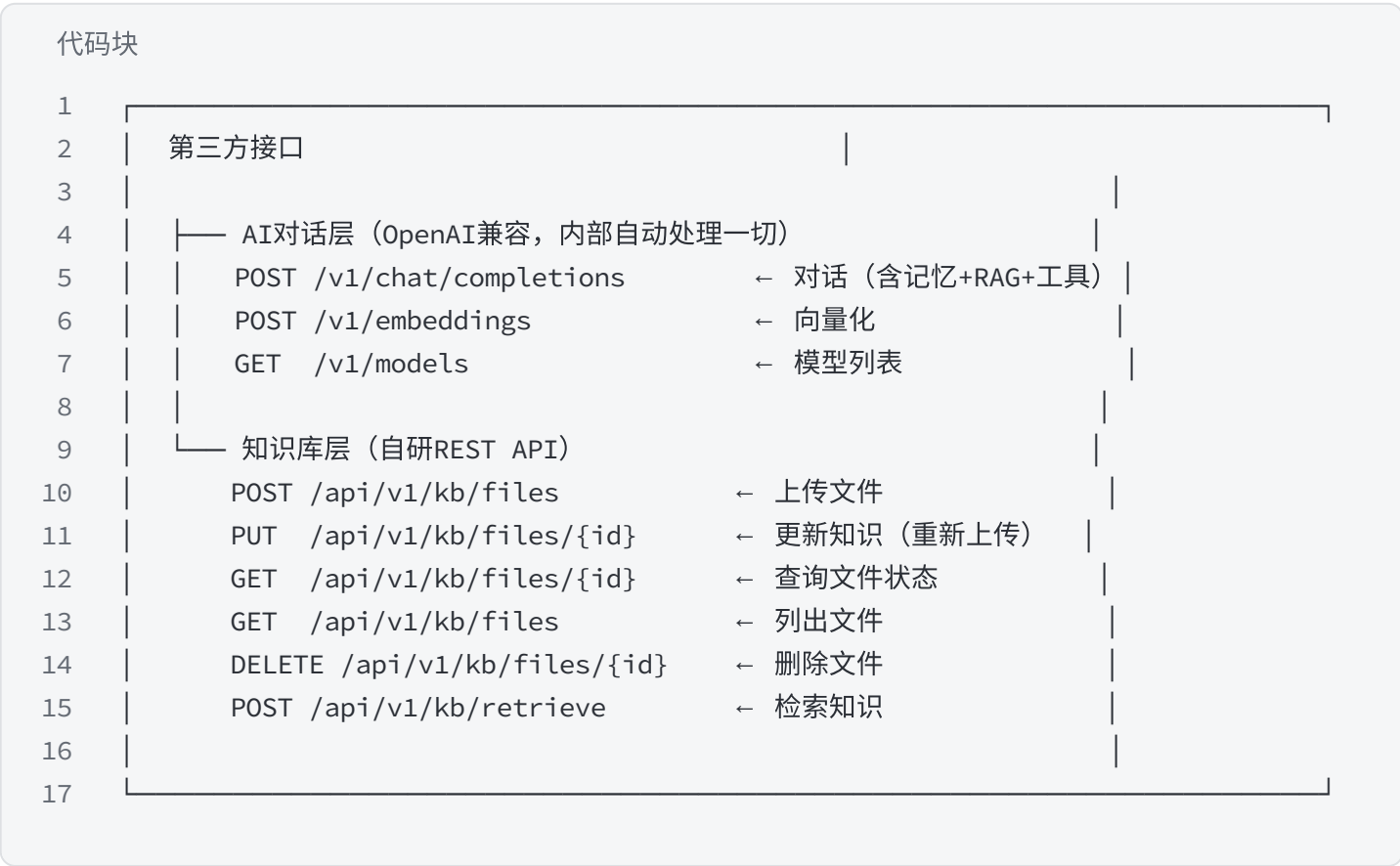
# 大模型对外API接口规范\_0509

版本：v1.0

日期：2026-05-09

协议：HTTPS + JSON + SSE

## 1. 接口架构



## 2. 快速开始

### 2.1 获取API Key

登录平台后台 → 开发者中心 → 创建API Key

### 2.2 接口地址

环境	地址
生产环境	<code>https://api.xxxx.com</code>
测试环境	<code>https://test-api.xxxx.com</code>

## 2.3 请求示例

代码块

```
1  from openai import OpenAI
2
3  client = OpenAI(
4      base_url="https://api.xxxx.com/v1",
5      api_key="sk-yak-xxxxxxxxxxxxxxxxxx"
6  )
7
8  response = client.chat.completions.create(
9      model="auto", # 自动选择智能体
10     messages=[{"role": "user", "content": "我家牦牛发烧了怎么办? "}]
11 )
12 print(response.choices[0].message.content)
```

返回示例：

代码块

```
1  {
2      "id": "chatcpl-abc123",
3      "object": "chat.completion",
4      "created": 1715248000,
5      "model": "yak-disease",
6      "choices": [{
7          "index": 0,
8          "message": {
9              "role": "assistant",
10             "content": "根据您的症状，牦牛体温升高可能由多种原因引起..."
11         },
12         "finish_reason": "stop"
13     }],
14     "usage": {
15         "prompt_tokens": 120,
```

```
16      "completion_tokens": 85,  
17      "total_tokens": 205  
18  }  
19 }
```

### 3. 认证方式

所有接口统一使用 Bearer Token 认证。

代码块

```
1 Authorization: Bearer {your_api_key}
```

请求头：

代码块

```
1 Content-Type: application/json  
2 Authorization: Bearer sk-yak-xxxxxxxxxxxxxxxxxx
```

Bearer Token 认证可选，随意字符串

### 4. 限流与配额

限制项	默认值	说明
每秒请求数（QPS）	10	超过返回429
每分钟请求数	600	超过返回429
单请求最大token	8192	输入+输出总和
知识库单文件大小	200MB	超过返回400
知识库总存储	100G	

## 5. 对话接口

### 5.1 接口概述

项目	说明
接口地址	POST /v1/chat/completions
协议	HTTPS + JSON
流式支持	SSE (Server-Sent Events)
OpenAI兼容	✅ 100%兼容，可用任意OpenAI SDK
内部能力	自动意图识别、记忆管理、RAG检索、工具调用（对第三方透明）

### 5.2 请求参数

参数	类型	必填	默认值	说明
model	string	✅	-	"auto" 自动选择，或指定智能体/模型ID
messages	array	✅	-	消息列表，标准OpenAI格式
stream	boolean	❌	false	是否流式返回
temperature	float	❌	0.7	采样温度，0-2
max_tokens	integer	❌	2048	最大生成token数
top_p	float	❌	1.0	核采样
user	string	❌	-	用户标识，用于记忆隔离和统计
session_id	string	❌	-	会话ID，不传则创建新会话

messages格式：

代码块

1 [

```
2     {"role": "system", "content": "自定义系统提示（可选，会覆盖默认）"},
3     {"role": "user", "content": "我家牛发烧了"},
4     {"role": "assistant", "content": "体温多少度? "},
5     {"role": "user", "content": "40.5度"}
6 ]
```

多模态输入（图片问诊）：

代码块

```
1  [
2    {
3      "role": "user",
4      "content": [
5        {"type": "text", "text": "这头牛皮肤怎么了? "},
6        {"type": "image_url", "image_url": {"url":
7          "https://example.com/cow.jpg"}}
8      ]
9    }
10 ]
```

5.3 model参数说明

取值	说明	使用场景
"auto"	<b>默认推荐。</b> 平台根据用户输入自动选择最合适的智能体	通用场景
"yak-general"	通用助手	通用问答
"yak-disease"	疫病诊疗助手	疾病诊断
"yak-feeding"	饲喂决策助手	饲料配方
"yak-device"	设备运维助手	设备故障
"yak-growth"	生长评估助手	出栏评估
"yak-grassland"	草畜平衡助手	草场管理
"qwen3.5-8b"	底层模型（高级）	绕过智能体层，直接调用LLM

5.4 非流式响应

代码块

```
1  {
2    "id": "chatcmpl-abc123",
3    "object": "chat.completion",
4    "created": 1715248000,
5    "model": "yak-disease",
6    "choices": [
7      {
8        "index": 0,
9        "message": {
10         "role": "assistant",
11         "content": "根据您的症状，牦牛体温40.5℃属于高热..."
12       },
13       "finish_reason": "stop"
14     ]
15   },
16   "usage": {
17     "prompt_tokens": 120,
18     "completion_tokens": 85,
19     "total_tokens": 205
20   }
21 }
```

响应字段说明：

字段	类型	说明
id	string	唯一标识
model	string	实际使用的模型/智能体
choices[0].message.content	string	回答内容
choices[0].finish_reason	string	stop 正常结束/ length 长度限制
usage.prompt_tokens	integer	输入token数

<code>usage.completion_tokens</code>	integer	输出token数
<code>usage.total_tokens</code>	integer	总token数

## 5.5 流式响应（SSE）

请求设置 `"stream": true`

### SSE事件流：

代码块

```
1  data: {"id":"chatcmpl-  
   abc","object":"chat.completion.chunk","created":1715248000,"model":"yak-  
   disease","choices":[{"index":0,"delta":  
   {"role":"assistant"},"finish_reason":null}]}
```

```
2  
3  data: {"id":"chatcmpl-  
   abc","object":"chat.completion.chunk","created":1715248000,"model":"yak-  
   disease","choices":[{"index":0,"delta":{"content":"根  
   据"},"finish_reason":null}]}
```

```
4  
5  data: {"id":"chatcmpl-  
   abc","object":"chat.completion.chunk","created":1715248000,"model":"yak-  
   disease","choices":[{"index":0,"delta":{"content":"您描  
   述"},"finish_reason":null}]}
```

```
6  
7  ...  
8  
9  data: {"id":"chatcmpl-  
   abc","object":"chat.completion.chunk","created":1715248000,"model":"yak-  
   disease","choices":[{"index":0,"delta":{"finish_reason":"stop"}]}
```

```
10  
11 data: [DONE]
```

### 客户端处理示例（Python）：

代码块

```
1  stream = client.chat.completions.create(  
2      model="auto",  
3      messages=[{"role": "user", "content": "牦牛怎么养? "}],
```

```
4     stream=True
5 )
6
7 for chunk in stream:
8     if chunk.choices[0].delta.content:
9         print(chunk.choices[0].delta.content, end="")
```

## 5.6 会话管理

平台自动维护会话上下文。通过 `user` 和 `session_id` 参数控制：

场景	传参方式
新用户首次对话	只传 <code>user</code> ，不传 <code>session_id</code>
继续已有会话	传 <code>user</code> + <code>session_id</code>
开始新会话（同一用户）	传 <code>user</code> ，不传 <code>session_id</code>

注意： `session_id` 在首次响应中返回，客户端需保存用于后续对话。

## 6. 向量化接口(内部使用，不用关心)

### 6.1 接口概述

项目	说明
接口地址	<code>POST /v1/embeddings</code>
功能	将文本转换为向量，用于语义检索、相似度计算

### 6.2 请求参数

参数	类型	必填	默认值	说明
<code>model</code>	string	<input checked="" type="checkbox"/>	-	<code>"auto"</code> 自动选择，或指定模型



<code>input</code>	string/array	✓	-	待量化的文本或文本数组
<code>encoding_format</code>	string	✗	<code>"float"</code>	<code>"float"</code> 或 <code>"base64"</code>

### 6.3 请求示例

代码块

```
1  {
2    "model": "auto",
3    "input": ["牦牛口蹄疫症状", "发烧用药指南"]
4  }
```

### 6.4 响应

代码块

```
1  {
2    "object": "list",
3    "data": [
4      {
5        "object": "embedding",
6        "index": 0,
7        "embedding": [0.023, -0.156, 0.089, ...],
8        "dimensions": 1024
9      },
10     {
11       "object": "embedding",
12       "index": 1,
13       "embedding": [0.089, -0.234, 0.156, ...],
14       "dimensions": 1024
15     }
16   ],
17   "model": "bge-m3-local",
18   "usage": {
19     "prompt_tokens": 15,
20     "total_tokens": 15
21   }
22 }
```

## 7. 模型列表接口

### 7.1 接口概述

项目	说明
接口地址	GET /v1/models
功能	获取平台所有可用模型/智能体
OpenAI兼容	 100%兼容

### 7.2 响应

代码块

```
1  {
2    "object": "list",
3    "data": [
4      {
5        "id": "auto",
6        "object": "model",
7        "created": 1715248000,
8        "owned_by": "xxxx",
9        "permission": []
10     },
11     {
12       "id": "yak-general",
13       "object": "model",
14       "created": 1715248000,
15       "owned_by": "xxxx"
16     },
17     {
18       "id": "yak-disease",
19       "object": "model",
20       "created": 1715248000,
21       "owned_by": "xxxx"
22     },
23     {
24       "id": "yak-feeding",
25       "object": "model",
```

```
26     "created": 1715248000,
27     "owned_by": "xxxx"
28 },
29 {
30     "id": "qwen3-8b-local",
31     "object": "model",
32     "created": 1715248000,
33     "owned_by": "xxxx"
34 },
35 {
36     "id": "bge-m3-local",
37     "object": "model",
38     "created": 1715248000,
39     "owned_by": "xxxx"
40 }
41 ]
42 }
```

## 8. 知识库接口

### 8.1 接口概述

知识库接口用于管理私有知识，支持多种文件格式上传、URL自动下载、自动解析、向量化、检索。

#### 支持两种添加方式

- 本地上传：**直接上传文件，支持文档/图片/音频/视频/压缩包等
- URL添加：**提供http/https链接，平台**智能识别内容类型，自动下载并处理：**
  - 网页链接 → 抓取正文，过滤广告导航
  - 文档链接 (\*.pdf/\*.docx等) → 下载后按文档解析
  - 图片链接 (\*.jpg/\*.png等) → 下载后OCR识别
  - 音频链接 (\*.mp3/\*.wav等) → 下载后语音转文字
  - 视频链接 (\*.mp4/\*.avi等) → 下载后提取字幕/音频
  - 压缩包链接 (\*.zip/\*.rar等) → 下载后解压递归处理

#### URL类型识别策略

- 默认`type\_hint=auto`，平台通过URL后缀+Content-Type+内容嗅探自动识别
- 可指定`type\_hint`加速处理，避免识别耗时
- 下载失败自动重试3次，支持断点续传
- 大文件分片下载，单文件最大200MB

**更新方式：**重新调用上传接口，相同URL或相同 `file_id` 会覆盖更新。

**支持的文件格式：**

格式类别	具体格式	扩展名/协议	处理方式
文档	PDF	<code>.pdf</code>	文本+图片提取，OCR识别扫描件
	Word	<code>.doc</code> , <code>.docx</code>	文本+表格+图片提取
	Markdown	<code>.md</code> , <code>.markdown</code>	保留格式直接入库
	纯文本	<code>.txt</code> , <code>.rtf</code>	直接入库
	EPUB	<code>.epub</code>	电子书解析
网页/URL	网页链接	<code>http://</code> , <code>https://</code>	智能识别类型，自动下载处理
	文档下载链接	<code>http://*.pdf</code> , <code>http://*.docx</code> 等	下载后按文档解析
	图片下载链接	<code>http://*.jpg</code> , <code>http://*.png</code> 等	下载后OCR识别
	音频下载链接	<code>http://*.mp3</code> , <code>http://*.wav</code> 等	下载后语音转文字
	视频下载链接	<del><code>http://*.mp4</code></del> , <del><code>http://*.avi</code></del> 等	下载后提取字幕/音频
	压缩包下载链接	<del><code>http://*.zip</code></del> , <del><code>http://*.rar</code></del> 等	下载后解压递归处理
表格	Excel	<code>.xls</code> , <code>.xlsx</code>	结构化数据，支持SQL查询
	CSV	<code>.csv</code>	结构化数据
演示	PPT	<code>.ppt</code> , <code>.pptx</code>	按页提取，保留结构
图片	图片	<code>.jpg</code> , <code>.jpeg</code> , <code>.png</code> , <code>.bmp</code> , <code>.tiff</code> , <code>.gif</code> , <code>.webp</code> , <code>.heic</code>	OCR提取文字+图片标注

音频	音频	.mp3 , .wav , .flac , .ogg , .m4a , .aac	语音转文字后入库
视频	视频	.mp4 , .avi , .webm , .mov , .mkv	提取字幕/音频转文字
代码	源代码	.py , .js , .java , .cpp , .c , .go , .rs , .ts	代码解析+注释提取
数据	JSON	.json	结构化解析
	XML	.xml	结构化解析
压缩包	压缩文件	.zip , .rar , .7z , .tar.gz	自动解压，递归处理内部文件

知识库分类（category）

分类ID	名称	说明
general	通用知识	通用养殖知识
disease	疫病诊疗	疾病诊断、治疗、预防
feeding	饲喂营养	饲料配方、营养标准
device	设备运维	设备操作、维修、保养
breeding	繁育育种	繁殖技术、遗传改良
grassland	草场管理	草畜平衡、轮牧方案
policy	政策法规	国家政策、补贴信息
market	市场行情	价格、交易、趋势

8.2 上传文件 POST /api/v1/kb/files

Content-Type: multipart/form-data

请求参数：

参数	类型	必填	说明
file	File	条件必填	file 和 url 二选一
url	string	条件必填	通用URL，支持网页/文档/图片/音频/视频/压缩包等下载链接，平台自动识别类型并下载处理

type_hint	string	✗	URL类型提示：auto (默认自动识别)/ webpage / document / image / audio / video / archive ，用于加速处理
knowledge_base	string	✗	知识库ID，默认 default
category	string	✗	分类标签，如 disease / feeding / device
description	string	✗	文件描述

请求示例：

代码块

```
1  # 方式1：上传本地文件
2  curl -X POST https://api.xxxx.com/api/v1/kb/files |
3      -H "Authorization: Bearer sk-yak-xxxxxxxxxxxxxxxxxx" |
4      -F "file=@/path/to/口蹄疫指南.pdf" |
5      -F "knowledge_base=default" |
6      -F "category=disease"
7
8  # 方式2：网页URL
9  curl -X POST https://api.xxxx.com/api/v1/kb/files |
10     -H "Authorization: Bearer sk-yak-xxxxxxxxxxxxxxxxxx" |
11     -H "Content-Type: application/json" |
12     -d '{
13         "url": "https://www.gov.cn/zhengce/2025-12-01/yak_policy.html",
14         "category": "policy",
15         "description": "国家牦牛产业政策2025"
16     }'
17
18 # 方式3：文档下载链接
19 curl -X POST https://api.xxxx.com/api/v1/kb/files |
20     -H "Authorization: Bearer sk-yak-xxxxxxxxxxxxxxxxxx" |
21     -H "Content-Type: application/json" |
22     -d '{
23         "url": "https://example.com/download/口蹄疫防治手册.pdf",
24         "type_hint": "document",
25         "category": "disease"
26     }'
27
28 # 方式4：图片下载链接
29 curl -X POST https://api.xxxx.com/api/v1/kb/files |
30     -H "Authorization: Bearer sk-yak-xxxxxxxxxxxxxxxxxx" |
31     -H "Content-Type: application/json" |
```

```

32     -d '{
33         "url": "https://example.com/images/disease-symptom.jpg",
34         "type_hint": "image",
35         "category": "disease"
36     }'
37
38 # 方式5: 音频下载链接
39 curl -X POST https://api.xxxx.com/api/v1/kb/files \
40     -H "Authorization: Bearer sk-yak-xxxxxxxxxxxxxxxxxx" \
41     -H "Content-Type: application/json" \
42     -d '{
43         "url": "https://example.com/audio/feeding-guide.mp3",
44         "type_hint": "audio",
45         "category": "feeding"
46     }'
47
48 # 方式6: 视频下载链接
49 curl -X POST https://api.xxxx.com/api/v1/kb/files \
50     -H "Authorization: Bearer sk-yak-xxxxxxxxxxxxxxxxxx" \
51     -H "Content-Type: application/json" \
52     -d '{
53         "url": "https://example.com/video/breeding-tutorial.mp4",
54         "type_hint": "video",
55         "category": "breeding"
56     }'
57
58 # 方式7: 压缩包下载链接
59 curl -X POST https://api.xxxx.com/api/v1/kb/files \
60     -H "Authorization: Bearer sk-yak-xxxxxxxxxxxxxxxxxx" \
61     -H "Content-Type: application/json" \
62     -d '{
63         "url": "https://example.com/archive/knowledge-base.zip",
64         "type_hint": "archive",
65         "category": "general"
66     }'

```

响应:

代码块

```

1  {
2      "code": 200,
3      "data": {
4          "file_id": "file_abc123",
5          "filename": "口蹄疫指南.pdf",

```

```
6      "source_type": "url",                // local/url
7      "source_url": "https://example.com/download/口蹄疫防治手册.pdf",
8      "status": "completed",
9      "detected_type": "document",        // 自动识别的类型
10     "type_hint": "document",            // 用户指定的类型提示
11     "knowledge_base": "default",
12     "category": "disease",
13     "chunks": 45,
14     "tokens": 12000,
15     "created_at": "2026-05-09T12:30:00Z",
16     "completed_at": "2026-05-09T12:31:30Z",
17     "poll_interval": 2,
18     "estimated_seconds": 30
19   }
20 }
```

8.2.1 异步处理状态获取（轮询查询）

上传文件后，系统异步处理（解析、分块、向量化）。第三方通过轮询查询获取最终状态。

轮询流程：

- 1. 调用 POST /api/v1/kb/files 上传文件/URL
- 2. 获取返回的 file\_id 和 poll\_interval
- 3. 循环调用 GET /api/v1/kb/files/{file\_id} 查询状态
- 4. 直到 status 变为 completed 或 failed，停止轮询

轮询策略：

处理阶段	轮询间隔	说明
初始阶段	按接口返回的 poll_interval（默认2秒）	快速感知小文件
超过30秒	5秒	中等文件
超过5分钟	10秒	大文件或队列积压
超过30分钟	停止轮询，标记超时	人工介入分析

处理状态说明：

--	--



状态	说明
processing	正在解析、分块、向量化, 继续轮询
completed	处理完成, 可检索
failed	处理失败, 查看error字段

## 代码示例 (python) :

代码块

```
1  import time
2  import requests
3
4  def upload_and_wait(file_path, api_key, max_wait=1800):
5      headers = {"Authorization": f"Bearer {api_key}"}
6
7      # 1. 上传文件
8      with open(file_path, "rb") as f:
9          resp = requests.post(
10              "https://api.xxxx.com/api/v1/kb/files",
11              headers=headers,
12              files={"file": f},
13              data={"category": "disease"}
14          )
15
16      result = resp.json()["data"]
17      file_id = result["file_id"]
18      status = result["status"]
19
20      # 2. 如果已完成, 直接返回
21      if status in ["completed", "failed"]:
22          return result
23
24      # 3. 轮询等待
25      start_time = time.time()
26      poll_interval = result.get("poll_interval", 2)
27
28      while status == "processing":
29          if time.time() - start_time > max_wait:
30              raise TimeoutError(f"处理超时, file_id={file_id}")
31
32          time.sleep(poll_interval)
33
34          resp = requests.get(
35              f"https://api.xxxx.com/api/v1/kb/files/{file_id}",
```

```

36         headers=headers
37     )
38     result = resp.json()["data"]
39     status = result["status"]
40
41     # 动态调整轮询间隔
42     elapsed = time.time() - start_time
43     if elapsed > 30:
44         poll_interval = 5
45     if elapsed > 300:
46         poll_interval = 10
47
48     return result
49
50 # 使用
51 result = upload_and_wait("/path/to/手册.pdf", "sk-yak-xxx")
52 print(f"处理结果: {result['status']}, 分块数: {result.get('chunks', 0)}")

```

## 代码示例 (JavaScript) :

### 代码块

```

1  async function uploadAndWait(file, apiKey, maxWait = 1800000) {
2      const headers = { "Authorization": `Bearer ${apiKey}` };
3
4      // 1. 上传文件
5      const formData = new FormData();
6      formData.append("file", file);
7      formData.append("category", "disease");
8
9      const uploadResp = await fetch("https://api.xxxx.com/api/v1/kb/files", {
10         method: "POST",
11         headers,
12         body: formData
13     });
14     const { data } = await uploadResp.json();
15     let { file_id, status, poll_interval = 2 } = data;
16
17     // 2. 如果已完成, 直接返回
18     if (status === "completed" || status === "failed") {
19         return data;
20     }
21
22     // 3. 轮询等待
23     const startTime = Date.now();
24

```

```

25     while (status === "processing") {
26         if (Date.now() - startTime > maxWait) {
27             throw new Error(`处理超时, file_id=${file_id}`);
28         }
29
30         await new Promise(r => setTimeout(r, poll_interval * 1000));
31
32         const resp = await
fetch(`https://api.xxxx.com/api/v1/kb/files/${file_id}`, {
33             headers
34         });
35         const result = await resp.json();
36         status = result.data.status;
37
38         // 动态调整轮询间隔
39         const elapsed = (Date.now() - startTime) / 1000;
40         if (elapsed > 30) poll_interval = 5;
41         if (elapsed > 300) poll_interval = 10;
42
43         if (status === "completed" || status === "failed") {
44             return result.data;
45         }
46     }
47 }
48
49 // 使用
50 uploadAndWait(fileInput.files[0], "sk-yak-xxx")
51   .then(result => console.log("完成:", result))
52   .catch(err => console.error("失败:", err));

```

## 8.3 更新知识 `PUT /api/v1/kb/files/{file_id}`

说明：重新上传同名文件或相同URL，覆盖原有内容。

请求参数：同 `POST`，`file_id` 在URL中指定

请求示例：

代码块

```

1  # 更新已有文件（重新上传）
2  curl -X PUT https://api.xxxx.com/api/v1/kb/files/file_abc123 \
3      -H "Authorization: Bearer sk-yak-xxxxxxxxxxxxxxxxxx" \
4      -F "file=@/path/to/口蹄疫指南_v2.pdf"
5
6  # 或更新URL

```

```
7 curl -X PUT https://api.xxxx.com/api/v1/kb/files/file_abc123 \  
8   -H "Authorization: Bearer sk-yak-xxxxxxxxxxxxxxxxxx" \  
9   -H "Content-Type: application/json" \  
10  -d '{  
11    "url": "https://example.com/download/口蹄疫防治手册_v2.pdf"  
12  }'
```

响应：同 POST ， version 字段+1

## 8.4 查询文件处理状态 GET /api/v1/kb/files/{file\_id}

代码块

```
1 GET /api/v1/kb/files/file_abc123  
2 Authorization: Bearer {token}
```

响应：

代码块

```
1  {  
2    "code": 200,  
3    "data": {  
4      "file_id": "file_abc123",  
5      "filename": "口蹄疫指南.pdf",  
6      "status": "completed",  
7      "knowledge_base": "default",  
8      "category": "disease",  
9      "chunks": 45,  
10     "tokens": 12000,  
11     "created_at": "2026-05-09T12:30:00Z",  
12     "completed_at": "2026-05-09T12:31:30Z"  
13   }  
14 }
```

## 8.5 列出文件 GET /api/v1/kb/files

代码块

```
1 GET /api/v1/kb/files?
  knowledge_base=default&category=disease&page=1&page_size=20
2 Authorization: Bearer {token}
```

响应:

代码块

```
1  {
2    "code": 200,
3    "data": {
4      "total": 156,
5      "page": 1,
6      "page_size": 20,
7      "files": [
8        {
9          "file_id": "file_abc123",
10         "filename": "口蹄疫指南.pdf",
11         "status": "completed",
12         "category": "disease",
13         "chunks": 45,
14         "created_at": "2026-05-09T12:30:00Z"
15       }
16     ]
17   }
18 }
```

## 8.6 删除文件 `DELETE /api/v1/kb/files/{file_id}`

代码块

```
1 DELETE /api/v1/kb/files/file_abc123
2 Authorization: Bearer {token}
```

响应:

代码块

```
1  {
2    "code": 200,
```

```
3  "message": "success",
4  "data": {
5    "deleted": true,
6    "file_id": "file_abc123"
7  }
8 }
```

8.7 检索知识 POST /api/v1/kb/retrieve

接口概括：语义检索，返回相关知识片段。

请求参数：

参数	类型	必填	默认值	说明
query	string	✓	-	查询文本
knowledge_base	string	✗	default	知识库ID
category	string	✗	-	分类过滤
top_k	integer	✗	5	返回条数
threshold	float	✗	0.7	相似度阈值

请求示例：

```
代码块
1  {
2    "query": "牦牛发烧用什么药? ",
3    "knowledge_base": "default",
4    "category": "disease",
5    "top_k": 5
6  }
```

响应：

```
1  {
2    "code": 200,
3    "message": "success",
4    "data": {
5      "query": "牦牛发烧用什么药？",
6      "results": [
7        {
8          "id": "chunk_001",
9          "content": "体温升高至40°C以上，可使用青霉素肌肉注射，每日2次...",
10         "content_type": "text",
11         "source_file": "口蹄疫指南.pdf",
12         "page": 12,
13         "category": "disease",
14         "relevance_score": 0.95
15       },
16       {
17         "id": "chunk_002",
18         "content": "高热性疾病治疗原则：先降温，再对因治疗...",
19         "content_type": "text",
20         "source_file": "牦牛疫病手册.docx",
21         "page": 45,
22         "category": "disease",
23         "relevance_score": 0.88
24       }
25     ],
26     "total": 15,
27     "latency_ms": 120
28   }
29 }
```

## 9. 错误码

### 9.1 HTTP状态码

状态码	说明
200	成功
400	请求参数错误
401	认证失败（API Key无效）
403	权限不足

429	请求频率限制
500	服务器内部错误
503	服务暂时不可用（模型过载）

## 9.2 业务错误码

错误码	说明	处理建议
<code>invalid_request</code>	请求格式错误	检查参数格式
<code>invalid_api_key</code>	API Key无效	检查API Key
<code>model_not_found</code>	模型不存在	检查model参数
<code>model_unavailable</code>	模型当前不可用	稍后重试或切换模型
<code>context_length_exceeded</code>	上下文超长	减少messages长度
<code>rate_limit_exceeded</code>	频率限制	降低请求频率
<code>kb_file_too_large</code>	文件过大	压缩或分卷上传
<code>kb_file_unsupported</code>	不支持的文件格式	转换格式后重试
<code>kb_processing_failed</code>	文件处理失败	检查文件是否损坏
<code>kb_url_download_failed</code>	URL下载失败	检查URL可访问性
<code>kb_url_not_found</code>	URL资源不存在	检查URL是否正确
<code>kb_polling_timeout</code>	轮询超时（超过30分钟）	人工分析

## 9.3 错误响应格式

代码块

```
1  {
2    "error": {
3      "message": "模型 yak-disease 当前不可用",
4      "type": "api_error",
5      "code": "model_unavailable",
```



```
6     "param": null
7   }
8 }
```

## 10. SDK示例

### 10.1 Python完整示例

代码块

```
1  from openai import OpenAI
2
3  # 初始化
4  client = OpenAI(
5      base_url="https://api.xxxx.com/v1",
6      api_key="sk-yak-xxxxxxxxxxxxxxxxxx"
7  )
8
9  # ===== 1. 普通对话 =====
10 response = client.chat.completions.create(
11     model="auto",
12     messages=[{"role": "user", "content": "我家牦牛发烧了怎么办? "}]
13 )
14 print(response.choices[0].message.content)
15
16 # ===== 2. 流式对话 =====
17 stream = client.chat.completions.create(
18     model="auto",
19     messages=[{"role": "user", "content": "牦牛怎么养? "}],
20     stream=True
21 )
22 for chunk in stream:
23     if chunk.choices[0].delta.content:
24         print(chunk.choices[0].delta.content, end="")
25
26 # ===== 3. 图片问诊 =====
27 response = client.chat.completions.create(
28     model="auto",
29     messages=[{
30         "role": "user",
31         "content": [
32             {"type": "text", "text": "这头牛皮肤怎么了? "},
```

```

33         {"type": "image_url", "image_url": {"url": "https://your-
cdn.com/cow.jpg"}}
34     ]
35 }]]
36 )
37 print(response.choices[0].message.content)
38
39 # ===== 4. 指定智能体 =====
40 response = client.chat.completions.create(
41     model="yak-disease",
42     messages=[{"role": "user", "content": "口蹄疫怎么治? "}]
43 )
44
45 # ===== 5. 向量化 =====
46 embeddings = client.embeddings.create(
47     model="auto",
48     input=["牦牛口蹄疫", "发烧用药"]
49 )
50 print(embeddings.data[0].embedding)

```

## 10.2 Node.js完整示例

代码块

```

1  import OpenAI from 'openai';
2
3  const client = new OpenAI({
4      baseURL: 'https://api.xxxx.com/v1',
5      apiKey: 'sk-yak-xxxxxxxxxxxxxxxxxx'
6  });
7
8  // 普通对话
9  const response = await client.chat.completions.create({
10     model: 'auto',
11     messages: [{ role: 'user', content: '我家牦牛发烧了怎么办? ' }]]
12 });
13 console.log(response.choices[0].message.content);
14
15 // 流式
16 const stream = await client.chat.completions.create({
17     model: 'auto',
18     messages: [{ role: 'user', content: '牦牛怎么养? ' }]],
19     stream: true
20 });

```

```
21   for await (const chunk of stream) {
22     process.stdout.write(chunk.choices[0]?.delta?.content || '');
23   }
```

## 10.3 cURL示例

代码块

```
1  # 普通对话
2  curl https://api.xxxx.com/v1/chat/completions \
3    -H "Authorization: Bearer sk-yak-xxxxxxxxxxxxxxxxxx" \
4    -H "Content-Type: application/json" \
5    -d '{
6      "model": "auto",
7      "messages": [{"role": "user", "content": "我家牦牛发烧了怎么办? "}]}
8  '
9
10 # 流式对话
11 curl https://api.xxxx.com/v1/chat/completions \
12   -H "Authorization: Bearer sk-yak-xxxxxxxxxxxxxxxxxx" \
13   -H "Content-Type: application/json" \
14   -d '{
15     "model": "auto",
16     "messages": [{"role": "user", "content": "你好"}],
17     "stream": true
18   }'
19
20 # 向量化
21 curl https://api.xxxx.com/v1/embeddings \
22 -H "Authorization: Bearer sk-yak-xxxxxxxxxxxxxxxxxx" \
23 -H "Content-Type: application/json" \
24 -d '{
25 - "model": "auto",
26 - "input": ["牦牛口蹄疫", "发烧用药"]
27 - }'
28
29 # 上传文件
30 curl -X POST https://api.xxxx.com/api/v1/kb/files \
31   -H "Authorization: Bearer sk-yak-xxxxxxxxxxxxxxxxxx" \
32   -F "file=@/path/to/手册.pdf" \
33   -F "category=disease"
34
35 # 检索知识
36 curl -X POST https://api.xxxx.com/api/v1/kb/retrieve \
```

```
37 -H "Authorization: Bearer sk-yak-xxxxxxxxxxxxxxxxxx" \  
38 -H "Content-Type: application/json" \  
39 -d '{  
40     "query": "牦牛发烧用什么药？",  
41     "category": "disease",  
42     "top_k": 5  
43 }'
```

## 附录：接口速查表

功能	方法	路径	认证
对话	POST	/v1/chat/completions	Bearer
向量化	POST	/v1/embeddings	Bearer
模型列表	GET	/v1/models	Bearer
上传/添加知识	POST	/api/v1/kb/files	Bearer
更新知识（覆盖）	PUT	/api/v1/kb/files/{file_id}	Bearer
查询文件	GET	/api/v1/kb/files/{file_id}	Bearer
列出文件	GET	/api/v1/kb/files	Bearer
删除文件	DELETE	/api/v1/kb/files/{file_id}	Bearer
检索知识	POST	/api/v1/kb/retrieve	Bearer

### 代码块

```
1  
2 版本: v1.0  
3 日期: 2026-05-09  
4 协议: HTTPS + JSON + SSE  
5  
6 接口架构
```

7	
8	第三方接口
9	
10	└─ AI对话层 (OpenAI兼容, 内部自动处理一切)
11	POST /v1/chat/completions      ← 对话 (含记忆+RAG+工具)
12	POST /v1/embeddings            ← 向量化
13	GET /v1/models                ← 模型列表
14	
15	└─ 知识库层 (自研REST API)
16	POST /api/v1/kb/files          ← 上传文件
17	PUT /api/v1/kb/files/{id}     ← 更新知识 (重新上传)
18	GET /api/v1/kb/files/{id}    ← 查询文件状态
19	GET /api/v1/kb/files          ← 列出文件
20	DELETE /api/v1/kb/files/{id} ← 删除文件
21	POST /api/v1/kb/retrieve      ← 检索知识
22	
23	

24 快速开始

25 获取API Key

26 登录平台后台 → 开发者中心 → 创建API Key

27

28 接口地址

29

30 请求示例

31 from openai import OpenAI

32

33 client = OpenAI(

34     base\_url="https://api.xxxx.com/v1",

35     api\_key="sk-yak-xxxxxxxxxxxxxxxxxx"

36 )

37

38 response = client.chat.completions.create(

39     model="auto", # 自动选择智能体

40     messages=[{"role": "user", "content": "我家牦牛发烧了怎么办? "}]

41 )

42 print(response.choices[0].message.content)

43

44 返回示例:

45 {

46     "id": "chatcmpl-abc123",

47     "object": "chat.completion",

48     "created": 1715248000,

49     "model": "yak-disease",

50     "choices": [{

51         "index": 0,

52         "message": {

53             "role": "assistant",

```
54     "content": "根据您的症状，牦牛体温升高可能由多种原因引起..."
55 },
56     "finish_reason": "stop"
57 }],
58     "usage": {
59         "prompt_tokens": 120,
60         "completion_tokens": 85,
61         "total_tokens": 205
62     }
63 }
```

64 认证方式

65 所有接口统一使用 Bearer Token 认证。

66 Authorization: Bearer {your\_api\_key}

67

68 **请求头:**

69 Content-Type: application/json

70 Authorization: Bearer sk-yak-xxxxxxxxxxxxxxxxxx

71 Bearer Token 认证可选，随意字符串

72

73 限流与配额

74

75 对话接口

76 接口概述

77

78 请求参数

79

80 **messages格式:**

```
81 [
82     {"role": "system", "content": "自定义系统提示（可选，会覆盖默认）"},
83     {"role": "user", "content": "我家牛发烧了"},
84     {"role": "assistant", "content": "体温多少度？"},
85     {"role": "user", "content": "40.5度"}
86 ]
87 ]
```

88

89 **多模态输入（图片问诊）:**

```
90 [
91     {
92         "role": "user",
93         "content": [
94             {"type": "text", "text": "这头牛皮肤怎么了？"},
95             {"type": "image_url", "image_url": {"url":
96                 "https://example.com/cow.jpg"}}
97         ]
98     }
99 ]
```

## model参数说明

### 非流式响应

```
{
  "id": "chatcmpl-abc123",
  "object": "chat.completion",
  "created": 1715248000,
  "model": "yak-disease",
  "choices": [
    {
      "index": 0,
      "message": {
        "role": "assistant",
        "content": "根据您的症状，牦牛体温40.5°C属于高热..."
      },
      "finish_reason": "stop"
    }
  ],
  "usage": {
    "prompt_tokens": 120,
    "completion_tokens": 85,
    "total_tokens": 205
  }
}
```

### 响应字段说明:

#### 流式响应 (SSE)

请求设置 `"stream": true`

#### SSE事件流:

```
data: {"id":"chatcmpl-abc","object":"chat.completion.chunk","created":1715248000,"model":"yak-disease","choices":[{"index":0,"delta":{"role":"assistant"},"finish_reason":null]}}
```

```
data: {"id":"chatcmpl-abc","object":"chat.completion.chunk","created":1715248000,"model":"yak-disease","choices":[{"index":0,"delta":{"content":"根据"},"finish_reason":null]}}
```

```
data: {"id":"chatcmpl-abc","object":"chat.completion.chunk","created":1715248000,"model":"yak-disease","choices":[{"index":0,"delta":{"content":"您描述"},"finish_reason":null]}}
```

...

```
138
139 data: {"id":"chatcmpl-
      abc","object":"chat.completion.chunk","created":1715248000,"model":"yak-
      disease","choices":[{"index":0,"delta":{},"finish_reason":"stop"}]}
```

```
140
141 data: [DONE]
```

```
142
143 客户端处理示例 (Python) :
```

```
144 stream = client.chat.completions.create(
145     model="auto",
146     messages=[{"role": "user", "content": "牦牛怎么养? "}],
147     stream=True
148 )
149
150 for chunk in stream:
151     if chunk.choices[0].delta.content:
152         print(chunk.choices[0].delta.content, end="")
```

```
153
154 会话管理
155 平台自动维护会话上下文。通过 user 和 session_id 参数控制：
156 注意：session_id在首次响应中返回，客户端需保存用于后续对话。
```

```
157
158 向量化接口(内部使用，不用关心)
```

```
159 接口概述
```

```
160
```

```
161 请求参数
```

```
162
```

```
163 请求示例
```

```
164 {
165     "model": "auto",
166     "input": ["牦牛口蹄疫症状", "发烧用药指南"]
167 }
```

```
168
```

```
169 响应
```

```
170 {
171     "object": "list",
172     "data": [
173         {
174             "object": "embedding",
175             "index": 0,
176             "embedding": [0.023, -0.156, 0.089, ...],
177             "dimensions": 1024
178         },
179         {
180             "object": "embedding",
181             "index": 1,
182             "embedding": [0.089, -0.234, 0.156, ...],
```



```
183     "dimensions": 1024
184   }
185 ],
186   "model": "bge-m3-local",
187   "usage": {
188     "prompt_tokens": 15,
189     "total_tokens": 15
190   }
191 }
192
193 模型列表接口
194 接口概述
195
196 响应
197 {
198   "object": "list",
199   "data": [
200     {
201       "id": "auto",
202       "object": "model",
203       "created": 1715248000,
204       "owned_by": "xxxx",
205       "permission": []
206     },
207     {
208       "id": "yak-general",
209       "object": "model",
210       "created": 1715248000,
211       "owned_by": "xxxx"
212     },
213     {
214       "id": "yak-disease",
215       "object": "model",
216       "created": 1715248000,
217       "owned_by": "xxxx"
218     },
219     {
220       "id": "yak-feeding",
221       "object": "model",
222       "created": 1715248000,
223       "owned_by": "xxxx"
224     },
225     {
226       "id": "qwen3-8b-local",
227       "object": "model",
228       "created": 1715248000,
229       "owned_by": "xxxx"
```

```
230     },
231     {
232         "id": "bge-m3-local",
233         "object": "model",
234         "created": 1715248000,
235         "owned_by": "xxxx"
236     }
237 ]
238 }
```

239

240 知识库接口

241 接口概述

242 知识库接口用于管理私有知识，支持多种文件格式上传、URL自动下载、自动解析、向量化、检索。

243

244 **支持两种添加方式**

245 1. **本地上传**：直接上传文件，支持文档/图片/音频/视频/压缩包等

246 2. **URL添加**：提供http/https链接，平台**智能识别内容类型，自动下载并处理**：

247 - 网页链接 → 抓取正文，过滤广告导航

248 - 文档链接 (\*.pdf/\*.docx等) → 下载后按文档解析

249 - 图片链接 (\*.jpg/\*.png等) → 下载后OCR识别

250 - 音频链接 (\*.mp3/\*.wav等) → 下载后语音转文字

251 ~~—— 视频链接 (\*.mp4/\*.avi等) → 下载后提取字幕/音频~~

252 ~~—— 压缩包链接 (\*.zip/\*.rar等) → 下载后解压递归处理~~

253

254 **URL类型识别策略**

255 - 默认`type\_hint=auto`，平台通过URL后缀+Content-Type+内容嗅探自动识别

256 - 可指定`type\_hint`加速处理，避免识别耗时

257 - 下载失败自动重试3次，支持断点续传

258 - 大文件分片下载，单文件最大200MB

259

260 **更新方式**：重新调用上传接口，相同URL或相同file\_id会覆盖更新。

261

262 **支持的文件格式：**

263 **知识库分类 (category)**

264

265 上传文件 POST /api/v1/kb/files

266 **Content-Type**: multipart/form-data

267

268 **请求参数：**

269

270 **请求示例：**

271 # 方式1：上传本地文件

272 curl -X POST https://api.xxxx.com/api/v1/kb/files \

273 -H "Authorization: Bearer sk-yak-xxxxxxxxxxxxxxxxxxxx" \

274 -F "file=@/path/to/口蹄疫指南.pdf" \

275 -F "knowledge\_base=default" \

276 -F "category=disease"

```
277
278 # 方式2: 网页URL
279 curl -X POST https://api.xxxx.com/api/v1/kb/files \
280     -H "Authorization: Bearer sk-yak-xxxxxxxxxxxxxxxxxx" \
281     -H "Content-Type: application/json" \
282     -d '{
283         "url": "https://www.gov.cn/zhengce/2025-12-01/yak_policy.html",
284         "category": "policy",
285         "description": "国家牦牛产业政策2025"
286     }'
287
288 # 方式3: 文档下载链接
289 curl -X POST https://api.xxxx.com/api/v1/kb/files \
290     -H "Authorization: Bearer sk-yak-xxxxxxxxxxxxxxxxxx" \
291     -H "Content-Type: application/json" \
292     -d '{
293         "url": "https://example.com/download/口蹄疫防治手册.pdf",
294         "type_hint": "document",
295         "category": "disease"
296     }'
297
298 # 方式4: 图片下载链接
299 curl -X POST https://api.xxxx.com/api/v1/kb/files \
300     -H "Authorization: Bearer sk-yak-xxxxxxxxxxxxxxxxxx" \
301     -H "Content-Type: application/json" \
302     -d '{
303         "url": "https://example.com/images/disease-symptom.jpg",
304         "type_hint": "image",
305         "category": "disease"
306     }'
307
308 # 方式5: 音频下载链接
309 curl -X POST https://api.xxxx.com/api/v1/kb/files \
310     -H "Authorization: Bearer sk-yak-xxxxxxxxxxxxxxxxxx" \
311     -H "Content-Type: application/json" \
312     -d '{
313         "url": "https://example.com/audio/feeding-guide.mp3",
314         "type_hint": "audio",
315         "category": "feeding"
316     }'
317
318 # 方式6: 视频下载链接
319 curl -X POST https://api.xxxx.com/api/v1/kb/files \
320 -H "Authorization: Bearer sk-yak-xxxxxxxxxxxxxxxxxx" \
321 -H "Content-Type: application/json" \
322 -d '{
323     "url": "https://example.com/video/breeding-tutorial.mp4",
```

```
324   —— "type_hint": "video",
325   —— "category": "breeding"
326   —— }'
327
328   # 方式7: 压缩包下载链接
329   curl -X POST https://api.xxxx.com/api/v1/kb/files \
330   ——H "Authorization: Bearer sk-yak-xxxxxxxxxxxxxxxxxx" \
331   ——H "Content-Type: application/json" \
332   ——d '{
333   —— "url": "https://example.com/archive/knowledge_base.zip",
334   —— "type_hint": "archive",
335   —— "category": "general"
336   —— }'
```

#### 响应:

```
339   {
340     "code": 200,
341     "data": {
342       "file_id": "file_abc123",
343       "filename": "口蹄疫指南.pdf",
344       "source_type": "url",                // local/url
345       "source_url": "https://example.com/download/口蹄疫防治手册.pdf",
346       "status": "completed",
347       "detected_type": "document",        // 自动识别的类型
348       "type_hint": "document",            // 用户指定的类型提示
349       "knowledge_base": "default",
350       "category": "disease",
351       "chunks": 45,
352       "tokens": 12000,
353       "created_at": "2026-05-09T12:30:00Z",
354       "completed_at": "2026-05-09T12:31:30Z",
355       "poll_interval": 2,
356       "estimated_seconds": 30
357     }
358   }
```

#### 异步处理状态获取（轮询查询）

上传文件后，系统异步处理（解析、分块、向量化）。第三方通过轮询查询获取最终状态。

#### 轮询流程:

1. 调用 POST /api/v1/kb/files 上传文件/URL
2. 获取返回的 file\_id 和 poll\_interval
3. 循环调用 GET /api/v1/kb/files/{file\_id} 查询状态
4. 直到 status 变为 completed 或 failed，停止轮询

#### 轮询策略:

处理状态说明:

代码示例 (python) :

```
import time
import requests

def upload_and_wait(file_path, api_key, max_wait=1800):
    headers = {"Authorization": f"Bearer {api_key}"}

    # 1. 上传文件
    with open(file_path, "rb") as f:
        resp = requests.post(
            "https://api.xxxx.com/api/v1/kb/files",
            headers=headers,
            files={"file": f},
            data={"category": "disease"}
        )

    result = resp.json()["data"]
    file_id = result["file_id"]
    status = result["status"]

    # 2. 如果已完成, 直接返回
    if status in ["completed", "failed"]:
        return result

    # 3. 轮询等待
    start_time = time.time()
    poll_interval = result.get("poll_interval", 2)

    while status == "processing":
        if time.time() - start_time > max_wait:
            raise TimeoutError(f"处理超时, file_id={file_id}")

        time.sleep(poll_interval)

        resp = requests.get(
            f"https://api.xxxx.com/api/v1/kb/files/{file_id}",
            headers=headers
        )
        result = resp.json()["data"]
        status = result["status"]

    # 动态调整轮询间隔
    elapsed = time.time() - start_time
    if elapsed > 30:
        poll_interval = 5
    if elapsed > 300:
```

```
418         poll_interval = 10
419
420     return result
421
422 # 使用
423 result = upload_and_wait("/path/to/手册.pdf", "sk-yak-xxx")
424 print(f"处理结果: {result['status']}, 分块数: {result.get('chunks', 0)}")
425 代码示例 (JavaScript) :
426 async function uploadAndWait(file, apiKey, maxWait = 1800000) {
427     const headers = { "Authorization": `Bearer ${apiKey}` };
428
429     // 1. 上传文件
430     const formData = new FormData();
431     formData.append("file", file);
432     formData.append("category", "disease");
433
434     const uploadResp = await fetch("https://api.xxxx.com/api/v1/kb/files", {
435         method: "POST",
436         headers,
437         body: formData
438     });
439     const { data } = await uploadResp.json();
440     let { file_id, status, poll_interval = 2 } = data;
441
442     // 2. 如果已完成, 直接返回
443     if (status === "completed" || status === "failed") {
444         return data;
445     }
446
447     // 3. 轮询等待
448     const startTime = Date.now();
449
450     while (status === "processing") {
451         if (Date.now() - startTime > maxWait) {
452             throw new Error(`处理超时, file_id=${file_id}`);
453         }
454
455         await new Promise(r => setTimeout(r, poll_interval * 1000));
456
457         const resp = await
458         fetch(`https://api.xxxx.com/api/v1/kb/files/${file_id}`, {
459             headers
460         });
461         const result = await resp.json();
462         status = result.data.status;
463
464         // 动态调整轮询间隔
```

```
464     const elapsed = (Date.now() - startTime) / 1000;
465     if (elapsed > 30) poll_interval = 5;
466     if (elapsed > 300) poll_interval = 10;
467
468     if (status === "completed" || status === "failed") {
469         return result.data;
470     }
471 }
472 }
473
474 // 使用
475 uploadAndWait(fileInput.files[0], "sk-yak-xxx")
476   .then(result => console.log("完成:", result))
477   .catch(err => console.error("失败:", err));
478
479 更新知识 PUT /api/v1/kb/files/{file_id}
480 说明：重新上传同名文件或相同URL，覆盖原有内容。
481 请求参数：同POST，file_id在URL中指定
482 请求示例：
483 # 更新已有文件（重新上传）
484 curl -X PUT https://api.xxxx.com/api/v1/kb/files/file_abc123 \
485   -H "Authorization: Bearer sk-yak-xxxxxxxxxxxxxxxxxx" \
486   -F "file=@/path/to/口蹄疫指南_v2.pdf"
487
488 # 或更新URL
489 curl -X PUT https://api.xxxx.com/api/v1/kb/files/file_abc123 \
490   -H "Authorization: Bearer sk-yak-xxxxxxxxxxxxxxxxxx" \
491   -H "Content-Type: application/json" \
492   -d '{
493     "url": "https://example.com/download/口蹄疫防治手册_v2.pdf"
494   }'
495 响应：同POST，version字段+1
496
497 查询文件处理状态 GET /api/v1/kb/files/{file_id}
498 GET /api/v1/kb/files/file_abc123
499 Authorization: Bearer {token}
500
501 响应：
502 {
503   "code": 200,
504   "data": {
505     "file_id": "file_abc123",
506     "filename": "口蹄疫指南.pdf",
507     "status": "completed",
508     "knowledge_base": "default",
509     "category": "disease",
510     "chunks": 45,
```

```
511     "tokens": 12000,
512     "created_at": "2026-05-09T12:30:00Z",
513     "completed_at": "2026-05-09T12:31:30Z"
514 }
515 }
516
517 列出文件 GET /api/v1/kb/files
518 GET /api/v1/kb/files?
knowledge_base=default&category=disease&page=1&page_size=20
519 Authorization: Bearer {token}
```

520  
521 **响应:**

```
522 {
523     "code": 200,
524     "data": {
525         "total": 156,
526         "page": 1,
527         "page_size": 20,
528         "files": [
529             {
530                 "file_id": "file_abc123",
531                 "filename": "口蹄疫指南.pdf",
532                 "status": "completed",
533                 "category": "disease",
534                 "chunks": 45,
535                 "created_at": "2026-05-09T12:30:00Z"
536             }
537         ]
538     }
539 }
```

```
540
541 删除文件 DELETE /api/v1/kb/files/{file_id}
542 DELETE /api/v1/kb/files/file_abc123
543 Authorization: Bearer {token}
```

544  
545 **响应:**

```
546 {
547     "code": 200,
548     "message": "success",
549     "data": {
550         "deleted": true,
551         "file_id": "file_abc123"
552     }
553 }
```

554  
555 检索知识 POST /api/v1/kb/retrieve  
556 **接口概括:** 语义检索, 返回相关知识片段。



557

558 请求参数:

559

560 请求示例:

561 {

562 "query": "牦牛发烧用什么药? ",

563 "knowledge\_base": "default",

564 "category": "disease",

565 "top\_k": 5

566 }

567

568 响应:

569 {

570 "code": 200,

571 "message": "success",

572 "data": {

573 "query": "牦牛发烧用什么药? ",

574 "results": [

575 {

576 "id": "chunk\_001",

577 "content": "体温升高至40°C以上,可使用青霉素肌肉注射,每日2次...",

578 "content\_type": "text",

579 "source\_file": "口蹄疫指南.pdf",

580 "page": 12,

581 "category": "disease",

582 "relevance\_score": 0.95

583 },

584 {

585 "id": "chunk\_002",

586 "content": "高热性疾病治疗原则:先降温,再对因治疗...",

587 "content\_type": "text",

588 "source\_file": "牦牛疫病手册.docx",

589 "page": 45,

590 "category": "disease",

591 "relevance\_score": 0.88

592 }

593 ],

594 "total": 15,

595 "latency\_ms": 120

596 }

597 }

598 错误码

599 HTTP状态码

600

601 业务错误码

602

603 错误响应格式

```

604 {
605     "error": {
606         "message": "模型 yak-disease 当前不可用",
607         "type": "api_error",
608         "code": "model_unavailable",
609         "param": null
610     }
611 }
612
613 SDK示例
614 Python完整示例
615 from openai import OpenAI
616
617 # 初始化
618 client = OpenAI(
619     base_url="https://api.xxxx.com/v1",
620     api_key="sk-yak-xxxxxxxxxxxxxxxxxx"
621 )
622
623 # ===== 1. 普通对话 =====
624 response = client.chat.completions.create(
625     model="auto",
626     messages=[{"role": "user", "content": "我家牦牛发烧了怎么办? "}]
627 )
628 print(response.choices[0].message.content)
629
630 # ===== 2. 流式对话 =====
631 stream = client.chat.completions.create(
632     model="auto",
633     messages=[{"role": "user", "content": "牦牛怎么养? "}],
634     stream=True
635 )
636 for chunk in stream:
637     if chunk.choices[0].delta.content:
638         print(chunk.choices[0].delta.content, end="")
639
640 # ===== 3. 图片问诊 =====
641 response = client.chat.completions.create(
642     model="auto",
643     messages=[{
644         "role": "user",
645         "content": [
646             {"type": "text", "text": "这头牛皮肤怎么了? "},
647             {"type": "image_url", "image_url": {"url": "https://your-
648 cdn.com/cow.jpg"}}
649         ]
650     }]

```

```

650 )
651 print(response.choices[0].message.content)
652
653 # ===== 4. 指定智能体 =====
654 response = client.chat.completions.create(
655     model="yak-disease",
656     messages=[{"role": "user", "content": "口蹄疫怎么治? "}]
657 )
658
659 # ===== 5. 向量化 =====
660 embeddings = client.embeddings.create(
661     model="auto",
662     input=["牦牛口蹄疫", "发烧用药"]
663 )
664 print(embeddings.data[0].embedding)
665
666 Node.js完整示例
667 import OpenAI from 'openai';
668
669 const client = new OpenAI({
670     baseURL: 'https://api.xxxx.com/v1',
671     apiKey: 'sk-yak-xxxxxxxxxxxxxxxxxx'
672 });
673
674 // 普通对话
675 const response = await client.chat.completions.create({
676     model: 'auto',
677     messages: [{ role: 'user', content: '我家牦牛发烧了怎么办? ' }]}
678 );
679 console.log(response.choices[0].message.content);
680
681 // 流式
682 const stream = await client.chat.completions.create({
683     model: 'auto',
684     messages: [{ role: 'user', content: '牦牛怎么养? ' }],
685     stream: true
686 });
687 for await (const chunk of stream) {
688     process.stdout.write(chunk.choices[0]?.delta?.content || '');
689 }
690
691 cURL示例
692 # 普通对话
693 curl https://api.xxxx.com/v1/chat/completions \
694     -H "Authorization: Bearer sk-yak-xxxxxxxxxxxxxxxxxx" \
695     -H "Content-Type: application/json" \
696     -d '{

```

```
697     "model": "auto",
698     "messages": [{"role": "user", "content": "我家牦牛发烧了怎么办? "}]
699   }'
700
701 # 流式对话
702 curl https://api.xxxx.com/v1/chat/completions \
703   -H "Authorization: Bearer sk-yak-xxxxxxxxxxxxxxxxxx" \
704   -H "Content-Type: application/json" \
705   -d '{
706     "model": "auto",
707     "messages": [{"role": "user", "content": "你好"}],
708     "stream": true
709   }'
710
711 # 向量化
712 curl https://api.xxxx.com/v1/embeddings \
713 -H "Authorization: Bearer sk-yak-xxxxxxxxxxxxxxxxxx" \
714 -H "Content-Type: application/json" \
715 -d '{
716   "model": "auto",
717   "input": ["牦牛口蹄疫", "发烧用药"]
718 }'
719
720 # 上传文件
721 curl -X POST https://api.xxxx.com/api/v1/kb/files \
722   -H "Authorization: Bearer sk-yak-xxxxxxxxxxxxxxxxxx" \
723   -F "file=@/path/to/手册.pdf" \
724   -F "category=disease"
725
726 # 检索知识
727 curl -X POST https://api.xxxx.com/api/v1/kb/retrieve \
728   -H "Authorization: Bearer sk-yak-xxxxxxxxxxxxxxxxxx" \
729   -H "Content-Type: application/json" \
730   -d '{
731     "query": "牦牛发烧用什么药? ",
732     "category": "disease",
733     "top_k": 5
734   }'
```