我需要先了解当前AI应用的架构和功能，以评估如何实现自定义知识库的关联功能。首先搜索与AI对话和知识处理相关的代码。

toolName: search\_codebase

status: success

query: AI robot dialog knowledge base chat

我需要更深入了解系统的数据库结构和模型定义，特别是与知识库可能相关的表结构。

toolName: search\_codebase

status: success

query: database model schema knowledge base table structure

我需要查看数据库安装脚本，了解系统的表结构设计，特别是与AI对话和知识库相关的表。

toolName: view\_files

status: success

filePath: /Users/yangboz/Documents/PoCs/chatAI-MPwithoutTrademark/chat/install.sql

我需要继续查看数据库安装脚本，特别关注与AI对话和知识库相关的表结构。

toolName: view\_files

status: success

filePath: /Users/yangboz/Documents/PoCs/chatAI-MPwithoutTrademark/chat/install.sql

我需要继续查看数据库安装脚本，特别关注与AI对话和知识库相关的表结构。

toolName: view\_files

status: success

filePath: /Users/yangboz/Documents/PoCs/chatAI-MPwithoutTrademark/chat/install.sql

我需要查找与知识库相关的代码实现，看看是否有现有的知识库功能或相关接口。

toolName: search\_codebase

status: success

query: knowledge base embedding vector search

我需要查找系统中与AI对话和模型相关的代码，以了解如何实现知识库功能的集成。

toolName: search\_codebase

status: success

query: robot model chat dialog api controller

我需要查看AI对话的核心实现，特别是Base.php和Chat.php文件，了解对话处理流程和如何集成知识库功能。

toolName: view\_files

status: success

filePath: /Users/yangboz/Documents/PoCs/chatAI-MPwithoutTrademark/chat/application/api/controller/robot/Base.php

我需要继续查看Base.php文件的剩余部分，特别是与对话生成相关的方法。

toolName: view\_files

status: success

filePath: /Users/yangboz/Documents/PoCs/chatAI-MPwithoutTrademark/chat/application/api/controller/robot/Base.php

我需要查看Chat.php文件，了解对话创建和处理的具体实现。

toolName: view\_files

status: success

filePath: /Users/yangboz/Documents/PoCs/chatAI-MPwithoutTrademark/chat/application/api/controller/robot/Chat.php

我需要继续查看Chat.php文件的剩余部分，特别是流式输出的处理逻辑。

toolName: view\_files

status: success

filePath: /Users/yangboz/Documents/PoCs/chatAI-MPwithoutTrademark/chat/application/api/controller/robot/Chat.php

我需要查看一下Chatgpt.php文件，了解AI对话实现的具体细节，特别是与知识库可能相关的部分。

toolName: view\_files

status: success

filePath: /Users/yangboz/Documents/PoCs/chatAI-MPwithoutTrademark/chat/application/api/library/robot/Chatgpt.php

我需要查找是否有与知识库相关的代码或功能，特别是向量搜索或嵌入相关的实现。

toolName: search\_codebase

status: success

query: knowledge base vector embedding search

我需要查找是否有与向量数据库或知识库存储相关的代码，特别是在数据库结构或模型定义中。

toolName: search\_codebase

status: success

query: vector database knowledge base storage model

我需要查找与AI对话模型相关的代码，特别是不同的AI模型实现，以了解如何集成知识库功能。

toolName: search\_codebase

status: success

query: AI model implementation Chatgpt Wenxin Xunfei

我需要查看一下Channel.php接口文件，了解AI模型接口的定义和实现方式。

toolName: view\_files

status: success

filePath: /Users/yangboz/Documents/PoCs/chatAI-MPwithoutTrademark/chat/application/api/library/robot/Channel.php

我需要查看Base.php文件中的\_create方法，了解对话生成的核心逻辑，以便分析如何集成知识库功能。

toolName: view\_files

status: success

filePath: /Users/yangboz/Documents/PoCs/chatAI-MPwithoutTrademark/chat/application/api/controller/robot/Base.php

# 基于现有PHP代码实现AI应用关联自定义知识库的技术可行性分析

经过对代码库的详细分析，我认为在现有系统基础上增加自定义知识库功能是技术上可行的。以下是实现方案和所需修改：

## 一、现有系统架构分析

### 1. 核心对话处理流程

现有系统采用了以下流程处理AI对话：

1. 用户通过`Chat.php`的`create`方法发起对话请求

2. 系统在`Base.php`的`\_create`方法中处理对话生成：

- 获取频道关联的模型列表

- 构建对话内容（包括历史对话）

- 按序调用不同模型的API接口

- 记录错误日志并尝试下一个模型

3. 各模型实现（如`Chatgpt.php`、`Wenxin.php`、`Xunfei.php`等）通过`Channel`接口提供统一的`chat`和`chatStream`方法

### 2. 数据库结构

系统使用多个表存储对话相关信息：

- `robot\_creation`：存储对话内容和结果

- `robot\_session`：管理对话会话

- `robot\_model`：配置不同AI模型

- `robot\_channel`：管理不同渠道

- `robot\_assistant`：配置AI助手

## 二、知识库功能实现方案

### 1. 数据库扩展

需要新增以下表：

```sql

CREATE TABLE `ai\_knowledge\_base` (

`id` int(11) NOT NULL AUTO\_INCREMENT,

`name` varchar(255) NOT NULL COMMENT '知识库名称',

`description` text COMMENT '知识库描述',

`user\_id` int(11) NOT NULL COMMENT '所属用户ID',

`status` tinyint(1) NOT NULL DEFAULT '1' COMMENT '状态：0=禁用，1=启用',

`createtime` int(10) NOT NULL COMMENT '创建时间',

`updatetime` int(10) NOT NULL COMMENT '更新时间',

PRIMARY KEY (`id`),

KEY `user\_id` (`user\_id`)

) ENGINE=InnoDB DEFAULT CHARSET=utf8mb4 COMMENT='知识库表';

CREATE TABLE `ai\_knowledge\_document` (

`id` int(11) NOT NULL AUTO\_INCREMENT,

`kb\_id` int(11) NOT NULL COMMENT '知识库ID',

`title` varchar(255) NOT NULL COMMENT '文档标题',

`content` longtext NOT NULL COMMENT '文档内容',

`file\_type` varchar(50) DEFAULT NULL COMMENT '文件类型',

`file\_path` varchar(255) DEFAULT NULL COMMENT '文件路径',

`status` tinyint(1) NOT NULL DEFAULT '1' COMMENT '状态：0=禁用，1=启用',

`createtime` int(10) NOT NULL COMMENT '创建时间',

`updatetime` int(10) NOT NULL COMMENT '更新时间',

PRIMARY KEY (`id`),

KEY `kb\_id` (`kb\_id`)

) ENGINE=InnoDB DEFAULT CHARSET=utf8mb4 COMMENT='知识库文档表';

CREATE TABLE `ai\_knowledge\_chunk` (

`id` int(11) NOT NULL AUTO\_INCREMENT,

`doc\_id` int(11) NOT NULL COMMENT '文档ID',

`content` text NOT NULL COMMENT '文本块内容',

`embedding` longtext COMMENT '向量嵌入JSON',

`metadata` text COMMENT '元数据',

`createtime` int(10) NOT NULL COMMENT '创建时间',

PRIMARY KEY (`id`),

KEY `doc\_id` (`doc\_id`)

) ENGINE=InnoDB DEFAULT CHARSET=utf8mb4 COMMENT='知识库文本块表';

CREATE TABLE `ai\_robot\_channel\_kb` (

`id` int(11) NOT NULL AUTO\_INCREMENT,

`channel\_id` int(11) NOT NULL COMMENT '渠道ID',

`kb\_id` int(11) NOT NULL COMMENT '知识库ID',

`createtime` int(10) NOT NULL COMMENT '创建时间',

PRIMARY KEY (`id`),

UNIQUE KEY `channel\_kb` (`channel\_id`,`kb\_id`)

) ENGINE=InnoDB DEFAULT CHARSET=utf8mb4 COMMENT='渠道知识库关联表';

```

### 2. 核心功能模块

#### 2.1 知识库管理模块

创建新的控制器`KnowledgeBase.php`，实现以下功能：

```php

<?php

namespace app\api\controller\robot;

use app\common\model\robot\KnowledgeBase as KBModel;

use app\common\model\robot\KnowledgeDocument as DocModel;

use app\common\model\robot\KnowledgeChunk as ChunkModel;

use think\Db;

class KnowledgeBase extends Base

{

/\*\*

\* 创建知识库

\*/

public function create()

{

if (!$this->uid) {

$this->response(1003);

}

$param = $this->request->post();

$name = isset($param['name']) ? $param['name'] : '';

$description = isset($param['description']) ? $param['description'] : '';

if (!$name) {

$this->response(9999, [], '请输入知识库名称');

}

$data = [

'name' => $name,

'description' => $description,

'user\_id' => $this->uid,

'status' => 1,

'createtime' => time(),

'updatetime' => time()

];

$model = new KBModel();

$id = $model->insertGetId($data);

if (!$id) {

$this->response(9999, [], '创建知识库失败');

}

$this->response(0, ['id' => $id]);

}

/\*\*

\* 上传文档到知识库

\*/

public function uploadDocument()

{

// 文档上传和处理逻辑

}

/\*\*

\* 生成文档向量嵌入

\*/

public function generateEmbeddings()

{

// 调用向量嵌入API生成向量

}

/\*\*

\* 关联知识库到渠道

\*/

public function linkToChannel()

{

// 关联知识库到特定渠道

}

}

```

#### 2.2 向量嵌入服务

创建新的库文件`VectorService.php`，实现向量嵌入和相似度搜索：

```php

<?php

namespace app\api\library\robot;

use think\Cache;

class VectorService extends Base

{

/\*\*

\* 生成文本向量嵌入

\*/

public function generateEmbedding($text, $model)

{

// 根据不同模型调用不同的嵌入API

switch ($model['model\_class']) {

case 'Chatgpt':

return $this->openaiEmbedding($text, $model);

case 'Wenxin':

return $this->baiduEmbedding($text, $model);

// 其他模型的嵌入实现

default:

return null;

}

}

/\*\*

\* OpenAI嵌入实现

\*/

private function openaiEmbedding($text, $model)

{

$apiKey = $model['authorization'];

$url = 'https://api.openai.com/v1/embeddings';

if ($model['api\_url']) {

$url = str\_replace('chat/completions', 'embeddings', $model['api\_url']);

}

$header = [

'Content-Type: application/json',

'Authorization: Bearer ' . $apiKey

];

$params = [

'input' => $text,

'model' => 'text-embedding-ada-002'

];

$res = $this->\_curl($url, 1, json\_encode($params), $header, [], 300);

$data = json\_decode($res, true);

if (isset($data['data'][0]['embedding'])) {

return $data['data'][0]['embedding'];

}

return null;

}

/\*\*

\* 百度文心嵌入实现

\*/

private function baiduEmbedding($text, $model)

{

// 实现百度文心的嵌入API调用

}

/\*\*

\* 向量相似度搜索

\*/

public function searchSimilar($query, $kbIds, $limit = 5)

{

// 1. 生成查询向量

// 2. 在知识库中搜索相似向量

// 3. 返回最相似的文本块

}

}

```

### 3. 集成到现有对话流程

修改`Base.php`中的`\_create`方法，增加知识库检索和上下文增强：

```php

protected function \_create($prompt, $sessionId, $stream = 1, $connection = '')

{

$model = new Channelmodel();

$models = $model->getModelListByChannel($this->channel['id']);

$models = array\_slice($models, 0, 4);

$res = [];

$content = [];

// 获取关联的知识库

$kbIds = Db::name('robot\_channel\_kb')

->where('channel\_id', $this->channel['id'])

->column('kb\_id');

// 如果有关联知识库，进行知识检索

$relevantContext = '';

if (!empty($kbIds)) {

$vectorService = new VectorService();

$similarDocs = $vectorService->searchSimilar($prompt, $kbIds);

if (!empty($similarDocs)) {

$relevantContext = "根据知识库中的以下信息回答：\n";

foreach ($similarDocs as $doc) {

$relevantContext .= $doc['content'] . "\n";

}

$relevantContext .= "\n用户问题：";

}

}

// 构建对话历史

if ($this->channel['multiple']) {

$map['user\_id'] = $this->uid;

$map['session\_id'] = $sessionId;

$map['assistant\_id'] = 0;

$history = Db::name('robot\_creation')->where($map)->order('id asc')->limit(2)->select();

foreach ($history as $k => $v) {

if ($v['content'] && $v['msg']) {

$content[] = ['role' => 'user', 'content' => $v['content']];

$content[] = ['role' => 'assistant', 'content' => $v['msg']];

}

}

}

// 如果有知识库上下文，添加到提示中

$finalPrompt = $prompt;

if (!empty($relevantContext)) {

$finalPrompt = $relevantContext . $prompt;

}

$content[] = ['role' => 'user', 'content' => $finalPrompt];

// 后续代码与原来相同...

}

```

## 三、前端界面扩展

### 1. 后台管理界面

需要在后台添加知识库管理页面，包括：

- 知识库列表管理

- 文档上传和管理

- 渠道关联知识库配置

### 2. 用户界面

在小程序和PC端添加知识库相关功能：

- 知识库创建和管理

- 文档上传和处理

- 对话时选择使用的知识库

## 四、技术挑战与解决方案

### 1. 向量数据存储

\*\*挑战\*\*：MySQL不是专为向量搜索设计的数据库

\*\*解决方案\*\*：

- 短期：将向量JSON序列化存储在MySQL中，使用PHP实现简单的余弦相似度计算

- 长期：集成专用向量数据库如Milvus、Pinecone或Qdrant

### 2. 文档处理

\*\*挑战\*\*：需要处理多种格式的文档（PDF、Word、Excel等）

\*\*解决方案\*\*：

- 使用PHP扩展库如PHPWord、TCPDF等解析文档

- 实现文档分块策略，将长文本分割成适合嵌入的小块

### 3. API限制

\*\*挑战\*\*：向量嵌入API可能有调用频率和成本限制

\*\*解决方案\*\*：

- 实现嵌入缓存机制

- 批量处理嵌入请求

- 实现队列系统处理大量文档

## 五、实施路径

### 第一阶段：基础架构

1. 创建数据库表结构

2. 实现知识库和文档管理API

3. 开发向量嵌入服务

### 第二阶段：核心功能

1. 实现文档处理和分块

2. 开发向量相似度搜索

3. 集成到对话流程

### 第三阶段：用户界面

1. 开发后台管理界面

2. 实现小程序和PC端知识库管理

3. 优化用户体验

## 六、结论

基于现有代码架构，实现AI应用关联自定义知识库功能在技术上是完全可行的。系统已有的模块化设计和接口抽象为扩展提供了良好基础。主要工作集中在向量嵌入服务的实现、知识库管理功能的开发，以及将检索结果集成到现有对话流程中。

通过这些扩展，系统将能够提供基于用户自定义知识的AI对话能力，大幅提升应用的实用性和定制化程度。

另

我需要查看现有代码库中是否有与Elasticsearch相关的配置或实现，以便提供基于ES的向量数据库实现方案。

toolName: view\_files

status: success

filePath: /Users/yangboz/Documents/PoCs/chatAI-MPwithoutTrademark/chat/application/database.php

我需要查找项目中是否有与Elasticsearch相关的代码或配置，以便了解如何集成ES作为向量数据库。

toolName: search\_codebase

status: success

query: elasticsearch elastic search ES

# 基于Elasticsearch实现向量数据库的方案

根据对代码库的分析，当前系统使用的是MySQL数据库，没有集成Elasticsearch。下面我将提供一个完整的基于Elasticsearch实现向量数据库的方案，用于支持知识库的向量检索功能。

## 一、Elasticsearch配置与安装

### 1. 安装Elasticsearch

```bash

# 使用Docker安装Elasticsearch

docker pull docker.elastic.co/elasticsearch/elasticsearch:8.10.4

# 运行Elasticsearch容器

docker run -d --name elasticsearch -p 9200:9200 -p 9300:9300 \

-e "discovery.type=single-node" \

-e "xpack.security.enabled=false" \

docker.elastic.co/elasticsearch/elasticsearch:8.10.4

```

### 2. 安装PHP的Elasticsearch客户端

```bash

cd /Users/yangboz/Documents/PoCs/chatAI-MPwithoutTrademark/chat

composer require elasticsearch/elasticsearch

```

## 二、创建Elasticsearch服务类

### 1. 创建ES配置文件

在`.env`文件中添加Elasticsearch配置：

```

[ELASTICSEARCH]

ES\_HOST=localhost

ES\_PORT=9200

ES\_INDEX\_PREFIX=ai\_kb\_

```

### 2. 创建Elasticsearch服务类

```php:chat/application/api/library/robot/ElasticsearchService.php

<?php

namespace app\api\library\robot;

use Elasticsearch\ClientBuilder;

use think\Env;

use think\Log;

class ElasticsearchService

{

protected $client;

protected $indexPrefix;

public function \_\_construct()

{

$host = Env::get('elasticsearch.es\_host', 'localhost');

$port = Env::get('elasticsearch.es\_port', '9200');

$this->indexPrefix = Env::get('elasticsearch.es\_index\_prefix', 'ai\_kb\_');

try {

$this->client = ClientBuilder::create()

->setHosts(["$host:$port"])

->build();

} catch (\Exception $e) {

Log::error('Elasticsearch连接失败: ' . $e->getMessage());

}

}

/\*\*

\* 创建知识库索引

\*/

public function createKnowledgeBaseIndex($kbId)

{

$indexName = $this->getIndexName($kbId);

// 检查索引是否存在

if ($this->client->indices()->exists(['index' => $indexName])) {

return true;

}

// 创建索引配置，包含向量字段

$params = [

'index' => $indexName,

'body' => [

'mappings' => [

'properties' => [

'doc\_id' => ['type' => 'integer'],

'chunk\_id' => ['type' => 'integer'],

'content' => ['type' => 'text', 'analyzer' => 'ik\_max\_word'],

'metadata' => ['type' => 'object'],

'vector' => [

'type' => 'dense\_vector',

'dims' => 1536, // OpenAI嵌入维度

'index' => true,

'similarity' => 'cosine'

]

]

],

'settings' => [

'number\_of\_shards' => 1,

'number\_of\_replicas' => 0

]

]

];

try {

$response = $this->client->indices()->create($params);

return isset($response['acknowledged']) && $response['acknowledged'];

} catch (\Exception $e) {

Log::error('创建Elasticsearch索引失败: ' . $e->getMessage());

return false;

}

}

/\*\*

\* 索引文本块及其向量

\*/

public function indexTextChunk($kbId, $docId, $chunkId, $content, $vector, $metadata = [])

{

$indexName = $this->getIndexName($kbId);

$params = [

'index' => $indexName,

'id' => $chunkId,

'body' => [

'doc\_id' => $docId,

'chunk\_id' => $chunkId,

'content' => $content,

'vector' => $vector,

'metadata' => $metadata

]

];

try {

$response = $this->client->index($params);

return isset($response['result']) && ($response['result'] == 'created' || $response['result'] == 'updated');

} catch (\Exception $e) {

Log::error('索引文本块失败: ' . $e->getMessage());

return false;

}

}

/\*\*

\* 向量相似度搜索

\*/

public function searchSimilar($kbIds, $vector, $limit = 5)

{

if (empty($kbIds)) {

return [];

}

$indices = [];

foreach ($kbIds as $kbId) {

$indices[] = $this->getIndexName($kbId);

}

$params = [

'index' => implode(',', $indices),

'body' => [

'size' => $limit,

'query' => [

'script\_score' => [

'query' => ['match\_all' => new \stdClass()],

'script' => [

'source' => "cosineSimilarity(params.query\_vector, 'vector') + 1.0",

'params' => ['query\_vector' => $vector]

]

]

],

'\_source' => ['doc\_id', 'chunk\_id', 'content', 'metadata']

]

];

try {

$response = $this->client->search($params);

$results = [];

if (isset($response['hits']['hits']) && !empty($response['hits']['hits'])) {

foreach ($response['hits']['hits'] as $hit) {

$results[] = [

'doc\_id' => $hit['\_source']['doc\_id'],

'chunk\_id' => $hit['\_source']['chunk\_id'],

'content' => $hit['\_source']['content'],

'metadata' => $hit['\_source']['metadata'],

'score' => $hit['\_score']

];

}

}

return $results;

} catch (\Exception $e) {

Log::error('向量搜索失败: ' . $e->getMessage());

return [];

}

}

/\*\*

\* 删除文档的所有文本块

\*/

public function deleteDocumentChunks($kbId, $docId)

{

$indexName = $this->getIndexName($kbId);

$params = [

'index' => $indexName,

'body' => [

'query' => [

'term' => ['doc\_id' => $docId]

]

]

];

try {

$response = $this->client->deleteByQuery($params);

return isset($response['deleted']) && $response['deleted'] > 0;

} catch (\Exception $e) {

Log::error('删除文档块失败: ' . $e->getMessage());

return false;

}

}

/\*\*

\* 删除知识库索引

\*/

public function deleteKnowledgeBaseIndex($kbId)

{

$indexName = $this->getIndexName($kbId);

if (!$this->client->indices()->exists(['index' => $indexName])) {

return true;

}

try {

$response = $this->client->indices()->delete(['index' => $indexName]);

return isset($response['acknowledged']) && $response['acknowledged'];

} catch (\Exception $e) {

Log::error('删除索引失败: ' . $e->getMessage());

return false;

}

}

/\*\*

\* 获取索引名称

\*/

protected function getIndexName($kbId)

{

return $this->indexPrefix . $kbId;

}

}

```

## 三、创建向量服务类

```php:chat/application/api/library/robot/VectorService.php

<?php

namespace app\api\library\robot;

use think\Db;

use think\Log;

class VectorService extends Base

{

protected $esService;

public function \_\_construct()

{

$this->esService = new ElasticsearchService();

}

/\*\*

\* 生成文本向量嵌入

\*/

public function generateEmbedding($text, $model)

{

// 根据不同模型调用不同的嵌入API

switch ($model['model\_class']) {

case 'Chatgpt':

return $this->openaiEmbedding($text, $model);

case 'Wenxin':

return $this->baiduEmbedding($text, $model);

// 其他模型的嵌入实现

default:

return null;

}

}

/\*\*

\* OpenAI嵌入实现

\*/

private function openaiEmbedding($text, $model)

{

$apiKey = $model['authorization'];

$url = 'https://api.openai.com/v1/embeddings';

if ($model['api\_url']) {

$url = str\_replace('chat/completions', 'embeddings', $model['api\_url']);

}

$header = [

'Content-Type: application/json',

'Authorization: Bearer ' . $apiKey

];

$params = [

'input' => $text,

'model' => 'text-embedding-ada-002'

];

$res = $this->\_curl($url, 1, json\_encode($params), $header, [], 300);

$data = json\_decode($res, true);

if (isset($data['data'][0]['embedding'])) {

return $data['data'][0]['embedding'];

}

return null;

}

/\*\*

\* 百度文心嵌入实现

\*/

private function baiduEmbedding($text, $model)

{

$appKey = $model['appkey'];

$appSecret = $model['appsecret'];

// 获取访问令牌

$tokenUrl = "https://aip.baidubce.com/oauth/2.0/token?grant\_type=client\_credentials&client\_id={$appKey}&client\_secret={$appSecret}";

$tokenRes = $this->\_curl($tokenUrl, 0, '', [], [], 60);

$tokenData = json\_decode($tokenRes, true);

if (!isset($tokenData['access\_token'])) {

Log::error('获取百度文心访问令牌失败: ' . $tokenRes);

return null;

}

$accessToken = $tokenData['access\_token'];

$url = "https://aip.baidubce.com/rpc/2.0/ai\_custom/v1/wenxinworkshop/embeddings/embedding-v1?access\_token={$accessToken}";

$header = ['Content-Type: application/json'];

$params = ['input' => $text];

$res = $this->\_curl($url, 1, json\_encode($params), $header, [], 300);

$data = json\_decode($res, true);

if (isset($data['data'][0]['embedding'])) {

return $data['data'][0]['embedding'];

}

return null;

}

/\*\*

\* 处理文档并生成向量嵌入

\*/

public function processDocument($kbId, $docId, $content, $model)

{

// 确保索引存在

$this->esService->createKnowledgeBaseIndex($kbId);

// 清除文档之前的块

$this->esService->deleteDocumentChunks($kbId, $docId);

// 分割文本为块

$chunks = $this->splitTextIntoChunks($content);

$results = [];

foreach ($chunks as $index => $chunk) {

// 生成向量嵌入

$vector = $this->generateEmbedding($chunk, $model);

if ($vector) {

// 保存到数据库

$chunkData = [

'doc\_id' => $docId,

'content' => $chunk,

'createtime' => time()

];

$chunkId = Db::name('knowledge\_chunk')->insertGetId($chunkData);

// 索引到ES

$metadata = ['chunk\_index' => $index];

$success = $this->esService->indexTextChunk($kbId, $docId, $chunkId, $chunk, $vector, $metadata);

if ($success) {

$results[] = $chunkId;

}

}

}

return $results;

}

/\*\*

\* 分割文本为块

\*/

private function splitTextIntoChunks($text, $maxChunkSize = 1000, $overlap = 100)

{

$text = str\_replace("\r\n", "\n", $text);

$paragraphs = explode("\n\n", $text);

$chunks = [];

$currentChunk = "";

foreach ($paragraphs as $paragraph) {

$paragraph = trim($paragraph);

if (empty($paragraph)) continue;

// 如果段落本身超过最大块大小，进一步分割

if (mb\_strlen($paragraph) > $maxChunkSize) {

$sentences = preg\_split('/(?<=[.!?。！？])\s+/', $paragraph);

foreach ($sentences as $sentence) {

if (mb\_strlen($currentChunk) + mb\_strlen($sentence) <= $maxChunkSize) {

$currentChunk .= ($currentChunk ? " " : "") . $sentence;

} else {

if (!empty($currentChunk)) {

$chunks[] = $currentChunk;

}

$currentChunk = $sentence;

}

}

} else {

// 检查添加这个段落是否会超过最大块大小

if (mb\_strlen($currentChunk) + mb\_strlen($paragraph) + 1 <= $maxChunkSize) {

$currentChunk .= ($currentChunk ? "\n" : "") . $paragraph;

} else {

if (!empty($currentChunk)) {

$chunks[] = $currentChunk;

}

$currentChunk = $paragraph;

}

}

}

// 添加最后一个块

if (!empty($currentChunk)) {

$chunks[] = $currentChunk;

}

// 处理重叠

if ($overlap > 0 && count($chunks) > 1) {

$overlappedChunks = [];

for ($i = 0; $i < count($chunks); $i++) {

if ($i > 0) {

// 获取前一个块的末尾

$prevChunk = $chunks[$i-1];

$overlapText = mb\_substr($prevChunk, -$overlap);

$chunks[$i] = $overlapText . " " . $chunks[$i];

}

$overlappedChunks[] = $chunks[$i];

}

return $overlappedChunks;

}

return $chunks;

}

/\*\*

\* 向量相似度搜索

\*/

public function searchSimilar($query, $kbIds, $model, $limit = 5)

{

// 生成查询向量

$queryVector = $this->generateEmbedding($query, $model);

if (!$queryVector) {

return [];

}

// 在ES中搜索相似向量

return $this->esService->searchSimilar($kbIds, $queryVector, $limit);

}

}

```

## 四、创建知识库控制器

```php:chat/application/api/controller/robot/KnowledgeBase.php

<?php

namespace app\api\controller\robot;

use app\api\library\robot\VectorService;

use app\common\model\robot\KnowledgeBase as KBModel;

use app\common\model\robot\KnowledgeDocument as DocModel;

use app\common\model\robot\Channelmodel;

use think\Db;

use think\Exception;

class KnowledgeBase extends Base

{

protected $vectorService;

public function \_\_construct()

{

parent::\_\_construct();

$this->vectorService = new VectorService();

}

/\*\*

\* 创建知识库

\*/

public function create()

{

if (!$this->uid) {

$this->response(1003);

}

$param = $this->request->post();

$name = isset($param['name']) ? $param['name'] : '';

$description = isset($param['description']) ? $param['description'] : '';

if (!$name) {

$this->response(9999, [], '请输入知识库名称');

}

$data = [

'name' => $name,

'description' => $description,

'user\_id' => $this->uid,

'status' => 1,

'createtime' => time(),

'updatetime' => time()

];

try {

$id = Db::name('knowledge\_base')->insertGetId($data);

if (!$id) {

$this->response(9999, [], '创建知识库失败');

}

$this->response(0, ['id' => $id]);

} catch (Exception $e) {

$this->response(9999, [], '创建知识库失败: ' . $e->getMessage());

}

}

/\*\*

\* 上传文档到知识库

\*/

public function uploadDocument()

{

if (!$this->uid) {

$this->response(1003);

}

$param = $this->request->post();

$kbId = isset($param['kb\_id']) ? intval($param['kb\_id']) : 0;

$title = isset($param['title']) ? $param['title'] : '';

$content = isset($param['content']) ? $param['content'] : '';

if (!$kbId || !$title || !$content) {

$this->response(9999, [], '参数不完整');

}

// 检查知识库是否存在且属于当前用户

$kb = Db::name('knowledge\_base')

->where('id', $kbId)

->where('user\_id', $this->uid)

->find();

if (!$kb) {

$this->response(9999, [], '知识库不存在或无权限');

}

// 保存文档

$docData = [

'kb\_id' => $kbId,

'title' => $title,

'content' => $content,

'status' => 1,

'createtime' => time(),

'updatetime' => time()

];

try {

$docId = Db::name('knowledge\_document')->insertGetId($docData);

if (!$docId) {

$this->response(9999, [], '保存文档失败');

}

// 获取默认模型用于生成向量

$modelModel = new Channelmodel();

$models = $modelModel->getModelListByChannel($this->channel['id']);

if (empty($models)) {

$this->response(9999, [], '无可用模型生成向量');

}

// 处理文档并生成向量

$chunkIds = $this->vectorService->processDocument($kbId, $docId, $content, $models[0]);

$this->response(0, [

'id' => $docId,

'chunks' => count($chunkIds)

]);

} catch (Exception $e) {

$this->response(9999, [], '处理文档失败: ' . $e->getMessage());

}

}

/\*\*

\* 关联知识库到渠道

\*/

public function linkToChannel()

{

if (!$this->uid) {

$this->response(1003);

}

$param = $this->request->post();

$kbId = isset($param['kb\_id']) ? intval($param['kb\_id']) : 0;

$channelId = isset($param['channel\_id']) ? intval($param['channel\_id']) : 0;

if (!$kbId || !$channelId) {

$this->response(9999, [], '参数不完整');

}

// 检查知识库是否存在且属于当前用户

$kb = Db::name('knowledge\_base')

->where('id', $kbId)

->where('user\_id', $this->uid)

->find();

if (!$kb) {

$this->response(9999, [], '知识库不存在或无权限');

}

// 检查渠道是否存在

$channel = Db::name('robot\_channel')

->where('id', $channelId)

->find();

if (!$channel) {

$this->response(9999, [], '渠道不存在');

}

// 关联知识库到渠道

$data = [

'channel\_id' => $channelId,

'kb\_id' => $kbId,

'createtime' => time()

];

try {

// 检查是否已关联

$exists = Db::name('robot\_channel\_kb')

->where('channel\_id', $channelId)

->where('kb\_id', $kbId)

->find();

if ($exists) {

$this->response(0, ['message' => '已关联']);

}

$result = Db::name('robot\_channel\_kb')->insert($data);

if (!$result) {

$this->response(9999, [], '关联失败');

}

$this->response(0, ['message' => '关联成功']);

} catch (Exception $e) {

$this->response(9999, [], '关联失败: ' . $e->getMessage());

}

}

/\*\*

\* 获取知识库列表

\*/

public function getList()

{

if (!$this->uid) {

$this->response(1003);

}

try {

$list = Db::name('knowledge\_base')

->where('user\_id', $this->uid)

->where('status', 1)

->field('id, name, description, createtime')

->order('id desc')

->select();

foreach ($list as &$item) {

$item['doc\_count'] = Db::name('knowledge\_document')

->where('kb\_id', $item['id'])

->where('status', 1)

->count();

$item['createtime'] = date('Y-m-d H:i:s', $item['createtime']);

}

$this->response(0, $list);

} catch (Exception $e) {

$this->response(9999, [], '获取知识库列表失败: ' . $e->getMessage());

}

}

/\*\*

\* 获取知识库文档列表

\*/

public function getDocuments()

{

if (!$this->uid) {

$this->response(1003);

}

$kbId = $this->request->get('kb\_id', 0);

if (!$kbId) {

$this->response(9999, [], '参数不完整');

}

try {

$list = Db::name('knowledge\_document')

->where('kb\_id', $kbId)

->where('status', 1)

->field('id, title, createtime')

->order('id desc')

->select();

foreach ($list as &$item) {

$item['chunk\_count'] = Db::name('knowledge\_chunk')

->where('doc\_id', $item['id'])

->count();

$item['createtime'] = date('Y-m-d H:i:s', $item['createtime']);

}

$this->response(0, $list);

} catch (Exception $e) {

$this->response(9999, [], '获取文档列表失败: ' . $e->getMessage());

}

}

}

```

## 五、修改Base.php中的\_create方法

修改`Base.php`中的`\_create`方法，增加知识库检索和上下文增强：

```php:chat/application/api/controller/robot/Base.php

protected function \_create($prompt, $sessionId, $stream = 1, $connection = '')

{

$model = new Channelmodel();

$models = $model->getModelListByChannel($this->channel['id']);

$models = array\_slice($models, 0, 4);

$res = [];

$content = [];

// 获取关联的知识库

$kbIds = Db::name('robot\_channel\_kb')

->where('channel\_id', $this->channel['id'])

->column('kb\_id');

// 如果有关联知识库，进行知识检索

$relevantContext = '';

if (!empty($kbIds)) {

$vectorService = new VectorService();

$similarDocs = $vectorService->searchSimilar($prompt, $kbIds, $models[0], 3);

if (!empty($similarDocs)) {

$relevantContext = "根据知识库中的以下信息回答：\n";

foreach ($similarDocs as $doc) {

$relevantContext .= $doc['content'] . "\n";

}

$relevantContext .= "\n用户问题：";

}

}

// 构建对话历史

if ($this->channel['multiple']) {

$map['user\_id'] = $this->uid;

$map['session\_id'] = $sessionId;

$map['assistant\_id'] = 0;

$history = Db::name('robot\_creation')->where($map)->order('id asc')->limit(2)->select();

foreach ($history as $k => $v) {

if ($v['content'] && $v['msg']) {

$content[] = ['role' => 'user', 'content' => $v['content']];

$content[] = ['role' => 'assistant', 'content' => $v['msg']];

}

}

}

// 如果有知识库上下文，添加到提示中

$finalPrompt = $prompt;

if (!empty($relevantContext)) {

$finalPrompt = $relevantContext . $prompt;

}

$content[] = ['role' => 'user', 'content' => $finalPrompt];

foreach ($models as $k => $model) {

$time = time();

$param['model'] = $model['model\_tag'];

$param['temperature'] = $model['temperature'];

$param['contents'] = $content;

$channel = $this->\_getChannelClass($model['model\_class']);

if ($stream == 2) {

$res = $channel->chatStream($param, $model, $connection);

} else {

$res = $channel->chat($param, $model);

}

if ($res['code'] == 0) {

$res['time'] = time() - $time;

$res['model\_id'] = $model['id'];

break;

}

try {

$error['channel'] = $this->channel['name'];

$error['model'] = $model['model\_tag'];

$error['user\_id'] = $this->uid;

$error['prompt'] = $prompt;

$error['code'] = $res['code'];

$error['msg'] = $res['msg'];

$error['stream'] = $stream;

$error['created\_date'] = date('Y-m-d H:i:s');

Db::name('robot\_error\_log')->insert($error);

} catch (\Exception $e) {

}

}

return $res;

}

```

## 六、创建数据库表

在`install.sql`文件中添加以下表结构：

```sql

-- ----------------------------

-- 知识库表

-- ----------------------------

CREATE TABLE `ai\_knowledge\_base` (

`id` int(11) NOT NULL AUTO\_INCREMENT,

`name` varchar(255) NOT NULL COMMENT '知识库名称',

`description` text COMMENT '知识库描述',

`user\_id` int(11) NOT NULL COMMENT '所属用户ID',

`status` tinyint(1) NOT NULL DEFAULT '1' COMMENT '状态：0=禁用，1=启用',

`createtime` int(10) NOT NULL COMMENT '创建时间',

`updatetime` int(10) NOT NULL COMMENT '更新时间',

PRIMARY KEY (`id`),

KEY `user\_id` (`user\_id`)

) ENGINE=InnoDB DEFAULT CHARSET=utf8mb4 COMMENT='知识库表';

-- ----------------------------

-- 知识库文档表

-- ----------------------------

CREATE TABLE `ai\_knowledge\_document` (

`id` int(11) NOT NULL AUTO\_INCREMENT,

`kb\_id` int(11) NOT NULL COMMENT '知识库ID',

`title` varchar(255) NOT NULL COMMENT '文档标题',

`content` longtext NOT NULL COMMENT '文档内容',

`file\_type` varchar(50) DEFAULT NULL COMMENT '文件类型',

`file\_path` varchar(255) DEFAULT NULL COMMENT '文件路径',

`status` tinyint(1) NOT NULL DEFAULT '1' COMMENT '状态：0=禁用，1=启用',

`createtime` int(10) NOT NULL COMMENT '创建时间',

`updatetime` int(10) NOT NULL COMMENT '更新时间',

PRIMARY KEY (`id`),

KEY `kb\_id` (`kb\_id`)

) ENGINE=InnoDB DEFAULT CHARSET=utf8mb4 COMMENT='知识库文档表';

-- ----------------------------

-- 知识库文本块表

-- ----------------------------

CREATE TABLE `ai\_knowledge\_chunk` (

`id` int(11) NOT NULL AUTO\_INCREMENT,

`doc\_id` int(11) NOT NULL COMMENT '文档ID',

`content` text NOT NULL COMMENT '文本块内容',

`createtime` int(10) NOT NULL COMMENT '创建时间',

PRIMARY KEY (`id`),

KEY `doc\_id` (`doc\_id`)

) ENGINE=InnoDB DEFAULT CHARSET=utf8mb4 COMMENT='知识库文本块表';

-- ----------------------------

-- 渠道知识库关联表

-- ----------------------------

CREATE TABLE `ai\_robot\_channel\_kb` (

`id` int(11) NOT NULL AUTO\_INCREMENT,

`channel\_id` int(11) NOT NULL COMMENT '渠道ID',

`kb\_id` int(11) NOT NULL COMMENT '知识库ID',

`createtime` int(10) NOT NULL COMMENT '创建时间',

PRIMARY KEY (`id`),

UNIQUE KEY `channel\_kb` (`channel\_id`,`kb\_id`)

) ENGINE=InnoDB DEFAULT CHARSET=utf8mb4 COMMENT='渠道知识库关联表';

```

## 七、添加路由配置

在`application/route.php`中添加知识库相关路由：

```php

// 知识库相关路由

Route::group('api/robot/kb', function () {

Route::post('create', 'api/robot.KnowledgeBase/create');

Route::post('upload', 'api/robot.KnowledgeBase/uploadDocument');

Route::post('link', 'api/robot.KnowledgeBase/linkToChannel');

Route::get('list', 'api/robot.KnowledgeBase/getList');

Route::get('docs', 'api/robot.KnowledgeBase/getDocuments');

});

```

## 八、使用方法

1. 安装Elasticsearch和PHP客户端

2. 创建上述所有文件和数据库表

3. 在后台添加知识库管理界面

4. 用户可以创建知识库、上传文档、关联知识库到渠道

5. 系统会自动处理文档、生成向量嵌入并存储到Elasticsearch

6. 用户提问时，系统会自动检索相关知识并增强提示

## 九、优势与注意事项

### 优势

1. \*\*高性能向量搜索\*\*：Elasticsearch 8.x版本原生支持向量搜索，性能优异

2. \*\*可扩展性\*\*：支持集群部署，可处理大规模知识库

3. \*\*全文搜索\*\*：除了向量搜索外，还可结合Elasticsearch的全文搜索能力

4. \*\*实时索引\*\*：支持实时添加和更新知识库内容

### 注意事项

1. \*\*资源消耗\*\*：Elasticsearch需要较多内存资源，建议至少4GB内存

2. \*\*安全配置\*\*：生产环境中需要配置适当的安全措施，如密码认证、HTTPS等

3. \*\*向量维度\*\*：不同模型的向量维度不同，需要在创建索引时指定正确的维度

4. \*\*备份策略\*\*：定期备份Elasticsearch索引数据

通过以上实现，您的系统将具备基于Elasticsearch的高性能向量数据库功能，支持知识库的创建、管理和智能检索，大幅提升AI对话的准确性和实用性。