ZhipuAI Open Platform Java SDK

maven-central v1.0.6
license MIT
java 1.8+

中文文档

The official Java SDK for ZhipuAI Open Platform Big Model API, enabling developers to easily integrate ZhipuAI's powerful AI capabilities into their Java applications.

✨ Features

- 🚀 **Type-safe API**: All interfaces are fully type-encapsulated, no need to consult API documentation
- 🔧 **Easy Integration**: Simple and intuitive API design for quick integration
- ⚡ **High Performance**: Built with modern Java libraries for optimal performance
- 🛡️ **Secure**: Built-in authentication and token management
- 📦 **Lightweight**: Minimal dependencies for easy project integration

📦 Installation

Requirements

- Java 1.8 or higher
- Maven or Gradle
- Not supported on Android platform

Maven

Add the following dependency to your `pom.xml`:

```
<dependency>
    <groupId>cn.bigmodel.openapi</groupId>
    <artifactId>oapi-java-sdk</artifactId>
    <version>release-V4-2.3.4</version>
</dependency>
```

Gradle

Add the following dependency to your `build.gradle` (for Groovy DSL):

```
dependencies {
    implementation 'cn.bigmodel.openapi:oapi-java-sdk:release-V4-2.3.4'
}
```

Or `build.gradle.kts` (for Kotlin DSL):

```
dependencies {
    implementation("cn.bigmodel.openapi:oapi-java-sdk:release-V4-2.3.4")
}
```

📋 Dependencies

This SDK uses the following core dependencies:

Library	Version
OkHttp	3.14.9
Java JWT	4.2.2
Jackson	2.11.3
Retrofit2	2.9.0

🚀 Quick Start

Basic Usage

1. **Create a Client** with your API key
2. **Call the desired API methods**

For complete examples, see V4Test.java. Remember to replace the API key with your own.

Client Configuration

The SDK provides a flexible `ClientV4` builder for customizing your client:

Configuration Options:

- `enableTokenCache()`: Enable token caching to reduce token requests
- `networkConfig()`: Configure connection, read, write timeouts, and ping intervals
- `connectionPool()`: Set up connection pooling

```java
String API_SECRET_KEY = "your_api_key_here";
private static final ClientV4 client = new ClientV4.Builder(API_SECRET_KEY)
        .enableTokenCache()
        .networkConfig(30, 10, 10, 10, TimeUnit.SECONDS)
        .connectionPool(new okhttp3.ConnectionPool(8, 1, TimeUnit.SECONDS))
        .build();
```

💡 Examples

Chat Model Invocation

Streaming Invocation (SSE)

- **Basic Chat**

```
List<ChatMessage> messages = new ArrayList<>();
ChatMessage chatMessage = new ChatMessage(ChatMessageRole.USER.value(), "What is the
relationship between ZhipuAI and ChatGLM?");
messages.add(chatMessage);
String requestId = String.format("your-request-id-%d", System.currentTimeMillis());
ChatCompletionRequest chatCompletionRequest = ChatCompletionRequest.builder()
        .model(Constants.ModelChatGLM4)
        .stream(Boolean.TRUE)
        .messages(messages)
        .requestId(requestId)
        .build();
ModelApiResponse sseModelApiResp = client.invokeModelApi(chatCompletionRequest);
if (sseModelApiResp.isSuccess()) {
    AtomicBoolean isFirst = new AtomicBoolean(true);
    ChatMessageAccumulator chatMessageAccumulator =
mapStreamToAccumulator(sseModelApiResp.getFlowable())
            .doOnNext(accumulator -> {
                // Process streaming results
                System.out.println("accumulator: " + accumulator);
            })
            .doOnComplete(System.out::println)
            .lastElement()
            .blockingGet();
}
```

- **Function-Calling**

```
List<ChatMessage> messages = new ArrayList<>();
ChatMessage chatMessage = new ChatMessage(ChatMessageRole.USER.value(), "How much is a
flight ticket from Chengdu to Beijing?");
messages.add(chatMessage);
String requestId = String.format("your-request-id-%d", System.currentTimeMillis());
// Function definition
List<ChatTool> chatToolList = new ArrayList<>();
ChatTool chatTool = new ChatTool();
chatTool.setType(ChatToolType.FUNCTION.value());
ChatFunctionParameters chatFunctionParameters = new ChatFunctionParameters();
chatFunctionParameters.setType("object");
Map<String, Object> properties = new HashMap<>();
properties.put("departure", new HashMap<String, Object>() {{
    put("type", "string");
    put("description", "Departure city");
}});
properties.put("destination", new HashMap<String, Object>() {{
    put("type", "string");
    put("description", "Destination city");
}});
chatFunctionParameters.setProperties(properties);
ChatFunction chatFunction = ChatFunction.builder()
        .name("query_flight_prices")
        .description("Query flight prices")
        .parameters(chatFunctionParameters)
        .build();
chatTool.setFunction(chatFunction);
chatToolList.add(chatTool);

ChatCompletionRequest chatCompletionRequest = ChatCompletionRequest.builder()
        .model(Constants.ModelChatGLM4)
        .stream(Boolean.TRUE)
        .messages(messages)
        .requestId(requestId)
        .tools(chatToolList)
        .toolChoice("auto")
        .build();
ModelApiResponse sseModelApiResp = client.invokeModelApi(chatCompletionRequest);
// Process the returned results
```

Synchronous Invocation

- **Basic Chat**

```
List<ChatMessage> messages = new ArrayList<>();
ChatMessage chatMessage = new ChatMessage(ChatMessageRole.USER.value(), "What is the
relationship between ZhipuAI and ChatGLM?");
messages.add(chatMessage);
String requestId = String.format("your-request-id-%d", System.currentTimeMillis());
ChatCompletionRequest chatCompletionRequest = ChatCompletionRequest.builder()
        .model(Constants.ModelChatGLM4)
        .stream(Boolean.FALSE)
        .invokeMethod(Constants.invokeMethod)
        .messages(messages)
        .requestId(requestId)
        .build();
ModelApiResponse invokeModelApiResp = client.invokeModelApi(chatCompletionRequest);
System.out.println("model output:" + new
ObjectMapper().writeValueAsString(invokeModelApiResp));
```

- **Function-Calling**

```
List<ChatMessage> messages = new ArrayList<>();
ChatMessage chatMessage = new ChatMessage(ChatMessageRole.USER.value(), "What can you do?");
messages.add(chatMessage);
String requestId = String.format("your-request-id-%d", System.currentTimeMillis());
// Function definition... (refer to streaming Function-Calling)
List<ChatTool> chatToolList = new ArrayList<>();
// ... Add Function and WebSearch tools
```

```
ChatCompletionRequest chatCompletionRequest = ChatCompletionRequest.builder()
        .model(Constants.ModelChatGLM4)
        .stream(Boolean.FALSE)
        .invokeMethod(Constants.invokeMethod)
        .messages(messages)
        .requestId(requestId)
        .tools(chatToolList)
        .toolChoice("auto")
        .build();
ModelApiResponse invokeModelApiResp = client.invokeModelApi(chatCompletionRequest);
```

Asynchronous Invocation

```
// 1. Initiate an asynchronous task
List<ChatMessage> messages = new ArrayList<>();
ChatMessage chatMessage = new ChatMessage(ChatMessageRole.USER.value(), "What is the
relationship between ZhipuAI and ChatGLM?");
messages.add(chatMessage);
ChatCompletionRequest chatCompletionRequest = ChatCompletionRequest.builder()
        .model(Constants.ModelChatGLM4)
        .stream(Boolean.FALSE)
        .invokeMethod(Constants.invokeMethodAsync)
        .messages(messages)
        .build();
ModelApiResponse invokeModelApiResp = client.invokeModelApi(chatCompletionRequest);
String taskId = invokeModelApiResp.getData().getTaskId();

// 2. Query the result by taskId
QueryModelResultRequest request = new QueryModelResultRequest();
request.setTaskId(taskId);
QueryModelResultResponse queryResultResp = client.queryModelResult(request);
```

Role Playing

```
List<ChatMessage> messages = new ArrayList<>();
ChatMessage chatMessage = new ChatMessage(ChatMessageRole.USER.value(), "How have you been
lately?");
messages.add(chatMessage);

ChatMeta meta = new ChatMeta();
meta.setUser_info("I am a film director, specializing in music-themed movies.");
meta.setBot_info("You are a popular female singer and actress in the country, with
outstanding musical talent.");
meta.setBot_name("Su Mengyuan");
meta.setUser_name("Lu Xingchen");

ChatCompletionRequest chatCompletionRequest = ChatCompletionRequest.builder()
        .model(Constants.ModelCharGLM3)
        .stream(Boolean.FALSE)
        .invokeMethod(Constants.invokeMethod)
        .messages(messages)
        .meta(meta)
        .build();
ModelApiResponse invokeModelApiResp = client.invokeModelApi(chatCompletionRequest);
```

Image Generation

```
CreateImageRequest createImageRequest = new CreateImageRequest();
createImageRequest.setModel(Constants.ModelCogView);
createImageRequest.setPrompt("A futuristic cloud data center");
ImageApiResponse imageApiResponse = client.createImage(createImageRequest);
```

Vector Models

```
EmbeddingRequest embeddingRequest = new EmbeddingRequest();
embeddingRequest.setInput("hello world");
embeddingRequest.setModel(Constants.ModelEmbedding2);
EmbeddingApiResponse apiResponse = client.invokeEmbeddingsApi(embeddingRequest);
```

Fine-tuning

Create Fine-tuning Job

```
FineTuningJobRequest request = new FineTuningJobRequest();
request.setModel("chatglm3-6b");
request.setTraining_file("your-file-id");
CreateFineTuningJobApiResponse createFineTuningJobApiResponse =
client.createFineTuningJob(request);
```

Retrieve Fine-tuning Job

```
QueryFineTuningJobRequest queryFineTuningJobRequest = new QueryFineTuningJobRequest();
queryFineTuningJobRequest.setJobId("your-job-id");
QueryFineTuningJobApiResponse queryFineTuningJobApiResponse =
client.retrieveFineTuningJobs(queryFineTuningJobRequest);
```

List Fine-tuning Jobs

```
QueryPersonalFineTuningJobRequest queryPersonalFineTuningJobRequest = new
QueryPersonalFineTuningJobRequest();
queryPersonalFineTuningJobRequest.setLimit(10);
QueryPersonalFineTuningJobApiResponse queryPersonalFineTuningJobApiResponse =
client.queryPersonalFineTuningJobs(queryPersonalFineTuningJobRequest);
```

List Fine-tuning Events

```
QueryFineTuningJobRequest queryFineTuningJobRequest = new QueryFineTuningJobRequest();
queryFineTuningJobRequest.setJobId("your-job-id");
QueryFineTuningEventApiResponse queryFineTuningEventApiResponse =
client.queryFineTuningJobsEvents(queryFineTuningJobRequest);
```

Cancel Fine-tuning Job

```
FineTuningJobIdRequest request = FineTuningJobIdRequest.builder().jobId("your-job-
id").build();
QueryFineTuningJobApiResponse queryFineTuningJobApiResponse =
client.cancelFineTuningJob(request);
```

Delete Fine-tuned Model

```
FineTuningJobModelRequest request =
FineTuningJobModelRequest.builder().fineTunedModel("your-fine-tuned-model").build();
FineTunedModelsStatusResponse fineTunedModelsStatusResponse =
client.deleteFineTuningModel(request);
```

Batch Processing

Create Batch Job

```
BatchCreateParams batchCreateParams = new BatchCreateParams(
        "24h",
        "/v4/chat/completions",
        "your-file-id",
        new HashMap<String, String>() {{
            put("model", "glm-4");
        }}
);
BatchResponse batchResponse = client.batchesCreate(batchCreateParams);
```

Retrieve Batch Job

```
BatchResponse batchResponse = client.batchesRetrieve("your-batch-id");
```

List Batch Jobs

```
QueryBatchRequest queryBatchRequest = new QueryBatchRequest();
queryBatchRequest.setLimit(10);
QueryBatchResponse queryBatchResponse = client.batchesList(queryBatchRequest);
```

Cancel Batch Job

```
BatchResponse batchResponse = client.batchesCancel("your-batch-id");
```

Spring Boot Integration

```
package com.zhipu.controller;

import com.fasterxml.jackson.core.JsonProcessingException;
import com.fasterxml.jackson.databind.ObjectMapper;
```

```java
import com.wd.common.core.domain.R;
import com.zhipu.oapi.ClientV4;
import com.zhipu.oapi.Constants;
import com.zhipu.oapi.service.v4.deserialize.MessageDeserializeFactory;
import com.zhipu.oapi.service.v4.model.ChatCompletionRequest;
import com.zhipu.oapi.service.v4.model.ModelApiResponse;
import com.zhipu.oapi.service.v4.model.ModelData;
import org.slf4j.Logger;
import org.slf4j.LoggerFactory;
import org.springframework.web.bind.annotation.RequestBody;
import org.springframework.web.bind.annotation.RequestMapping;
import org.springframework.web.bind.annotation.RestController;

import java.util.concurrent.TimeUnit;

@RestController
public class TestController {

  private final static Logger logger = LoggerFactory.getLogger(TestController.class);
  private static final String API_SECRET_KEY = Constants.getApiKey();

  private static final ClientV4 client = new ClientV4.Builder(API_SECRET_KEY)
          .networkConfig(300, 100, 100, 100, TimeUnit.SECONDS)
          .connectionPool(new okhttp3.ConnectionPool(8, 1, TimeUnit.SECONDS))
          .build();
  private static final ObjectMapper mapper =
MessageDeserializeFactory.defaultObjectMapper();


  @RequestMapping("/test")
  public R<ModelData> test(@RequestBody ChatCompletionRequest chatCompletionRequest) {
    ModelApiResponse sseModelApiResp = client.invokeModelApi(chatCompletionRequest);

    return R.ok(sseModelApiResp.getData());
  }
}
```

📈 Release Notes

For detailed release notes and version history, please see Release-Note.md.

📄 License

This project is licensed under the MIT License - see the LICENSE file for details.

🤝 Contributing

We welcome contributions! Please feel free to submit a Pull Request.

📞 Support

For questions and support, please visit the ZhipuAI Open Platform or check our documentation.