|  |  |  |  |  |
| --- | --- | --- | --- | --- |
|  |  | | |  |
| **A10\_应用开发SDK** | | |  |
| **A10\_应用开发SDK** | | |
| **文档版本** | **01** | |
| **发布日期** | **2024-02-27** | |
|  | | | | |
|  | **华为技术有限公司** | |  |  |

|  |
| --- |
| 版权所有 © 华为技术有限公司2024。 保留一切权利。  非经本公司书面许可，任何单位和个人不得擅自摘抄、复制本文档内容的部分或全部，并不得以任何形式传播。  商标声明  和其他华为商标均为华为技术有限公司的商标。  本文档提及的其他所有商标或注册商标，由各自的所有人拥有。  注意  您购买的产品、服务或特性等应受华为公司商业合同和条款的约束，本文档中描述的全部或部分产品、服务或特性可能不在您的购买或使用范围之内。除非合同另有约定，华为公司对本文档内容不做任何明示或暗示的声明或保证。  由于产品版本升级或其他原因，本文档内容会不定期进行更新。除非另有约定，本文档仅作为使用指导，本文档中的所有陈述、信息和建议不构成任何明示或暗示的担保。 |

|  |  |
| --- | --- |
| 华为技术有限公司 | |
| 地址： | 深圳市龙岗区坂田华为总部办公楼 邮编：518129 |
| 网址： | <https://www.huawei.com> |
| 客户服务邮箱： | [support@huawei.com](mailto:support@huawei.com) |
| 客户服务电话： | 4008302118 |

|  |
| --- |
| 安全声明 |
| 漏洞处理流程  华为公司对产品漏洞管理的规定以“漏洞处理流程”为准，该流程的详细内容请参见如下网址：<https://www.huawei.com/cn/psirt/vul-response-process>  如企业客户须获取漏洞信息，请参见如下网址：  <https://securitybulletin.huawei.com/enterprise/cn/security-advisory> |

目 录

[1 盘古应用开发SDK简介 1](#_Toc256000000)

[2 准备工作 2](#_Toc256000001)

[3 Java SDK 6](#_Toc256000002)

[3.1 安装SDK 6](#_Toc256000003)

[3.2 配置SDK 7](#_Toc256000004)

[3.3 LLMs（语言模型） 11](#_Toc256000005)

[3.4 Prompt（提示词模板） 14](#_Toc256000006)

[3.5 Memory（记忆） 16](#_Toc256000007)

[3.5.1 Cache 16](#_Toc256000008)

[3.5.2 Vector 19](#_Toc256000009)

[3.5.3 History 21](#_Toc256000010)

[3.6 Skill（技能） 23](#_Toc256000011)

[3.6.1 基础问答 23](#_Toc256000012)

[3.6.2 多轮对话 23](#_Toc256000013)

[3.6.3 文档问答 24](#_Toc256000014)

[3.6.4 文档摘要 26](#_Toc256000015)

[3.7 Agent（智能代理） 27](#_Toc256000016)

[3.7.1 实例化Tool 28](#_Toc256000017)

[3.7.2 实例化Agent 30](#_Toc256000018)

[3.7.3 运行Agent 31](#_Toc256000019)

[3.7.4 监听Agent 33](#_Toc256000020)

[3.7.5 Agent流式输出 36](#_Toc256000021)

[3.7.6 Tool Retriever 36](#_Toc256000022)

[3.8 应用示例 39](#_Toc256000023)

[3.8.1 搜索增强 39](#_Toc256000024)

[3.8.2 长文本摘要 40](#_Toc256000025)

[3.9 常见问题 41](#_Toc256000026)

[4 Python SDK 42](#_Toc256000027)

[4.1 安装SDK 42](#_Toc256000028)

[4.2 配置SDK 43](#_Toc256000029)

[4.3 LLMs（语言模型） 46](#_Toc256000030)

[4.4 Prompt（提示词模板） 47](#_Toc256000031)

[4.5 Memory（记忆） 48](#_Toc256000032)

[4.5.1 Cache 48](#_Toc256000033)

[4.5.2 Vector 50](#_Toc256000034)

[4.5.3 History 52](#_Toc256000035)

[4.6 Skill（技能） 54](#_Toc256000036)

[4.6.1 基础问答 54](#_Toc256000037)

[4.6.2 多轮对话 54](#_Toc256000038)

[4.6.3 文档问答 55](#_Toc256000039)

[4.6.4 文档摘要 56](#_Toc256000040)

[4.7 Agent（智能代理） 58](#_Toc256000041)

[4.7.1 实例化Tool 58](#_Toc256000042)

[4.7.2 实例化Agent 60](#_Toc256000043)

[4.7.3 运行Agent 60](#_Toc256000044)

[4.7.4 监听Agent 62](#_Toc256000045)

[4.7.5 Agent流式输出 64](#_Toc256000046)

[4.7.6 Tool Retriever 65](#_Toc256000047)

[4.8 应用示例 68](#_Toc256000048)

[4.8.1 搜索增强 68](#_Toc256000049)

[4.8.2 长文本摘要 70](#_Toc256000050)

[4.9 常见问题 70](#_Toc256000051)

[5 应用实践 71](#_Toc256000052)

[5.1 基础问答 71](#_Toc256000053)

[5.2 长文本摘要 73](#_Toc256000054)

[5.3 搜索增强 76](#_Toc256000055)

[6 常见问题 81](#_Toc256000056)

[7 版本变更记录 82](#_Toc256000057)

# 盘古应用开发SDK简介

应用开发SDK概述

应用开发SDK针对大模型开发的场景，对LLMs（大语言模型，包括盘古系列、开源系列）进行的封装，提供了提示词模板、记忆、技能、智能代理，以简化用户的开发工作，帮助用户快速开发一个大模型应用。当前应用开发SDK支持如下语言：

* Java
* Python

开发环境要求

华为云盘古大模型应用开发SDK要求JAVA SDK 1.8 及其以上版本，Python 3.9 及以上版本。

# 准备工作

使用盘古大模型 Java SDK 时，需要在代码中配置以下信息，请提前收集。

资源列表

| 类型 | 资源 | 是否必选 | 依赖信息 | 参考文档 | 备注 |
| --- | --- | --- | --- | --- | --- |
| 大语言模型 | 华为云盘古 | 是（大语言模型至少选一个） | * 盘古模型API 调用URL。 * 华为云IAM账号认证信息。 | * 盘古大模型API参考文档：   申请资源时，可联系技术支持获取   * IAM帮助文档：   https://support.huaweicloud.com/api-identitycenter/iic\_04\_0032.html | * 模型API调用文档中完整url格式示例为：https://{endpoint}/v1/{project\_id}/deployments/{deployment\_id}/chat/completions ； SDK配置https://{endpoint}/v1/{project\_id}/deployments/{deployment\_id} 即可（/chat/completionsSDK代码中指定了） * IAM endpoint需要根据服务所在的区域正确配置， 参考帮助文档“终端节点”章节查找 * 参考IAM帮助文档，获取账号相关信息 |
| 华为云Gallery托管三方模型 | 否 | * Gallery三方托管模型API 调用URL。 * 华为云IAM账号认证信息。 | * 三方大模型API参考文档：   申请资源时，可联系客户支持获取   * IAM帮助文档：   https://support.huaweicloud.com/api-identitycenter/iic\_04\_0032.html | * 参考三方大模型API参考文档“API”章节，获取URI信息； 格式示例：https://{endpoint}/v1/infers/{deployment\_id} * 参考IAM帮助文档，获取账号相关信息 |
| Embedding模型 | 华为云CSS | 否 | * CSS Embedding模型API 调用URL。 * 华为云IAM账号认证信息。 | * CSS Embedding模型API参考文档：   申请资源时，可联系客户支持获取   * IAM帮助文档：   https://support.huaweicloud.com/api-identitycenter/iic\_04\_0032.html | * 参考CSS Embedding模型API参考文档，获取URL信息；格式示例：https://{endpoint}/v1/{project\_id}/applications/{app\_id}/{model\_version} * 参考IAM帮助文档，获取账号相关信息 |
| 文档加载 | 华为云CSS | 否 | * CSS 文档解析服务API 调用URL； * 华为云IAM账号认证信息； | * CSS 文档解析服务API参考文档：   申请资源时，可联系客户支持获取   * IAM帮助文档：   https://support.huaweicloud.com/api-identitycenter/iic\_04\_0032.html | * 参考CSS Embedding模型API参考文档，获取URL信息；格式示例：https://{endpoint}/v1/{project\_id}/applications/{app\_id} * 参考IAM帮助文档，获取账号相关信息 |
| 向量记忆 | 华为云CSS | 否 | * 集群hosts * 用户认证信息 | * 云搜索服务 CSS   https://support.huaweicloud.com/intl/zh-cn/css/index.html | * 参考CSS服务“快速入门”章节创建机器后， 在集群信息中获取hosts信息，示例： https://10.0.0.1:9200,https://10.0.0.2:9200 * 用户认证信息就是创建集群时设置的用户/密码 |
| 华为云CSS（集成Embedding） | 否 | * 集群hosts * 用户认证信息 | * 云搜索服务 CSS   https://support.huaweicloud.com/intl/zh-cn/css/index.html | * 参考CSS服务“快速入门”章节创建机器后， 在集群信息中获取hosts信息，示例： https://10.0.0.1:9200,https://10.0.0.2:9200 * 用户认证信息就是创建集群时设置的用户/密码 |
| Elastic Search | 否 | * 集群host信息 * 用户认证信息 | * ES官网：   https://www.elastic.co/cn/elasticsearch/ | hosts示例： https://10.0.0.1:9200,https://10.0.0.2:9200 |
| 标量存储 | 华为云DCS | 否 | * host * 认证信息 | * 分布式缓存DCS   https://support.huaweicloud.com/intl/zh-cn/dcs/index.html | host示例： redis://10.0.0.1:6379 |
| Redis | 否 | * host * 认证信息 | * Redis官网：   https://redis.io/ | host示例： redis://10.0.0.1:6379 |
| 华为云RDS | 否 | * host * 认证信息 | * 云数据库RDS：   https://support.huaweicloud.com/intl/zh-cn/productdesc-rds/zh-cn\_topic\_dashboard.html | - |
| Mysql | 否 | * host * 认证信息 | * Mysql官网：   https://www.mysql.com/ | - |

iam认证与SDK配置项的映射关系如下：

POST https://iam.cn-southwest-2.myhuaweicloud.com/v3/auth/tokens // iam的endpoint，sdk.iam.url

{

"auth": {

"identity": {

"methods": [

"password"

],

"password": {

"user": {

"name": "your iam user name", //IAM用户名，sdk.iam.user

"password": "your password", //华为云帐号密码，sdk.iam.password

"domain": {

"name": "your domain name" //帐号名，sdk.iam.domain

}

}

}

},

"scope": {

"project": {

"name": "cn-southwest-2" //盘古大模型当前部署在“西南-贵阳一”区域，取值为cn-southwest-2，sdk.iam.project

}

}

}

}

# Java SDK

[3.1 安装SDK](#_ZH-CN_TOPIC_0000001665279616" \o " )

[3.2 配置SDK](#_ZH-CN_TOPIC_0000001665439320" \o " )

[3.3 LLMs（语言模型）](#_ZH-CN_TOPIC_0000001665443412" \o " )

[3.4 Prompt（提示词模板）](#_ZH-CN_TOPIC_0000001665283724" \o " )

[3.5 Memory（记忆）](#_ZH-CN_TOPIC_0000001713283245" \o " )

[3.6 Skill（技能）](#_ZH-CN_TOPIC_0000001665443416" \o " )

[3.7 Agent（智能代理）](#_ZH-CN_TOPIC_0000001679800770" \o " )

[3.8 应用示例](#_ZH-CN_TOPIC_0000001713363025" \o " )

[3.9 常见问题](#_ZH-CN_TOPIC_0000001713279129" \o " )

## 安装SDK

Maven中央仓导入

在项目pom.xml中参考以下方式添加依赖。

<dependency>   
 <groupId>com.huaweicloud</groupId>   
 <artifactId>pangu-kits-app-dev-java</artifactId>   
 <version>2.0.1</version>   
</dependency>

本地导入

1. 下载SDK包，将 pangu-kits-app\*.jar文件放在项目目录lib文件夹下（.jar文件放置路径也可以自定义）。
2. 在项目pom.xml中参考以下方式添加依赖。

<dependency>   
 <groupId>com.huaweicloud</groupId>   
 <artifactId>pangu-kits-app-dev-java-bundle</artifactId>   
 <version>2.0.1</version>   
 <scope>system</scope>   
 <systemPath>${project.basedir}/lib/pangu-kits-app-dev-java-bundle-2.0.1.jar</systemPath>   
</dependency>

## 配置SDK

基础配置项

SDK依赖的配置项主要通过读取llm.properties配置文件；如果配置文件名不为llm.properties，需要在项目中主动设置，方法如下。

1. 在resources路径下，创建llm.properties文件，并根据实际需要配置相应的值。
2. 如果需要自定义配置文件名，可以参考以下代码设置。

// 建议在业务项目入口处配置   
// 不需要添加.properties后缀   
ConfigLoadUtil.setBaseName("application");

1. 完整配置项如下。

配置项中的密码等字段建议在配置文件或者环境变量中密文存放，使用时解密，确保安全，详见[配置文件敏感信息加密配置](#d0e696" \o " )。

################################ GENERIC CONFIG ###############################   
   
## User-defined Prompt.   
#   
sdk.prompt.path=   
   
## Proxy.   
# Examples: http://127.0.0.1:8000 ;   
#   
# sdk.proxy.enabled=   
# sdk.proxy.url=   
# sdk.proxy.user=   
# sdk.proxy.password=   
   
## Generic IAM info. This config is used when the specified IAM is not configured.   
## Either user password authentication or AK/SK authentication.   
#   
sdk.iam.url=   
sdk.iam.domain=   
sdk.iam.user=   
sdk.iam.password=   
sdk.iam.project=   
sdk.iam.ak=   
sdk.iam.sk=   
sdk.iam.disabled=   
   
   
################################ LLM ###############################   
   
## Pangu   
# Examples: https://{endPoint}/v1/{projectId}/deployments/{deploymentId} ;   
#   
sdk.llm.pangu.url=   
## If necessary, you can specify the IAM configuration.   
# sdk.llm.pangu.iam.url=   
# sdk.llm.pangu.iam.domain=   
# sdk.llm.pangu.iam.user=   
# sdk.llm.pangu.iam.password=   
# sdk.llm.pangu.iam.project=   
# sdk.llm.pangu.iam.disabled=   
## If necessary, you can specify the proxy status.   
# sdk.llm.pangu.proxy.enabled=   
   
## Gallery   
# Examples: https://{endPoint}/v1/infers/{deploymentId} ;   
#   
sdk.llm.gallery.url=   
## If necessary, you can specify the IAM configuration.   
# sdk.llm.gallery.iam.url=   
# sdk.llm.gallery.iam.domain=   
# sdk.llm.gallery.iam.user=   
# sdk.llm.gallery.iam.password=   
# sdk.llm.gallery.iam.project=   
# sdk.llm.gallery.iam.disabled=   
## If necessary, you can specify the proxy status.   
# sdk.llm.gallery.proxy.enabled=   
   
   
################################ EMBEDDINGS ###############################   
   
## CSS   
# Examples: https://{endPoint}/v1/{projectId}/applications/{appId}/{modelVersion} ;   
#   
sdk.embedding.css.url=   
## If necessary, you can specify the IAM configuration.   
# sdk.embedding.css.iam.url=   
# sdk.embedding.css.iam.domain=   
# sdk.embedding.css.iam.user=   
# sdk.embedding.css.iam.password=   
# sdk.embedding.css.iam.project=   
# sdk.embedding.css.iam.disabled=   
## If necessary, you can specify the proxy status.   
# sdk.embedding.css.proxy.enabled=   
   
   
################################ MEMORY ###############################   
   
## CSS or ES   
# Examples: http://127.0.0.1:9200,http://127.0.0.2:9200 ;   
#   
sdk.memory.css.url=   
sdk.memory.css.user=   
sdk.memory.css.password=   
   
## DCS or Redis   
# Examples: 127.0.0.1:6379 ;   
#   
# sdk.memory.dcs.url=   
# sdk.memory.dcs.user=   
# sdk.memory.dcs.password=   
   
## RDS or Mysql   
# Examples: jdbc:mariadb://127.0.0.1:3306/sdk?useSSL=false&useUnicode=true&characterEncoding=UTF-8&serverTimezone=Asia/Shanghai ;   
#   
# sdk.memory.rds.url=   
# sdk.memory.rds.user=   
# sdk.memory.rds.password=   
# sdk.memory.rds.poolSize=   
   
   
################################ DOC SPLIT ###############################   
   
## CSS   
# Examples: https://{endPoint}/v1/{projectId}/applications/{appId} ;   
#   
# sdk.doc.split.css.url=   
# sdk.doc.split.css.filepath=   
# sdk.doc.split.css.mode=   
## If necessary, you can specify the IAM configuration.   
# sdk.doc.split.css.iam.url=   
# sdk.doc.split.css.iam.domain=   
# sdk.doc.split.css.iam.user=   
# sdk.doc.split.css.iam.password=   
# sdk.doc.split.css.iam.project=   
# sdk.doc.split.css.iam.disabled=   
## If necessary, you can specify the proxy status.   
# sdk.doc.split.css.proxy.enabled=

日志打印配置

SDK日志采用slf4j框架，业务项目中可以使用logback、log4j、slf4j默认等具体实现（实现任一即可）。

* 通过logback实现
  1. 参考以下代码在pom文件中引入 logback 相关依赖（建议 1.3.1--1.3.8 版本）。

<dependency>   
 <groupId>ch.qos.logback</groupId>   
 <artifactId>logback-classic</artifactId>   
 <version>1.3.8</version>   
</dependency>   
<dependency>   
 <groupId>ch.qos.logback</groupId>   
 <artifactId>logback-core</artifactId>   
 <version>1.3.8</version>   
</dependency>

* 1. 在resource目录下创建logback.xml，用于配置日志级别、格式、输出位置等，示例如下。

<?xml version="1.0" encoding="UTF-8"?>   
<configuration>   
 <appender name="CONSOLE" class="ch.qos.logback.core.ConsoleAppender">   
 <encoder>   
 <pattern>%d{yyyy-MM-dd HH:mm:ss.SSS} [%thread] %-5level %logger{36} - %msg%n</pattern>   
 </encoder>   
 </appender>   
 <root level="DEBUG">   
 <appender-ref ref="CONSOLE" />   
 </root>   
</configuration>

* 通过log4j实现
  1. 参考以下代码在pom文件中引入 log4j适配slf4j 依赖（建议 2.0.7 版本）。

<dependency>   
 <groupId>org.slf4j</groupId>   
 <artifactId>slf4j-log4j12</artifactId>   
 <version>2.0.7</version>   
</dependency>

* 1. 在resource目录下创建log4j.xml，用于配置日志级别、格式、输出位置等，示例如下。

<?xml version="1.0" encoding="UTF-8"?>   
<!DOCTYPE log4j:configuration SYSTEM "log4j.dtd">   
<log4j:configuration>   
 <appender name="console" class="org.apache.log4j.ConsoleAppender">   
 <layout class="org.apache.log4j.PatternLayout">   
 <param name="ConversionPattern" value="%d{yyyy-MM-dd HH:mm:ss} %-5p %c{1}:%L - %m%n" />   
 </layout>   
 </appender>   
 <root>   
 org.slf4j <priority value="debug" />   
 <appender-ref ref="console" />   
 </root>   
</log4j:configuration>

* 通过slf4j simple实现
  1. 参考以下代码在pom文件中引入依赖。

<dependency>   
 <groupId>org.slf4j</groupId>   
 <artifactId>slf4j-simple</artifactId>   
 <version>1.7.36</version>   
</dependency>

* 1. 在resource目录下创建simplelogger.properties配置文件，示例如下。

org.slf4j.simpleLogger.defaultLogLevel=DEBUG   
org.slf4j.simpleLogger.log.org.apache.hc.client5=INFO   
org.slf4j.simpleLogger.showDateTime=true   
org.slf4j.simpleLogger.dateTimeFormat=yyyy-MM-dd HH:mm:ss   
org.slf4j.simpleLogger.showThreadName=true   
org.slf4j.simpleLogger.showLogName=true   
org.slf4j.simpleLogger.logFile=System.out   
#org.slf4j.simpleLogger.showShortLogName=false

配置文件敏感信息加密配置

配置项中的密码等字段需要加密存储时，可以自定义实现解密方法。

* 实现解密类

// 配置类实现   
public class CryptoCustom implements ConfigCryptor {   
 @Override   
 public String decrypt(String customTag, String encryptedValue) throws DecryptFailedException {   
 // todo   
 }   
}

* 配置项中指定解密类

# 配置项指定解密类   
sdk.crypto.implementation.path=CryptoCustom

* 配置项中可正常配置密文，示例如下

# 配置项示例：   
sdk.llm.iam.iamPwd={Crypto.customTag}encryptedvalue

## LLMs（语言模型）

LLMs模块用于对大语言模型API的适配封装，提供统一的接口快速地调用盘古、GALLERY三方模型等模型API。

* 初始化

根据相应模型定义LLM类，如使用盘古LLM为： LLMs.of(LLMs.PANGU)。

import com.huaweicloud.pangu.dev.sdk.api.llms.LLM;   
import com.huaweicloud.pangu.dev.sdk.api.llms.LLMs;   
   
// 初始化盘古LLM   
LLM llm = LLMs.of(LLMs.PANGU);   
   
// 初始化三方Gallery 托管 LLM   
LLM llm = LLMs.of(LLMs.GALLERY);

* 基础问答

基础的模型文本问答（template等参数采用模型默认的设置）。

llm.ask("你是谁？").getAnswer();

* 自定义参数问答

自定义设置如template等参数，获得对应的效果。

import com.huaweicloud.pangu.dev.sdk.api.llms.LLM;   
import com.huaweicloud.pangu.dev.sdk.api.llms.LLMs;   
import com.huaweicloud.pangu.dev.sdk.api.llms.config.LLMConfig;   
import com.huaweicloud.pangu.dev.sdk.api.llms.config.LLMConfigGallery;   
import com.huaweicloud.pangu.dev.sdk.api.llms.config.LLMParamConfig;   
   
// 设置模型参数，temperature为0.9   
LLMConfig llmConfig = LLMConfig.builder().llmParamConfig(LLMParamConfig.builder().temperature(0.9).build()).build();   
   
// 如使用Gallery三方模型，使用以下配置   
// LLMConfig llmConfig = LLMConfigGallery.builder().llmParamConfig(LLMParamConfig.builder().temperature(0.9).build()).build();   
   
// 初始化带参数的盘古LLM   
LLM pangu = LLMs.of(LLMs.PANGU, llmConfig);   
pangu.ask("写一篇五言律诗").getAnswer();

支持调整的参数解释。

private int maxTokens; // 完成时要生成的令牌的最大数量   
private double temperature; // 调整随机抽样的程度，温度值越高，随机性越大; 范围见模型API规范   
private double topP; // 核采样值， 和temperature不同时配置   
private double presencePenalty; // 存在惩罚，增加模型谈论新主题的可能性; 范围见模型API规范；   
private double frequencyPenalty; // 频率惩罚，降低模型重复的可能性，提高文本多样性、创造型; 范围见具体模型API规范；   
private int bestOf; // 服务侧生成优选的回答数   
private boolean stream; // 是否开启流式调用

* 流式问答

模型问答，开启流式效果，响应消息流式打印。

import com.huaweicloud.pangu.dev.sdk.api.callback.StreamCallBack;   
import com.huaweicloud.pangu.dev.sdk.api.callback.StreamResult;   
import com.huaweicloud.pangu.dev.sdk.api.llms.LLM;   
import com.huaweicloud.pangu.dev.sdk.api.llms.LLMs;   
import com.huaweicloud.pangu.dev.sdk.api.llms.config.LLMConfig;   
import com.huaweicloud.pangu.dev.sdk.api.llms.config.LLMParamConfig;   
import com.huaweicloud.pangu.dev.sdk.api.llms.response.LLMResp;   
   
import lombok.extern.slf4j.Slf4j;   
   
// 设置模型参数，stream为true   
final LLMConfig llmConfig = LLMConfig.builder().llmParamConfig(LLMParamConfig.builder().stream(true).build()).build();   
   
// 盘古LLM   
LLM pangu = LLMs.of(LLMs.PANGU, llmConfig);   
// 设置回调处理逻辑   
pangu.setStreamCallback(new StreamCallBackImp());   
pangu.ask("写一篇200字的散文").getAnswer();   
   
// 构造回调函数（以log打印为例，业务可自定义）   
@Slf4j   
public class StreamCallBackImp implements StreamCallBack {   
 @Override   
 public void onStart(String callBackId) {   
 log.info("StreamCallBack onStart: callBackId ----> {}", callBackId);   
 }   
   
 @Override   
 public void onEnd(String callBackId, StreamResult streamResult, LLMResp llmResp) {   
 log.info("StreamCallBack onEnd: callBackId ----> {} || llmResp ----> {}", callBackId, llmResp);   
 }   
   
 @Override   
 public void onError(String callBackId, StreamResult streamResult) {   
 log.error("StreamCallBack onError: callBackId ----> {}", callBackId);   
 }   
   
 @Override   
 public void onNewToken(String callBackId, LLMResp llmResp) {   
 log.info("StreamCallBack onNewToken: callBackId ----> {} || llmResp ----> {}", callBackId, llmResp);   
 }   
}

* 多轮对话问答

传递历史问答记录，实现多轮对话问答能力，同时支持自定义参数问答、流式问答。

import com.huaweicloud.pangu.dev.sdk.api.llms.LLM;   
import com.huaweicloud.pangu.dev.sdk.api.llms.LLMs;   
import com.huaweicloud.pangu.dev.sdk.api.llms.request.ConversationMessage;   
import com.huaweicloud.pangu.dev.sdk.api.llms.request.Role;   
import java.util.ArrayList;   
import java.util.List;   
   
// 构造多轮对话：历史问答记录 + 最新问题   
private List<ConversationMessage> buildMultiTurnChatMessages() {   
 List<ConversationMessage> messages = new ArrayList<>();   
 messages.add(ConversationMessage.builder().role(Role.SYSTEM).content("You are a helpful assistant.").build());   
 messages.add(ConversationMessage.builder().role(Role.USER).content("Who won the world series in 2020?").build());   
 messages.add(ConversationMessage.builder()   
 .role(Role.ASSISTANT)   
 .content("The Los Angeles Dodgers won the World Series in 2020.")   
 .build());   
 messages.add(ConversationMessage.builder().role(Role.USER).content("Where was it played?").build());   
 return messages;   
}   
   
// 初始化盘古LLM，开启问答   
LLM llm = LLMs.of(LLMs.PANGU);   
llm.ask(buildMultiTurnChatMessages()).getAnswer();

* 带人设的问答

支持在LLM配置项中设置人设，在LLM问答时系统会自动加上该人设，同时支持以上问答功能（暂不支持GALLERY三方模型）。

import com.huaweicloud.pangu.dev.sdk.api.llms.LLM;   
import com.huaweicloud.pangu.dev.sdk.api.llms.LLMs;   
import com.huaweicloud.pangu.dev.sdk.api.llms.config.LLMConfig;   
import com.huaweicloud.pangu.dev.sdk.api.llms.config.LLMModuleConfig;   
   
// 设置模型系统人设   
LLMConfig llmConfig = LLMConfig.builder()   
 .llmModuleConfig(LLMModuleConfig.builder().systemPrompt("You are a helpful assistant.").build())   
 .build();   
   
// 初始化盘古LLM，开启问答   
LLM llm = LLMs.of(LLMs.PANGU, llmConfig);   
llm.ask("写一首五言律诗").getAnswer();

## Prompt（提示词模板）

提示词模板模块提供模板格式化、自定义配置、few-shot管理功能。

* 模板格式化

import com.huaweicloud.pangu.dev.sdk.template.KV;   
import com.huaweicloud.pangu.dev.sdk.template.PromptTemplate;   
import org.junit.jupiter.api.Assertions;   
   
import java.util.HashMap;   
   
// 初始化Prompt模板对象   
PromptTemplate promptTemplate = new PromptTemplate("Tell me a {{adjective}} joke about {{content}}");   
   
// 支持Map<String, Object>格式匹配替换Prompt模板   
String format1 = promptTemplate.format(new HashMap<String, Object>() {   
 {   
 put("adjective", "funny");   
 put("content", "chickens");   
 }   
 });   
Assertions.assertEquals("Tell me a funny joke about chickens", format1);   
   
// 支持KV格式匹配替换Prompt模板   
String format2 = promptTemplate.format(KV.of("adjective", "funny"), KV.of("content", "chickens"));   
Assertions.assertEquals("Tell me a funny joke about chickens", format2);

* few-shot

基于长度

// 构造Prompt模板   
interface Antonyms {   
 String FIND\_ANTONYMS = "给定一个单词：{{word}}，返回一个反义词：{{antonym}}";   
}

import java.util.ArrayList;   
import java.util.HashMap;   
import java.util.List;   
import java.util.Map;   
   
// 构造示例   
public interface ExampleSample {   
 interface Antonyms {   
 List<Map<String, Object>> antonyms = new ArrayList<Map<String, Object>>() {   
 {   
 add(new HashMap<String, Object>() {   
 {   
 put("word", "short");   
 put("antonym", "long");   
 }   
 });   
 add(new HashMap<String, Object>() {   
 {   
 put("word", "energetic");   
 put("antonym", "lethargic");   
 }   
 });   
 add(new HashMap<String, Object>() {   
 {   
 put("word", "windy");   
 put("antonym", "calm");   
 }   
 });   
 }   
 };   
 }   
}

import com.huaweicloud.pangu.dev.sdk.prompts.exampleselector.LengthBasedES;   
import com.huaweicloud.pangu.dev.sdk.template.PromptTemplate;   
import com.huaweicloud.pangu.dev.sdk.prompts.FewShotPromptTemplate;   
import java.util.ArrayList;   
import java.util.HashMap;   
import java.util.List;   
import java.util.Map;   
   
// 构造prompt模板   
PromptTemplate promptTemplate = new PromptTemplate(Antonyms.FIND\_ANTONYMS);   
   
// 构造fewshotPrompt模板：   
// 基于长度策略（长度限制50）   
// 传入prompt的开头 "参考以下示例，给出给定单词的反义词："   
// 传入分隔符 "\n\n"   
// 传入prompt的结尾 "给定一个单词：{{word}}，返回一个反义词："   
FewShotPromptTemplate fewShotPromptTemplate =   
 new FewShotPromptTemplate(new LengthBasedES(ExampleSample.Antonyms.antonyms, promptTemplate, 50),   
 promptTemplate, "参考以下示例，给出给定单词的反义词：", "给定一个单词：{{word}}，返回一个反义词：", "\n\n");   
   
// 输入   
Map<String, Object> inputs = new HashMap<String, Object>() {   
 {   
 put("word", "hard");   
 }   
};   
   
// 根据输入匹配替换fewShotPromptTemplate，得到最终fewShotPrompt   
String prompt = fewShotPromptTemplate.format(inputs);

* 自定义prompt

// 按约定的格式准备prompt文件；   
// 文档结构和文件名参考提供的系统预置prompts文件   
   
文档结构示例：   
……   
 prompts   
 -- default   
 -- documents   
 -- stuff.pt

// 配置sdk配置项，指定prompt文件根路径，以 /home 目录下为例   
sdk.prompt.path=/home/prompts/default

## Memory（记忆）

Memory（记忆）模块结合外部存储为LLM应用提供长短期记忆能力，用于支持上下文记忆的对话、搜索增强等场景。

Memory（记忆）支持多种不同的存储方式和功能。

* Cache 缓存

这是一种临时存储数据的方法，它可以提高数据的访问速度和效率。缓存可以根据不同的存储方式进行初始化、更新、查找和清理操作。缓存还可以支持语义匹配和查询，通过向量和相似度的计算，实现对数据的语义理解和检索。

* Vector 向量存储

这是一种将数据转换为数学表示的方法，它可以度量数据之间的关系和相似度。向量存储可以根据不同的词向量模型进行初始化、更新、查找和清理操作。向量存储还可以支持多种相似算法，如余弦相似度、欧氏距离、曼哈顿距离等，实现对数据的相似度评分和排序。

* History 对话消息存储

这是一种将对话消息保存在内存中的方法，它可以记录和管理对话历史。对话消息存储可以根据不同的会话标识进行初始化、更新、查找和清理操作。对话消息存储还可以支持多种过滤条件，如时间范围、用户标识、消息类型等，实现对对话消息的筛选和分析。

### Cache

Cache 缓存是一种临时存储数据的方法，它可以把常用的数据保存在内存或者其它设备中，这样当我们需要访问这些数据时，就不用再去原始的数据源查找，而是直接从缓存中获取，从而节省时间和资源。

对LLM使用缓存：

LLM llm = LLMs.of(LLMs.PANGU, llmConfig);   
llm.setCache(Caches.of(Caches.IN\_MEMORY));   
llm.ask("你能讲一个笑话吗？")

此时，再次使用同样的问题，则不会再调用大模型，而是直接从内存返回：

llm.ask("你能讲一个笑话吗？")

Cache 缓存有以下几个操作。

* 初始化

指定缓存使用哪种存储方式，例如，使用内存型缓存可以设置为Cache cache = Caches.of(Caches.IN\_MEMORY);。

import com.huaweicloud.pangu.dev.sdk.api.memory.cache.Cache;   
import com.huaweicloud.pangu.dev.sdk.api.memory.cache.Caches;   
   
// 内存型   
Cache cache = Caches.of(Caches.IN\_MEMORY);   
// Redis   
Cache cache = Caches.of(Caches.REDIS);   
// mysql   
Cache cache = Caches.of(Caches.SQL);

* 更新数据

指向缓存中添加或修改数据，需要指定数据的键值对和结果对象。

例如，把1+1这个问题和对应的答案2保存到缓存中，可以参考以下示例。

import com.huaweicloud.pangu.dev.sdk.api.llms.response.LLMResp;   
   
//更新数据   
cache.update("1+1", LLMResp.builder().answer("2").build());

* 查询数据

从缓存中获取数据，需要指定数据的键值对。

例如，查找1+1这个问题对应的答案，可以参考以下示例。

import com.huaweicloud.pangu.dev.sdk.api.llms.response.LLMResp;   
   
// 查找数据   
LLMResp cacheValue = cache.lookup("1+1");

* 清理数据

删除缓存中的数据。

例如，删除对应的缓存数据，可以参考以下示例。

// 清理   
cache.clear()

* 配置过期策略

设置缓存有效期，支持基于时间和大小的限制。

// 设置缓存数据10s 后过期   
Cache cache = Caches.of(Caches.IN\_MEMORY, CacheStoreConfig.builder().expireAfterWrite(10).build());

* 参数解释

用于设置缓存对象的一些基本信息，如过期时间等。

/\*\*   
 \* 会话标识，业务确定   
 \*/   
@Builder.Default   
private String sessionTag = "";   
   
/\*\*   
 \* 访问后到期时间，单位为秒， 默认不设置过期   
 \*/   
@Builder.Default   
private int expireAfterAccess = -1;   
   
/\*\*   
 \* 写入后到期时间，单位为秒， 默认不设置过期   
 \*/   
@Builder.Default   
private int expireAfterWrite = -1;   
   
/\*\*   
 \* 最大个数， 默认不设置过期   
 \*/   
@Builder.Default   
private int maximumSize = -1;

* 语义缓存

语义缓存是一种基于向量和相似度的缓存方法，它可以实现对数据的语义匹配和查询。语义缓存可以根据不同的向量存储、相似算法、评分规则和阈值进行配置，并且可以使用不同的词向量模型进行嵌入。

import com.huaweicloud.pangu.dev.sdk.api.embedings.Embeddings;   
import com.huaweicloud.pangu.dev.sdk.api.llms.response.LLMResp;   
import com.huaweicloud.pangu.dev.sdk.api.memory.cache.Cache;   
import com.huaweicloud.pangu.dev.sdk.api.memory.cache.Caches;   
import com.huaweicloud.pangu.dev.sdk.api.memory.config.CacheStoreConfig;   
import com.huaweicloud.pangu.dev.sdk.api.memory.config.ServerInfoRedis;   
import com.huaweicloud.pangu.dev.sdk.api.memory.vector.Vectors;   
import com.huaweicloud.pangu.dev.sdk.vectorstore.DistanceStrategy;   
   
import org.junit.jupiter.api.Assertions;   
   
//redis向量   
// 不同的向量存储， 不同的相似算法；计算的评分规则不同； 可以同过scoreThreshold 设置相似性判断阈值   
// 例如使用Redis向量、余弦相似度、CSS词向量模型，并且设置相似性判断阈值为0.1f，代码示例如下   
Cache cache = Caches.of(CacheStoreConfig.builder()   
.storeName(Caches.SEMANTIC\_REDIS)   
.vectorStoreName(Vectors.REDIS)   
.serverInfo(ServerInfoRedis.builder().build())   
.distanceStrategy(DistanceStrategy.COSINE)   
.scoreThreshold(0.1f)   
.embedding(Embeddings.of(Embeddings.CSS))   
.build());   
   
   
//会话标识   
String sessionTag="test-semantic-cache-vector-001";   
cache.setSessionTag(sessionTag);   
cache.clear();   
   
//数据已清空   
Assertions.assertNull(cache.lookup("缓存是否存在？"));   
   
//更新数据   
//指向语义缓存中添加或修改数据，需要指定数据的键值对和结果对象   
//例如，把“把缓存是否存在？”这个问题和“test-semantic-cache-vector-001”这个会话标识对应的答案“存在”保存到语义缓存中，代码示例如下   
cache.update("缓存是否存在？",LLMResp.builder().answer("存在。").build());   
   
//校验，一致   
//用于检查缓存中的数据是否与查询的数据是否一致，如果一致，就返回缓存中的结果对象   
//例如，查询“缓存是否存在？”这个问题和“test-semantic-cache-vector-001”这个会话标识，就可以从缓存中获取到之前保存的答案“存在”   
String query="缓存是否存在？";   
LLMResp cacheValueAfter=cache.lookup(query);   
Assertions.assertNotNull(cacheValueAfter);   
System.out.println(query+cacheValueAfter.getAnswer());   
   
//校验，相似   
//用于检查缓存中的数据是否与查询的数据语义相似，如果相似，就返回缓存中的结果对象。这个操作需要使用向量和相似度的计算，以及设置的阈值来判断   
//例如，查询“缓存存在？”这个问题和“test-semantic-cache-vector-001”这个会话标识，就可以从缓存中获取到之前保存的答案“存在”，因为这个问题和“缓存是否存在？”问题语义相似   
String query\_sim="缓存存在？";   
LLMResp cacheValueSemantic=cache.lookup(query\_sim);   
Assertions.assertNotNull(cacheValueSemantic);   
System.out.println(query\_sim+cacheValueSemantic.getAnswer());   
   
//校验，不一致   
//用于检查缓存中的数据是否与查询的数据不一致，如果不一致，返回空值   
//例如，查询“有没有数据？”这个问题和“test-semantic-cache-vector-001”这个会话标识，就无法从缓存中获取到任何答案，因为这个问题和之前保存的问题都不一致   
String query\_not="有没有数据？";   
Assertions.assertNull(cache.lookup(query\_not));

### Vector

Embedding

Embedding模块用于对Embedding模型API的适配封装，提供统一的接口快速地调用CSS模型embedding能力。

* 初始化

根据相应模型定义Embedding类，如使用华为CSS Embedding为：Embeddings.of(Embeddings.CSS);。

import com.huaweicloud.pangu.dev.sdk.api.embedings.Embedding;   
import com.huaweicloud.pangu.dev.sdk.api.embedings.Embeddings;   
   
// 初始化 Css Embedding   
Embedding css = Embeddings.of(Embeddings.CSS);

* embedding 单文本

把单个字符串转换为向量数据。（向量维度由模型确定）。

import java.util.List;   
   
String text = "this is a test text.";   
// embed query.   
List<Float> embedding = css.embedQuery(text);   
embedding.forEach(s -> System.out.println(s.toString()));

* embedding 批量文档

把文档批量转换为向量数据。

import java.util.Arrays;   
import java.util.List;   
   
String text = "this is a test text.";   
// embed documents.   
List<List<Float>> embeddings = css.embedDocuments(Arrays.asList(text));   
embeddings.forEach(s -> System.out.println(s.toString()));

Splitter

文档拆分解析，提供对文档数据进行拆分解析能力，支持pdf/doc/docx/ppt/pptx/xls/xlsx/png/jpg/jpeg/bmp/gif/tiff/webp/pcx/ico/psd等格式文档。

* 初始化

根据相应解析接口定义DocSplit类，如使用华为Pangu DocSplit为例;

其中filePath指的是需要解析的文档路径；mode为分割解析模式，具体定义如下：

0 - 返回文档的原始段落，不做其它处理。

1 - 根据标注的书签或目录分段，一般适合有层级标签的word文档。

2 - 根据内容里的章节条分段，适合制度类文档。

3 - 根据长度分段，默认按照500字拆分，会尽量保留完整句子。

import com.huaweicloud.pangu.dev.sdk.api.doc.splitter.DocSplit;   
import com.huaweicloud.pangu.dev.sdk.api.doc.splitter.DocSplits;   
import com.huaweicloud.pangu.dev.sdk.api.doc.splitter.config.SplitConfig;   
   
// 初始化 pangudoc split（直接指定filePath和mode）   
String filePath = "D:/test.doc";   
DocSplit docPanguSplit =   
 DocSplits.of(SplitConfig.builder().splitName(DocSplits.PANGUDOC).filePath(filePath).mode("1").build());

// 初始化pangudoc split（通过配置文件指定filePath和mode）   
DocSplit docPanguSplit = DocSplits.of(DocSplits.PANGUDOC);

* 文档解析

import com.alibaba.fastjson.JSONObject;   
import com.huaweicloud.pangu.dev.sdk.api.doc.splitter.DocSplit;   
import com.huaweicloud.pangu.dev.sdk.api.doc.splitter.DocSplits;   
import com.huaweicloud.pangu.dev.sdk.client.css.doc.splitter.CSSDocResultResp;   
   
DocSplit docPanguSplit = DocSplits.of(DocSplits.PANGUDOC);   
CSSDocResultResp resp = docPanguSplit.loadDocument();   
System.out.println(JSONObject.toJSONString(resp.getResult()));

向量库

向量库，作为向量数据存储； 并提供向量数据检索能力。

* 初始化

以使用华为CSS示例。

import com.huaweicloud.pangu.dev.sdk.api.embedings.Embeddings;   
import com.huaweicloud.pangu.dev.sdk.api.memory.bo.Document;   
import com.huaweicloud.pangu.dev.sdk.api.memory.config.ServerInfoCss;   
import com.huaweicloud.pangu.dev.sdk.api.memory.config.VectorStoreConfig;   
import com.huaweicloud.pangu.dev.sdk.api.memory.vector.Vector;   
import com.huaweicloud.pangu.dev.sdk.api.memory.vector.Vectors;   
import java.util.Collections;   
   
Vector cssVector = Vectors.of(Vectors.CSS,   
 VectorStoreConfig.builder()   
 .embedding(Embeddings.of(Embeddings.CSS))   
 .indexName("you\_index\_name")   
 .vectorFields(Collections.singletonList("description"))   
 .textKey("name")   
 .serverInfo(ServerInfoCss.builder().build())   
 .build());

* 数据入库

import com.huaweicloud.pangu.dev.sdk.vectorstore.bo.BulkData;   
import java.util.ArrayList;   
import java.util.Arrays;   
import java.util.Collections;   
import java.util.HashMap;   
import java.util.List;   
import java.util.Map;   
   
Map<String, Object> data = new HashMap<>();   
data.put("name", "名称name");   
data.put("description", "描述description");   
final BulkData bulkData = new BulkData();   
bulkData.setId("1");   
bulkData.setData(data);   
cssVector.addDocs(Collections.singletonList(bulkData));

* 数据检索

List<Document> docs = cssVector.similaritySearch("bar", 2);

* 数据清理

List<Document> docs = cssVector.clear();

* CSS插件模式（内部已集成Embedding， 支持多字段组合向量检索）。

CSS插件模式，需要提前手工创建索引（因索引中需要指定embdding/rank模型，SDK不能简单自动创建）。

import com.huaweicloud.pangu.dev.sdk.api.embedings.Embeddings;   
import com.huaweicloud.pangu.dev.sdk.api.memory.bo.Document;   
import com.huaweicloud.pangu.dev.sdk.api.memory.config.ServerInfoCss;   
import com.huaweicloud.pangu.dev.sdk.api.memory.config.VectorStoreConfig;   
import com.huaweicloud.pangu.dev.sdk.api.memory.vector.Vector;   
import com.huaweicloud.pangu.dev.sdk.api.memory.vector.Vectors;   
   
import java.util.Arrays;   
import java.util.HashMap;   
import java.util.List;   
import java.util.Map;   
   
Vector cssVector = Vectors.of(Vectors.CSS,   
 VectorStoreConfig.builder()   
 .indexName("your\_index\_name")   
 .vectorFields(Arrays.asList("name", "description"))   
 .textKey("name")   
 .serverInfo(ServerInfoCss.builder().build())   
 .build());

通过vectorStoreConfig判断使用CSS的插件模式和非插件模式，如果配置了embedding模型，则使用非插件模式；否则使用插件模式。注意，在非插件模式下，vectorFields有且只有1个。

### History

History 缓存，用于存储历史对话信息，用于辅助模型理解上下文信息； 历史消息对有固定窗口、消息摘要等策略。

* 初始化

消息记录支持不同的存储方式， 如内存、DCS（Redis）、RDS（Sql）。

import com.huaweicloud.pangu.dev.sdk.api.memory.config.MessageHistoryConfig;   
import com.huaweicloud.pangu.dev.sdk.api.memory.config.ServerInfoRedis;   
import com.huaweicloud.pangu.dev.sdk.api.memory.config.ServerInfoSql;   
import com.huaweicloud.pangu.dev.sdk.memory.messagehistory.ChatMessageHistory;   
import com.huaweicloud.pangu.dev.sdk.memory.messagehistory.RedisChatMessageHistory;   
import com.huaweicloud.pangu.dev.sdk.memory.messagehistory.SqlChatMessageHistory;   
   
String sessionTag = "session\_tag\_test";   
// 内存型   
ChatMessageHistory chatMessage = new ChatMessageHistory();   
// Redis   
RedisChatMessageHistory redisChat = new RedisChatMessageHistory(MessageHistoryConfig.builder()   
 .serverInfo(ServerInfoRedis.builder().build())   
 .sessionTag("test-memory-0624")   
 .build());   
// Sql   
SqlChatMessageHistory sqlChat = new SqlChatMessageHistory(   
 MessageHistoryConfig.builder().serverInfo(ServerInfoSql.builder().build()).sessionTag(sessionTag).build());

* 添加、查找、删除数据

//更新数据   
chatMessage.addAIMessage("i am ai.");   
chatMessage.addUserMessage("i am tester.");   
// 查找数据   
chatMessage.getMessages().toString();   
// 清理   
chatMessage.clear();

* 消息策略（windows-size）

将固定轮次历史对话信息，作为历史上下文信息。

import com.huaweicloud.pangu.dev.sdk.api.memory.config.MemoryStoreConfig;   
import com.huaweicloud.pangu.dev.sdk.api.memory.config.MessageHistoryConfig;   
import com.huaweicloud.pangu.dev.sdk.llms.model.MessagePo;   
import com.huaweicloud.pangu.dev.sdk.memory.messagehistory.RedisChatMessageHistory;   
import com.huaweicloud.pangu.dev.sdk.api.memory.config.ServerInfoRedis;   
import com.huaweicloud.pangu.dev.sdk.memory.ConversationBufferMemory;   
   
// 历史消息存储   
RedisChatMessageHistory redisChat = new RedisChatMessageHistory(MessageHistoryConfig.builder()   
 .serverInfo(ServerInfoRedis.builder().build())   
 .sessionTag("test-memory")   
 .build());   
// 固定窗口策略   
ConversationBufferMemory memory = new ConversationBufferMemory(MemoryStoreConfig.builder().messageHistory(redisChat).build());   
   
memory.getChatMessage().addUserMessage("This is me, the human");   
memory.getChatMessage().addAIMessage("This is me, the AI");   
System.out.print(MessagePo.toString(memory.getChatMessage().getMessages()));   
memory.getChatMessage().clear();

* 消息策略（信息摘要）

将历史消息进行摘要后，作为历史上下文信息。

import com.huaweicloud.pangu.dev.sdk.api.memory.config.MemoryStoreConfig;   
import com.huaweicloud.pangu.dev.sdk.api.memory.config.MessageHistoryConfig;   
import com.huaweicloud.pangu.dev.sdk.llms.model.MessagePo;   
import com.huaweicloud.pangu.dev.sdk.memory.messagehistory.RedisChatMessageHistory;   
import com.huaweicloud.pangu.dev.sdk.api.memory.config.ServerInfoRedis;   
import com.huaweicloud.pangu.dev.sdk.memory.ConversationSummaryMemory;   
   
// 历史消息存储   
RedisChatMessageHistory redisChat = new RedisChatMessageHistory(MessageHistoryConfig.builder()   
 .serverInfo(ServerInfoRedis.builder().build())   
 .sessionTag("test-memory")   
 .build());   
// 摘要策略   
ConversationSummaryMemory memory = new ConversationSummaryMemory(MemoryStoreConfig.builder().messageHistory(redisChat).llm(LLMs.of(LLMs.PANGU)).build());   
   
memory.getChatMessage().addUserMessage("This is me, the human");   
memory.getChatMessage().addAIMessage("This is me, the AI");   
System.out.print(MessagePo.toString(memory.getChatMessage().getMessages()));   
memory.getChatMessage().clear();

## Skill（技能）

### 基础问答

SimpleSkill提供基础的对话实现。

* 初始化

import com.huaweicloud.pangu.dev.sdk.api.llms.LLMs;   
import com.huaweicloud.pangu.dev.sdk.template.KV;   
import com.huaweicloud.pangu.dev.sdk.api.skill.base.SimpleSkill;   
   
final String promptTemplate = "讲一个关于{{subject}}的笑话，字数{{count}}字以内";   
SimpleSkill skill = new SimpleSkill(LLMs.of(LLMs.PANGU), promptTemplate);

* 问答

import com.huaweicloud.pangu.dev.sdk.api.llms.config.LLMParamConfig;   
   
// 问答   
skill.execute(KV.of("subject", "哈士奇"), KV.of("count", "25"));   
   
// 带参数问答   
Map<String, Object> inputs = new HashMap<>();   
inputs.put("subject", "哈士奇");   
inputs.put("count", "25");   
LLMParamConfig param = LLMParamConfig.builder().temperature(0.9).build();   
skill.execute(inputs, param);

### 多轮对话

支持上下文记忆的多轮对话。

* 初始化

import com.huaweicloud.pangu.dev.sdk.api.llms.LLMs;   
import com.huaweicloud.pangu.dev.sdk.api.skill.Skills;   
import com.huaweicloud.pangu.dev.sdk.skill.ConversationSkill;   
   
ConversationSkill skill = Skills.newConversationSkill(LLMs.of(LLMs.PANGU));

* 问答

import com.huaweicloud.pangu.dev.sdk.memory.messagehistory.ChatMessageHistory;   
import com.huaweicloud.pangu.dev.sdk.memory.ConversationBufferMemory;   
import com.huaweicloud.pangu.dev.sdk.api.memory.config.MemoryStoreConfig;   
   
import org.junit.jupiter.api.Assertions;   
   
// 定义存储策略   
skill.setMemory(new ConversationBufferMemory(MemoryStoreConfig.builder().windowSize(3).messageHistory(new ChatMessageHistory()).build()));   
String answer = skill.execute("中国首都是哪个城市？");   
System.out.println(answer);   
Assertions.assertTrue(answer.contains("北京"));   
answer = skill.execute("它有什么好玩的地方？");   
System.out.println(answer);   
Assertions.assertTrue(answer.contains("故宫"));

### 文档问答

基于已有的知识库，进行回答。有stuff、refine、map-reduce策略。

* Stuff

将所有文档直接填充到prompt中，提给模型回答； 适合文档较少场景。

import com.huaweicloud.pangu.dev.sdk.api.llms.LLMs;   
import com.huaweicloud.pangu.dev.sdk.api.memory.bo.Document;   
import com.huaweicloud.pangu.dev.sdk.skill.DocSkill;   
import com.huaweicloud.pangu.dev.sdk.api.skill.Skills;   
import com.huaweicloud.pangu.dev.sdk.api.memory.vector.Vector;   
import com.huaweicloud.pangu.dev.sdk.api.memory.vector.Vectors;   
import com.huaweicloud.pangu.dev.sdk.api.memory.config.VectorStoreConfig;   
import com.huaweicloud.pangu.dev.sdk.api.embedings.Embeddings;   
   
import java.util.List;   
   
Vector cssVector = Vectors.of(Vectors.CSS,   
 VectorStoreConfig.builder()   
 .embedding(Embeddings.of(Embeddings.CSS))   
 .indexName("test-stuff-document-062102")   
 .build());   
   
// 检索   
String query = "杜甫的诗代表了什么主义诗歌艺术的高峰？";   
List<Document> docs = cssVector.similaritySearch(query, 4, 105);   
   
// 问答   
DocSkill docSkill = Skills.Document.newDocAskStuffSkill(LLMs.of(LLMs.PANGU));   
   
System.out.println(docSkill.executeWithDocs(docs, query));

* Refine

基于首个文档问答，并循环后续文档来迭代更新答案。

import com.huaweicloud.pangu.dev.sdk.api.llms.LLMs;   
import com.huaweicloud.pangu.dev.sdk.api.memory.bo.Document;   
import com.huaweicloud.pangu.dev.sdk.skill.DocSkill;   
import com.huaweicloud.pangu.dev.sdk.api.skill.Skills;   
import com.huaweicloud.pangu.dev.sdk.api.memory.vector.Vector;   
import com.huaweicloud.pangu.dev.sdk.api.memory.vector.Vectors;   
import com.huaweicloud.pangu.dev.sdk.api.memory.config.VectorStoreConfig;   
import com.huaweicloud.pangu.dev.sdk.api.embedings.Embeddings;   
   
import java.util.List;   
   
Vector cssVector = Vectors.of(Vectors.CSS,   
 VectorStoreConfig.builder()   
 .embedding(Embeddings.of(Embeddings.CSS))   
 .indexName("test-stuff-document-062102")   
 .build());   
   
// 检索   
String query = "杜甫的诗代表了什么主义诗歌艺术的高峰？";   
List<Document> docs = cssVector.similaritySearch(query, 4, 105);   
   
// 问答   
DocSkill docSkill = Skills.Document.newDocAskRefineSkill(LLMs.of(LLMs.PANGU));   
   
System.out.println(docSkill.executeWithDocs(docs, query));

* Map-Reduce

先将文档单独进行摘要， 将摘要后的文档再提交给模型。 必要时，会循环迭代摘要。

import com.huaweicloud.pangu.dev.sdk.api.llms.LLMs;   
import com.huaweicloud.pangu.dev.sdk.api.memory.bo.Document;   
import com.huaweicloud.pangu.dev.sdk.skill.DocSkill;   
import com.huaweicloud.pangu.dev.sdk.api.skill.Skills;   
import com.huaweicloud.pangu.dev.sdk.api.memory.vector.Vector;   
import com.huaweicloud.pangu.dev.sdk.api.memory.vector.Vectors;   
import com.huaweicloud.pangu.dev.sdk.api.memory.config.VectorStoreConfig;   
import com.huaweicloud.pangu.dev.sdk.api.embedings.Embeddings;   
   
import java.util.List;   
   
Vector cssVector = Vectors.of(Vectors.CSS,   
 VectorStoreConfig.builder()   
 .embedding(Embeddings.of(Embeddings.CSS))   
 .indexName("test-stuff-document-062102")   
 .build());   
   
// 检索   
String query = "杜甫的诗代表了什么主义诗歌艺术的高峰？";   
List<Document> docs = cssVector.similaritySearch(query, 4, 105);   
   
// 问答   
DocSkill docSkill = Skills.Document.newDocAskMapReduceSkill(LLMs.of(LLMs.PANGU));   
   
System.out.println(docSkill.executeWithDocs(docs, query));

### 文档摘要

基于已有的知识库，进行摘要总结。有stuff、refine， map-reduce策略。

* Stuff

将所有文档直接填充到prompt中，提给模型处理； 适合文档较少场景。

import com.huaweicloud.pangu.dev.sdk.api.llms.LLMs;   
import com.huaweicloud.pangu.dev.sdk.api.memory.bo.Document;   
import com.huaweicloud.pangu.dev.sdk.skill.DocSkill;   
import com.huaweicloud.pangu.dev.sdk.api.skill.Skills;   
import com.huaweicloud.pangu.dev.sdk.api.memory.vector.Vector;   
import com.huaweicloud.pangu.dev.sdk.api.memory.vector.Vectors;   
import com.huaweicloud.pangu.dev.sdk.api.memory.config.VectorStoreConfig;   
import com.huaweicloud.pangu.dev.sdk.api.embedings.Embeddings;   
   
import java.util.List;   
   
Vector cssVector = Vectors.of(Vectors.CSS,   
 VectorStoreConfig.builder()   
 .embedding(Embeddings.of(Embeddings.CSS))   
 .indexName("test-stuff-document-062102")   
 .build());   
   
// 检索   
String query = "杜甫";   
List<Document> docs = cssVector.similaritySearch(query, 4, 105);   
   
// 摘要   
DocSkill docSkill = Skills.Document.newDocSummarizeStuffSkill(LLMs.of(LLMs.PANGU));   
   
System.out.println(docSkill.executeWithDocs(docs));

* Refine

基于首个文档摘要，并循环后续文档来迭代更新。

import com.huaweicloud.pangu.dev.sdk.api.llms.LLMs;   
import com.huaweicloud.pangu.dev.sdk.api.memory.bo.Document;   
import com.huaweicloud.pangu.dev.sdk.skill.DocSkill;   
import com.huaweicloud.pangu.dev.sdk.api.skill.Skills;   
import com.huaweicloud.pangu.dev.sdk.api.memory.vector.Vector;   
import com.huaweicloud.pangu.dev.sdk.api.memory.vector.Vectors;   
import com.huaweicloud.pangu.dev.sdk.api.memory.config.VectorStoreConfig;   
import com.huaweicloud.pangu.dev.sdk.api.embedings.Embeddings;   
   
import java.util.List;   
   
Vector cssVector = Vectors.of(Vectors.CSS,   
 VectorStoreConfig.builder()   
 .embedding(Embeddings.of(Embeddings.CSS))   
 .indexName("test-stuff-document-062102")   
 .build());   
   
// 检索   
String query = "杜甫";   
List<Document> docs = cssVector.similaritySearch(query, 4, 105);   
   
// 摘要   
DocSkill docSkill = Skills.Document.newDocSummarizeRefineSkill(LLMs.of(LLMs.PANGU));   
   
System.out.println(docSkill.executeWithDocs(docs));

* Map-Reduce

先将文档单独进行摘要， 将摘要后的文档再提交给模型。 必要时，会循环迭代摘要。

import com.huaweicloud.pangu.dev.sdk.api.llms.LLMs;   
import com.huaweicloud.pangu.dev.sdk.api.memory.bo.Document;   
import com.huaweicloud.pangu.dev.sdk.skill.DocSkill;   
import com.huaweicloud.pangu.dev.sdk.api.skill.Skills;   
import com.huaweicloud.pangu.dev.sdk.api.memory.vector.Vector;   
import com.huaweicloud.pangu.dev.sdk.api.memory.vector.Vectors;   
import com.huaweicloud.pangu.dev.sdk.api.memory.config.VectorStoreConfig;   
import com.huaweicloud.pangu.dev.sdk.api.embedings.Embeddings;   
   
import java.util.List;   
   
Vector cssVector = Vectors.of(Vectors.CSS,   
 VectorStoreConfig.builder()   
 .embedding(Embeddings.of(Embeddings.CSS))   
 .indexName("test-stuff-document-062102")   
 .build());   
   
// 检索   
String query = "杜甫";   
List<Document> docs = cssVector.similaritySearch(query, 4, 105);   
   
// 摘要   
DocSkill docSkill = Skills.Document.newDocSummarizeMapReduceSkill(LLMs.of(LLMs.PANGU));   
   
System.out.println(docSkill.executeWithDocs(docs));

## Agent（智能代理）

Agent（智能代理），用于对复杂任务的自动拆解与外部工具调用执行，一般包括任务规划、记忆系统、执行系统：

* 任务规划：将复杂目标任务分解为小的可执行子任务，通过评估、自我反思等方式提升规划成功率。
* 记忆系统：通过构建记忆模块去管理历史任务和策略，并让Agent结合记忆模块中相关的信息以获取最优化任务解决策略。
* 任务执行：能通过工具与外界发生联系并产生影响，工具可以自定义，包括查询信息、调用服务、网络搜索、文件管理、调用云服务等，通过Agent构建一个让LLM按照特定的规则迭代运行的Prompt，直到任务完成或者达到终止条件（如设置迭代次数）。

### 实例化Tool

Tool分为StaticTool（静态工具）和DynamicTool（动态工具）两类，静态工具需要开发者事先定义好，即在编译期定义与实例化；动态工具开发者可以在系统运行时动态构建，即在运行态定义与实例化。

StaticTool（静态工具）

静态工具可以通过注解的方式新增，在run接口中实现工具的功能，例如：

import com.huaweicloud.pangu.dev.sdk.api.annotation.AgentTool;   
import com.huaweicloud.pangu.dev.sdk.api.annotation.AgentToolParam;   
import com.huaweicloud.pangu.dev.sdk.api.tool.StaticTool;   
import lombok.Data;   
   
@AgentTool(toolId = "reserve\_meeting\_room", toolDesc = "预订会议室", toolPrinciple = "请在需要预订会议室时调用此工具",   
 inputDesc = "会议开始结束时间，会议室", outPutDesc = "预订会议室的结果")   
public class ReserveMeetingRoom extends StaticTool<ReserveMeetingRoom.InputParam, String> {   
 @Override   
 public String run(InputParam input) {   
 return String.format("%s到%s的%s已预订成功", input.start, input.end, input.meetingRoom);   
 }   
 @Data   
 public static class InputParam {   
 @AgentToolParam(description = "会议开始时间，格式为HH:mm")   
 private String start;   
 @AgentToolParam(description = "会议结束时间，格式为HH:mm")   
 private String end;   
 @AgentToolParam(description = "会议室")   
 private String meetingRoom;   
 }   
}

@AgentTool注解说明：

toolId，工具的标识，建议为英文且与实际工具含义匹配，在同一个Agent中唯一。

toolDesc，【重要】工具的描述，尽可能的准确简短描述工具的用途。

toolPrinciple，【重要】何时使用该工具，该描述直接影响LLM对工具使用的判断，尽量描述清楚，如果Agent实际执行效果不符合预期，可以调整。

inputDesc，【重要】工具的入参描述 ，该描述直接影响LLM对入参的提取，尽量描述清楚，如果Agent实际执行效果不符合预期，可以调整。

outputDesc，工具的出参描述，当前对Agent的表现无重要影响。

如果输入输出参数为复杂类型，则需要通过AgentToolParam注解定义复杂类型的参数描述，此时inputDesc、outputDesc可以填空字符串，但仍然建议给出简要的描述。当前版本不支持复杂类型中再嵌套复杂类型，只支持基本类型：String、Number、Boolean，建议参数数量不超过5个。

@AgentToolParam注解说明：

description，【重要】参数的描述，该描述直接影响LLM对入参的提取，尽量描述清楚，如果Agent实际执行效果不符合预期，可以调整。

required，是否为可选参数

注意：字段的命名需要以小写字母开头，否则在转换成标准的Json schema时会出现问题，导致模型精度受到影响。

上例中的InputParam为一个复杂的入参，如果工具的入参为一个基本类型，则不需要再额外定一个结构体，例如：

import com.huaweicloud.pangu.dev.sdk.api.annotation.AgentTool;   
import com.huaweicloud.pangu.dev.sdk.api.tool.StaticTool;   
   
@AgentTool(toolId = "capital", toolDesc = "资产注册查询", toolPrinciple = "请在需要查询各个公司的资产注册情况时调用此工具",   
 inputDesc = "需要查询的公司名称，一次只支持查询一家公司", outPutDesc = "公司的资产注册规模")   
public class CapitalQuery extends StaticTool<String, String> {   
 @Override   
 public String run(String input) {   
 switch (input) {   
 case "x公司":   
 return "x亿人民币";   
 case "y公司":   
 return "y亿人民币";   
 case "z公司":   
 return "z亿人民币";   
 default:   
 return "未知";   
 }   
 }   
}

DynamicTool（动态工具）

* 动态工具可以在业务运行态动态新增或修改：

import com.alibaba.fastjson.JSON;   
import com.alibaba.fastjson.JSONObject;   
import com.huaweicloud.pangu.dev.sdk.api.tool.DynamicTool;   
   
private static DynamicTool buildAddTool() {   
 final DynamicTool addTool = new DynamicTool();   
 addTool.setToolId("add");   
 addTool.setToolDesc("加法运算");   
 addTool.setToolPrinciple("请在需要做两个整型的加法运算时调用此工具");   
 addTool.setInputDesc("加法输入");   
 addTool.setOutputDesc("加法运算的结果");   
 addTool.setInputSchema(   
 "{\"type\":\"object\",\"properties\":{\"a\":{\"type\":\"integer\",\"description\":\"加法运算的数字\"},\"b\":{\"type\":\"integer\",\"description\":\"加法运算的数字\"}},\"required\":[\"a\",\"b\"]}");   
 addTool.setOutputSchema("{\"type\":\"integer\"}");   
 addTool.setFunction(s -> {   
 final JSONObject jsonObject = JSON.parseObject(s);   
 final Integer a = jsonObject.getInteger("a");   
 final Integer b = jsonObject.getInteger("b");   
 return a + b;   
 });   
 return addTool;   
}

toolId、toolDesc、toolPrinciple、inputDesc、outputDesc的定义与说明与静态工具相同。

inputSchema：工具入参的JsonSchema。

outputSchema：工具出参的JsonSchema。

* 复杂对象JsonSchema示例：

{   
 "type": "object",   
 "properties": {   
 "a": {   
 "description": "加法运算的数字",   
 "type": "integer"   
 },   
 "b": {   
 "description": "加法运算的数字",   
 "type": "integer"   
 }   
 },   
 "required": ["a","b"]   
}

* 简单对象JsonSchema示例：

{   
 "type": "integer"   
}

* 枚举对象JsonSchema示例：

{   
 "type": "object",   
 "properties": {   
 "chargeType": {   
 "type": "string",   
 "enum": ["Alipay", "Wechat", "Bankcard"],   
 "description": "充值类型"   
 },   
 "amount": {   
 "type": "integer",   
 "description": "充值金额"   
 }   
 },   
 "required": ["chargeType", "amount"]   
}

### 实例化Agent

Agent实例化过程包括两个部分，注册LLM、注册工具。

import com.huaweicloud.pangu.dev.sdk.agent.ReactPanguAgent;   
import com.huaweicloud.pangu.dev.sdk.api.llms.LLMs;   
   
public static void initAgent() {   
 ReactPanguAgent agent = new ReactPanguAgent(LLMs.of(LLMs.PANGU,   
 LLMConfig.builder().llmModuleConfig(LLMModuleConfig.builder().moduleVersion("N2\_agent\_v2").build()).build()));   
 agent.setMaxIterations(5);   
 agent.addTool(new ReverseTool());   
 agent.addTool(new AddTool());   
 agent.addTool(new SearchTool());   
}

静态工具和动态工具的注册方式相同，通过addTool接口进行注册。

通过setMaxIterations可以设置最大迭代次数，控制Agent子规划的最大迭代步数，防止无限制的迭代或出现死循环情况。

Agent使用的模型必须为Pangu-NLP-N2-Agent-L0.C模型，或其衍生模型，使用通用模型或其它模型无法运行。如上例所示，当前的moduleVersion需要配置为“N2\_agent\_v2”，模型的相关配置需要改为Pangu-NLP-N2-Agent-L0.C模型的地址。

### 运行Agent

调用run接口运行一个Agent：

agent.run("帮我定个下午3点到8点2303会议室");

Agent的运行时会进行自我迭代，并且选择合适的工具，在日志中打印最终的执行结果：

用户:帮我定个下午3点到8点2303会议室   
计划已执行完成,自动编排步骤:   
步骤1:   
思考:根据人类的需求，我需要使用预订会议室的工具，将会议室设置为2303，开始时间设置为15:00，结束时间设置为20:00。   
行动:使用工具[reserve\_meeting\_room],传入参数{"meetingRoom":"2303","start":"15:00","end":"20:00"}   
工具返回:15:00到20:00的2303已预订成功

多轮接口：

List<ConversationMessage> messages = new ArrayList<>();   
messages.add(ConversationMessage.builder().role(Role.USER).content("定个2点的会议").build());   
messages.add(   
 ConversationMessage.builder().role(Role.ASSISTANT).content("请问您的会议预计何时结束？另外，您是需要预订线上会议还是实体会议室？").build());   
messages.add(ConversationMessage.builder().role(Role.USER).content("4点结束，线上会议").build());   
agent.run(messages);

运行结果示例：

用户: 定个2点的会议   
助手: 请问您的会议预计何时结束？另外，您是需要预订线上会议还是实体会议室？   
用户: 4点结束，线上会议   
计划已执行完成,自动编排步骤:   
行动:使用工具[reserve\_meeting],传入参数{"start":"14:00","end":"16:00"}   
思考:用户需要预订一个从2点到4点的线上会议，我可以使用reserve\_meeting工具来帮助他预订。   
工具返回:14:00到16:00的线上会议已预订成功

有些时候，我们并不希望Agent完全自主执行，在某些关键节点，让用户先进行确认，确认后再执行，或者用户对模型的结果有异议或者想法有变化，想对当前结果进行更改。此时我们可以单步运行Agnet：

/\*\*   
 \* 单步执行Agent，提供干预能力   
 \*   
 \* @param agentSession 包括初始状态，以及执行步骤间的agentSession，可以使用AgentSessionHelper类辅助处理   
 \* @return Agent执行的结果   
 \*/   
AgentSession runStep(AgentSession agentSession);

以下为一个完整的示例：

agent.clearTool();   
agent.addTool(new RiskDetection());   
   
AgentSession session = AgentSessionHelper.initAgentSession("请帮我查一下方欣科技有限公司今年1月的经营异常风险");   
// 预期Agent返回reportType为经营异常风险检测的Json，呈现给终端用户   
session = agent.runStep(session);   
// 终端用户反悔，想改成欠税信息检测，修改信息后继续让Agent执行   
AgentSessionHelper.setUserFeedback(session, "改为欠税信息");   
   
// 预期Agent返回reportType为欠税信息体检的Json，呈现给终端用户   
session = agent.runStep(session);   
// 终端用户确认，调用外部系统，进一步确认公司名称和编号，补充信息后，让Agent继续执行   
AgentSessionHelper.setUserFeedback(session, "公司改为金财互联数据服务有限公司，nsrsbh为91440101MA59JN5456");   
   
// 预期Agent更新companyName和nsrsbh   
session = agent.runStep(session);   
// 此时已经得到完整调用信息了，调用risk\_detection API，将API返回结果给Agent继续执行   
AgentSessionHelper.setToolOutput(session, "金财互联数据服务有限公司1月份存在异常经营风险");   
   
session = agent.runStep(session);   
// 打印   
if (session.getAgentSessionStatus() == AgentSessionStatus.FINISHED) {   
 log.info(AgentSessionHelper.printPlan(session));   
}

上面的例子中，终端用户对Agent即将要执行的API做了2次修改，最终进行了执行。runStep的入参为AgentSession，有较大的灵活性，因此提供了辅助类AgentSessionHelper帮助开发者更容易处理：

initAgentSession：使用用户消息初始化一个AgentSession

setToolOutput：向Agent的当前步骤设置工具返回结果

setUserFeedback：向Agent的当前步骤设置用户反馈

打印的结果为：

用户: 请帮我查一下方欣科技有限公司今年1月的经营异常风险   
计划已执行完成,自动编排步骤:   
步骤1:   
思考:好的，我将使用"risk\_detection"工具来进行企业健康体检。根据您的需求，需要检测的是方欣科技有限公司今年1月的经营异常风险。今年1月的日期范围是从2024-01-01到2024-01-31。现在我将调用工具进行体检。   
行动:使用工具[risk\_detection],传入参数"{\"reportType\": \"经营异常风险检测\",\"companyName\": \"方欣科技有限公司\",\"skssqq\": \"2024-01-01\",\"skssqz\": \"2024-01-31\"}"   
用户反馈:改为欠税信息   
步骤2:   
思考:了解，我将重新调用"risk\_detection"工具，这次检测的是方欣科技有限公司今年1月的欠税信息。   
行动:使用工具[risk\_detection],传入参数"{\"reportType\": \"欠税信息体检\",\"companyName\": \"方欣科技有限公司\",\"skssqq\": \"2024-01-01\",\"skssqz\": \"2024-01-31\"}"   
用户反馈:公司改为金财互联数据服务有限公司，nsrsbh为91440101MA59JN1111   
步骤3:   
思考:好的，我将继续使用"risk\_detection"工具，这次检测的是金财互联数据服务有限公司今年1月的欠税信息，并提供了企业编号91440101MA59JN1111。   
行动:使用工具[risk\_detection],传入参数"{\"reportType\": \"欠税信息体检\",\"companyName\": \"金财互联数据服务有限公司\",\"skssqq\": \"2024-01-01\",\"skssqz\": \"2024-01-31\",\"nsrsbh\": \"91440101MA59JN5456\"}"   
工具返回:金财互联数据服务有限公司1月份存在异常经营风险   
步骤4:   
思考:"根据体检结果，金财互联数据服务有限公司在今年1月份存在异常经营风险。"

有些时候，我们会需要Agent对之前的思考、行动回答用户的最终问题，如果对模型返回的结果不满意，我们可以选择使用AgentSessionSkill解决：

final AgentSession agentSession = agent.run("我的数学成绩和语文成绩分别是多少");   
final LLM llm = LLMs.of(LLMs.PANGU, LLMConfig.builder()   
 .llmModuleConfig(LLMModuleConfig.builder()   
 .url(   
 "https://pangu.cn-southwest-2.myhuaweicloud.com/v1/infers/N2-基础模型的调用地址")   
 .build())   
 .build());   
final AgentSessionSkill agentSessionSkill = new AgentSessionSkill(llm);   
System.out.println("最终结果: " + agentSessionSkill.summary(agentSession));

运行结果示例：

用户: 我的数学成绩和语文成绩分别是多少   
计划已执行完成,自动编排步骤:   
行动:使用工具[queryScore],传入参数"{\"subjectName\": \"数学\"}"   
思考:好的，我需要调用 queryScore 工具来查询您的数学和语文成绩。首先查询数学成绩。   
工具返回:99   
行动:使用工具[queryScore],传入参数"{\"subjectName\": \"语文\"}"   
思考:您的数学成绩是99分。接下来查询语文成绩。   
工具返回:99   
问题已求解:您的语文成绩也是99分

最终结果: 您的数学成绩是99分，您的语文成绩也是99分。

AgentSessionSkill使用的大模型建议为N2-基础模型或者其它同等类型的模型。

### 监听Agent

一次Agent的响应如果涉及到多个任务的分解，往往会执行比较长的时间，此时可以对agent的执行过程进行监听。

AgentListener的定义如下：

public interface AgentListener {   
 /\*\*   
 \* Session启动时调用   
 \*   
 \* @param agentSession AgentSession   
 \*/   
 default void onSessionStart(AgentSession agentSession) {   
 }   
 /\*\*   
 \* Session迭代过程中调用   
 \*   
 \* @param agentSession AgentSession   
 \*/   
 default void onSessionIteration(AgentSession agentSession) {   
 }   
 /\*\*   
 \* onSessionIteration调用结束后，检查Agent是否需要终止，如果需要终止，则返回true，默认不终止   
 \* 可以在终止前对agentSession进行修改，如：修改agent的finalAnswer   
 \*   
 \* @param agentSession AgentSession   
 \*/   
 default boolean onCheckInterruptRequirement(AgentSession agentSession) {   
 return false;   
 }   
 /\*\*   
 \* Session结束时调用   
 \*   
 \* @param agentSession AgentSession   
 \*/   
 default void onSessionEnd(AgentSession agentSession) {   
 }   
}

定义一个监听器

通过实现AgentListener定义一个监听器：

import com.huaweicloud.pangu.dev.sdk.api.agent.AgentListener;   
   
public static class TestAgentListener implements AgentListener {   
 @Override   
 public void onSessionStart(AgentSession agentSession) {   
 System.out.println(agentSession);   
 }   
 @Override   
 public void onSessionIteration(AgentSession agentSession) {   
 System.out.println(agentSession);   
 }   
 @Override   
 public void onSessionEnd(AgentSession agentSession) {   
 System.out.println(agentSession);   
 }   
}

分别对应了Agent的开始、中间过程、结束阶段。

为Agent添加一个监听器

通多调用Agent的addListener接口添加一个监听器：

import com.huaweicloud.pangu.dev.sdk.agent.ReactPanguAgent;   
import com.huaweicloud.pangu.dev.sdk.api.llms.LLMs;   
   
public static void initAgent() {   
 ReactPanguAgent agent = new ReactPanguAgent(LLMs.of(LLMs.PANGU));   
 final TestAgentListener testAgentListener = new TestAgentListener();   
 agent.addListener(testAgentListener);   
}

listener会在Agent运行时生效。

监听的对象

监听的对象为一个AgentSession：

public class AgentSession {   
 /\*\*   
 \* UUID，在一个session内唯一   
 \*/   
 private String sessionId;   
 /\*\*   
 \* Agent返回的最终答案（最后一个AgentAction的输出）   
 \*/   
 private String finalAnswer = "";   
 /\*\*   
 \* 本次session的用户query   
 \*/   
 private List<ConversationMessage> messages;   
 /\*\*   
 \* 历史Action   
 \*/   
 private List<AgentAction> historyAction;   
 /\*\*   
 \* 当前Action   
 \*/   
 private AgentAction currentAction;   
 /\*\*   
 \* Agent状态   
 \*/   
 private AgentSessionStatus agentSessionStatus;   
}

AgentAction包含Agent的工具选择、工具执行结果、思考等信息，AgentSessionStatus为一个枚举，包含Agnet的执行状态。

通过监听终止Agent的执行

当需要在Agent的执行过程中终止执行时，除了通过setMaxIterations设置Agent的最大迭代次数，也可以通过实现监听器的onCheckInterruptRequirement实现。

agent.addListener(new AgentListener() {   
 @Override   
 public boolean onCheckInterruptRequirement(AgentSession agentSession) {   
 final AgentAction currentAction = agentSession.getCurrentAction();   
 // 如果当前的action为capital，则返回capital的工具调用原始返回值   
 if ("capital".equals(currentAction.getAction())) {   
 agentSession.setFinalAnswer(currentAction.getObservation());   
 return true;   
 }   
 return false;   
 }   
});

上面的例子中，当满足if判断条件时，就会直接终止agent的执行，并且agent的finalAnswer被设置为工具的原始返回值。

### Agent流式输出

Agent用于工具调用场景，与普通的LLM流式输出相比，区分了文本流与工具流。文本流将输出模型的思考过程和最终结果；工具流将输出工具的调用过程，而工具的调用的执行结果是通过监听获取的。

通过如下接口为Agent添加流式输出的回调：

/\*\*   
 \* 设置流式接口回调函数   
 \*   
 \* @param textStreamCallback Agent文本输出的StreamCallback   
 \* @param toolStreamCallback Agent工具输出的StreamCallBack   
 \*/   
void setStreamCallback(StreamCallBack textStreamCallback, StreamCallBack toolStreamCallback);

StreamCallBack的实现与定义与LLM的回调完全相同。

### Tool Retriever

Agent在实际生产应用中往往涉及到的工具数量较多，如果把所用的工具全部添加至Agent会产生如下问题：

* 占用大量输入token
* 和问题无关的工具太多，影响模型的判断

通过Tool Retriever可以解决这个问题，其原理是在Agent运行前，先从所有可用的工具中选择与问题最相关的工具，再交给Agent去处理。

定义一个Tool Retriever：

final List<Tool> toolList =   
 Arrays.asList(new MetricQuery(), new ReserveMeeting(), new ReserveMeetingRoom(), new StuffQuery());   
   
// 新增InMemoryToolProvider，添加工具集   
final InMemoryToolProvider inMemoryToolProvider = new InMemoryToolProvider();   
inMemoryToolProvider.add(toolList);   
   
// 初始化CSSToolRetriever   
final CSSToolRetriever cssToolRetriever = new CSSToolRetriever(inMemoryToolProvider,   
 VectorStoreConfig.builder()   
 .indexName(TestConstant.CSS\_TOOL\_RETRIEVER\_INDEX)   
 .vectorFields(Arrays.asList("name", "description"))   
 .build());

定义一个ToolRetriever包含2个参数，一个ToolProvider，一个向量数据库配置。其中，ToolProvider的作用为根据工具检索的结果组装工具。

上述例子使用了一个简单的InMemoryToolProvider，InMemoryToolProvider的原理为将完整的工具存入内存，再根据工具检索的结果（toolId）将其从内存中取出。一般来说，ToolProvider将由用户自定义，后续会有例子说明。

上述例子使用的向量数据库配置指定索引名称，以及使用name和description作为向量化字段，因此工具入库时，会将工具的name和description进行向量化，并在后续的检索中生效。

注意，上述toolList中包含的工具在SDK中并不存在，需要替换成实际的工具。

向ToolRetriever中添加工具：

// 添加工具   
cssToolRetriever.addTools(toolList);

工具添加后，会存储在向量库的索引中，并将指定的字段向量化。

从ToolRetriever中查找工具：

// 查找工具   
List<Tool> result = cssToolRetriever.search("预订会议室", 2);

返回的result中，包含与预订会议室最相关的工具。搜索支持topK和阈值2个参数，例如上例指定topK=2，则最多返回2个工具。

从ToolRetriever中删除工具：

// 删除工具   
cssToolRetriever.remove(Collections.singletonList("a\_tool\_id"));

以上为一个比较基础的用法，在实际使用过程中会有更加灵活的场景，我们可以通过自定义ToolProvider的方式解决。

自定义ToolProvider：

// 初始化CSSToolRetriever，使用ToolProviderWithMetadata作为ToolProvider   
final CSSToolRetriever cssToolRetriever = new CSSToolRetriever(new ToolProviderWithMetadata(),   
 VectorStoreConfig.builder()   
 .indexName(TestConstant.CSS\_TOOL\_RETRIEVER\_INDEX)   
 .vectorFields(Arrays.asList("name", "description"))   
 .build());

其中ToolProviderWithMetadata为自定义ToolProvider：

private static class ToolProviderWithMetadata implements ToolProvider {   
 @Override   
 public List<Tool> provide(List<RetrievedTool> retrievedTools, String query) {   
 doSomeFilter(retrievedTools, query);   
 return retrievedTools.stream().map((Function<RetrievedTool, Tool>) retrievedTool -> {   
 final DynamicTool tool = new DynamicTool();   
 tool.setToolId(retrievedTool.getToolId());   
 final Map<String, Object> toolMetadata = retrievedTool.getToolMetadata();   
 tool.setToolDesc(toolMetadata.get("description").toString());   
 tool.setToolPrinciple(toolMetadata.get("description").toString());   
 // 从plugin\_schema和openapi\_schema构建工具参数信息   
 tool.setInputDesc("会议开始结束时间，会议室");   
 tool.setOutputDesc("会议预订结果");   
 tool.setInputSchema(   
 "{\"type\":\"object\",\"properties\":{\"meetingRoom\":{\"type\":\"string\",\"description\":\"会议室\"},\"start\":{\"type\":\"string\",\"description\":\"会议开始时间，格式为HH:mm\"},\"end\":{\"type\":\"string\",\"description\":\"会议结束时间，格式为HH:mm\"}},\"required\":[\"meetingRoom\",\"start\",\"end\"]}");   
 tool.setOutputSchema("");   
 tool.setFunction(s -> {   
 log.info("do the open api call plugin\_schema={} openapi\_schema={}",   
 toolMetadata.get("plugin\_schema"), toolMetadata.get("openapi\_schema"));   
 return null;   
 });   
 return tool;   
 }).collect(Collectors.toList());   
 }   
 private void doSomeFilter(List<RetrievedTool> retrievedTools, String query) {   
 // do some filter here   
 log.info("{} {}", query, retrievedTools);   
 }   
}

如上的toolProvider中，实现了provide接口，可以利用工具检索的返回动态构建出工具列表，同时也可以加一些后处理工作，例如根据黑白名单做工具的过滤。

与上述的toolProvide呼应，在向toolRetriever中添加工具时，可以添加任意的元数据，用于在tooProvider中把工具组装出来：

// 构造工具元数据   
Map<String, Object> toolMetaData = new HashMap<>();   
toolMetaData.put("name\_for\_human", "预订会议室");   
toolMetaData.put("name", "reserve\_meeting\_room");   
toolMetaData.put("description\_for\_human", "预订会议室");   
toolMetaData.put("description", "预订会议室，请在需要预订会议室时调用此工具");   
toolMetaData.put("status", "on");   
toolMetaData.put("plugin\_type", "API");   
toolMetaData.put("plugin\_schema", "this is a plugin schema");   
toolMetaData.put("openapi\_schema", "this is a openapi schema");   
final ToolMetadata toolMetadata = new ToolMetadata();   
toolMetadata.setToolId("reserve\_meeting\_room");   
toolMetadata.setToolMetadata(toolMetaData);   
   
// 工具管理面添加工具到toolRetriever，这里实际可以添加若干个工具   
cssToolRetriever.addToolsFromMetadata(Collections.singletonList(toolMetadata));   
   
// 运行时检索工具，并添加到Agent执行   
final List<Tool> toolList = cssToolRetriever.search("预订会议室", 1, 0.8f);

工具的检索与之前的用法一致。

以下是一个将Tool Retriever集成在Agent中的完整示例：

// 工具集   
final List<Tool> toolList =   
 Arrays.asList(new MetricQuery(), new ReserveMeeting(), new ReserveMeetingRoom(), new StuffQuery());   
   
// 新增InMemoryToolProvider，添加工具集   
final InMemoryToolProvider inMemoryToolProvider = new InMemoryToolProvider();   
inMemoryToolProvider.add(toolList);   
   
// 初始化CSSToolRetriever   
final VectorStoreConfig vectorStoreConfig = VectorStoreConfig.builder()   
 .indexName(TestConstant.CSS\_TOOL\_RETRIEVER\_INDEX)   
 .vectorFields(Arrays.asList("name", "description"))   
 .build();   
final CSSToolRetriever cssToolRetriever = new CSSToolRetriever(inMemoryToolProvider, vectorStoreConfig);   
   
// 添加工具   
cssToolRetriever.addTools(toolList);   
   
// 添加多轮改写   
cssToolRetriever   
 .setQueryPreprocessor(messages -> new ConversationRewriteSkill(LLMs.of(LLMs.PANGU)).rewrite(messages));   
   
// 为Agent添加ToolRetriever   
Agent agent = new ReactPanguAgent(   
 new YundaoLLM(LLMConfig.builder().llmModuleConfig(LLMModuleConfig.builder().build()).build()));   
agent.setToolRetriever(cssToolRetriever);   
   
// 多轮对话调用   
List<ConversationMessage> messages = new ArrayList<>();   
messages.add(ConversationMessage.builder().role(Role.USER).content("定个2点的会议").build());   
messages.add(   
 ConversationMessage.builder().role(Role.ASSISTANT).content("请问您的会议预计何时结束？另外，您是需要预订线上会议还是实体会议室？").build());   
messages.add(ConversationMessage.builder().role(Role.USER).content("4点结束，线上会议").build());   
agent.run(messages);   
   
// 删除工具   
cssToolRetriever.remove(toolList.stream().map(Tool::getToolId).collect(Collectors.toList()));

其中有两个变化值得关注：

第一，我们为ToolRetriever添加了一个queryPreprocessor，它的作用为对用户输入的多轮对话进行改写，会将改写后的结果作为工具检索的输入，这里我们使用了系统内置的ConversationRewriteSkill，它的作用为将多轮对话改写为单轮。

第二，我们在创建一个Agent后，调用了setToolRetriever方法为其添加了一个ToolRetriever，这样Agent所使用的工具会根据用户的对话动态的选择。

## 应用示例

### 搜索增强

场景介绍

私有化场景下，大模型需要基于现存的私有数据提供服务。通过外挂知识库（Embedding、向量库）方式提供通用的、标准化的文档问答场景。

工程实现

1. 准备知识库。
2. 获取并安装SDK包。
3. 在配置文件(llm.properties)中配置模型信息。

# 盘古模型IAM 认证信息，根据实际填写   
sdk.llm.pangu.iam.url=   
sdk.llm.pangu.iam.domain=   
sdk.llm.pangu.iam.user=   
sdk.llm.pangu.iam.password=   
sdk.llm.pangu.project=   
   
## 盘古模型信息，根据实际情况填写   
sdk.llm.pangu.url=   
   
## CSS Embedding模型api   
sdk.embedding.css.url=   
sdk.embedding.css.iam.url=   
sdk.embedding.css.iam.domain=   
sdk.embedding.css.iam.user=   
sdk.embedding.css.iam.password=   
sdk.embedding.css.iam.project=   
   
## CSS 向量库   
sdk.memory.css.url=   
sdk.memory.css.user=   
sdk.memory.css.password=

1. 工程实现。

import com.huaweicloud.pangu.dev.sdk.skill.DocSkill;   
import com.huaweicloud.pangu.dev.sdk.api.skill.Skills;   
import com.huaweicloud.pangu.dev.sdk.api.memory.vector.Vector;   
import com.huaweicloud.pangu.dev.sdk.api.memory.vector.Vectors;   
import com.huaweicloud.pangu.dev.sdk.api.memory.config.VectorStoreConfig;   
import com.huaweicloud.pangu.dev.sdk.api.embedings.Embeddings;   
   
String query = "杜甫的诗代表了什么主义诗歌艺术的高峰？";   
// 初始化向量库   
Vector cssVector = Vectors.of(Vectors.CSS,   
 VectorStoreConfig.builder()   
 .embedding(Embeddings.of(Embeddings.CSS))   
 .indexName("sdk-test-dataset-webqa-10")   
 .build());   
   
// 检索文档；   
docs = cssVector.similaritySearch(query, 4, 105);   
   
// 文档问答   
DocSkill skill = Skills.Document.newDocAskStuffSkill(LLMs.of(LLMs.PANGU));   
String answer = skill.executeWithDocs(docs, query);

### 长文本摘要

场景介绍

切割长文本，利用大模型逐步总结。 如对会议/报告/文章等较长内容总结概述。

工程实现

1. 获取并安装SDK包。
2. 在配置文件(llm.properties)中配置模型信息。

# IAM 认证信息，根据实际填写   
sdk.llm.pangu.iam.url=   
sdk.llm.pangu.iam.domain=   
sdk.llm.pangu.iam.user=   
sdk.llm.pangu.iam.password=   
sdk.llm.pangu.project=   
   
## 盘古模型信息，根据实际情况填写   
sdk.llm.pangu.url=

1. 工程实现。

import com.huaweicloud.pangu.dev.sdk.skill.DocSkill;   
import com.huaweicloud.pangu.dev.sdk.api.skill.Skills;   
import com.huaweicloud.pangu.dev.sdk.documentloader.splitter.TextSplitter;   
   
import org.apache.commons.io.FileUtils;   
   
// 加载原始内容， 需根据文件源自行实现读取步骤   
// 以txt文件为例；   
String context = FileUtils.readFileToString(new File("D:/test.txt"), StandardCharsets.UTF\_8);   
   
// 通过分割符分割文档；   
// 以 分割符“\n”， 每段大小最大1000字符 为例   
List<String> docs = TextSplitter.splitByChars(context, "\n", 1000);   
   
// 对文档进行摘要   
DocSkill skill = Skills.Document.newDocSummarizeMapReduceSkill(LLMs.of(LLMs.PANGU));   
String summarize= skill.executeWithTexts(docs);

## 常见问题

# Python SDK

[4.1 安装SDK](#_ZH-CN_TOPIC_0000001669070498" \o " )

[4.2 配置SDK](#_ZH-CN_TOPIC_0000001717109949" \o " )

[4.3 LLMs（语言模型）](#_ZH-CN_TOPIC_0000001716597361" \o " )

[4.4 Prompt（提示词模板）](#_ZH-CN_TOPIC_0000001668637648" \o " )

[4.5 Memory（记忆）](#_ZH-CN_TOPIC_0000001716517225" \o " )

[4.6 Skill（技能）](#_ZH-CN_TOPIC_0000001716517229" \o " )

[4.7 Agent（智能代理）](#_ZH-CN_TOPIC_0000001680598294" \o " )

[4.8 应用示例](#_ZH-CN_TOPIC_0000001668477924" \o " )

[4.9 常见问题](#_ZH-CN_TOPIC_0000001716517233" \o " )

## 安装SDK

pip直接安装

pip install pangu\_kits\_app\_dev\_py

本地导入

1. 从support网站上下载pangu-kits-app-dev-py的whl包。
2. 建议使用conda创建一个新的python环境，python版本选择3.9。
3. 在whl包同级目录下，执行如下命令安装。

pip install pangu\_kits\_app\_dev\_py-2.0.0-py3-none-any.whl

安装可选

1. 安装全部依赖项

pip install pangu\_kits\_app\_dev\_py[all]

1. cache相关依赖

pip install redis~=4.5.5   
pip install gptcache~=0.1.37   
pip install redis-om~=0.1.3   
pip install pymysql~=1.1.0   
pip install SQLAlchemy~=2.0.19

## 配置SDK

基础配置项

SDK依赖的配置项主要通过加载llm.properties配置文件。

1. 在项目路径下，创建llm.properties文件，并根据实际需要配置相应的值。
2. 在环境变量中配置"SDK\_CONF\_PATH"指向该配置文件。

# 建议在业务项目入口处配置   
import os   
os.environ["SDK\_CONFIG\_PATH"] = "./llm.properties"

1. 完整配置项如下。

配置项中的密码等字段建议在配置文件或者环境变量中密文存放，使用时解密，确保安全，详见[配置文件敏感信息加密配置](#d0e4270" \o " )。

################################ GENERIC CONFIG ###############################   
   
## User-defined Prompt.   
#   
sdk.prompt.path=   
   
## Proxy.   
# Examples: http://127.0.0.1:8000 ;   
#   
# sdk.proxy.enabled=   
# sdk.proxy.url=   
# sdk.proxy.user=   
# sdk.proxy.password=   
   
## Generic IAM info. This config is used when the specified IAM is not configured.   
## Either user password authentication or AK/SK authentication.   
#   
sdk.iam.url=   
sdk.iam.domain=   
sdk.iam.user=   
sdk.iam.password=   
sdk.iam.project=   
sdk.iam.ak=   
sdk.iam.sk=   
sdk.iam.disabled=   
   
   
################################ LLM ###############################   
   
## Pangu   
# Examples: https://{endPoint}/v1/{projectId}/deployments/{deploymentId} ;   
#   
sdk.llm.pangu.url=   
## If necessary, you can specify the IAM configuration.   
# sdk.llm.pangu.iam.url=   
# sdk.llm.pangu.iam.domain=   
# sdk.llm.pangu.iam.user=   
# sdk.llm.pangu.iam.password=   
# sdk.llm.pangu.iam.project=   
# sdk.llm.pangu.iam.disabled=   
## If necessary, you can specify the proxy status.   
# sdk.llm.pangu.proxy.enabled=   
   
## Gallery   
# Examples: https://{endPoint}/v1/infers/{deploymentId} ;   
#   
sdk.llm.gallery.url=   
## If necessary, you can specify the IAM configuration.   
# sdk.llm.gallery.iam.url=   
# sdk.llm.gallery.iam.domain=   
# sdk.llm.gallery.iam.user=   
# sdk.llm.gallery.iam.password=   
# sdk.llm.gallery.iam.project=   
# sdk.llm.gallery.iam.disabled=   
## If necessary, you can specify the proxy status.   
# sdk.llm.gallery.proxy.enabled=   
   
   
################################ EMBEDDINGS ###############################   
   
## CSS   
# Examples: https://{endPoint}/v1/{projectId}/applications/{appId}/{modelVersion} ;   
#   
sdk.embedding.css.url=   
## If necessary, you can specify the IAM configuration.   
# sdk.embedding.css.iam.url=   
# sdk.embedding.css.iam.domain=   
# sdk.embedding.css.iam.user=   
# sdk.embedding.css.iam.password=   
# sdk.embedding.css.iam.project=   
# sdk.embedding.css.iam.disabled=   
## If necessary, you can specify the proxy status.   
# sdk.embedding.css.proxy.enabled=   
   
   
################################ MEMORY ###############################   
   
## CSS or ES   
# Examples: http://127.0.0.1:9200,http://127.0.0.2:9200 ;   
#   
sdk.memory.css.url=   
sdk.memory.css.user=   
sdk.memory.css.password=   
   
## DCS or Redis   
# Examples: 127.0.0.1:6379 ;   
#   
# sdk.memory.dcs.url=   
# sdk.memory.dcs.user=   
# sdk.memory.dcs.password=   
   
## RDS or Mysql   
# Examples: jdbc:mariadb://127.0.0.1:3306/sdk?useSSL=false&useUnicode=true&characterEncoding=UTF-8&serverTimezone=Asia/Shanghai ;   
#   
# sdk.memory.rds.url=   
# sdk.memory.rds.user=   
# sdk.memory.rds.password=   
# sdk.memory.rds.poolSize=   
   
   
################################ DOC SPLIT ###############################   
   
## CSS   
# Examples: https://{endPoint}/v1/{projectId}/applications/{appId} ;   
#   
# sdk.doc.split.css.url=   
# sdk.doc.split.css.filepath=   
# sdk.doc.split.css.mode=   
## If necessary, you can specify the IAM configuration.   
# sdk.doc.split.css.iam.url=   
# sdk.doc.split.css.iam.domain=   
# sdk.doc.split.css.iam.user=   
# sdk.doc.split.css.iam.password=   
# sdk.doc.split.css.iam.project=   
# sdk.doc.split.css.iam.disabled=   
## If necessary, you can specify the proxy status.   
# sdk.doc.split.css.proxy.enabled=

日志打印配置

SDK日志采用logging模块。

* 参考以下代码开启相应日志打印信息。

import logging   
# 打印在命令行（与打印在文件不同时生效）   
logging.basicConfig(level=logging.DEBUG)   
   
# 打印在日志文件（与打印在命令行不同时生效）   
logging.basicConfig(level=logging.DEBUG, # 控制台打印的日志级别   
 filename='new.log',   
 filemode='a',   
 format='%(asctime)s - %(pathname)s[line:%(lineno)d] - %(levelname)s: %(message)s')

配置文件敏感信息加密配置

配置项中的认证凭据等信息不建议明文配置在配置文件中。可以通过下述方式扩展自定义的加解密组件。

1. 在一个module(yourmodule)自定义一个解密方法，decrypt\_func(key\_id, cipher)，要求可以通过`from yourmodule import decrypt\_func`这样的方式使用该方法。
2. 在配置文件中配置`sdk.crypto.implementation.path=yourmodule.decrypt\_func`指向自定义的解密方法的引用。程序加载时会通过import\_lib加载该方法。
3. 配置文件中配置密文的格式：`sdk.llm.iam.iamPwd={Crypto.key\_id}cipher`，其中key\_id和cipher会在配置项读取时被解析传递进decrypt\_func方法中，进行解密。

## LLMs（语言模型）

LLMs模块用于对大语言模型API的适配封装，提供统一的接口快速地调用盘古、GALLERY三方模型等模型API。

* 初始化

根据相应模型定义LLM类，如使用盘古LLM为： LLMs.of("pangu")。

from pangukitsappdev.api.llms.factory import LLMs   
# 初始化盘古LLM   
llm\_api = LLMs.of("pangu")   
   
# 初始化三方Gallery 托管 LLM   
llm\_api = LLMs.of("gallery")

* 基础问答

基础的模型文本问答（template等参数采用模型默认的设置）。

llm\_api.ask("你是谁？").answer

* 自定义参数问答

自定义设置如template等参数，获得对应的效果。

from pangukitsappdev.api.llms.llm\_config import LLMConfig, LLMParamConfig   
from pangukitsappdev.api.llms.factory import LLMs   
# 设置模型参数，temperature为0.9   
llm\_config = LLMConfig(llm\_param\_config=LLMParamConfig(temperature=0.9))   
   
# 初始化带参数的盘古LLM   
pangu\_api = LLMs.of("pangu", llm\_config)   
pangu\_api.ask("写一篇五言律诗").answer

支持调整的参数解释

max\_tokens: Optional[int] # 完成时要生成的令牌的最大数量   
temperature: Optional[float] # 调整随机抽样的程度，温度值越高，随机性越大; 范围见模型API规范   
top\_p: Optional[float] # 核采样值， 和temperature不同时配置   
presence\_penalty: Optional[float] # 存在惩罚，增加模型谈论新主题的可能性，范围见具体模型API规范   
frequency\_penalty: Optional[float] # 频率惩罚，降低模型重复的可能性，提高文本的多样性、创新性（从参数盘古大模型暂不支持）   
best\_of: Optional[int] # 服务侧生成优选的回答数   
stream: Optional[bool] # 是否开启流式调用

* 流式问答

模型问答，开启流式效果，响应消息流式打印。

import sys   
from pangukitsappdev.api.llms.factory import LLMs   
from pangukitsappdev.api.llms.llm\_config import LLMConfig, LLMParamConfig   
# 设置模型参数，stream为true   
llm\_config = LLMConfig(llm\_param\_config=LLMParamConfig(stream=True))   
# 盘古LLM   
pangu\_llm = LLMs.of("pangu", llm\_config)   
   
tokens = pangu\_llm.ask("写一篇200字的散文")   
   
actual\_tokens = []   
for token in tokens:   
 sys.stdout.write(token)   
 sys.stdout.flush()   
 actual\_tokens.append(token)

* 多轮对话问答

传递历史问答记录，实现多轮对话问答能力，同时支持自定义参数问答、流式问答。

from langchain.schema import ChatMessage   
from pangukitsappdev.api.llms.factory import LLMs   
   
messages = [ChatMessage(role="system", content="你是一个乐于助人的助手"), ChatMessage(role="user", content="北京有什么好玩的地方"), ChatMessage(role="assistant", content="长城"),   
 ChatMessage(role="user", content="具体介绍一下")]   
pangu\_llm = LLMs.of("pangu")   
pangu\_llm.ask(messages).answer

* 带人设的问答

支持在LLM配置项中设置人设，在LLM问答时系统会自动加上该人设，同时支持以上问答功能（暂不支持GALLERY三方模型）。

import sys   
from pangukitsappdev.api.llms.factory import LLMs   
from pangukitsappdev.api.llms.llm\_config import LLMConfig, LLMParamConfig   
   
llm\_config = LLMConfig()   
llm\_config.llm\_module\_config.system\_prompt = "你是华为开发的AI助手"   
# 盘古LLM   
pangu\_llm = LLMs.of("pangu", llm\_config)   
   
answer = pangu\_llm.ask("你是谁")

## Prompt（提示词模板）

提示词模板模块提供模板格式化、自定义配置管理功能。

* 模板格式化

from langchain import PromptTemplate   
from pangukitsappdev.api.llms.factory import LLMs   
from pangukitsappdev.api.llms.llm\_config import LLMConfig, LLMParamConfig   
# 初始化Prompt模板对象   
prompt\_template = PromptTemplate.from\_template("Tell me a {{adjective}} joke about {{content}}", template\_format="jinja2")   
   
# 支持dict格式匹配替换Prompt模板   
format1 = prompt\_template.format(\*\*{"adjective": "funny", "content": "chickens"})   
assert "Tell me a funny joke about chickens" == format1   
   
# 支持\*\*kwargs匹配替换Prompt模板   
format2 = prompt\_template.format(adjective="funny", content="chickens")   
assert "Tell me a funny joke about chickens" == format2

* 自定义prompt

# 按约定的格式准备prompt文件；   
# 文档结构和文件名参考提供的系统预置prompts文件   
   
文档结构示例：   
……   
 prompts   
 -- default   
 -- documents   
 -- stuff.pt

# 配置sdk配置项，指定prompt文件绝对路径，以 /home 路径为例   
   
sdk.prompt.path=/home/prompts/default

## Memory（记忆）

Memory（记忆）模块结合外部存储为LLM应用提供长短期记忆功能，用于支持上下文记忆的对话、搜索增强等场景。

Memory（记忆）支持多种不同的存储方式和功能。

* Cache 缓存

这是一种临时存储数据的方法，它可以提高数据的访问速度和效率。缓存可以根据不同的存储方式进行初始化、更新、查找和清理操作。缓存还可以支持语义匹配和查询，通过向量和相似度的计算，实现对数据的语义理解和检索。

* Vector 向量存储

这是一种将数据转换为数学表示的方法，它可以度量数据之间的关系和相似度。向量存储可以根据不同的词向量模型进行初始化、更新、查找和清理操作。向量存储还可以支持多种相似算法，如余弦相似度、欧氏距离、曼哈顿距离等，实现对数据的相似度评分和排序。

* History 对话消息存储

这是一种将对话消息保存在内存中的方法，它可以记录和管理对话历史。对话消息存储可以根据不同的会话标识进行初始化、更新、查找和清理操作。对话消息存储还可以支持多种过滤条件，如时间范围、用户标识、消息类型等，实现对对话消息的筛选和分析。

### Cache

Cache 缓存是一种临时存储数据的方法，它可以把常用的数据保存在内存或者其它设备中，这样当我们需要访问这些数据时，就不用再去原始的数据源查找，而是直接从缓存中获取，从而节省时间和资源。

Cache 缓存有以下几个操作。

* 初始化

指定缓存使用哪种存储方式，例如，使用内存型缓存可以设置为memory\_cache = Caches.of("inMemory")。

from pangukitsappdev.api.memory.cache.factory import Caches   
# 内存型   
memory\_cache = Caches.of("inMemory")   
# Redis   
redis\_cache = Caches.of("redis")   
# mysql   
sql\_cache = Caches.of("sql")

* 更新数据

指向缓存中添加或修改数据，需要指定数据的键值对和结果对象。

例如，把1+1这个问题和用户cache会话下对应的答案2保存到缓存中，可以参考以下示例。

from pangukitsappdev.api.schema import LLMResp   
cache = Caches.of("inMemory")   
# 更新数据   
cache.update("1+1", LLMResp(answer=2))

* 查询数据

从缓存中获取数据，需要指定数据的键值对。

例如，查找1+1这个问题对应的答案，可以参考以下示例。

# 查找数据   
cache\_value = cache.lookup("1+1")

* 清理数据

删除用户cache会话下缓存中的数据。

例如，删除所有缓存数据，可以参考以下示例。

# 清理   
cache.clear()

* 参数解释

用于设置缓存对象的一些基本信息，如过期时间、session\_tag等。

expire\_after\_access: int # 缓存失效策略-基于访问后到期时间（支持inMemeory缓存）   
expire\_after\_write: int # 缓存失效策略-基于写入后到期时间（支持redis缓存）   
maximum\_size: int # 缓存失效策略-基于个数大小（支持inMemory、sql缓存）   
session\_tag: str # 用户指定cache会话标志

* 缓存失效策略

from pangukitsappdev.api.memory.cache.cache\_config import CacheStoreConfig   
   
# redis缓存配置写入2s后到期   
redis\_cache = Caches.of("redis", CacheStoreConfig(expire\_after\_write=2))   
# inMemory缓存配置缓存窗口数量为3，访问后2s到期   
memory\_cache = Caches.of("inMemory", CacheStoreConfig(maximum\_size=3, expire\_after\_access=2))   
# sql缓存配置缓存窗口数量为3   
sql\_cache = Caches.of("sql", CacheStoreConfig(maximum\_size=3))

* 语义缓存（同步适配langchain语义缓存暂时不支持expire\_after\_write）

语义缓存是一种基于向量和相似度的缓存方法，它可以实现对数据的语义匹配和查询。语义缓存可以根据不同的向量存储、相似算法、评分规则和阈值进行配置，并且可以使用不同的词向量模型进行嵌入。

from pangukitsappdev.api.memory.cache.cache\_config import CacheStoreConfig, ServerInfoRedis   
from pangukitsappdev.api.embeddings.factory import Embeddings   
# redis向量   
# 不同的向量存储， 不同的相似算法；计算的评分规则不同； 可以同过scoreThreshold 设置相似性判断阈值   
# 例如使用Redis向量、余弦相似度、CSS词向量模型，并且设置相似性判断阈值为0.1f，代码示例如下   
embedding\_api = Embeddings.of("css")   
cache\_config = CacheStoreConfig(store\_name="semantic\_redis",   
 server\_info=ServerInfoRedis(env\_prefix="sdk.memory.dcs"),   
 embedding=embedding\_api,   
 vector\_store\_name="redis",   
 distance\_strategy="cosine",   
 score\_threshold=0.1,   
 session\_tag="test-semantic-cache-vector-001")   
cache = Caches.of("semantic\_redis", cache\_config)   
   
# 更新数据   
# 指向语义缓存中添加或修改数据，需要指定数据的键值对和结果对象   
# 例如，把“把缓存是否存在？”这个问题和“test-semantic-cache-vector-001”这个会话标识对应的答案“存在”保存到语义缓存中，代码示例如下   
cache.update("缓存是否存在？", LLMResp(answer="存在。"))   
   
# 校验，一致   
# 用于检查缓存中的数据是否与查询的数据是否一致，如果一致，就返回缓存中的结果对象   
# 例如，查询“缓存是否存在？”这个问题和“test-semantic-cache-vector-001”这个会话标识，就可以从缓存中获取到之前保存的答案“存在”   
query = "缓存是否存在？"   
cache\_value\_after = cache.lookup(query)   
assert cache\_value\_after is not None   
print(cache\_value\_after.answer)   
   
# 校验，相似   
# 用于检查缓存中的数据是否与查询的数据语义相似，如果相似，就返回缓存中的结果对象。这个操作需要使用向量和相似度的计算，以及设置的阈值来判断   
# 例如，查询“缓存存在？”这个问题和“test-semantic-cache-vector-001”这个会话标识，就可以从缓存中获取到之前保存的答案“存在”，因为这个问题和“缓存是否存在？”问题语义相似   
query\_sim = "缓存存在？"   
cache\_value\_semantic = cache.lookup(query\_sim)   
assert cache\_value\_semantic is not None   
print(cache\_value\_semantic.answer)   
   
# 校验，不一致   
# 用于检查缓存中的数据是否与查询的数据不一致，如果不一致，返回空值   
# 例如，查询“有没有数据？”这个问题和“test-semantic-cache-vector-001”这个会话标识，就无法从缓存中获取到任何答案，因为这个问题和之前保存的问题都不一致   
query\_not = "有没有数据？"   
assert cache.lookup(query\_not) is None

### Vector

Embedding

Emebedding模块用于对Emebedding模型API的适配封装，提供统一的接口快速地调用CSS等模型emebedding能力。

* 初始化

根据相应模型定义Emebedding类，如使用华为CSS Embedding为：Embeddings.of("css");。

from pangukitsappdev.api.embeddings.factory import Embeddings   
# 初始化 Css Embedding   
embedding\_api = Embeddings.of("css")

* embedding 单文本

把单个字符串转换为向量数据。（向量维度由模型确定）。

text = "this is a test text."   
# embed query.   
embedding = embedding\_api.embed\_query(text)   
print(embedding)

* embedding 批量文档

把文档批量转换为向量数据。

text = "this is a test text."   
# embed documents.   
embeddings = embedding\_api.embed\_documents([text])   
print(embeddings)

Splitter

文档拆分解析，提供对文档数据进行拆分解析能力，支持pdf/doc/docx/ppt/pptx/xls/xlsx/png/jpg/jpeg/bmp/gif/tiff/webp/pcx/ico/psd等格式文档。

* 初始化

根据相应解析接口定义DocSplit类，如使用华为Pangu DocSplit为例。

其中filePath指的是需要解析的文档路径；mode为分割解析模式，具体定义如下：

0 - 返回文档的原始段落，不做其它处理。

1 - 根据标注的书签或目录分段，一般适合有层级标签的word文档。

2 - 根据内容里的章节条分段，适合制度类文档。

3 - 根据长度分段，默认按照500字拆分，会尽量保留完整句子。

from pangukitsappdev.api.doc\_split.factory import DocSplits   
from pangukitsappdev.api.doc\_split.split\_config import SplitConfig   
split\_config = SplitConfig()   
split\_config.file\_path='/data/xxx.docx'   
split\_api = DocSplits.of("pangu-doc", split\_config)

* 文档解析

doc\_list = split\_api.load()   
for doc in doc\_list:   
 print(doc.page\_content)

向量库

向量库，作为向量数据存储； 并提供向量数据检索能力。

* 初始化

以使用华为CSS示例。

from pangukitsappdev.api.memory.vector.factory import Vectors   
from pangukitsappdev.api.memory.vector.vector\_config import VectorStoreConfig, ServerInfoCss   
from pangukitsappdev.api.embeddings.factory import Embeddings   
vector\_store\_config = VectorStoreConfig(store\_name="css",   
 index\_name="your\_index\_name",   
 embedding=Embeddings.of("css"),   
 text\_key="name",   
 vector\_fields=["description"],   
 distance\_strategy="inner\_product",   
 server\_info=ServerInfoCss(env\_prefix="sdk.memory.css"))   
vector\_api = Vectors.of("css", vector\_store\_config)

* 数据入库

from pangukitsappdev.vectorstores.bulk\_data import BulkData   
bulk\_list = [BulkData(id="1", data={"name": "名称name1", "description": "foo"}),   
 BulkData(id="2", data={"name": "名称name2", "description": "bar"}),   
 BulkData(id="3", data={"name": "名称name3", "description": "baz"})]   
]   
vector\_api.add\_docs(bulk\_list)

* 数据检索

docs = vector\_api.similarity\_search("bar", top\_k=2)

* 数据清理

vector\_api.clear()

* CSS插件模式（内部已集成Embedding， 支持多字段组合向量检索）。

CSS插件模式，需要提前手工创建索引（因索引中需要指定embdding/rank模型，SDK不能简单自动创建）。

CSS插件模式，不支持clear删除索引接口（索引外部创建，应由外部删除）。

vector\_store\_config = VectorStoreConfig(store\_name="css",   
 index\_name="your\_index\_name",   
 text\_key="name",   
 vector\_fields=["name", "description"],   
 server\_info=ServerInfoCss(env\_prefix="sdk.memory.css"))   
vector\_api = Vectors.of("css", vector\_store\_config)   
   
# 检索   
docs = vector\_api.similarity\_search("bar", top\_k=2)   
   
# 添加   
bulk\_list = [BulkData(id="1", data={"name": "名称name1", "description": "foo"}),   
 BulkData(id="2", data={"name": "名称name2", "description": "bar"}),   
 BulkData(id="3", data={"name": "名称name3", "description": "baz"})]   
]   
vector\_api.add\_docs(bulk\_list)

通过vectorStoreConfig判断使用CSS的插件模式和非插件模式，如果配置了embedding模型，则使用非插件模式；否则使用插件模式。注意，在非插件模式下，vectorFields有且只有1个。

### History

History 缓存，用于存储历史对话信息，用于辅助模型理解上下文信息； 历史消息对有固定窗口、消息摘要等策略。

* 初始化

消息记录支持不同的存储方式， 如内存、DCS（Redis）、RDS（Sql）。

from pangukitsappdev.memory.sql\_message\_history import SQLMessageHistory   
from pangukitsappdev.api.memory.cache.cache\_config import ServerInfoSql, ServerInfoRedis   
from pangukitsappdev.api.memory.message\_history\_config import MessageHistoryConfig   
from pangukitsappdev.memory.redis\_message\_history import RedisMessageHistory   
from langchain.memory import ChatMessageHistory   
# 内存型:   
chat\_message = ChatMessageHistory()   
   
# Redis:   
redis\_chat\_message = RedisMessageHistory(msg\_history\_config=MessageHistoryConfig(store\_name="redis",   
 server\_info=ServerInfoRedis(env\_prefix="sdk.memory.dcs"),   
 session\_tag="test-memory-0624"))   
# Sql:   
sql\_chat\_message = SQLMessageHistory(msg\_history\_config=MessageHistoryConfig(store\_name="sql",   
 server\_info=ServerInfoSql(env\_prefix="sdk.memory.rds"),   
 session\_tag="test-memory-0624"))

* 添加、查找、删除数据

# 更新数据   
chat\_message.add\_ai\_message("i am ai.")   
chat\_message.add\_user\_message("i am tester.")   
# 查找数据   
contents = [msg.content for msg in chat\_message.messages]   
# 清理   
chat\_message.clear()

* 消息策略（windows-size）

将固定轮次历史对话信息，作为历史上下文信息。

from langchain.memory import ChatMessageHistory, ConversationBufferWindowMemory   
from pangukitsappdev.api.memory.cache.cache\_config import ServerInfoRedis   
# 历史消息存储   
redis\_chat\_message = RedisMessageHistory(msg\_history\_config=MessageHistoryConfig(store\_name="redis",   
 server\_info=ServerInfoRedis(env\_prefix="sdk.memory.dcs"),   
 session\_tag="test-memory"))   
# 固定窗口策略: langchain.memory.buffer\_window.ConversationBufferWindowMemory   
memory = ConversationBufferWindowMemory(k=3, chat\_memory=redis\_chat\_message)   
   
memory.chat\_memory.add\_user\_message("This is me, the human")   
memory.chat\_memory.add\_ai\_message("This is me, the AI")   
print(memory.chat\_memory.messages)   
memory.chat\_memory.clear()

* 消息策略（信息摘要）

将历史消息进行摘要后，作为历史上下文信息。

from pangukitsappdev.memory.conversation\_summory\_memory import ConversationSummaryBufferMemory   
from pangukitsappdev.api.skill.base import SimpleSkill   
from pangukitsappdev.prompt.prompt\_tmpl import PromptTemplates   
# 历史消息存储   
redis\_chat\_message = RedisMessageHistory(msg\_history\_config=MessageHistoryConfig(store\_name="redis",   
 server\_info=ServerInfoRedis(env\_prefix="sdk.memory.dcs"),   
 session\_tag="test-memory"))   
# 摘要策略   
memory = ConversationSummaryBufferMemory(summary\_skill=SimpleSkill(PromptTemplates.get("memory\_summary"), LLMs.of("pangu")),   
 chat\_memory=redis\_chat\_message)   
   
memory.chat\_memory.add\_user\_message("This is me, the human")   
memory.chat\_memory.add\_ai\_message("This is me, the AI")   
print(memory.chat\_memory.messages)   
memory.chat\_memory.clear()

## Skill（技能）

### 基础问答

提供简单的对话实现。

* 初始化

from pangukitsappdev.api.llms.factory import LLMs   
from pangukitsappdev.api.skill.base import SimpleSkill   
from langchain.prompts import PromptTemplate   
# 自定义模板   
prompt\_template = PromptTemplate.from\_template("讲一个关于{{subject}}的笑话，字数{{count}}字以内", template\_format="jinja2")   
skill = SimpleSkill(prompt\_template=prompt\_template, llm\_api=LLMs.of("pangu"))

* 问答

from pangukitsappdev.api.llms.llm\_config import LLMParamConfig   
   
# 不带参数的问答   
skill.execute({"subject": "哈士奇", "count": 20})   
   
# 带参数的问答   
llm\_param\_config = LLMParamConfig(temperature=0.9)   
skill.execute({"subject": "哈士奇", "count": 20}, llm\_param\_config)

### 多轮对话

支持上下文记忆的多轮对话。

* 初始化

from pangukitsappdev.skill.conversation\_skill import ConversationSkill   
from pangukitsappdev.api.llms.factory import LLMs   
skill = ConversationSkill(LLMs.of("pangu"))

* 问答

from pangukitsappdev.memory.redis\_message\_history import RedisMessageHistory   
from langchain.memory import ConversationBufferWindowMemory   
# 定义存储策略   
skill.set\_memory(ConversationBufferWindowMemory(k=3, chat\_memory=RedisMessageHistory()))   
answer = skill.execute("中国首都是哪个城市？")   
print(answer)   
assert "北京" in answer   
answer = skill.execute("它有什么好玩的地方？")   
print(answer)   
assert "故宫" in answer

### 文档问答

基于已有的知识库，进行回答。有stuff、refine， map-reduce策略。

* Stuff

将所有文档直接填充到prompt中，提给模型回答； 适合文档较少场景。

from pangukitsappdev.api.embeddings.factory import Embeddings   
from pangukitsappdev.api.llms.factory import LLMs   
from pangukitsappdev.api.memory.vector.factory import Vectors   
from pangukitsappdev.api.memory.vector.vector\_config import VectorStoreConfig, ServerInfoCss   
from pangukitsappdev.skill.doc.ask import DocAskStuffSkill   
vector\_store\_config = VectorStoreConfig(store\_name="css",   
 index\_name="your\_index\_name",   
 embedding=Embeddings.of("css"),   
 text\_key="name",   
 vector\_fields=["description"],   
 distance\_strategy="inner\_product",   
 server\_info=ServerInfoCss(env\_prefix="sdk.memory.css"))   
vector\_api = Vectors.of("css", vector\_store\_config)   
# 检索   
query = "杜甫的诗代表了什么主义诗歌艺术的高峰？"   
docs = vector\_api.similarity\_search(query, 4)   
   
# 问答   
doc\_skill = DocAskStuffSkill(LLMs.of("pangu"))   
   
print(doc\_skill.execute({"documents": docs, "question": query}))

* Refine

基于首个文档，并循环后续文档来迭代更新答案。

from pangukitsappdev.api.embeddings.factory import Embeddings   
from pangukitsappdev.api.llms.factory import LLMs   
from pangukitsappdev.api.memory.vector.factory import Vectors   
from pangukitsappdev.api.memory.vector.vector\_config import VectorStoreConfig, ServerInfoCss   
from pangukitsappdev.skill.doc.ask import DocAskRefineSkill   
vector\_store\_config = VectorStoreConfig(store\_name="css",   
 index\_name="your\_index\_name",   
 embedding=Embeddings.of("css"),   
 text\_key="name",   
 vector\_fields=["description"],   
 distance\_strategy="inner\_product",   
 server\_info=ServerInfoCss(env\_prefix="sdk.memory.css"))   
vector\_api = Vectors.of("css", vector\_store\_config)   
   
# 检索   
query = "杜甫的诗代表了什么主义诗歌艺术的高峰？"   
docs = vector\_api.similarity\_search(query, 4)   
   
# 问答   
doc\_skill = DocAskRefineSkill(LLMs.of("pangu"))   
   
print(doc\_skill.execute({"documents": docs, "question": query}))

* Map-Reduce

先将文档单独进行摘要， 将摘要后的文档再提交给模型。 必要时，会循环迭代摘要。

from pangukitsappdev.api.embeddings.factory import Embeddings   
from pangukitsappdev.api.llms.factory import LLMs   
from pangukitsappdev.api.memory.vector.factory import Vectors   
from pangukitsappdev.api.memory.vector.vector\_config import VectorStoreConfig, ServerInfoCss   
from pangukitsappdev.skill.doc.ask import DocAskMapReduceSkill   
vector\_store\_config = VectorStoreConfig(store\_name="css",   
 index\_name="your\_index\_name",   
 embedding=Embeddings.of("css"),   
 text\_key="name",   
 vector\_fields=["description"],   
 distance\_strategy="inner\_product",   
 server\_info=ServerInfoCss(env\_prefix="sdk.memory.css"))   
vector\_api = Vectors.of("css", vector\_store\_config)   
   
# 检索   
query = "杜甫的诗代表了什么主义诗歌艺术的高峰？"   
docs = vector\_api.similarity\_search(query, 4)   
   
# 问答   
doc\_skill = DocAskMapReduceSkill(LLMs.of("pangu"))   
   
print(doc\_skill.execute({"documents": docs, "question": query}))

### 文档摘要

基于已有的知识库，进行摘要总结。有stuff、refine、map-reduce策略。

* Stuff

将所有文档直接填充到prompt中，提给模型处理； 适合文档较少场景。

from pangukitsappdev.api.embeddings.factory import Embeddings   
from pangukitsappdev.api.llms.factory import LLMs   
from pangukitsappdev.api.memory.vector.factory import Vectors   
from pangukitsappdev.api.memory.vector.vector\_config import VectorStoreConfig, ServerInfoCss   
from pangukitsappdev.skill.doc.summary import DocSummaryStuffSkill   
vector\_store\_config = VectorStoreConfig(store\_name="css",   
 index\_name="your\_index\_name",   
 embedding=Embeddings.of("css"),   
 text\_key="name",   
 vector\_fields=["description"],   
 distance\_strategy="inner\_product",   
 server\_info=ServerInfoCss(env\_prefix="sdk.memory.css"))   
vector\_api = Vectors.of("css", vector\_store\_config)   
   
# 检索   
query = "杜甫"   
docs = vector\_api.similarity\_search(query, 4)   
   
# 摘要   
doc\_skill = DocSummaryStuffSkill(LLMs.of("pangu"))   
   
print(doc\_skill.execute({"documents": docs}))

* Refine

基于首个文档，并循环后续文档来迭代更新答案。

from pangukitsappdev.api.embeddings.factory import Embeddings   
from pangukitsappdev.api.llms.factory import LLMs   
from pangukitsappdev.api.memory.vector.factory import Vectors   
from pangukitsappdev.api.memory.vector.vector\_config import VectorStoreConfig, ServerInfoCss   
from pangukitsappdev.skill.doc.summary import DocSummaryRefineSkill   
vector\_store\_config = VectorStoreConfig(store\_name="css",   
 index\_name="your\_index\_name",   
 embedding=Embeddings.of("css"),   
 text\_key="name",   
 vector\_fields=["description"],   
 distance\_strategy="inner\_product",   
 server\_info=ServerInfoCss(env\_prefix="sdk.memory.css"))   
   
vector\_api = Vectors.of("css", vector\_store\_config)   
   
# 检索   
query = "杜甫"   
docs = vector\_api.similarity\_search(query, 4)   
   
# 摘要   
doc\_skill = DocSummaryRefineSkill(LLMs.of("pangu"))   
   
print(doc\_skill.execute({"documents": docs}))

* Map-Reduce

先将文档单独进行摘要， 将摘要后的文档再提交给模型。 必要时，会循环迭代摘要。

from pangukitsappdev.api.embeddings.factory import Embeddings   
from pangukitsappdev.api.llms.factory import LLMs   
from pangukitsappdev.api.memory.vector.factory import Vectors   
from pangukitsappdev.api.memory.vector.vector\_config import VectorStoreConfig, ServerInfoCss   
from pangukitsappdev.skill.doc.summary import DocSummaryMapReduceSkill   
vector\_store\_config = VectorStoreConfig(store\_name="css",   
 index\_name="your\_index\_name",   
 embedding=Embeddings.of("css"),   
 text\_key="name",   
 vector\_fields=["description"],   
 distance\_strategy="inner\_product",   
 server\_info=ServerInfoCss(env\_prefix="sdk.memory.css"))   
   
vector\_api = Vectors.of("css", vector\_store\_config)   
   
# 检索   
query = "杜甫"   
docs = vector\_api.similarity\_search(query, 4)   
   
# 摘要   
doc\_skill = DocSummaryMapReduceSkill(LLMs.of("pangu"))   
   
print(doc\_skill.execute({"documents": docs}))

## Agent（智能代理）

Agent（智能代理），用于对复杂任务的自动拆解与外部工具调用执行，一般包括任务规划、记忆系统、执行系统。

* 任务规划：将复杂目标任务分解为小的可执行子任务，通过评估、自我反思等方式提升规划成功率。
* 记忆系统：通过构建记忆模块去管理历史任务和策略，并让Agent结合记忆模块中相关的信息以获取最优化任务解决策略。
* 任务执行：能通过工具与外界发生联系并产生影响，工具可以自定义，包括查询信息、调用服务、网络搜索、文件管理、调用云服务等，通过Agent构建一个让LLM按照特定的规则迭代运行的Prompt，直到任务完成或者达到终止条件（如设置迭代次数）。

### 实例化Tool

Tool分为StaticTool（静态工具）和DynamicTool（动态工具）两类，静态工具需要开发者事先定义好，即在编译期定义与实例化；动态工具开发者可以在系统运行时动态构建，即在运行态定义与实例化。

StaticTool（静态工具）

静态工具可以通过继承Tool的方式新增，在\_run接口中实现工具的功能，例如：

from typing import Type   
from pangukitsappdev.tool.tool import Tool   
from pydantic import BaseModel, Field   
   
   
class AddTool(Tool):   
 class AddParam(BaseModel):   
 a: int = Field(description="加法运算的数字")   
 b: int = Field(description="加法运算的数字")   
   
 name = "add"   
 description = "加法运算"   
 args\_schema: Type[BaseModel] = AddParam   
 principle = "请在需要做两个整型的加法运算时调用此工具"   
 input\_desc = "加法输入"   
 output\_desc = "加法运算的结果"   
   
 def \_run(self, a: int, b: int) -> int:   
 return a + b

@Tool说明：

name，工具的标识，建议为英文且与实际工具含义匹配，在同一个Agent中唯一。

description，工具的描述，建议为中文，尽可能的简短描述工具。

principle，【重要】何时使用该工具，该描述直接影响LLM对工具使用的判断，尽量描述清楚，如果Agent实际执行效果不符合预期，可以调整。

input\_desc，【重要】工具的入参描述 ，该描述直接影响LLM对入参的提取，尽量描述清楚，如果Agent实际执行效果不符合预期，可以调整。

output\_desc，工具的出参描述，当前对Agent的表现无重要影响。

args\_schema，【重要】工具入参类型，入参继承BaseModel的类型需额外指定，简单类型无需指定。

return\_type，【可选】指定工具返回类型，如\_run方法未指定返回类型时必选。

如果输入输出参数为复杂类型，则需要通过继承BaseModel定义复杂类型的参数描述，此时input\_desc、output\_desc可以填空字符串，但仍然建议给出简要的描述。当前版本不支持复杂类型中再嵌套复杂类型，只支持基本类型：str、int、float、bool，建议参数数量不超过5个。

@Field说明：

description，【重要】参数的描述，该描述直接影响LLM对入参的提取，尽量描述清楚，如果Agent实际执行效果不符合预期，可以调整。

上例中的args\_schema为一个复杂的入参，如果工具的入参为一个基本类型，则不需要再额外定一个结构体，例如：

from typing import Type   
from pangukitsappdev.tool.tool import Tool   
from pydantic import BaseModel, Field   
   
   
class ReverseTool(Tool):   
 name = "reverse"   
 description = "字符串翻转"   
 principle = "请在需要字符串翻转时调用此工具"   
 input\_desc = "输入的字符串"   
 output\_desc = "反转的结果"   
   
 def \_run(self, s: str) -> str:   
 return s[::-1]

DynamicTool（动态工具）

动态工具可以在业务运行态动态新增或修改：

from pangukitsappdev.tool.tool import Tool   
from pydantic import BaseModel, Field, create\_model   
   
def add(a: int, b: int) -> int:   
 return a + b   
   
# 定义函数方式   
add\_tool = Tool.from\_function(func=add,   
 name="add",   
 description="加法运算",   
 principle="请在需要做两个整型的加法运算时调用此工具",   
 input\_desc="加法输入",   
 output\_desc="加法运算的结果",   
 args\_schema=create\_model('AddTool', a=(int, Field(description="加法运算的数字")), b=(int, Field(description="加法运算的数字")))   
)   
# lambda匿名方式   
lambda\_add\_tool = Tool.from\_function(func=lambda a, b: a+b,   
 name="add",   
 description="加法运算",   
 principle="请在需要做两个整型的加法运算时调用此工具",   
 input\_desc="加法输入",   
 output\_desc="加法运算的结果",   
 args\_schema=create\_model('AddTool', a=(int, Field(description="加法运算的数字")), b=(int, Field(description="加法运算的数字"))),   
 return\_type=int)

name、description、principle、input\_desc、output\_desc, args\_schema的定义与说明与静态工具相同。

return\_type：【可选】如func为未指定返回值类型的callable类型，必须通过return\_type指定返回值类型。

### 实例化Agent

Agent实例化过程包括两个部分，注册LLM、注册工具。

from pangukitsappdev.agent.react\_pangu\_agent import ReactPanguAgent   
from pangukitsappdev.api.llms.factory import LLMs   
   
agent = ReactPanguAgent(LLMs.of("pangu"))   
agent.set\_max\_iterations(5)   
agent.add\_tool(ReverseTool())   
agent.add\_tool(AddTool())   
agent.add\_tool(SearchTool())

静态工具和动态工具的注册方式相同，通过addTool接口进行注册。

通过set\_max\_iterations可以设置最大迭代次数，控制Agent子规划的最大迭代步数，防止无限制的迭代或出现死循环情况。

Agent使用的模型必须为Pangu-NLP-N2-Agent-L0.C模型，或其衍生模型，使用通用模型或其它模型无法运行。如上例所示，当前的moduleVersion需要配置为“38B”，模型的相关配置需要改为Pangu-NLP-N2-Agent-L0.C模型的地址。

### 运行Agent

调用run接口运行一个Agent：

agent.run("帮我定个下午3点到8点2303会议室")

Agent的运行时会进行自我迭代，并且选择合适的工具，在日志中打印最终的执行结果：

用户问题为:帮我定个下午3点到8点2303会议室   
计划已执行完成,自动编排步骤:   
思考:根据人类的需求，我需要使用预订会议室的工具，将会议室设置为2303，开始时间设置为15:00，结束时间设置为20:00。   
行动:使用工具[reserve\_meeting\_room],传入参数{"meetingRoom":"2303","start":"15:00","end":"20:00"}工具返回:15:00到20:00的2303已预订成功

多轮接口：

messages = [ChatMessage(role="user", content="定个2点的会议"), ChatMessage(role="assistant", content="请问您的会议预计何时结束？另外，您是需要预订线上会议还是实体会议室？"),   
 ChatMessage(role="user", content="4点结束，线上会议")]   
agent.run(messages).final\_answer

运行结果示例：

user: 定个2点的会议   
assistant: 请问您的会议预计何时结束？另外，您是需要预订线上会议还是实体会议室？   
user: 4点结束，线上会议   
计划已执行完成,自动编排步骤:   
工具返回:2023-06-2214:00到2023-06-2216:00的线上会议已预定成功   
行动:使用工具[reserve\_meeting],传入参数{"start": "2023-06-2214:00","end": "2023-06-2216:00"}   
思考:好的，我将为您预定2点至4点的线上会议。现在开始调用工具。   
思考:已为您预定2023年6月22日下午2点至4点的线上会议，请准时参加。

有些时候，我们并不希望Agent完全自主执行，在某些关键节点，让用户先进行确认，确认后再执行，或者用户对模型的结果有异议或者想法有变化，想对当前结果进行更改。此时我们可以单步运行Agnet：

"""   
单步执行Agent，提供干预能力   
:param agent\_session: 包括初始状态，以及执行步骤间的agentSession，可以使用AgentSessionHelper类辅助处理   
:return: Agent执行的结果   
"""   
run\_step(agent\_session: AgentSession) -> AgentSession

以下为一个完整的示例：

agent.clear\_tool()   
agent.add\_tool(RiskDetectionTool())   
session = AgentSessionHelper.init\_agent\_session("请帮我查一下方欣科技有限公司今年1月的经营异常风险")   
# 预期Agent返回reportType为经营异常风险检测的Json，呈现给终端用户   
session = agent.run\_step(session)   
# 终端用户反悔，想改成欠税信息检测，修改信息后继续让Agent执行   
AgentSessionHelper.set\_user\_feedback(session, "改为欠税信息")   
   
# 预期Agent返回reportType为欠税信息体检的Json，呈现给终端用户   
session = agent.run\_step(session)   
# 终端用户确认，调用外部系统，进一步确认公司名称和编号，补充信息后，让Agent继续执行   
AgentSessionHelper.set\_user\_feedback(session, "公司改为金财互联数据服务有限公司，nsrsbh为91440101MA59JN5456")   
   
# 预期Agent更新companyName和nsrsbh   
session = agent.run\_step(session)   
# 此时已经得到完整调用信息了，调用risk\_detection API，将API返回结果给Agent继续执行   
AgentSessionHelper.set\_tool\_output(session, "金财互联数据服务有限公司1月份存在异常经营风险")   
   
session = agent.run\_step(session)   
# 打印   
if session.agent\_session\_status == "FINISHED":   
 print(AgentSessionHelper.print\_plan(session))

上面的例子中，终端用户对Agent即将要执行的API做了2次修改，最终进行了执行。run\_step的入参为AgentSession，有较大的灵活性，因此提供了辅助类AgentSessionHelper帮助开发者更容易处理：

init\_agent\_session：使用用户消息初始化一个AgentSession

set\_tool\_output：向Agent的当前步骤设置工具返回结果

set\_user\_feedback：向Agent的当前步骤设置用户反馈

打印的结果为：

用户: 请帮我查一下方欣科技有限公司今年1月的经营异常风险   
计划已执行完成,自动编排步骤:   
步骤1:   
思考:好的，我将使用"risk\_detection"工具来进行企业健康体检。根据您的需求，需要检测的是方欣科技有限公司今年1月的经营异常风险。今年1月的日期范围是从2024-01-01到2024-01-31。现在我将调用工具进行体检。   
行动:使用工具[risk\_detection],传入参数"{\"reportType\": \"经营异常风险检测\",\"companyName\": \"方欣科技有限公司\",\"skssqq\": \"2024-01-01\",\"skssqz\": \"2024-01-31\"}"   
用户反馈:改为欠税信息   
步骤2:   
思考:了解，我将重新调用"risk\_detection"工具，这次检测的是方欣科技有限公司今年1月的欠税信息。   
行动:使用工具[risk\_detection],传入参数"{\"reportType\": \"欠税信息体检\",\"companyName\": \"方欣科技有限公司\",\"skssqq\": \"2024-01-01\",\"skssqz\": \"2024-01-31\"}"   
用户反馈:公司改为金财互联数据服务有限公司，nsrsbh为91440101MA59JN1111   
步骤3:   
思考:好的，我将继续使用"risk\_detection"工具，这次检测的是金财互联数据服务有限公司今年1月的欠税信息，并提供了企业编号91440101MA59JN1111。   
行动:使用工具[risk\_detection],传入参数"{\"reportType\": \"欠税信息体检\",\"companyName\": \"金财互联数据服务有限公司\",\"skssqq\": \"2024-01-01\",\"skssqz\": \"2024-01-31\",\"nsrsbh\": \"91440101MA59JN5456\"}"   
工具返回:金财互联数据服务有限公司1月份存在异常经营风险   
步骤4:   
思考:"根据体检结果，金财互联数据服务有限公司在今年1月份存在异常经营风险。"

有些时候，我们会需要Agent对之前的思考、行动回答用户的最终问题，如果对模型返回的结果不满意，我们可以选择使用AgentSessionSkill解决：

llm\_api = LLMs.of("pangu", llm\_config=LLMConfig(llm\_param\_config=LLMParamConfig(temperature=0.01), llm\_module\_config=LLMModuleConfig( module\_version="N2\_agent\_v2")))   
agent = ReactPanguAgent(llm=llm\_api)   
agent.set\_max\_iterations(5)   
agent.add\_tool(QueryScoreTool())   
   
agent\_session = agent.run("我的数学成绩和语文成绩分别是多少")   
   
summary\_llm\_api = LLMs.of("pangu", llm\_config=LLMConfig(llm\_module\_config=LLMModuleConfig(url="https://pangu.cn-southwest-2.myhuaweicloud.com/v1/infers/N2-基础模型的调用地址")))   
agent\_session\_skill = AgentSessionSkill(llm\_api=summary\_llm\_api)   
print(agent\_session\_skill.summary(agent\_session))

运行结果示例：

user: 我的数学成绩和语文成绩分别是多少   
计划已执行完成,自动编排步骤:   
工具返回:你的数学的成绩是99分   
行动:使用工具[query\_score],传入参数{"subject\_name": "数学"}   
思考:好的，我可以帮你查询。首先，我将调用查询成绩工具来查询你的数学成绩。   
   
思考:你的数学成绩是99分。接下来，我将查询你的语文成绩。   
行动:使用工具[query\_score],传入参数{"subject\_name": "语文"}   
工具返回:你的语文的成绩是99分   
问题已求解:您的语文成绩也是99分

最终结果: 您的数学成绩是99分，您的语文成绩也是99分。

AgentSessionSkill使用的大模型建议为N2-基础模型或者其它同等类型的模型。

### 监听Agent

一次Agent的响应如果涉及到多个任务的分解，往往会执行比较长的时间，此时可以对agent的执行过程进行监听，输出中间步骤。

定义一个监听器

通过实现AgentListener定义一个监听器：

from pangukitsappdev.agent.agent\_session import AgentSession   
from pangukitsappdev.api.agent.base import AgentListener   
   
class TestListener(AgentListener):   
 def on\_session\_start(self, agent\_session: AgentSession):   
 print(agent\_session.dict().get("current\_action"))   
   
 def on\_session\_iteration(self, agent\_session: AgentSession):   
 print(agent\_session.dict().get("current\_action"))   
   
 def on\_session\_end(self, agent\_session: AgentSession):   
 print(agent\_session.dict().get("current\_action"))

分别对应了Agent的开始、中间过程、结束阶段。

为Agent添加一个监听器

通多调用Agent的addListener接口添加一个监听器：

from pangukitsappdev.agent.react\_pangu\_agent import ReactPanguAgent   
from pangukitsappdev.api.llms.factory import LLMs   
   
agent = ReactPanguAgent(LLMs.of("pangu"))   
agent.add\_listener(TestListener())

listener会在Agent运行时生效。

监听的对象

监听的对象为一个AgentSession：

class AgentSession(BaseModel):   
 """   
 Agent运行Session，包含历史Action，当前Action，状态   
 Attributes:   
 messages: 本次session的用户的输入   
 session\_id: UUID，在一个session内唯一   
 final\_answer: Agent返回的最终答案（最后一个AgentAction的输出）   
 history\_action: 历史Action   
 current\_action: 当前Action   
 agent\_session\_status: Agent状态   
 """   
 messages: List[BaseMessage]   
 session\_id: str   
 final\_answer: Optional[str]   
 history\_action: List[AgentAction]   
 current\_action: Optional[AgentAction]   
 agent\_session\_status: str

AgentAction包含Agent的工具选择、工具执行结果、思考等信息，AgentSessionStatus为Agnet的执行状态。

通过监听终止Agent的执行

当需要在Agent的执行过程中终止执行时，除了通过setMaxIterations设置Agent的最大迭代次数，也可以通过实现监听器的on\_check\_interrupt\_requirement实现。

class InterruptListener(AgentListener):   
 def on\_check\_interrupt\_requirements(self, agent\_session: AgentSession):   
 if "capital" == agent\_session.current\_action.action:   
 # 如果当前的action为captial，则返回capital工具调用的原始返回值   
 agent\_session.final\_answer = agent\_session.current\_action.observation   
 return True   
 return False   
agent.add\_listener(InterruptListener())

上面的例子中，当满足if判断条件时，就会直接终止agent的执行，并且agent的finalAnswer被设置为工具的原始返回值。

### Agent流式输出

Agent用于工具调用场景，与普通的LLM流式输出相比，区分了文本流与工具流。文本流将输出模型的思考过程和最终结果；工具流将输出工具的调用过程，而工具的调用的执行结果是通过监听获取的。

通过如下接口为Agent添加流式输出的回调：

from pangukitsappdev.callback.StreamCallbackHandler import StreamCallbackHandler   
# 以下为两个自定义StreamCallbackHandler示例   
class TextStreamCallBack(StreamCallbackHandler):   
 def \_\_init\_\_(self):   
 # agent文本输出工具类   
 super().\_\_init\_\_()   
 self.stream = ""   
   
 def on\_chat\_model\_start(   
 self,   
 serialized: Dict[str, Any],   
 messages: List[List[BaseMessage]],   
 \*\*kwargs: Any,   
 ) -> Any:   
 self.stream += "[Text stream start]"   
   
 def on\_llm\_new\_token(   
 self,   
 token: str,   
 \*\*kwargs: Any,   
 ) -> Any:   
 self.stream += token   
   
 def on\_llm\_end(   
 self,   
 response: LLMResult,   
 \*\*kwargs: Any,   
 ) -> Any:   
 self.stream += "[Text stream end]"   
   
   
class ToolStreamCallBack(StreamCallbackHandler):   
 def \_\_init\_\_(self):   
 # agent工具输出工具流   
 super().\_\_init\_\_()   
 self.stream = ""   
   
 def on\_chat\_model\_start(   
 self,   
 serialized: Dict[str, Any],   
 messages: List[List[BaseMessage]],   
 \*\*kwargs: Any,   
 ) -> Any:   
 self.stream += "[Tool stream start]"   
   
 def on\_llm\_new\_token(   
 self,   
 token: str,   
 \*\*kwargs: Any,   
 ) -> Any:   
 self.stream += token   
   
 def on\_llm\_end(   
 self,   
 response: LLMResult,   
 \*\*kwargs: Any,   
 ) -> Any:   
 self.stream += "[Tool stream end]"   
text\_stream\_callback = TextStreamCallBack()   
tool\_stream\_callback = ToolStreamCallBack()   
agent.set\_stream\_callback(text\_stream\_callback, tool\_stream\_callback)

StreamCallBack的实现与定义与LLM的回调完全相同。

### Tool Retriever

Agent在实际生产应用中往往涉及到的工具数量较多，如果把所用的工具全部添加至Agent会产生如下问题：

* 占用大量输入token
* 和问题无关的工具太多，影响模型的判断

通过Tool Retriever可以解决这个问题，其原理是在Agent运行前，先从所有可用的工具中选择与问题最相关的工具，再交给Agent去处理。

定义一个Tool Retriever：

from pangukitsappdev.tool.in\_memory\_tool\_provider import InMemoryToolProvider   
from pangukitsappdev.retriever.css\_tool\_retriever import CSSToolRetriever   
   
# 新增InMemoryToolProvider，添加工具集   
in\_memory\_tool\_provider = InMemoryToolProvider()   
tool\_list = [AddTool(), ReverseTool(), ReserveMeeting(), ReserveMeetingRoom()]   
in\_memory\_tool\_provider.add(tool\_list)   
   
# 初始化CSSToolRetriever   
vector\_config = VectorStoreConfig(index\_name="your\_index\_name",   
 verify\_certs=False,   
 text\_key="name",   
 vector\_fields=["name", "description"])   
css\_tool\_retriever = CSSToolRetriever(tool\_provider, vector\_config)

定义一个ToolRetriever包含2个参数，一个ToolProvider，一个向量数据库配置。其中，ToolProvider的作用为根据工具检索的结果组装工具。

上述例子使用了一个简单的InMemoryToolProvider，InMemoryToolProvider的原理为将完整的工具存入内存，再根据工具检索的结果（tool\_id）将其从内存中取出。一般来说，ToolProvider将由用户自定义，后续会有例子说明。

上述例子使用的向量数据库配置指定索引名称，以及使用name和description作为向量化字段，因此工具入库时，会将工具的name和description进行向量化，并在后续的检索中生效。

注意，上述tool\_list中包含的工具在SDK中并不存在，需要替换成实际的工具。

向ToolRetriever中添加工具：

# 添加工具   
css\_tool\_retriever.add\_tools(tool\_list)

工具添加后，会存储在向量库的索引中，并将指定的字段向量化。

从ToolRetriever中查找工具：

# 查找工具   
result = css\_tool\_retriever.search("预订会议室", 2)

返回的result中，包含与预订会议室最相关的工具。搜索支持topK和阈值2个参数，例如上例指定topK=2，则最多返回2个工具。

从ToolRetriever中删除工具：

# 删除工具   
css\_tool\_retriever.remove(["add", "reverse"])

以上为一个比较基础的用法，在实际使用过程中会有更加灵活的场景，我们可以通过自定义ToolProvider的方式解决。

自定义ToolProvider：

# 初始化CSSToolRetriever，使用ToolProviderWithMetadata作为ToolProvider   
vector\_config = VectorStoreConfig(index\_name="your\_index\_name",   
 verify\_certs=False,   
 text\_key="name",   
 vector\_fields=["name", "description", "principle"])   
css\_tool\_retriever = CSSToolRetriever(ToolProviderWithMetadata(), vector\_config)

其中ToolProviderWithMetadata为自定义ToolProvider：

import pickle   
   
class ToolProviderWithMetadata(ToolProvider):   
   
 def provide(self, retrieved\_tools: List[RetrievedTool], query: str) -> List[AbstractTool]:   
 retrieved\_tools = self.do\_some\_filter(retrieved\_tools, query)   
 return [Tool.from\_function(func=pickle.loads(eval(retrieved\_tool.tool\_metadata.get("function"))),   
 name=retrieved\_tool.tool\_id,   
 description=retrieved\_tool.tool\_metadata.get("description"),   
 principle=retrieved\_tool.tool\_metadata.get("principle"),   
 input\_desc=retrieved\_tool.tool\_metadata.get("input\_desc"),   
 output\_desc=retrieved\_tool.tool\_metadata.get("output\_desc"),   
 args\_schema=pickle.loads(eval(retrieved\_tool.tool\_metadata.get("args\_schema"))),   
 return\_type=pickle.loads(eval(retrieved\_tool.tool\_metadata.get("return\_type"))))   
 for retrieved\_tool in retrieved\_tools]   
   
 @staticmethod   
 def do\_some\_filter(retrieved\_tools: List[RetrievedTool], query: str) -> List[RetrievedTool]:   
 print(f"{retrieved\_tools}, {query}")   
 return retrieved\_tools

如上的tool\_provider中，实现了provide接口，可以利用工具检索的返回动态构建出工具列表，同时也可以加一些后处理工作，例如根据黑白名单做工具的过滤。

与上述的tool\_provide呼应，在向tool\_retriever中添加工具时，可以添加任意的元数据，python需要借助pickle将函数或类转换成字节流字符串存入CSS中，用于在tool\_provider中把工具组装出来：

from pydantic import BaseModel, Field   
import pickle   
   
# 构造工具元数据   
class MeetingInfo(BaseModel):   
 id: str = Field(description="会议ID")   
 info: str = Field(description="会议信息")   
   
def list\_meeting(inputs: NoneType) -> List[MeetingInfo]:   
 return [MeetingInfo(id=1, info="金桥2023"), MeetingInfo(id=2, info="金桥203")]   
   
tool\_meta\_data = {   
 "name": "list\_meeting",   
 "description": "查询员工的会议预订状态，返回已经预订的会议和其会议ID",   
 "principle":"请在需要查询员工已预订会议室列表时使用",   
 "input\_desc": "",   
 "output\_desc": "已预订会议室列表",   
 "args\_schema": str(pickle.dumps(None)),   
 "function": str(pickle.dumps(list\_meeting)),   
 "return\_type": str(pickle.dumps(MeetingInfo))   
}   
# 工具管理面添加工具到toolRetriever，这里实际可以添加若干个工具   
css\_tool\_retriever.add\_tools\_from\_metadata([tool\_meta\_data])   
   
# 运行时检索工具，并添加到Agent执行   
tool\_list = css\_tool\_retriever.search("查询会议室预订状态", 1, 0.8)

工具的检索与之前的用法一致。

以下是一个将Tool Retriever集成在Agent中的完整示例：

from pangukitsappdev.skill.conversation\_rewrite\_skill import ConversationRewriteSkill   
   
# 工具集   
toolList = [AddTool(), ReverseTool(), ReserveMeeting(), ReserveMeetingRoom()]   
   
# 新增InMemoryToolProvider，添加工具集   
in\_memory\_tool\_provider = InMemoryToolProvider()   
tool\_list = [AddTool(), ReverseTool(), ReserveMeeting(), ReserveMeetingRoom()]   
in\_memory\_tool\_provider.add(tool\_list)   
   
# 初始化CSSToolRetriever   
vector\_config = VectorStoreConfig(index\_name="your\_index\_name",   
 verify\_certs=False,   
 text\_key="name",   
 vector\_fields=["name", "description"])   
css\_tool\_retriever = CSSToolRetriever(tool\_provider, vector\_config)   
   
   
# 添加工具   
css\_tool\_retriever.add\_tools(tool\_list)   
   
# 添加多轮改写   
css\_tool\_retriever.set\_query\_preprocessor(ConversationRewriteSkill(LLMs.of("pangu")).rewrite)   
   
# 为Agent添加ToolRetriever   
agent = ReactPanguAgent(LLMs.of("yundao"))   
agent.set\_tool\_retriever(css\_tool\_retriever);   
   
# 多轮对话调用   
messages = [ChatMessage(role="user", content="定个2点的会议"),   
 ChatMessage(role="assistant", content="请问您的会议预计何时结束？另外，您是需要预订线上会议还是实体会议室？"),   
 ChatMessage(role="user", content="4点结束，线上会议")]   
print(agent.run(messages))   
   
# 删除工具   
cssToolRetriever.remove([tool.name for tool in tool\_list])

其中有两个变化值得关注：

第一，我们为ToolRetriever添加了一个query\_preprocessor，它的作用为对用户输入的多轮对话进行改写，会将改写后的结果作为工具检索的输入，这里我们使用了系统内置的ConversationRewriteSkill，它的作用为将多轮对话改写为单轮。

第二，我们在创建一个Agent后，调用了set\_tool\_retriever方法为其添加了一个ToolRetriever，这样Agent所使用的工具会根据用户的对话动态的选择。

## 应用示例

### 搜索增强

场景介绍

私有化场景下，大模型需要基于现存的私有数据提供服务。通过外挂知识库（Embedding、向量库）方式提供通用的、标准化的文档问答场景。

工程实现

1. 准备知识库。
2. 获取并安装SDK包。
3. 在配置文件(llm.properties)中配置模型信息。

# 盘古模型IAM 认证信息，根据实际填写   
sdk.llm.pangu.iam.url=   
sdk.llm.pangu.iam.domain=   
sdk.llm.pangu.iam.user=   
sdk.llm.pangu.iam.password=   
sdk.llm.pangu.project=   
   
## 盘古模型信息，根据实际情况填写   
sdk.llm.pangu.url=   
   
## CSS Embedding模型api   
sdk.embedding.css.url=   
sdk.embedding.css.iam.url=   
sdk.embedding.css.iam.domain=   
sdk.embedding.css.iam.user=   
sdk.embedding.css.iam.password=   
sdk.embedding.css.iam.project=   
   
## CSS 向量库   
sdk.memory.css.url=   
sdk.memory.css.user=   
sdk.memory.css.password=

1. 工程实现。

from pangukitsappdev.api.embeddings.factory import Embeddings   
from pangukitsappdev.api.llms.factory import LLMs   
from pangukitsappdev.api.memory.vector.factory import Vectors   
from pangukitsappdev.api.memory.vector.vector\_config import VectorStoreConfig, ServerInfoCss   
from pangukitsappdev.skill.doc.ask import DocAskStuffSkill   
query = "杜甫的诗代表了什么主义诗歌艺术的高峰？"   
# 初始化向量库   
vector\_store\_config = VectorStoreConfig(store\_name="css",   
 index\_name="test-vector-css",   
 embedding=Embeddings.of("css"),   
 server\_info=ServerInfoCss(env\_prefix="sdk.memory.css"))   
vector\_api = Vectors.of("css", vector\_store\_config)   
   
# 检索文档   
docs = vector\_api.similarity\_search(query, 4)   
# 文档问答   
doc\_skill = DocAskStuffSkill(LLMs.of("pangu"))   
print(doc\_skill.execute({"documents": docs, "question": query}))

### 长文本摘要

场景介绍

切割长文本，利用大模型逐步总结。 如对会议/报告/文章等较长内容总结概述。

工程实现

1. 获取并安装SDK包。
2. 在配置文件(llm.properties)中配置模型信息。

# IAM 认证信息，根据实际填写   
sdk.llm.pangu.iam.url=   
sdk.llm.pangu.iam.domail=   
sdk.llm.pangu.iam.user=   
sdk.llm.pangu.iam.password=   
sdk.llm.pangu.iam.project=   
   
## 盘古模型信息，根据实际情况填写   
sdk.llm.pangu.url=

1. 工程实现。

import docx   
from pangukitsappdev.api.llms.factory import LLMs   
from pangukitsappdev.skill.doc.summary import DocSummaryMapReduceSkill   
# 加载原始内容， 需根据文件源自行实现读取步骤   
# 以word文件为例，需安装docx库   
doc = docx.Document(r'报告.docx')   
documents = [d.text for d in doc.paragraphs]   
   
# 对文档进行摘要   
skill = DocSummaryMapReduceSkill(LLMs.of("pangu"))   
summarize = skill.execute\_with\_texts(documents)

## 常见问题

# 应用实践

[5.1 基础问答](#_ZH-CN_TOPIC_0000001727269261" \o " )

[5.2 长文本摘要](#_ZH-CN_TOPIC_0000001679270020" \o " )

[5.3 搜索增强](#_ZH-CN_TOPIC_0000001727109389" \o " )

## 基础问答

应用介绍

基础的大语言模型问答场景。涉及模型问答，流式效果等相关特性。

环境准备

* python3.9 及以上版本
* 安装依赖的组件包， pip install pangu\_kits\_app\_dev\_py gradio
* 盘古大语言模型

开发实现

* 创建配置文件llm.properties， 正确配置iam和pangu配置项。（信息收集可参考 2 准备工作章节）

#   
# Copyright (c) Huawei Technologies Co., Ltd. 2023-2023. All rights reserved.   
#   
   
################################ GENERIC CONFIG ###############################   
   
## If necessary, you can specify the http proxy configuration.   
# sdk.proxy.enabled=true   
# sdk.proxy.url=   
# sdk.proxy.user=   
# sdk.proxy.password=   
   
   
## Generic IAM info. This config is used when the specified IAM is not configured.   
## Either user password authentication or AK/SK authentication.   
#   
sdk.iam.url=   
sdk.iam.domain=   
sdk.iam.user=   
sdk.iam.password=   
sdk.iam.project=   
   
   
## Pangu   
# Examples: https://{endPoint}/v1/{projectId}/deployments/{deploymentId} ;   
#   
sdk.llm.pangu.url=

* 创建代码文件（chat.py），示例如下。

import os   
import sys   
import gradio as gr   
   
from pangukitsappdev.api.llms.llm\_config import LLMParamConfig   
from pangukitsappdev.api.llms.factory import LLMs   
   
# 设置SDK使用的配置文件   
os.environ["SDK\_CONFIG\_PATH"] = "./llm.properties"   
   
# 初始化LLMs   
llm\_api = LLMs.of("pangu")   
   
with gr.Blocks() as demo:   
 chatbot = gr.Chatbot()   
 msg = gr.Textbox()   
 clear = gr.Button("清除")   
   
 def user(user\_message, history):   
 return "", history + [[user\_message, None]]   
   
 def llm(history):   
 history[-1][1] = ""   
   
 # 流式调用大模型   
 tokens = llm\_api.ask(history[-1][0], LLMParamConfig(stream=True))   
 for token in tokens:   
 history[-1][1] += token   
 yield history   
   
 msg.submit(user, [msg, chatbot], [msg, chatbot], queue=False).then(   
 llm, chatbot, chatbot   
 )   
 clear.click(lambda: None, None, chatbot, queue=False)   
   
demo.queue()   
demo.launch()

* 终端命令行下执行 python3 chat.py 运行应用，效果如下。



## 长文本摘要

应用介绍

切割长文本，利用大模型逐步总结，如对会议/报告/文章等总结概述。涉及模型、长文本分割、摘要等相关特性。

环境准备

* python3.9 及以上版本
* 安装依赖的组件包， pip install pangu\_kits\_app\_dev\_py gradio python-docx
* 盘古大语言模型

开发实现

* 创建配置文件llm.properties， 正确配置iam和pangu配置项。（信息收集可参考 2 准备工作章节）

#   
# Copyright (c) Huawei Technologies Co., Ltd. 2023-2023. All rights reserved.   
#   
   
################################ GENERIC CONFIG ###############################   
   
## If necessary, you can specify the http proxy configuration.   
# sdk.proxy.enabled=true   
# sdk.proxy.url=   
# sdk.proxy.user=   
# sdk.proxy.password=   
   
   
## Generic IAM info. This config is used when the specified IAM is not configured.   
## Either user password authentication or AK/SK authentication.   
#   
sdk.iam.url=   
sdk.iam.domain=   
sdk.iam.user=   
sdk.iam.password=   
sdk.iam.project=   
   
   
## Pangu   
# Examples: https://{endPoint}/v1/{projectId}/deployments/{deploymentId} ;   
#   
sdk.llm.pangu.url=

* 创建代码文件（doc\_summary.py），示例如下。

import os   
import gradio as gr   
import docx   
import time   
   
from pangukitsappdev.skill.doc.summary import DocSummaryMapReduceSkill   
from pangukitsappdev.api.llms.factory import LLMs   
   
# 设置SDK使用的配置文件   
os.environ["SDK\_CONFIG\_PATH"] = "./llm.properties"   
   
# 初始化文档问答Skill   
doc\_skill = DocSummaryMapReduceSkill(LLMs.of("pangu"))   
   
# 合并长度过小的段落，不超过maxLength   
def merge\_docs(docs, maxLength, separator):   
 result = []   
 current = []   
 length = 0   
 for doc in docs:   
 if (length + len(doc) > maxLength):   
 if current:   
 result.append(separator.join(current))   
 current = [doc]   
 length = len(doc)   
 else:   
 current.append(doc)   
 length = length + len(doc)   
 if current:   
 result.append(separator.join(current))   
 return result   
   
# 加载文档，支持word和txt文本文件   
def load\_file(name):   
 docs = []   
 if (name.endswith("doc") or name.endswith("docx")):   
 doc = docx.Document(name)   
 for grah in doc.paragraphs:   
 docs.append(grah.text)   
 else:   
 data = ""   
 with open(name, 'r', encoding='utf-8') as f:   
 data = f.read()   
 docs = data.split("\n")   
 return docs   
   
   
def summary(files):   
 sum\_texts = []   
 for file in files:   
 docs = load\_file(file.name)   
 # print(docs)   
 docs\_merge = merge\_docs(docs, 1000, "\n")   
   
 # 大模型对文档做摘要   
 sum\_texts.append(doc\_skill.execute\_with\_texts(docs\_merge))   
   
 # 设置延时，避免访问太频繁   
 time.sleep(10)   
   
 return sum\_texts[0] if len(sum\_texts) == 1 else doc\_skill.execute\_with\_texts(sum\_texts)   
   
def upload\_file(files):   
 file\_paths = [file.name for file in files]   
 return file\_paths   
   
with gr.Blocks() as demo:   
 upload\_bt = gr.UploadButton("选择文档", file\_types=["text"], file\_count="multiple")   
 file\_output = gr.Files()   
 upload\_bt.upload(upload\_file, upload\_bt, file\_output)   
   
 greet\_btn = gr.Button("生成摘要")   
 output = gr.Textbox(label="输出")   
   
 greet\_btn.click(fn=summary, inputs=file\_output, outputs=output, api\_name="summary")   
   
   
demo.launch()

* 终端命令行下执行 python3 doc\_summary.py 运行应用，效果如下。



## 搜索增强

应用介绍

通过外挂知识库（向量库存储）方式提供通用的、标准化的文档问答。涉及模型、Embedding、向量存储等相关特性。

环境准备

* python3.9 及以上版本
* 安装依赖的组件包， pip install pangu\_kits\_app\_dev\_py gradio python-docx
* 盘古大语言模型、CSS Embedding模型、CSS向量库

开发实现

* 创建配置文件llm.properties， 正确配置iam、pangu、css embedding、css向量库配置项。（信息收集可参考 2 准备工作’章节）

#   
# Copyright (c) Huawei Technologies Co., Ltd. 2023-2023. All rights reserved.   
#   
   
################################ GENERIC CONFIG ###############################   
   
## If necessary, you can specify the http proxy configuration.   
# sdk.proxy.enabled=true   
# sdk.proxy.url=   
# sdk.proxy.user=   
# sdk.proxy.password=   
   
   
## Generic IAM info. This config is used when the specified IAM is not configured.   
## Either user password authentication or AK/SK authentication.   
#   
sdk.iam.url=   
sdk.iam.domain=   
sdk.iam.user=   
sdk.iam.password=   
sdk.iam.project=   
   
   
## Pangu   
# Examples: https://{endPoint}/v1/{projectId}/deployments/{deploymentId} ;   
#   
sdk.llm.pangu.url=   
   
   
## CSS embedding   
# Examples: https://{endPoint}/v1/{projectId}/applications/{appId}/{modelVersion}/embedding ;   
#   
sdk.embedding.css.url=   
## If necessary, you can specify the IAM configuration.   
#sdk.embedding.css.iam.url=   
#sdk.embedding.css.iam.domain=   
#sdk.embedding.css.iam.user=   
#sdk.embedding.css.iam.password=   
#sdk.embedding.css.iam.project=   
# sdk.embedding.css.iam.disabled=   
## If necessary, you can specify the proxy status.   
#sdk.embedding.css.proxy.enabled=   
   
   
## CSS or ES   
# Examples: http://127.0.0.1:9200,http://127.0.0.2:9200 ;   
#   
sdk.memory.css.url=   
sdk.memory.css.user=   
sdk.memory.css.password=

* 创建代码文件（doc\_ask.py），示例如下。

import os   
import gradio as gr   
import docx   
import time   
   
from pangukitsappdev.skill.doc.ask import DocAskStuffSkill   
from pangukitsappdev.api.llms.factory import LLMs   
from pangukitsappdev.api.memory.vector.vector\_config import VectorStoreConfig   
from pangukitsappdev.api.memory.vector.vector\_config import ServerInfoCss   
from pangukitsappdev.api.embeddings.factory import Embeddings   
from pangukitsappdev.api.memory.vector.factory import Vectors   
from pangukitsappdev.vectorstores.bulk\_data import BulkData   
   
# 设置SDK使用的配置文件   
os.environ["SDK\_CONFIG\_PATH"] = "./llm.properties"   
   
vector\_index\_name = "test-vector-css-demo"   
# 初始化向量库   
vector\_store\_config = VectorStoreConfig(store\_name="css",   
 index\_name=vector\_index\_name,   
 embedding=Embeddings.of("css"),   
 text\_key="name",   
 vector\_fields=["description"],   
 distance\_strategy="inner\_product",   
 server\_info=ServerInfoCss(env\_prefix="sdk.memory.css"))   
vector\_api = Vectors.of("css", vector\_store\_config)   
   
   
# 初始化文档问答Skill   
doc\_skill = DocAskStuffSkill(LLMs.of("pangu"))   
   
# 加载文档内容，支持word和txt文本文件   
def load\_file(name):   
 docs = []   
 if (name.endswith("doc") or name.endswith("docx")):   
 doc = docx.Document(name)   
 for grah in doc.paragraphs:   
 docs.append(grah.text)   
 else:   
 data = ""   
 with open(name, 'r', encoding='utf-8') as f:   
 data = f.read()   
 docs = data.split("\n")   
 return docs   
   
# 保存知识库； 如果已有知识库，可忽略   
def save(files):   
 sum\_texts = []   
 for file in files:   
 docs = load\_file(file.name)   
 # print(docs)   
 # 向量入库   
 bulk\_list = [BulkData(id=str(i), data={"name": doc, "description": doc}) for i,doc in enumerate(docs)]   
 vector\_api.add\_docs(bulk\_list)   
   
 # 设置延时，避免访问太频繁   
 time.sleep(1)   
   
 return vector\_index\_name   
   
def upload\_file(files):   
 file\_paths = [file.name for file in files]   
 return file\_paths   
   
def user(user\_message, history):   
 return "", history + [[user\_message, None]]   
   
# 基于已有知识库，问答   
def doc\_ask(history):   
 history[-1][1] = ""   
   
 query = history[-1][0]   
 # 先检索相似文档   
 docs = vector\_api.similarity\_search(query, top\_k=4)   
   
 # print(docs)   
 # 基于文档问答   
 result = doc\_skill.execute({"documents": docs, "question": query})   
   
 # 模拟打印机效果   
 for char in result:   
 history[-1][1] += char   
 yield history   
   
with gr.Blocks() as demo:   
 with gr.Tab("问答"):   
 # 问答   
 chatbot = gr.Chatbot()   
 msg = gr.Textbox()   
 clear = gr.Button("清除")   
   
 msg.submit(user, [msg, chatbot], [msg, chatbot], queue=False).then(   
 doc\_ask, chatbot, chatbot   
 )   
 clear.click(lambda: None, None, chatbot, queue=False)   
   
 with gr.Tab("知识库"):   
 # 上传知识库   
 file\_output = gr.Files()   
 upload\_bt = gr.UploadButton("上传文档", file\_types=["text"], file\_count="multiple")   
 upload\_bt.upload(upload\_file, upload\_bt, file\_output)   
 # 保存   
 greet\_btn = gr.Button("保存知识库")   
 output = gr.Textbox(label="向量库索引")   
 greet\_btn.click(fn=save, inputs=file\_output, outputs=output, api\_name="save")   
   
demo.queue()   
demo.launch()

* 终端命令行下执行 python3 doc\_ask.py 运行应用，效果如下。



# 常见问题

常见问题1

常见问题2

# 版本变更记录

盘古大模型 Java SDK版本变更记录如下。

盘古大模型 Java SDK变更记录

| 版本 | 变更类型 | 说明 |
| --- | --- | --- |
| 1.4.0 | - | 第一次正式发布。 |