

Chatopera 云服务

用户使用文档

<https://bot.chatopera.com>



Copyright © (2018-2023) 北京华夏春松科技有限公司版权所有

Table of Contents

服务概述 服务概述

入门教程 入门教程

<1/5> 创建机器人 <1/5> 创建机器人

<2/5> 添加对话语料 <2/5> 添加对话语料

<3/5> 添加脚本和函数 <3/5> 添加脚本和函数

<4/5> 添加意图对话 <4/5> 添加意图对话

<5/5> 查看使用情况 <5/5> 查看使用情况

使用指南 使用指南

账号 账号

注册使用 Chatopera 云服务 注册使用 Chatopera 云服务

词典 词典

系统词典 系统词典

词汇表词典 词汇表词典

正则表达式词典 正则表达式词典

知识库 知识库

知识库问答对维护 知识库问答对维护

知识库问答及调优测试 知识库问答及调优测试

知识库问答对导入和导出 知识库问答对导入和导出

自定义词典增强知识库 自定义词典增强知识库

知识库热门问题查看和导出 知识库热门问题查看和导出

使用函数获得动态答案 使用函数获得动态答案

话术助手快速检索话术 话术助手快速检索话术

意图识别 意图识别

创建和训练意图识别模型 创建和训练意图识别模型

发布上线意图识别生产版本 发布上线意图识别生产版本

多轮对话 多轮对话

多轮对话设计器安装 多轮对话设计器安装

使用通配符匹配器 使用通配符匹配器

使用模糊匹配器 使用模糊匹配器

使用意图匹配器 使用意图匹配器

设置回复 设置回复

使用上下轮钩子 使用上下轮钩子

使用函数 使用函数

配置环境变量	配置环境变量
管理对话状态	管理对话状态
设置定时任务	设置定时任务
语音识别	语音识别
调用语音识别服务	调用语音识别服务
对话调优	对话调优
对话测试	对话测试
对话历史分析	对话历史分析
CLI 连接多轮对话	CLI 连接多轮对话
渠道	渠道
H5 聊天控件	H5 聊天控件
飞书	飞书
春松客服	春松客服
系统集成	系统集成
SDK	SDK
CLI 安装和配置	CLI 安装和配置
CLI 导入和导出对话语料	CLI 导入和导出对话语料
参考手册	参考手册
术语	术语
通配符匹配器	通配符匹配器
内置函数库	内置函数库
basics	basics
message	message
user	user
maestro	maestro
3rd-party	3rd-party
函数返回值	函数返回值
命令行界面 (CLI)	命令行界面 (CLI)
SDK	SDK
Chatopera 类	Chatopera 类
构造函数	构造函数
接口规范	接口规范
机器人管理	机器人管理
Chatbot 类	Chatbot 类
构造函数	构造函数
接口规范	接口规范
机器人基本管理	机器人基本管理

- 对话检索 对话检索
- 词典管理 词典管理
- 知识库管理 知识库管理
- 用户和对话历史 用户和对话历史
- 语音识别 语音识别
- 常见问题 常见问题
- 计费及保障 计费及保障
- 计费及发票 计费及发票
- 私有部署 私有部署
- 服务水平协议 服务水平协议
- 服务条款 服务条款
- 背景知识 背景知识
 - 深度学习 深度学习
 - 词典 词典
 - 知识库 知识库
 - 意图识别 意图识别
 - 对话脚本 对话脚本
 - 模糊匹配器 模糊匹配器
 - 意图匹配器 意图匹配器
 - 多轮对话的检索 多轮对话的检索
 - 视频教程 视频教程
- 附录 附录
 - 词性标注 词性标注

服务概述

Chatopera 机器人平台

用聊天机器人做市场营销、客户服务和自动化流程？如何实现多轮对话？

Chatopera 机器人平台帮助企业和开发者使用低代码方式定制智能对话机器人。

Chatopera 云服务是Chatopera 机器人平台的公有云实例，Chatopera 云服务尤其对中小型企业非常友好，按需消费，一站式上线智能对话机器人。

Chatopera 云服务使用按量计费，提供多语言 SDK 集成；同时 Chatopera 可为企业私有部署 Chatopera 机器人平台。

Chatopera 机器人平台目前支持语言：

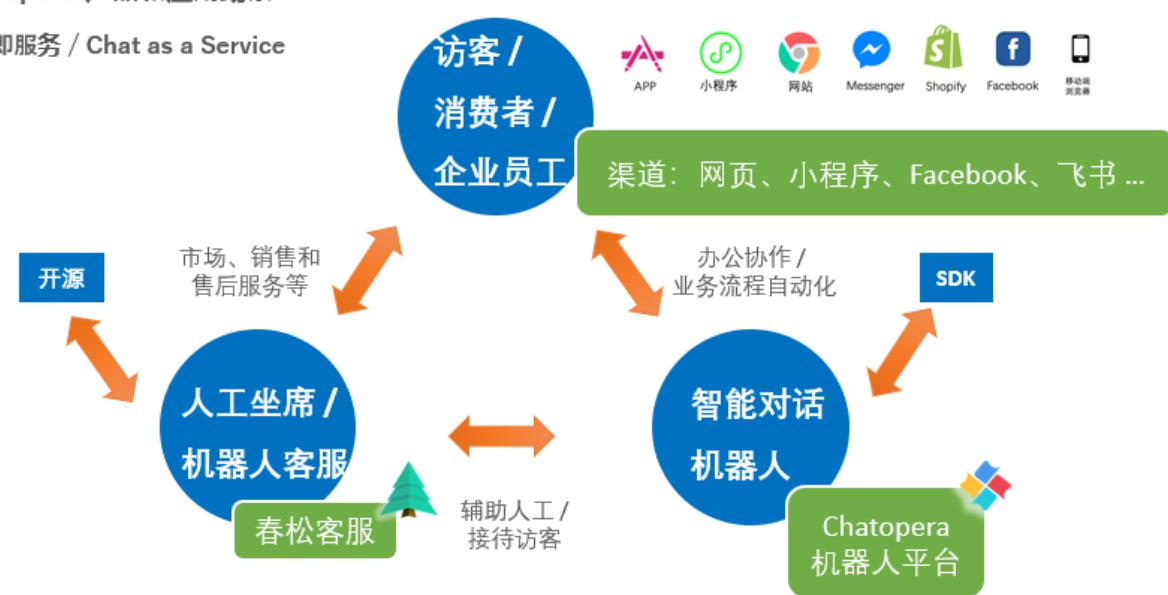
- 中文简体 (zh_CN)
- 中文繁体 (zh_TW)
- 英语 (en_US)
- 日语 (ja)
- 泰语 (th)

应用场景

智能对话机器人在 Chatopera 的“聊天即服务”整体系统中，占据重要位置。

Chatopera 产品和应用场景

聊天即服务 / Chat as a Service



Chatopera 融合了不同形式的实现聊天机器人的方法，支持定制多轮对话，从而满足企业各种需求，管理对话机器人对话内容。此外，Chatopera 机器人平台还有聊天历史管理、聚类分析和语音识别等模块，这些服务都是紧紧围绕智能对话机器人的上线展开的。

下一步

- 新手任务：使用入门教程一步步实现智能对话机器人

入门教程

入门教程帮助你系统性的了解 Chatopera 云服务。在系统性的了解后，才能快速的利用 Chatopera 云服务定制聊天机器人。

准备工作

确保你已经注册了 [Chatopera 云服务账号](#)。接下来就发布第一个聊天机器人吧！

新手任务

入门教程包括 5 个简单的新手任务，你只需要一步步的按照顺序操作即可：点击、复制、粘贴。完成入门教程需要 ~20 分钟。

[**<1/5> 创建机器人**](#)

[**<2/5> 添加对话语料**](#)

[**<3/5> 添加脚本和函数**](#)

[**<4/5> 添加意图对话**](#)

[**<5/5> 查看使用情况**](#)

<1/5> 创建机器人

<< 上一步: [入门教程首页](#) | ⏳ 阅读本节内容大约需要 3 mins

登录

登录 Chatopera 云服务管理控制台: <https://bot.chatopera.com/dashboard>。

创建机器人

在工具条菜单中, 点击【创建机器人】。填入表单:

表单项	值
机器人名称	阿Q

其它信息默认, 点击【确认】。

The screenshot shows the Chatopera control panel interface. At the top, there's a navigation bar with the Chatopera logo and the text "重新定义聊天机器人". Below it, a sub-navigation bar has "创建机器人" (Create Robot) highlighted with a red circle containing the number 1. On the right of this bar is a link to "入门教程" (Getting Started Guide). The main content area is titled "新建机器人" (New Robot). It contains a form with the following fields:

- "机器人名称" (Robot Name) field with the value "阿Q" highlighted with a red circle containing the number 2.
- "机器人头像" (Robot Avatar) section with a "上传图标" (Upload Icon) button and a placeholder text: "请上传1:1的图片,不上传将使用默认图标". To the right is a "头像预览" (Avatar Preview) box showing a blue robot head icon.

On the left side of the main content area, there's a sidebar with several icons and text labels, including "OpenTEKr春人" (OpenTEKr Spring Person), "Chatopera 官网联系商务" (Chatopera Official Website Business Contact), "Chatopera 文档中心助手" (Chatopera Document Center Assistant), and "OhMyCskefu" (OhMyCskefu).

创建成功后, 会自动跳转到 [阿Q](#) 的概况页。



阿Q

创建机器人



机器人对话内容概况

知识库问答对条数

0

知识库扩展问条数

0

自定义词典数

0

多轮对话话题数

1

意图识别意图数

0

系统词典数

5

自定义词典词条

0

今天

过去24小时

过去30天

过去3个月

2023-05-02 08:44 - 2023-05-03 08:44



发布聊天机器人

步骤如下，进入系统集成，点击【H5 聊天控件[^h5-channel] - 接入设置】。



阿Q

创建机器人



系统集成

1

企业应用

企业应用是已经集成了 Chatopera 机器人平台的企业软件，用户只需要配置即可使用。

H5 聊天控件	春松客服 chatopera.com	Facebook Messenger
接入设置 ② 使用说明	使用说明 私有部署 定制开发	演示示例 私有部署 定制开发

找到【文字链代码】，复制该 URL。

H5 聊天控件 [回到系统集成](#)[基本设置](#)[邀请设置](#)[客服信息](#)

接入代码

请将以下代码添加到你的网站 HTML 源代码中，放在<head></head>标签之间。

```
<script defer="true" src="https://h5.chatopera.com/im/0QAxgY.html"></script>
```

[复制](#)

文字链代码

请将以下代码添加到你的网站链接代码上，自由定义链接的内容形式，或将以下 URL 生成二维码。

<https://h5.chatopera.com/im/text/0QAxgY.html> ①

[复制](#)

访客对话入口样式

1、访客入口样式



将【文字链代码】URL 在你的浏览器地址栏打开，然后就可以作为访客体验聊天机器人了：发送【你好】，机器人有回应，这是一个默认的对话技能。



电脑客户端网页



手机客户端网页

恭喜你完成本节任务！



<< 上一步: [入门教程首页](#) | >> 下一步: [*<2/5> 添加对话语料*](#)

[^h5-channel]: H5 聊天控件是一个面向访客的发布渠道, 访客可以是你自己、你的同事或消费者。H5 聊天控件是聊天机器人快速的提供服务的一种方式。

<2/5> 添加对话语料

<< 上一步: <1/5> 创建机器人 | 阅读本节内容大约需要 5 mins

现在我们添加对话语料, 让阿Q可以回答更多的问题。先从知识库开始: 设定问答对。

管理知识库问答对

进入阿Q的知识库管理页。

- 通过维护问答对的形式提供检索服务, 使用标准问、相似问和词典管理等。
- 在测试对话页面, 调试知识库, 通过 SDK 接口等形式访问知识库, 更多 [详细介绍](#)。
- 使用 Chatopera CLI 进行自动化知识库导入、导出操作, 参考 [详细介绍](#)。
- [话术助手](#)桌面软件支持查询知识库, 获得智能建议回复。

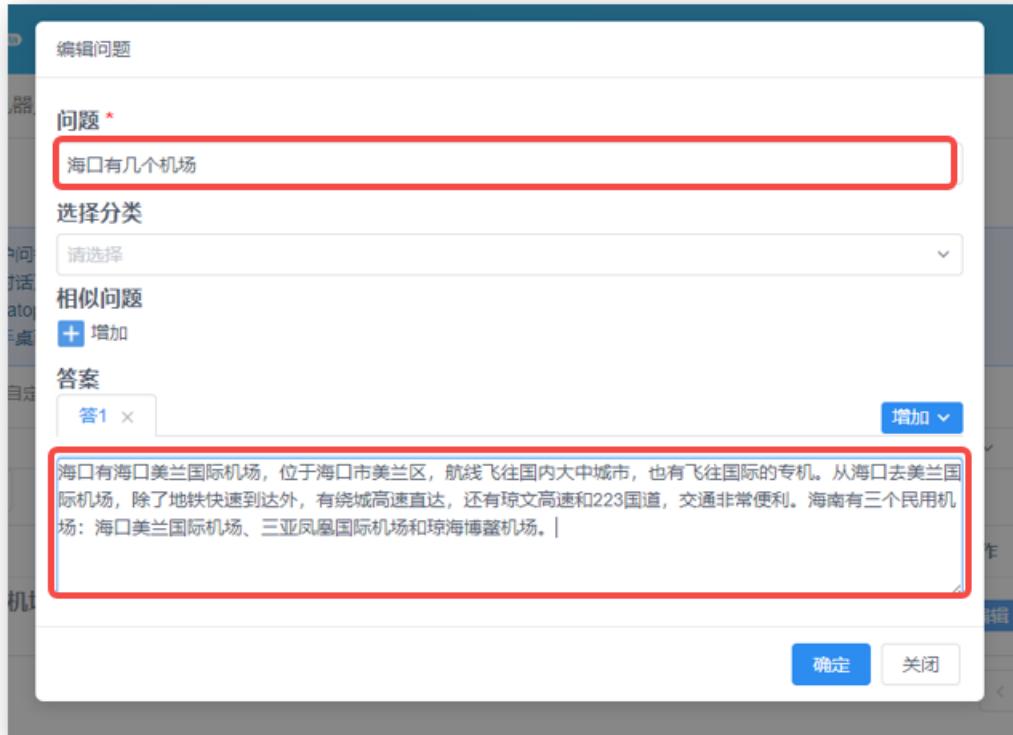
知识库已同步自定义词典信息 ✓

+	问题	选择分类: 请选择	搜索	批量导入
分类管理				

添加知识库问答对

在知识库菜单的左侧点击加号【+】，这时弹出创建问答对的表单。复制粘贴下面的问题和答案:

海口有几个机场
海口有海口美兰国际机场, 位于海口市美兰区, 航线飞往国内大中城市, 也有飞往国际的专机。从海口去美兰国际机场, 除了地铁快速到达外, 有绕城高速直达, 还有琼文高速和 223国道 , 交通非常便利。海南有三个民用机场: 海口美兰国际机场、三亚凤凰国际机场和琼海博鳌机场。



然后点击【确定】，这样就为阿Q增加了一条“知识”，它可以回答这个问题。

测试知识库问答对

进入阿Q的测试对话页，并在下图 1 位置输入：海口有几个机场。点击【发送】。阿Q 的回复类似下面。

对话

ME

机器人

您是否想问以下问题

1.00 海口有几个机场

发送

调试信息

阈值: 0

滑动以测试不同阈值的结果，阈值范围 [0, 1]。

检索接口返回值，SDK 调用得到的 JSON 结果:

```
[{"id": "AYffTaquLXiNiuvu_uXR", "score": 1, "post": "海口有几个机场", "replies": [{"rtype": "plain", "enabled": true, "content": "海口有海口美兰国际机场，位于海口市美兰区，航线飞往国内大中城市，也有飞往国际的专机。从海口去美兰国际机场，除了地铁快速到达外，有绕城高速直达，还有琼文高速和223国道，交通非常便利。海南有三个民用机场：海口美兰国际机场、三亚凤凰国际机场和琼海博鳌机场。"}], "categories": []}]
```

验证通过。我们继续介绍另外一个强大的对话管理模块: 多轮对话。

管理多轮对话脚本

安装多轮对话设计器

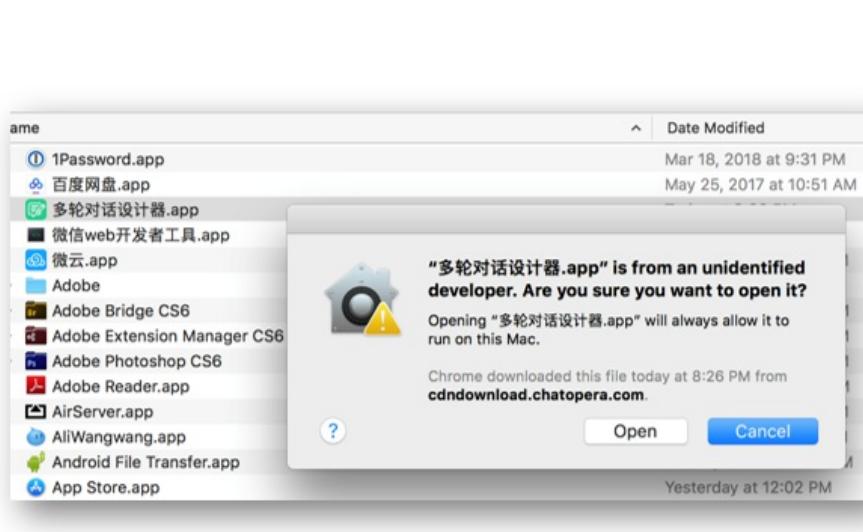
版本	下载地址
v2.5.7	macOS Windows

提示:

- 点击下载地址未能开始下载时, 将下载地址拖拽到新的 Tab 页, 启动下载
- 下载后, 得到应用安装包, 双击后根据提示进行安装[^install-cde](#)

添加机器人

安装完成后, 在电脑应用中心, 启动【多轮对话设计器】。



Mac OSX or higher



Windows

在多轮对话设计器一级菜单中, 点击【添加】, 此处需要 [阿Q](#) 的 Client Id 和 Secret。

聊天机器人列表

名字	语言	Client Id	机器人平台	操作
Recruiter	zh_CN	60a		发布 导入 版
Test007	zh_CN	60d	机器人平台 <input type="text" value="https://bot.chatopera.com"/>	发布 导入 版
localhost机器人	zh_CN	60c	clientID (必填) <input type="text" value="请输入clientID"/>	发布 导入 版
dev1078	zh_CN	611	secret (必填) <input type="text" value="请输入secret"/>	发布 导入 版
春松机器人	zh_CN	613	< 还没有聊天机器人? 去Chatopera云服务	取消 确认

在浏览器中，进入 `阿Q` 的设置页。

设置

基本信息

Client Id: 6451bd4f6baba000133bdadc 2

Secret: 3

* 机器人名称: 阿Q

* 语言: zh_CN

是否开启繁体中文自动翻译: 关闭

描述: 请描述机器人

会话模型

欢迎语: 你好！我是机器人客服。 11/100

复制 `Client Id` 和 `Secret` 到多轮对话设计器【添加表单】，点击【确认】^{help1}。

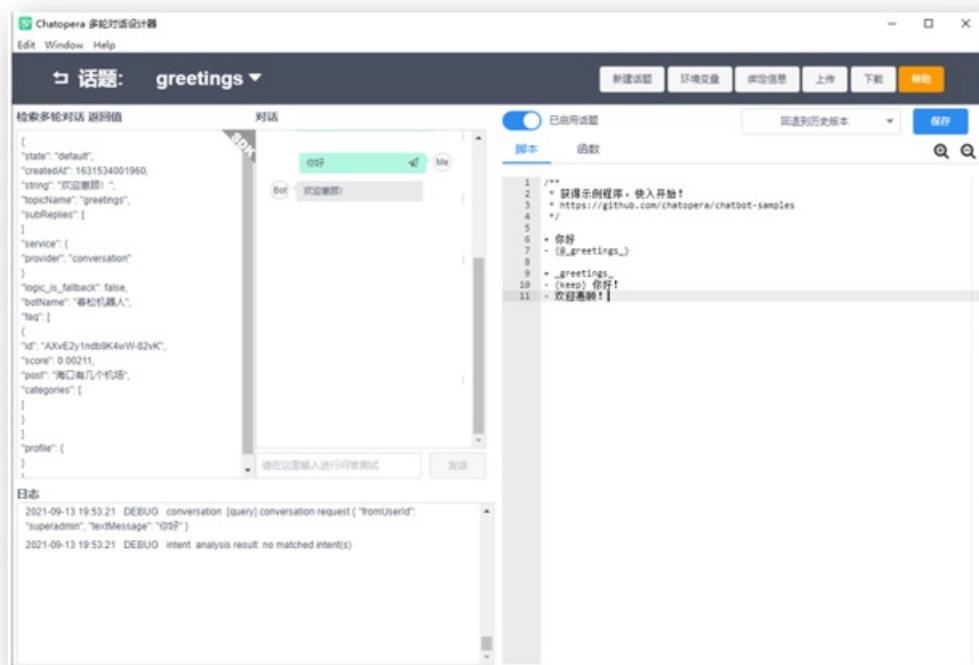
回到列表后，找到 `阿Q`，点击右侧操作中的【打开】，进入阿Q的话题列表页。这时，可以看到默认的话题: `greetings`。

阿Q (zh_CN)

话题名称	是否启用
greetings	已启用话题

和机器人进行对话

点击 **greetings** 话题的【编辑】按钮，进入 阿Q 的 **greetings** 脚本编辑窗口。



在脚本编辑窗口中间，有对话框，在文本发送区域，输入 **你好**，然后点击 **【发送】**。

当你看到机器人回复了 **你好！**，那么本步骤就完成了！恭喜你完成本节任务！



<< 上一步: <1/5> 创建机器人 | >> 下一步: <3/5> 添加脚本和函数

可能遇到的问题

信息不匹配，请确认机器人信息

提示如下：

名字	语言	Client Id	机器人平台	操作
Recruiter	zh_CN	60a	添加聊天机器人	<button>发布</button> <button>导入</button> <button>版本管理</button>
Test007	zh_CN	60d	机器人平台 clientID (必填) secret (必填)	<button>发布</button> <button>导入</button> <button>版本管理</button>
localhost机器人	zh_CN	60c	60c05dcda375e0001b660696	<button>发布</button> <button>导入</button> <button>版本管理</button>
dev1078	zh_CN	611	<button>发布</button> <button>导入</button> <button>版本管理</button>
春松机器人	zh_CN	613	<还没有聊天机器人? 去Chatopera云服务	<button>取消</button> <button>确认</button> <button>发布</button> <button>导入</button> <button>版本管理</button>

如果经过验证，你填写的信息没有错误，那么可能是电脑的时间日期与互联网标准时间之间有很大误差，需要先在操作系统上同步互联网时间，以下是 Windows 上同步互联网时间的方法，你也可以手动设置，其它操作系统，都有类似的操作。



在 Windows 上同步互联网时间

<3/5> 添加脚本和函数

<< 上一步: <2/5> 添加对话语料 | 阅读本节内容大约需要 5 mins

接下来, 我们为 阿Q 添加个性化的问候语。

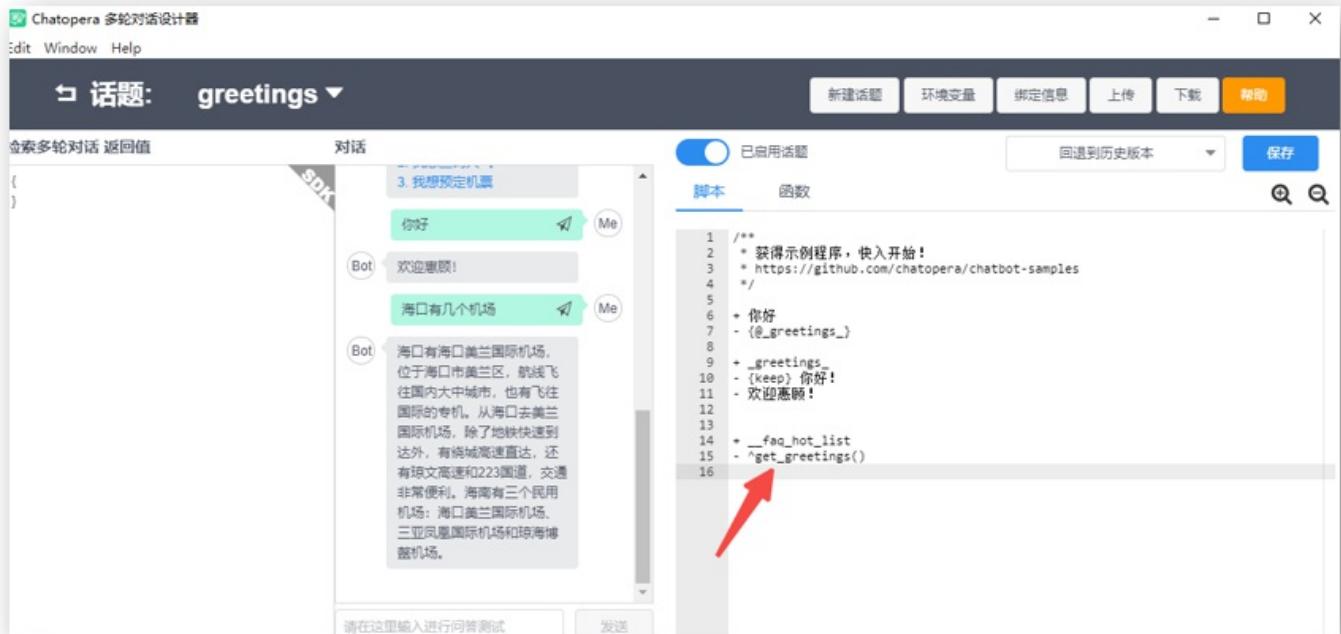
添加脚本

在多轮对话设计器中, 打开 阿Q 话题 greetings 脚本编辑窗口。

追加下面的内容:

```
+ __faq_hot_list
- {keep} ^get_greetings()
```

点击保存。

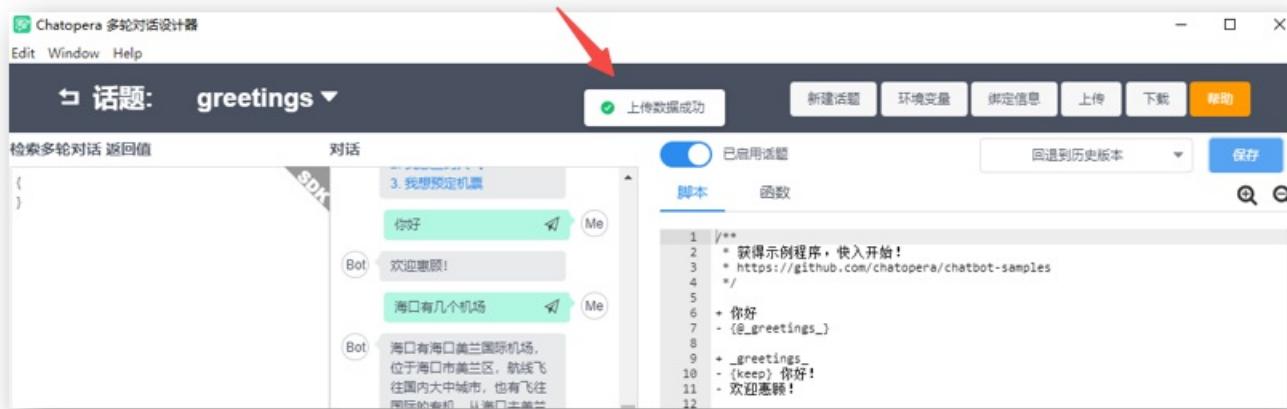


添加函数

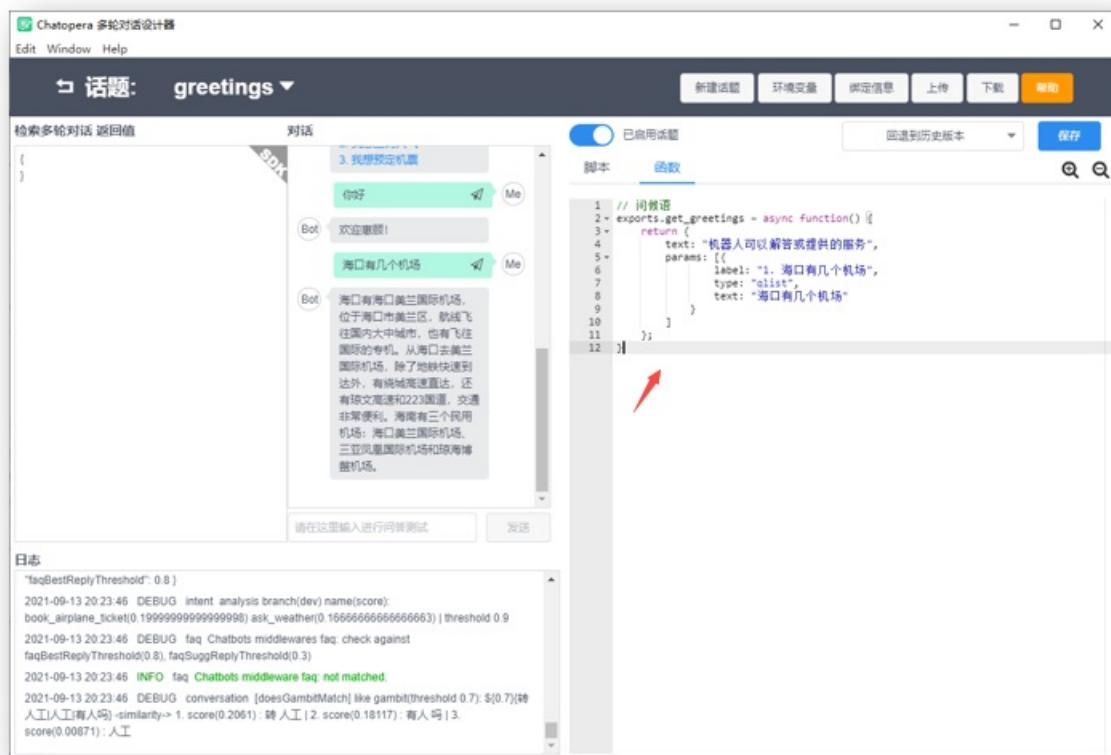
点击【脚本】旁边的【函数】, 进入函数编辑窗口, 追加下面的内容[^function-js]:

```
// 问候语中关联常见问题
exports.get_greetings = async function() {
  return {
    text: "机器人可以解答或提供的服务",
    params: [
      {
        label: "1. 海口有几个机场",
        type: "qlist",
        text: "海口有几个机场"
      }
    ]
  };
}
```

点击【保存】，此时，得到提示信息: 上传数据成功。



添加后，函数编辑区域看起来是这样。



测试对话

在聊天对话框，输入：

faq_hot_list

机器人回复：

机器人可以解答或提供的服务

1. 海口有几个机场

The screenshot shows the Chatopera Multi-turn Conversation Designer interface. At the top, there's a navigation bar with 'Edit', 'Window', 'Help', and tabs for '话题' (Topic) set to 'greetings'. Below the navigation is a toolbar with buttons for '新建话题' (New Topic), '环境变量' (Environment Variables), '绑定信息' (Bind Information), '上传' (Upload), '下载' (Download), and '帮助' (Help). A large central area contains a conversation log and a script editor.

对话 (Conversation):

- Bot: _faq_hot_list
- Me: 机器人可以解答或提供的服务
- Bot: 1. 海口有几个机场

脚本 (Script):

```
1 /**
2 * 获得示例程序，快入开始！
3 * https://github.com/chatopera/chatbot-samples
4 */
5
6 + 你好
7 - {@.greetings...}
8
9 + _greetings_
10 - {keep} 你好！
11 - 欢迎惠顾！
12
13
14 + _faq_hot_list
15 - "get_greetings()"
```

当你看到了这样的回答，那么本步骤就完成了！恭喜你完成本节任务！



<< 上一步: <2/5> 添加对话语料 | >> 下一步: <4/5> 添加意图对话

[^function-js]: 这是一段 JavaScript 代码，JavaScript 是非常容易掌握的编程语言。

<4/5> 添加意图对话

<< 上一步: <3/5> 添加脚本和函数 | 阅读本节内容大约需要 7 mins

如何让阿Q可以引导访客完成预约机票呢? 这需要使用意图识别模块和多轮对话设计器。

引用系统词典

阿Q需要识别访客的输入文本中包含的地名和时间信息, 比如出发城市、到达城市和航班时间。

在浏览器中, 进入阿Q的词典页, 点击【引用系统词典】。

词典管理

- 词典包括自定义词典和系统词典。自定义词典是开发者自己创建的词典; 系统词典是平台提供给开发者使用的常用词典。
- 词典主要用于意图中槽位的识别和填充, 不同机器人的自定义词典不能共享。
- 使用 Chatopera CLI 进行自动化词典引用、导入、导出和同步操作, 参考 [详细介绍](#)。

系统词典

以下是已经引用的系统词典列表, 引用的系统词典用于意图识别

词典标识名	词典中文名	发布者	词典示例	更新时间	创建时间
暂无数据					

找到 @TIME 和 @LOC, 点击【引用】, 如下图所示。

引用系统词典

- 系统词典可以直接引用, 无需配置词条。
- 被引用了的系统词典可以在槽位配置时使用。

词典标识名	词典中文名	词典示例	操作
@ANY	任意字符串	任意字符串	引用
@PER	人名	郭德纲;于谦	引用
@TIME	时间	今天;下午一时	取消引用
@LOC	地名	北京市;东京	取消引用
@ORG	组织机构	北京华夏春松科技有限公司	引用

共 5 条 < > 1 >

创建意图

在浏览器中，进入 阿Q 的意图管理页面。

The screenshot shows the Chatopera AI Platform interface. At the top, there is a blue header bar with the Chatopera logo and the text "重新定义聊天机器人". Below the header, the navigation bar includes "阿Q" and "创建机器人". On the right side of the navigation bar, there are several icons: a profile picture, a gear, a graduation cap, a link, a wrench, a microphone, a speaker, and a gear. A red circle with the number "1" is positioned above the gear icon. The main content area is titled "意图" (Intent). It contains a list of bullet points about intents:

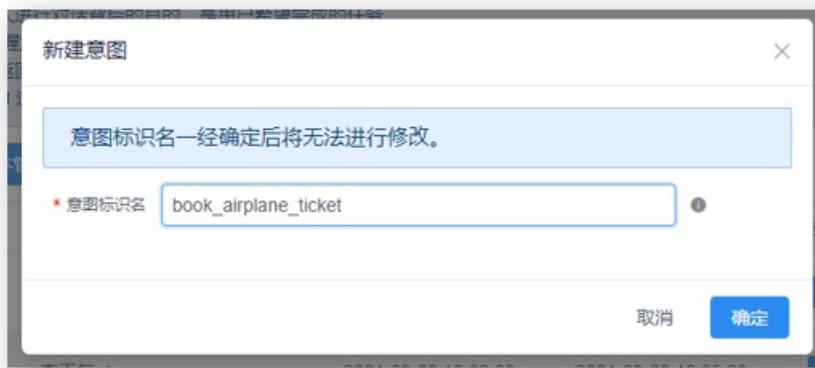
- 意图是用户与机器人进行对话背后的目的，是用户希望完成的任务。
- 查看 快速开始，掌握意图识别模块上线聊天机器人过程。
- 意图对话的结果是返回意图和槽位信息，开发者通过 意图识别匹配器集成到其它系统进一步完成任务。
- 使用 Chatopera CLI 进行自动化意图说法、槽位的导入、导出和训练操作，参考 详细介绍。

Below the list, there are three buttons: "新建意图" (Create New Intent), "版本管理" (Version Management), and a status message "意图识别模块调试分支已训练" with a green checkmark. The main table has columns: "意图标识名" (Intent Identifier Name), "意图中文名" (Intent Chinese Name), "更新时间" (Last Updated Time), "创建时间" (Creation Time), and "操作" (Operations). The table displays the message "暂无数据" (No Data Available). At the bottom right of the table, it says "共 0 条" (0 items).

点击【新建意图】，复制粘贴以下内容到表单中。

book_airplane_ticket

点击【确定】。现在，就有了一个意图，接下来为这个意图添加训练数据：槽位和说法。



- 槽位：该意图中的关键信息。
- 说法：表明意图的开场白。

添加槽位

在 `book_airplane_ticket` 的操作中，点击【编辑】，进入意图识别编辑页面。

意图

- 意图是用户与机器人进行对话背后的目的，是用户希望完成的任务。
- 查看 快速开始，掌握意图识别模块上线聊天机器人过程。
- 意图对话的结果是返回意图和槽位信息，开发者通过 意图识别匹配器集成到其它系统进一步完成任务。
- 使用 Chatopera CLI 进行自动化意图说法、槽位的导入、导出和训练操作，参考 [详细介绍](#)。

新建意图 版本管理 意图识别模块调试分支已训练 ✓

意图标识名	意图中文名	更新时间	创建时间	操作
book_airplane_ticket		2023-05-03 12:01:53	2023-05-03 12:01:53	1 共 1 条

我们开始添加槽位信息，槽位编辑面板在【用户说法】的下面，按照如下信息逐个【添加】：

槽位名称	词典（下拉选择）	必填	追问
fromPlace	@LOC	是	您从哪个城市或机场出发？
date	@TIME	是	您的计划出发日期是什么时候？
destPlace	@LOC	是	您要去的目的城市或机场是哪里？

添加完成后，看起来是这样的。

槽位

槽位名称 例如：CityName 词典 请选择 必填

追问 例如：请问是哪个城市

添加

槽位名称	词典	必填	追问	操作
fromPlace	@LOC	<input checked="" type="checkbox"/>	您从哪个城市或机场出发？	
date	@TIME	<input checked="" type="checkbox"/>	您的计划出发日期是什么时候？	
destPlace	@LOC	<input checked="" type="checkbox"/>	您要去的目的城市或机场是哪里？	

添加说法

接下来，我们为预约机票添加一些说法。复制下面的内容；粘贴到【用户说法】中；点击【添加】。

预约机票
预定飞机票
我想预约机票
我要预约从{fromPlace}出发的机票
帮我预约{date}的机票

用户说法 ①

预约机票 ①
预定飞机票
我想预约机票
我要预约从{fromPlace}出发的机票
帮我预约{date}的机票

添加 ②

使用 Enter 换行，编写多个说法，同时添加

添加完成后，看起来是这样的。

用户说法 ①

输入本意图可能的说法，用{}引用槽位，例如：
我想预订{CityName}的机票

添加

使用 Enter 换行，编写多个说法，同时添加

用户说法	操作
预约机票	删除
预定飞机票	删除
我想预约机票	删除
我要预约从{fromPlace}出发的机票	删除
帮我预约{date}的机票	删除

训练意图识别模型

滚动到槽位表格下面，点击【保存】。

槽位名称	词典
date	@TIME
fromPlace	@LOC
destPlace	@LOC

1 

保存 取消

在保存后，会提示进行模型的训练，大约几秒钟后，提示训练成功，可进行测试。

测试意图识别

在浏览器中，进入[阿Q](#)的测试对话页。

在测试对话页面，选择【意图识别】，然后在聊天窗口中，发送：

我想预约机票

这时候阿Q会回答： 您从哪个城市或机场出发？ 或 您的计划出发日期是什么时候？。

测试对话

知识库 意图识别 多轮对话

对话

机器人 请问您需要什么帮助?

ME 我想预约机票

机器人 您从哪个城市或机场出发?

ME 南京

机器人 您要去的目的城市或机场是哪里?

刷新 **发送**

调试信息

意图名称: book_airplane_ticket

序号	槽位	值	是否必填	绑定词典
1	fromPlace	南京	是	@LOC
2	destPlace		是	@LOC
3	date		是	@TIME

现在阿Q可以识别意图了，但是要将意图识别和其它的对话语料融合，还需要按照下面步骤操作。

添加到多轮对话

回到多轮对话设计器，在刚刚打开的 `greetings` 话题编辑窗口。

Chatopera 多轮对话设计器

Edit Window Help

话题: greetings ▾

新建话题 环境变量 绑定信息 上传

检索多轮对话 返回值 对话 已启用话题

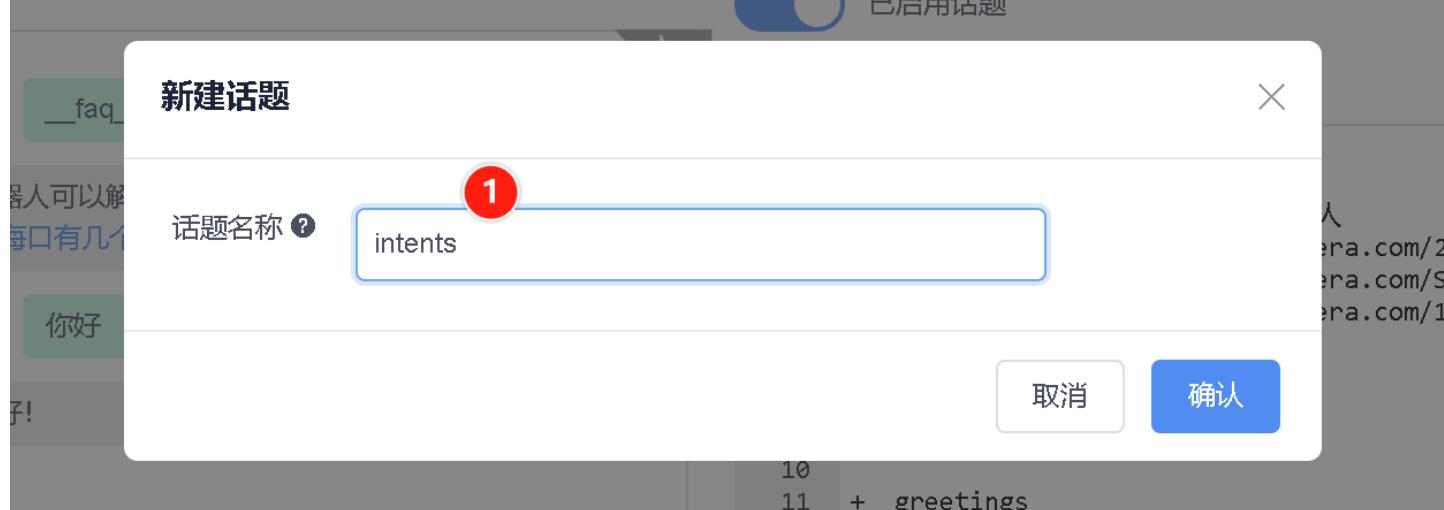
脚本 函数

```
{
  "state": "default",
  "createdAt": 1631536471115,
  "string": "机器人可以解答或提供的服务",
  "topicName": "greetings",
  "...": ...
}
```

SDK

有琼文高速和223国道，交通非常便利。海南有三个民用机场：海口美兰国际机场、三亚凤凰国际机场和琼海博鳌机场。

点击【新建话题】，话题名称填写 `intents`。



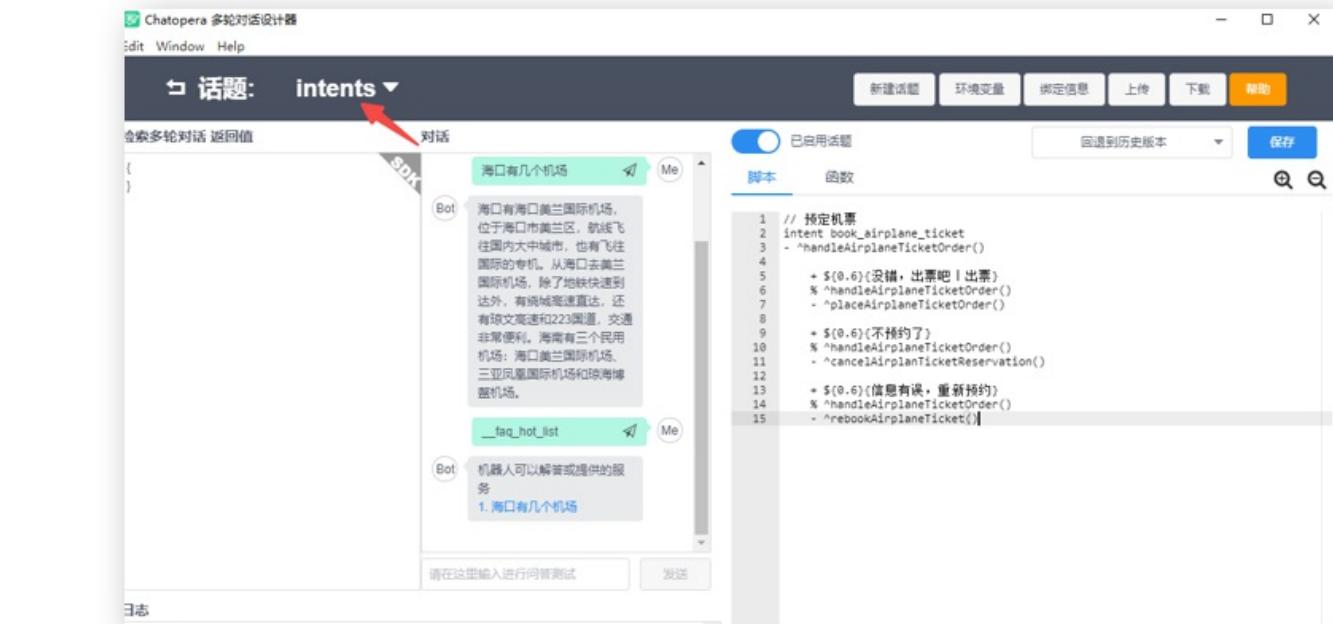
此时会进入一个新的脚本编辑窗口，在脚本编辑区域，复制粘贴以下的内容：

```
// 预约机票
intent {keep} book_airplane_ticket
- ^handleAirplaneTicketOrder()

+ ${0.6}{没错，出票吧！出票}
% ^handleAirplaneTicketOrder()
- {keep} ^placeAirplaneTicketOrder()

+ ${0.6}{不预约了}
% ^handleAirplaneTicketOrder()
- {keep} ^cancelAirplaneTicketReservation()

+ ${0.6}{信息有误，重新预约}
% ^handleAirplaneTicketOrder()
- {keep} ^rebookAirplaneTicket()
```



点击【保存】。然后，点击【函数】，进入函数编辑窗口，在之前编辑的函数后，复制粘贴下面的内容：

```
// 提取时间实体
async function extractTimeEntity(maestro, entities, property) {
```

```
debug("extractTimeEntity name %s, value %s", property, entities[property]["val"])
let dates = await maestro.extractTime(entities[property]["val"], "YYYY年MM月DD日 HH:mm");
return dates.length > 0 ? dates[0] : "";
}

// 确认订单信息
exports.handleAirplaneTicketOrder = async function() {

    debug("[handleAirplaneTicketOrder] this.intent", JSON.stringify(this.intent))
    let entities = _.keyBy(this.intent.entities, 'name');
    let date = await extractTimeEntity(this.maestro, entities, "date");

    this.intent.extras = {
        date: date
    }

    return {
        text: `和您确认一下信息，出发地${entities["fromPlace"]["val"]}, 目的地${entities["destPlace"]["val"]}, 出发时间${this.intent.extras.date}`,
        params: [
            {
                label: "没错，出票吧",
                type: "button",
                text: "没错，出票吧"
            },
            {
                label: "信息有误，重新预约",
                type: "button",
                text: "我想预约机票"
            },
            {
                label: "不预约了",
                type: "button",
                text: "不预约了"
            },
        ],
    }
}

// 下单
exports.placeAirplaneTicketOrder = async function() {
    this.intent.drop = true;
    let entities = _.keyBy(this.intent.entities, 'name');

    return {
        text: "{CLEAR} 已帮您购买",
        params: [
            {
                type: 'card',
                title: "查看详情",
                thumbnail:
"https://gitee.com/chatopera/cskefu/attach_files/1143210/download/AIRPLANE_20220801113300.jpg",
                summary: `${this.intent.extras.date}, 国泰航空 CA001, 国泰机场, ${entities["fromPlace"]["val"]} - ${entities["destPlace"]["val"]}`,
                hyperlink: "https://www.chatopera.com/"
            }
        ]
    }
}

// 不预约了
exports.cancelAirplanTicketReservation = async function() {
    this.intent.drop = true;
    return {
        text: "{CLEAR} 好的，下次再帮您预约"
    }
}

// 重新预约机票
```

```
exports.rebookAirplaneTicket = async function() {
    debug("rebookAirplaneTicket this.intent", this.intent);
    this.intent.drop = true;
    return `^topicRedirect(\"intents\", \"book_airplane_ticket\", true)`
}
```

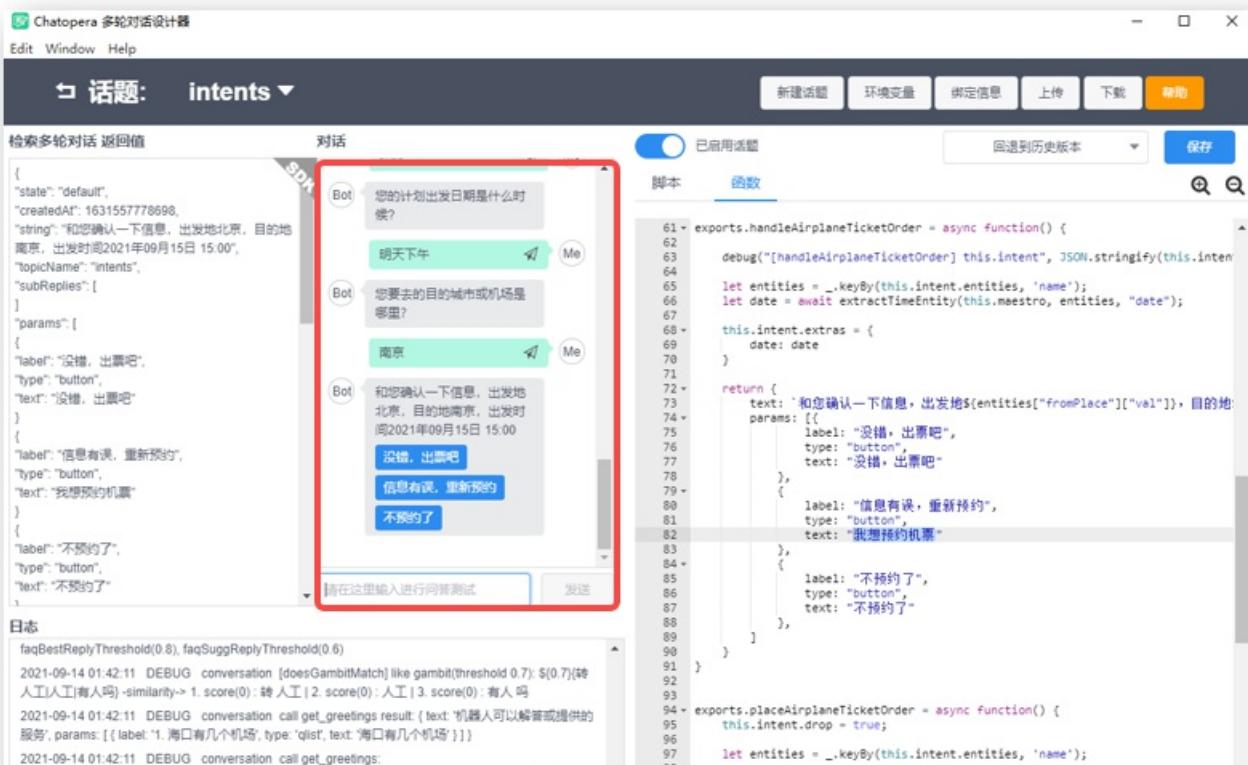
点击【保存】。

测试多轮对话

在多轮对话设计器内的对话窗口，发送文本：

我想预约机票

这时，阿Q识别了意图，并进行交互。



如果你看到了类似下面的消息：

对话

已启用话题

SDK

时间



请在这里对话

发送

重置

脚本

函数

```

1 // 问候语中关联常见意图
2 exports.get_greeting = {
3   return {
4     text: "机器",
5     params: [
6       {
7         label: "出发地",
8         type: "place",
9         text: ""
10      }
11    ]
12  };
13 // 提取时间实体
14 exports.extractTime = async function extractTime() {
15   debug("extractTime");
16   let dates = await this.$store.dispatch("fetchDates");
17   return dates.list;
18 }
19 // 确认订单信息
20 exports.handleAirplaneTicket = {
21   debug("[handleAirplaneTicket]"),
22   let entities = this.$store.state.entities,
23   let date = await this.$store.dispatch("fetchDate");
24   this.intent.extractTime();
25   let intent = this.$store.state.intent;
26   let date = await this.$store.dispatch("fetchDate");
27   this.$store.dispatch("setIntent", intent);
28   this.$store.dispatch("setDate", date);
29 }
30
31 // 重新预约
32 exports.reschedule = {
33   return {
34     text: `和您确认一下信息，出发地${entities.place[0].text}，目的地${entities.place[1].text}，出发时间${date.text}`,
35     params: [
36       {
37         label: "出发地",
38         type: "place",
39         text: entities.place[0].text
40       },
41       {
42         label: "目的地",
43         type: "place",
44         text: entities.place[1].text
45       },
46       {
47         label: "出发时间",
48         type: "date",
49         text: date.text
50       }
51     ]
52   }
53 }
54
55 // 重新预约
56 exports.reschedule = {
57   return {
58     text: `和您确认一下信息，出发地${entities.place[0].text}，目的地${entities.place[1].text}，出发时间${date.text}`,
59     params: [
60       {
61         label: "出发地",
62         type: "place",
63         text: entities.place[0].text
64       },
65       {
66         label: "目的地",
67         type: "place",
68         text: entities.place[1].text
69       },
70       {
71         label: "出发时间",
72         type: "date",
73         text: date.text
74       }
75     ]
76   }
77 }
78
79 // 重新预约
80 exports.reschedule = {
81   return {
82     text: `和您确认一下信息，出发地${entities.place[0].text}，目的地${entities.place[1].text}，出发时间${date.text}`,
83     params: [
84       {
85         label: "出发地",
86         type: "place",
87         text: entities.place[0].text
88       },
89       {
90         label: "目的地",
91         type: "place",
92         text: entities.place[1].text
93       },
94       {
95         label: "出发时间",
96         type: "date",
97         text: date.text
98       }
99     ]
100   }
101 }
102
103 // 重新预约
104 exports.reschedule = {
105   return {
106     text: `和您确认一下信息，出发地${entities.place[0].text}，目的地${entities.place[1].text}，出发时间${date.text}`,
107     params: [
108       {
109         label: "出发地",
110         type: "place",
111         text: entities.place[0].text
112       },
113       {
114         label: "目的地",
115         type: "place",
116         text: entities.place[1].text
117       },
118       {
119         label: "出发时间",
120         type: "date",
121         text: date.text
122       }
123     ]
124   }
125 }
126
127 // 重新预约
128 exports.reschedule = {
129   return {
130     text: `和您确认一下信息，出发地${entities.place[0].text}，目的地${entities.place[1].text}，出发时间${date.text}`,
131     params: [
132       {
133         label: "出发地",
134         type: "place",
135         text: entities.place[0].text
136       },
137       {
138         label: "目的地",
139         type: "place",
140         text: entities.place[1].text
141       },
142       {
143         label: "出发时间",
144         type: "date",
145         text: date.text
146       }
147     ]
148   }
149 }
150
151 // 重新预约
152 exports.reschedule = {
153   return {
154     text: `和您确认一下信息，出发地${entities.place[0].text}，目的地${entities.place[1].text}，出发时间${date.text}`,
155     params: [
156       {
157         label: "出发地",
158         type: "place",
159         text: entities.place[0].text
160       },
161       {
162         label: "目的地",
163         type: "place",
164         text: entities.place[1].text
165       },
166       {
167         label: "出发时间",
168         type: "date",
169         text: date.text
170       }
171     ]
172   }
173 }
174
175 // 重新预约
176 exports.reschedule = {
177   return {
178     text: `和您确认一下信息，出发地${entities.place[0].text}，目的地${entities.place[1].text}，出发时间${date.text}`,
179     params: [
180       {
181         label: "出发地",
182         type: "place",
183         text: entities.place[0].text
184       },
185       {
186         label: "目的地",
187         type: "place",
188         text: entities.place[1].text
189       },
190       {
191         label: "出发时间",
192         type: "date",
193         text: date.text
194       }
195     ]
196   }
197 }
198
199 // 重新预约
200 exports.reschedule = {
201   return {
202     text: `和您确认一下信息，出发地${entities.place[0].text}，目的地${entities.place[1].text}，出发时间${date.text}`,
203     params: [
204       {
205         label: "出发地",
206         type: "place",
207         text: entities.place[0].text
208       },
209       {
210         label: "目的地",
211         type: "place",
212         text: entities.place[1].text
213       },
214       {
215         label: "出发时间",
216         type: "date",
217         text: date.text
218       }
219     ]
220   }
221 }
222
223 // 重新预约
224 exports.reschedule = {
225   return {
226     text: `和您确认一下信息，出发地${entities.place[0].text}，目的地${entities.place[1].text}，出发时间${date.text}`,
227     params: [
228       {
229         label: "出发地",
230         type: "place",
231         text: entities.place[0].text
232       },
233       {
234         label: "目的地",
235         type: "place",
236         text: entities.place[1].text
237       },
238       {
239         label: "出发时间",
240         type: "date",
241         text: date.text
242       }
243     ]
244   }
245 }
246
247 // 重新预约
248 exports.reschedule = {
249   return {
250     text: `和您确认一下信息，出发地${entities.place[0].text}，目的地${entities.place[1].text}，出发时间${date.text}`,
251     params: [
252       {
253         label: "出发地",
254         type: "place",
255         text: entities.place[0].text
256       },
257       {
258         label: "目的地",
259         type: "place",
260         text: entities.place[1].text
261       },
262       {
263         label: "出发时间",
264         type: "date",
265         text: date.text
266       }
267     ]
268   }
269 }
270
271 // 重新预约
272 exports.reschedule = {
273   return {
274     text: `和您确认一下信息，出发地${entities.place[0].text}，目的地${entities.place[1].text}，出发时间${date.text}`,
275     params: [
276       {
277         label: "出发地",
278         type: "place",
279         text: entities.place[0].text
280       },
281       {
282         label: "目的地",
283         type: "place",
284         text: entities.place[1].text
285       },
286       {
287         label: "出发时间",
288         type: "date",
289         text: date.text
290       }
291     ]
292   }
293 }
294
295 // 重新预约
296 exports.reschedule = {
297   return {
298     text: `和您确认一下信息，出发地${entities.place[0].text}，目的地${entities.place[1].text}，出发时间${date.text}`,
299     params: [
300       {
301         label: "出发地",
302         type: "place",
303         text: entities.place[0].text
304       },
305       {
306         label: "目的地",
307         type: "place",
308         text: entities.place[1].text
309       },
310       {
311         label: "出发时间",
312         type: "date",
313         text: date.text
314       }
315     ]
316   }
317 }
318
319 // 重新预约
320 exports.reschedule = {
321   return {
322     text: `和您确认一下信息，出发地${entities.place[0].text}，目的地${entities.place[1].text}，出发时间${date.text}`,
323     params: [
324       {
325         label: "出发地",
326         type: "place",
327         text: entities.place[0].text
328       },
329       {
330         label: "目的地",
331         type: "place",
332         text: entities.place[1].text
333       },
334       {
335         label: "出发时间",
336         type: "date",
337         text: date.text
338       }
339     ]
340   }
341 }
342
343 // 重新预约
344 exports.reschedule = {
345   return {
346     text: `和您确认一下信息，出发地${entities.place[0].text}，目的地${entities.place[1].text}，出发时间${date.text}`,
347     params: [
348       {
349         label: "出发地",
350         type: "place",
351         text: entities.place[0].text
352       },
353       {
354         label: "目的地",
355         type: "place",
356         text: entities.place[1].text
357       },
358       {
359         label: "出发时间",
360         type: "date",
361         text: date.text
362       }
363     ]
364   }
365 }
366
367 // 重新预约
368 exports.reschedule = {
369   return {
370     text: `和您确认一下信息，出发地${entities.place[0].text}，目的地${entities.place[1].text}，出发时间${date.text}`,
371     params: [
372       {
373         label: "出发地",
374         type: "place",
375         text: entities.place[0].text
376       },
377       {
378         label: "目的地",
379         type: "place",
380         text: entities.place[1].text
381       },
382       {
383         label: "出发时间",
384         type: "date",
385         text: date.text
386       }
387     ]
388   }
389 }
390
391 // 重新预约
392 exports.reschedule = {
393   return {
394     text: `和您确认一下信息，出发地${entities.place[0].text}，目的地${entities.place[1].text}，出发时间${date.text}`,
395     params: [
396       {
397         label: "出发地",
398         type: "place",
399         text: entities.place[0].text
400       },
401       {
402         label: "目的地",
403         type: "place",
404         text: entities.place[1].text
405       },
406       {
407         label: "出发时间",
408         type: "date",
409         text: date.text
410       }
411     ]
412   }
413 }
414
415 // 重新预约
416 exports.reschedule = {
417   return {
418     text: `和您确认一下信息，出发地${entities.place[0].text}，目的地${entities.place[1].text}，出发时间${date.text}`,
419     params: [
420       {
421         label: "出发地",
422         type: "place",
423         text: entities.place[0].text
424       },
425       {
426         label: "目的地",
427         type: "place",
428         text: entities.place[1].text
429       },
430       {
431         label: "出发时间",
432         type: "date",
433         text: date.text
434       }
435     ]
436   }
437 }
438
439 // 重新预约
440 exports.reschedule = {
441   return {
442     text: `和您确认一下信息，出发地${entities.place[0].text}，目的地${entities.place[1].text}，出发时间${date.text}`,
443     params: [
444       {
445         label: "出发地",
446         type: "place",
447         text: entities.place[0].text
448       },
449       {
450         label: "目的地",
451         type: "place",
452         text: entities.place[1].text
453       },
454       {
455         label: "出发时间",
456         type: "date",
457         text: date.text
458       }
459     ]
460   }
461 }
462
463 // 重新预约
464 exports.reschedule = {
465   return {
466     text: `和您确认一下信息，出发地${entities.place[0].text}，目的地${entities.place[1].text}，出发时间${date.text}`,
467     params: [
468       {
469         label: "出发地",
470         type: "place",
471         text: entities.place[0].text
472       },
473       {
474         label: "目的地",
475         type: "place",
476         text: entities.place[1].text
477       },
478       {
479         label: "出发时间",
480         type: "date",
481         text: date.text
482       }
483     ]
484   }
485 }
486
487 // 重新预约
488 exports.reschedule = {
489   return {
490     text: `和您确认一下信息，出发地${entities.place[0].text}，目的地${entities.place[1].text}，出发时间${date.text}`,
491     params: [
492       {
493         label: "出发地",
494         type: "place",
495         text: entities.place[0].text
496       },
497       {
498         label: "目的地",
499         type: "place",
500         text: entities.place[1].text
501       },
502       {
503         label: "出发时间",
504         type: "date",
505         text: date.text
506       }
507     ]
508   }
509 }
510
511 // 重新预约
512 exports.reschedule = {
513   return {
514     text: `和您确认一下信息，出发地${entities.place[0].text}，目的地${entities.place[1].text}，出发时间${date.text}`,
515     params: [
516       {
517         label: "出发地",
518         type: "place",
519         text: entities.place[0].text
520       },
521       {
522         label: "目的地",
523         type: "place",
524         text: entities.place[1].text
525       },
526       {
527         label: "出发时间",
528         type: "date",
529         text: date.text
530       }
531     ]
532   }
533 }
534
535 // 重新预约
536 exports.reschedule = {
537   return {
538     text: `和您确认一下信息，出发地${entities.place[0].text}，目的地${entities.place[1].text}，出发时间${date.text}`,
539     params: [
540       {
541         label: "出发地",
542         type: "place",
543         text: entities.place[0].text
544       },
545       {
546         label: "目的地",
547         type: "place",
548         text: entities.place[1].text
549       },
550       {
551         label: "出发时间",
552         type: "date",
553         text: date.text
554       }
555     ]
556   }
557 }
558
559 // 重新预约
560 exports.reschedule = {
561   return {
562     text: `和您确认一下信息，出发地${entities.place[0].text}，目的地${entities.place[1].text}，出发时间${date.text}`,
563     params: [
564       {
565         label: "出发地",
566         type: "place",
567         text: entities.place[0].text
568       },
569       {
570         label: "目的地",
571         type: "place",
572         text: entities.place[1].text
573       },
574       {
575         label: "出发时间",
576         type: "date",
577         text: date.text
578       }
579     ]
580   }
581 }
582
583 // 重新预约
584 exports.reschedule = {
585   return {
586     text: `和您确认一下信息，出发地${entities.place[0].text}，目的地${entities.place[1].text}，出发时间${date.text}`,
587     params: [
588       {
589         label: "出发地",
590         type: "place",
591         text: entities.place[0].text
592       },
593       {
594         label: "目的地",
595         type: "place",
596         text: entities.place[1].text
597       },
598       {
599         label: "出发时间",
600         type: "date",
601         text: date.text
602       }
603     ]
604   }
605 }
606
607 // 重新预约
608 exports.reschedule = {
609   return {
610     text: `和您确认一下信息，出发地${entities.place[0].text}，目的地${entities.place[1].text}，出发时间${date.text}`,
611     params: [
612       {
613         label: "出发地",
614         type: "place",
615         text: entities.place[0].text
616       },
617       {
618         label: "目的地",
619         type: "place",
620         text: entities.place[1].text
621       },
622       {
623         label: "出发时间",
624         type: "date",
625         text: date.text
626       }
627     ]
628   }
629 }
630
631 // 重新预约
632 exports.reschedule = {
633   return {
634     text: `和您确认一下信息，出发地${entities.place[0].text}，目的地${entities.place[1].text}，出发时间${date.text}`,
635     params: [
636       {
637         label: "出发地",
638         type: "place",
639         text: entities.place[0].text
640       },
641       {
642         label: "目的地",
643         type: "place",
644         text: entities.place[1].text
645       },
646       {
647         label: "出发时间",
648         type: "date",
649         text: date.text
650       }
651     ]
652   }
653 }
654
655 // 重新预约
656 exports.reschedule = {
657   return {
658     text: `和您确认一下信息，出发地${entities.place[0].text}，目的地${entities.place[1].text}，出发时间${date.text}`,
659     params: [
660       {
661         label: "出发地",
662         type: "place",
663         text: entities.place[0].text
664       },
665       {
666         label: "目的地",
667         type: "place",
668         text: entities.place[1].text
669       },
670       {
671         label: "出发时间",
672         type: "date",
673         text: date.text
674       }
675     ]
676   }
677 }
678
679 // 重新预约
680 exports.reschedule = {
681   return {
682     text: `和您确认一下信息，出发地${entities.place[0].text}，目的地${entities.place[1].text}，出发时间${date.text}`,
683     params: [
684       {
685         label: "出发地",
686         type: "place",
687         text: entities.place[0].text
688       },
689       {
690         label: "目的地",
691         type: "place",
692         text: entities.place[1].text
693       },
694       {
695         label: "出发时间",
696         type: "date",
697         text: date.text
698       }
699     ]
700   }
701 }
702
703 // 重新预约
704 exports.reschedule = {
705   return {
706     text: `和您确认一下信息，出发地${entities.place[0].text}，目的地${entities.place[1].text}，出发时间${date.text}`,
707     params: [
708       {
709         label: "出发地",
710         type: "place",
711         text: entities.place[0].text
712       },
713       {
714         label: "目的地",
715         type: "place",
716         text: entities.place[1].text
717       },
718       {
719         label: "出发时间",
720         type: "date",
721         text: date.text
722       }
723     ]
724   }
725 }
726
727 // 重新预约
728 exports.reschedule = {
729   return {
730     text: `和您确认一下信息，出发地${entities.place[0].text}，目的地${entities.place[1].text}，出发时间${date.text}`,
731     params: [
732       {
733         label: "出发地",
734         type: "place",
735         text: entities.place[0].text
736       },
737       {
738         label: "目的地",
739         type: "place",
740         text: entities.place[1].text
741       },
742       {
743         label: "出发时间",
744         type: "date",
745         text: date.text
746       }
747     ]
748   }
749 }
750
751 // 重新预约
752 exports.reschedule = {
753   return {
754     text: `和您确认一下信息，出发地${entities.place[0].text}，目的地${entities.place[1].text}，出发时间${date.text}`,
755     params: [
756       {
757         label: "出发地",
758         type: "place",
759         text: entities.place[0].text
760       },
761       {
762         label: "目的地",
763         type: "place",
764         text: entities.place[1].text
765       },
766       {
767         label: "出发时间",
768         type: "date",
769         text: date.text
770       }
771     ]
772   }
773 }
774
775 // 重新预约
776 exports.reschedule = {
777   return {
778     text: `和您确认一下信息，出发地${entities.place[0].text}，目的地${entities.place[1].text}，出发时间${date.text}`,
779     params: [
780       {
781         label: "出发地",
782         type: "place",
783         text: entities.place[0].text
784       },
785       {
786         label: "目的地",
787         type: "place",
788         text: entities.place[1].text
789       },
790       {
791         label: "出发时间",
792         type: "date",
793         text: date.text
794       }
795     ]
796   }
797 }
798
799 // 重新预约
800 exports.reschedule = {
801   return {
802     text: `和您确认一下信息，出发地${entities.place[0].text}，目的地${entities.place[1].text}，出发时间${date.text}`,
803     params: [
804       {
805         label: "出发地",
806         type: "place",
807         text: entities.place[0].text
808       },
809       {
810         label: "目的地",
811         type: "place",
812         text: entities.place[1].text
813       },
814       {
815         label: "出发时间",
816         type: "date",
817         text: date.text
818       }
819     ]
820   }
821 }
822
823 // 重新预约
824 exports.reschedule = {
825   return {
826     text: `和您确认一下信息，出发地${entities.place[0].text}，目的地${entities.place[1].text}，出发时间${date.text}`,
827     params: [
828       {
829         label: "出发地",
830         type: "place",
831         text: entities.place[0].text
832       },
833       {
834         label: "目的地",
835         type: "place",
836         text: entities.place[1].text
837       },
838       {
839         label: "出发时间",
840         type: "date",
841         text: date.text
842       }
843     ]
844   }
845 }
846
847 // 重新预约
848 exports.reschedule = {
849   return {
850     text: `和您确认一下信息，出发地${entities.place[0].text}，目的地${entities.place[1].text}，出发时间${date.text}`,
851     params: [
852       {
853         label: "出发地",
854         type: "place",
855         text: entities.place[0].text
856       },
857       {
858         label: "目的地",
859         type: "place",
860         text: entities.place[1].text
861       },
862       {
863         label: "出发时间",
864         type: "date",
865         text: date.text
866       }
867     ]
868   }
869 }
870
871 // 重新预约
872 exports.reschedule = {
873   return {
874     text: `和您确认一下信息，出发地${entities.place[0].text}，目的地${entities.place[1].text}，出发时间${date.text}`,
875     params: [
876       {
877         label: "出发地",
878         type: "place",
879         text: entities.place[0].text
880       },
881       {
882         label: "目的地",
883         type: "place",
884         text: entities.place[1].text
885       },
886       {
887         label: "出发时间",
888         type: "date",
889         text: date.text
890       }
891     ]
892   }
893 }
894
895 // 重新预约
896 exports.reschedule = {
897   return {
898     text: `和您确认一下信息，出发地${entities.place[0].text}，目的地${entities.place[1].text}，出发时间${date.text}`,
899     params: [
900       {
901         label: "出发地",
902         type: "place",
903         text: entities.place[0].text
904       },
905       {
906         label: "目的地",
907         type: "place",
908         text: entities.place[1].text
909       },
910       {
911         label: "出发时间",
912         type: "date",
913         text: date.text
914       }
915     ]
916   }
917 }
918
919 // 重新预约
920 exports.reschedule = {
921   return {
922     text: `和您确认一下信息，出发地${entities.place[0].text}，目的地${entities.place[1].text}，出发时间${date.text}`,
923     params: [
924       {
925         label: "出发地",
926         type: "place",
927         text: entities.place[0].text
928       },
929       {
930         label: "目的地",
931         type: "place",
932         text: entities.place[1].text
933       },
934       {
935         label: "出发时间",
936         type: "date",
937         text: date.text
938       }
939     ]
940   }
941 }
942
943 // 重新预约
944 exports.reschedule = {
945   return {
946     text: `和您确认一下信息，出发地${entities.place[0].text}，目的地${entities.place[1].text}，出发时间${date.text}`,
947     params: [
948       {
949         label: "出发地",
950         type: "place",
951         text: entities.place[0].text
952       },
953       {
954         label: "目的地",
955         type: "place",
956         text: entities.place[1].text
957       },
958       {
959         label: "出发时间",
960         type: "date",
961         text: date.text
962       }
963     ]
964   }
965 }
966
967 // 重新预约
968 exports.reschedule = {
969   return {
970     text: `和您确认一下信息，出发地${entities.place[0].text}，目的地${entities.place[1].text}，出发时间${date.text}`,
971     params: [
972       {
973         label: "出发地",
974         type: "place",
975         text: entities.place[0].text
976       },
977       {
978         label: "目的地",
979         type: "place",
980         text: entities.place[1].text
981       },
982       {
983         label: "出发时间",
984         type: "date",
985         text: date.text
986       }
987     ]
988   }
989 }
990
991 // 重新预约
992 exports.reschedule = {
993   return {
994     text: `和您确认一下信息，出发地${entities.place[0].text}，目的地${entities.place[1].text}，出发时间${date.text}`,
995     params: [
996       {
997         label: "出发地",
998         type: "place",
999         text: entities.place[0].text
1000      },
1001      {
1002        label: "目的地",
1003        type: "place",
1004        text: entities.place[1].text
1005      },
1006      {
1007        label: "出发时间",
1008        type: "date",
1009        text: date.text
1010      }
1011    ]
1012  }
1013 }
```



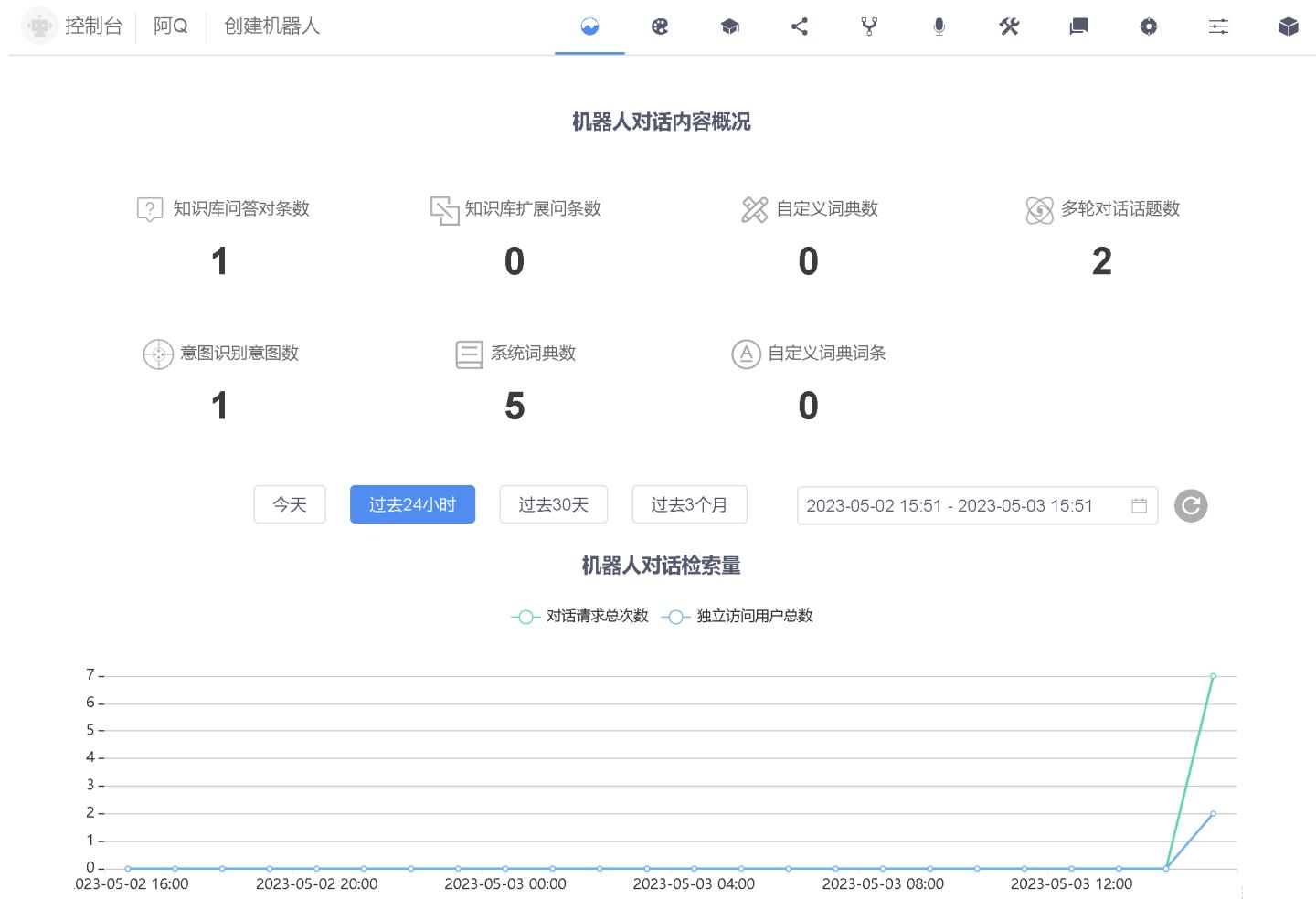
<< 上一步: <3/5> 添加脚本和函数 | >> 下一步: <5/5> 查看使用情况

<5/5> 查看使用情况

<< 上一步: <4/5> 添加意图对话 | ⏱ 阅读本节内容大约需要 1 min

查看机器人对话数据

经过了上面的几个环节, 阿Q 就能提供服务了。查看 阿Q 的概况页, 对话语料数据和请求数据现在有了变化。



你创造了一个智能对话机器人

太棒了！你已经完成了新手入门任务！



<< 上一步: <4/5> 添加意图对话

接下来

本篇是 Chatopera 云服务入门教程的最后一节，当你看到这里，就可以正式使用 Chatopera 云服务了，衷心的感谢你选择 Chatopera 云服务！回到 [首页](#)。

使用指南

账号

- [注册使用 Chatopera 云服务](#)

词典

- [系统词典](#)
- [词汇表词典](#)
- [正则表达式词典](#)

知识库

- [维护知识库问答对](#)
- [知识库问答及调优测试](#)
- [知识库问答对导入和导出](#)
- [自定义词典增强知识库](#)
- [知识库热门问题查看和导出](#)
- [使用函数获得动态答案](#)
- [话术助手快速检索话术](#)

意图识别

- [创建和训练意图识别模型](#)
- [发布上线意图识别生产版本](#)

多轮对话

- [多轮对话设计器安装](#)
- [通配符匹配器](#)
- [使用模糊匹配器](#)
- [使用意图匹配器](#)
- [设置回复](#)
- [使用上下轮钩子](#)
- [使用函数](#)
- [配置环境变量](#)
- [管理对话状态](#)
- [设置定时任务](#)

语音识别

- [调用语音识别服务](#)

对话调优

- [对话测试](#)
- [对话历史分析](#)
- [Chatopera CLI 连接多轮对话](#)

渠道

- [H5 聊天控件](#)

- [春松客服](#)

- [飞书](#)

系统集成

- [SDK](#)
- [CLI 安装和配置](#)
- [CLI 导入和导出对话语料](#)

注册使用 Chatopera 云服务

注册

Chatopera 云服务的地址是: <https://bot.chatopera.com>

从浏览器打开后, 点击“立即使用”, 跳转到登录/注册页面, 输入邮箱和密码, 使用“回车键”【Enter】提交, 即完成。

在登录后, 没有做手机号认证的, Chatopera 云服务会主动提示用户绑定手机号, 该项目是必须的, 未填写手机号的用户无法使用 Chatopera 云服务。

注意事项:

- 目前, 用于邮件通知服务的提供商暂未支持 Gmail 服务, 为方便重置密码、获得产品更新信息, 请使用其它邮件地址注册。
- 请保证填写正确的邮箱地址用于找回密码, 接收系统通知, 该邮箱不会泄漏给第三方。
- 用于手机号验证的 SMS 服务仅支持中国大陆运营商, 为了保证您的正常注册和使用, 请使用相关运营商的手机号。

更新密码

登录后, 在右上角看到头像, 点击头像, 在菜单中点击“个人中心”, 来到“个人中心”页面后, 可填入新的密码, 点击“保存”。

找回密码

进入登录页面 <https://bot.chatopera.com/login>, 点击“忘记密码”, 根据提示完成。

此处会使用邮箱验证账号。

修改头像

登录后, 在右上角看到头像, 点击头像, 在菜单中点击“个人中心”, 来到“个人中心”页面后, 点击屏幕中间的“头像”, 弹出上传头像表单。

系统词典

现有系统词典

系统词典可以直接引用，无需配置词条。

已有的系统词典包括：地名(@LOC)，人名(@PER)、时间(@TIME)、组织机构(@ORG)和任何字符串(@ANY)。在引用系统词典页面中可以看到

词典标识名	词典中文名	词典示例	操作
@PER	人名	郭德纲;于谦	<button>引用</button>
@TIME	时间	今天;下午一时	<button>引用</button>
@LOC	地名	北京市;东京	<button>引用</button>
@ORG	组织机构	北京华夏春松科技有限公司	<button>引用</button>

共 4 条 < 1 >

各个系统词典的作用

名称	描述	示例
@LOC	地名	北京市、东京
@PER	人名	郭德纲、于谦
@TIME	时间	今天、下午一时
@ORG	组织机构	北京华夏春松科技有限公司
@ANY	任何字符串	请求发送的任何字符串

引用与取消引用

对于系统词典可以引用和取消引用。被引用了的系统词典可以在槽位配置时使用



控制台 | ceshi | + 创建机器人

[← 引用系统词典](#)

- 系统词典可以直接引用，无需配置词条。
- 被引用了的系统词典可以在槽位配置时使用。

词典标识名	词典中文名	词典示例	操作
@PER	人名	郭德纲;于谦	<button>引用</button>
@TIME	时间	今天;下午一时	<button>引用</button>
@LOC	地名	北京市;东京	<button>取消引用</button>
@ORG	组织机构	北京华夏春松科技有限公司	<button>引用</button>

共 4 条



1



词汇表词典

词汇表词典就是管理词条，一个词汇是否属于某词汇表词典，就是检查该词汇是否属于该词典的词条集合。

创建

命名规则:

- 词典标识名为字母、数字、下划线的组合，1-32位，如: RailTypes
- 同一个机器人下的词典标识名不能重复
- 词典标识名一经确认后无法进行修改

在词典管理页面点击新建自定义词典，在弹出的对话框里输入正确的自定义词典名称，词典类型选择“词汇表”，点击确定

The screenshot shows the Chatopera platform's dictionary management interface. At the top, there is a navigation bar with icons for control panel, a user named 'ceshi2', and a '+ Create Robot'. Below the navigation bar, the title 'Dictionary Management' is displayed. A callout box contains two bullet points:

- Dictionaries include custom dictionaries and system dictionaries. Custom dictionaries are created by developers; system dictionaries are provided by the platform for common use.
- Dictionaries are mainly used for slot recognition and filling in intentions; custom dictionaries for different robots cannot be shared.

A red arrow points to the 'New Custom Dictionary' button, which is highlighted with a blue background. Below this button is a section titled 'System Dictionary' with the sub-instruction 'Below is the list of dictionaries already引用 (referenced)'. A table below shows columns for dictionary identifier, Chinese name, type, example, update time, creation time, and operations. The message 'No data found' is displayed. Further down, there is a section for 'Custom Dictionary' with a similar table structure and 'No data found' message. Navigation controls at the bottom right show '0 items', page number '1', and arrows for navigation.



控制台 | ceshi2 | + 创建机器人



词典管理

- 词典包括自定义词典
- 词典主要用于意图识别

新建自定义词典

词典标识名一经确定后将无法进行修改。

* 词典标识名

food



取消

确定

系统词典

以下是在引用的系统词典列表

词典标识名	词典中文名	词典类型	更新时间	创建时间	操作
暂无数据					

暂无数据

自定义词典

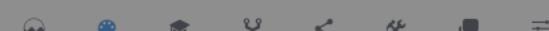
词典标识名	词典中文名	词典类型	更新时间	创建时间	操作
暂无数据					

共 0 条 < 1 >

如果创建的自定义词典名称不符合要求，会有相应的提示



控制台 | ceshi2 | + 创建机器人



词典管理

- 词典包括自定义词典
- 词典主要用于意图识别

新建自定义词典

词典标识名一经确定后将无法进行修改。

* 词典标识名

食物

词典标识名为字母、数字、下划线的组合, 1-32位

不可以是中文

取消

确定

系统词典

以下是在引用的系统词典列表

词典标识名	词典中文名	词典类型	更新时间	创建时间	操作
food	/	自定义词典	2019-09-07 16:02:17	2019-09-07 16:02:17	<button>编辑</button> <button>删除</button>

共 1 条 < 1 >

自定义词典

词典标识名	词典中文名	词典类型	更新时间	创建时间	操作
food	/	自定义词典	2019-09-07 16:02:17	2019-09-07 16:02:17	<button>编辑</button> <button>删除</button>

共 1 条 < 1 >

更新

对于已经创建的自定义词典可以修改中文名

The screenshot shows the Chatopera Bot control panel with the following interface elements:

- Header:** Chatopera Bot, a yellow sun-like icon, and a user profile icon.
- Top Bar:** Control台 (Control Panel), ceshi (Test), + 创建机器人 (Create Robot), and several small icons for settings and notifications.
- Section Header:** 词典管理 (Dictionary Management).
- Information Box:** A blue box containing two bullet points:
 - 词典包括自定义词典和系统词典。自定义词典是开发者自己创建的词典；系统词典是平台提供给开发者使用的常用词典。
 - 词典主要用于意图中槽位的识别和填充，不同机器人的自定义词典不能共享。
- Buttons:** 新建自定义词典 (Create Custom Dictionary) and 引用系统词典 (Reference System Dictionary).
- Section: 系统词典 (System Dictionary):** Subtitle: 以下是已经引用的系统词典列表. Table: A table with columns: 词典标识名, 词典中文名, 词典类型, 词典示例, 更新时间, 创建时间, 操作. Subtext: 暂无数据.
- Section: 自定义词典 (Custom Dictionary):** Table: A table with columns: 词典标识名, 词典中文名, 词典类型, 词典示例, 更新时间, 创建时间, 操作. Data rows:
 - num: 词典中文名 is empty, Type: 自定义词典, Updated: 2019-09-07 16:08:27, Created: 2019-09-07 16:08:27, Operations: 编辑 (Edit), 删 (Delete)
 - taset: 词典中文名 is empty, Type: 自定义词典, Updated: 2019-09-06 23:14:30, Created: 2019-09-06 23:14:30, Operations: 编辑 (Edit), 删 (Delete)
 - food: 词典中文名 is highlighted with a red box and arrow, Type: 自定义词典, Updated: 2019-09-06 23:13:15, Created: 2019-09-06 23:13:15, Operations: 编辑 (Edit), 删 (Delete)Subtext: 共 3 条, with navigation buttons <, 1, >.

删除

对于不需要的自定义词典点击删除按钮即可

 控制台 | ceshi | + 创建机器人

词典管理

- 词典包括自定义词典和系统词典。自定义词典是开发者自己创建的词典；系统词典是平台提供给开发者使用的常用词典。
- 词典主要用于意图中槽位的识别和填充，不同机器人的自定义词典不能共享。

[新建自定义词典](#)[引用系统词典](#)

系统词典

以下是从已经引用的系统词典列表

词典标识名	词典中文名	词典类型	词典示例	更新时间	创建时间	操作
暂无数据						

自定义词典

词典标识名	词典中文名	词典类型	更新时间	创建时间	操作
car	/	自定义词典	2019-09-07 16:33:40	2019-09-07 16:33:40	编辑 删除
num	数量 /	自定义词典	2019-09-07 16:33:05	2019-09-07 16:08:27	编辑 删除
taset	口味 /	自定义词典	2019-09-07 16:32:59	2019-09-06 23:14:30	编辑 删除
food	食物 /	自定义词典	2019-09-07 16:32:54	2019-09-06 23:13:15	编辑 删除

共 4 条



1



Chatopera Beta

控制台 ceshi + 创建机器人

词典管理

词典删除后将无法进行恢复, 确定删除吗?

请确认该词典没有正在使用再删除, 以免影响用户端的体验

取消 确定

新建自定义词典 引用系统词典

系统词典
以下是已经引用的系统词典列表

词典标识名	词典中文名	词典类型	词典示例	更新时间	创建时间	操作
暂无数据						

自定义词典

词典标识名	词典中文名	词典类型	更新时间	创建时间	操作
car	/	自定义词典	2019-09-07 16:33:40	2019-09-07 16:33:40	<button>编辑</button> <button>删除</button>
num	数量 /	自定义词典	2019-09-07 16:33:05	2019-09-07 16:08:27	<button>编辑</button> <button>删除</button>
taset	口味 /	自定义词典	2019-09-07 16:32:59	2019-09-06 23:14:30	<button>编辑</button> <button>删除</button>
food	食物 /	自定义词典	2019-09-07 16:32:54	2019-09-06 23:13:15	<button>编辑</button> <button>删除</button>

共 4 条 < 1 >

对于已经绑定了槽位的自定义词典不可删除



该词典已经绑定槽位，不可删除



控制台 ceshi + 创建机器人



词典管理

- 词典包括自定义词典和系统词典。自定义词典是开发者自己创建的词典；系统词典是平台提供给开发者使用的常用词典。
- 词典主要用于意图中槽位的识别和填充，不同机器人的自定义词典不能共享。

[新建自定义词典](#)[引用系统词典](#)

系统词典

以下是从已经引用的系统词典列表

词典标识名	词典中文名	词典类型	词典示例	更新时间	创建时间	操作
暂无数据						

自定义词典

词典标识名	词典中文名	词典类型	更新时间	创建时间	操作
num	数量	自定义词典	2019-09-07 16:33:05	2019-09-07 16:08:27	编辑 删除
taset	口味	自定义词典	2019-09-07 16:32:59	2019-09-06 23:14:30	编辑 删除
food	食物	自定义词典	2019-09-07 16:32:54	2019-09-06 23:13:15	编辑 删除

共 3 条



1



添加词条

请使用分号；分隔词项，第一个词项将作为标准词，后面的词项将作为同义词。

词条的增删改查

添加词条：编辑完成后点击添加或者回车。



控制台 | ceshi | + 创建机器人



← food

- 开发者可以根据业务需要为自己创建的自定义词典添加多个词条。
- 一个词条包含至少一个标准词和不限个数的同义词，同义词会被识别为标准词。

添加词条

标准词: 饺子 同义词: 水饺

添加

按Enter键快速添加

输入词项进行搜索

标准词 ①	同义词 ②	操作
米饭		
面条		
馒头		

1

保存

取消

删除词条: 点击删除按钮即可。

修改词条: 一个词条的标准词一旦确定不可修改, 但是同义词可以修改, 点击编辑按钮, 出现输入框, 可以对同义词进行修改。



控制台 | ceshi | + 创建机器人



← food

- 开发者可以根据业务需要为自己创建的自定义词典添加多个词条。
- 一个词条包含至少一个标准词和不限个数的同义词，同义词会被识别为标准词。

添加词条

请使用分号；分隔词项，第一个词项将作为标准词，后面的词项将作为同义词

添加 按Enter键快速添加

输入词项进行搜索 

标准词 	同义词 	操作
饺子	水饺	 
米饭	 大米  米  大米饭  请使用分号；分隔多个同义词	 
面条		 
馒头		 

< 1 >

 保存 取消

查找词条：对于词条较多对词典来说，可以快速搜索词条。在搜索框输入需要搜索的词条之后点击放大镜或者点击回车键即可搜索。搜索完成后搜索框中出现一个清除按钮，点击之后清除本次搜索。



控制台 | ceshi | + 创建机器人



food

- 开发者可以根据业务需要为自己创建的自定义词典添加多个词条。
- 一个词条包含至少一个标准词和不限个数的同义词，同义词会被识别为标准词。

添加词条

请使用分号；分隔词项，第一个词项将作为标准词，后面的词项将作为同义词

添加 按Enter键快速添加



标准词 ①	同义词 ②	操作
米饭	大米;米;大米饭	编辑 删除

保存

取消

保存

在编辑完所有词条之后，需要进行保存，否则变更不生效。比如本页词条有变更的情况下进行翻页或者搜索，都需要先进行保存。保存成功后，会重新训练机器人。



控制台 | ceshi | + 创建机器人



← food

- 开发者可以根据业务需要为自己创建的自定义词典添加多个词条。
- 一个词条包含至少一个标准词和不限个数的同义词，同义词会被识别为标准词。

添加词条

请使用分号：分隔词项，第一个词项将作为标准词，后面的词项将作为同义词

添加

按Enter键快速添加

输入词项进行搜索



标准词 ①	同义词 ②	操作
饺子	水饺	编辑 删除
米饭	大米;米;大米饭	编辑 删除
面条		编辑 删除
馒头		编辑 删除

< 1 >

保存中

取消

保存成功，正在训练



The screenshot shows a user interface for managing a custom vocabulary. A modal dialog box is centered, asking "是否保存本次变更" (Save changes) with "不保存" (Not Save) and "保存" (Save) buttons. The background shows a table of words and their synonyms, with buttons for adding new entries and saving changes.

标准词 ⓘ	同义词 ⓘ	操作
饺子	水饺	<button>编辑</button> <button>删除</button>
米饭	大米;米;大米饭	<button>编辑</button> <button>删除</button>
面条		<button>编辑</button> <button>删除</button>
馒头		<button>编辑</button> <button>删除</button>

Buttons at the bottom include "保存" (Save) and "取消" (Cancel).

维护

为了用户体验更好，需要对自定义词典进行维护与管理

创建时：在创建词典添加词条时，尽可能多的，全面的添加不同的词条和同义词，便于在槽位中被识别出来（如果用户说的词条在该自定义词典中没有，就会识别不出来）

使用时：在历史消息中可以查看所有机器人与用户的对话，如果发现用户说的词条在该自定义词典中没有，应该立即加上。

正则表达式词典

正则表达式词典是通过正则表达式的形式约束一个词汇的集合，正则表达式是以一定语法作为规则的语法。

创建

命名规则:

- 词典标识名为字母、数字、下划线的组合，1-32位，如: RailTypes
- 同一个机器人下的词典标识名不能重复
- 词典标识名一经确认后无法进行修改

在词典管理页面点击新建自定义词典，在弹出的对话框里输入正确的自定义词典名称，词典类型选择“正则表达式”，点击确定

The screenshot shows the Chatopera platform's dictionary management interface. At the top, there is a navigation bar with icons for control panel, a user named 'ceshi2', and a '+ Create Robot'. Below the navigation bar, the title 'Dictionary Management' is displayed. A callout box contains instructions: 'Dictionary includes custom dictionaries and system dictionaries. Custom dictionaries are created by developers; system dictionaries are provided by the platform for common use. Dictionaries are mainly used for slot recognition and filling in the intent, and custom dictionaries cannot be shared between different robots.' Two buttons are present: 'New Custom Dictionary' (highlighted with a red arrow) and 'Use System Dictionary'. Below these buttons, a section titled 'System Dictionary' shows a table with one row: '暂无数据' (No data). Another section titled 'Custom Dictionary' also shows a table with one row: '暂无数据' (No data). At the bottom right, there are pagination controls: '共 0 条' (0 items), a page number '1' (highlighted with a blue box), and arrows for navigating through pages.



如果创建的自定义词典名称不符合要求，会有相应的提示

A screenshot of the Chatopera platform's dictionary management interface. A modal window titled '新建自定义词典' (Create New Custom Dictionary) is open. It contains a warning message: '词典标识名一经确定后将无法进行修改。' (Dictionary identifier name cannot be changed once set). Below this, there is a field labeled '词典标识名' (Dictionary Identifier Name) with the value '食物'. This field is highlighted with a red border and has a red arrow pointing to it from the left. Below the identifier name is a note: '词典标识名为字母、数字、下划线的组合, 1-32位' (Dictionary identifier name consists of letters, digits, underscores, 1-32 characters). At the bottom right of the modal are two buttons: '取消' (Cancel) and '确定' (Confirm). In the background, a table lists system dictionaries, including one named 'ceshi2'. The table columns include '词典标识名' (Dictionary Identifier Name), '词典中文名' (Dictionary Chinese Name), '词典类型' (Dictionary Type), '更新时间' (Last Update Time), '创建时间' (Creation Time), and '操作' (Operations). A note at the bottom of the table says '暂无数据' (No data available).

编辑

Chatopera 机器人平台中，正则表达式词典支持的表达式规范为PCRE 标准。

添加正则表达式定义，进入【词典管理>>自定义词典】列表，选择正则表达式词典，点击“编辑”。



← IDNumber

- 开发者可以根据业务需要为自己创建的自定义词典添加正则表达式。
- 添加的正则表达式应符合规范

添加表达式

```
\w[-\w.+]@([A-Za-z0-9][-A-Za-z0-9]+\.)+[A-Za-z]{2,14}
```

添加 按Enter键快速添加

正则表达式

```
\w[-\w.+]@([A-Za-z0-9][-A-Za-z0-9]+\.)+[A-Za-z]{2,14}
```

操作

编辑 **删除**

保存

✓ 训练成功，可进行测试

重置

在“添加表达式”项目下，写入表达式，点击“添加”，然后点击“保存”。保存成功后会重新训练机器人，如果遇到错误会弹出提示消息，对于没有意图或意图中没有说法的情况，训练机器人会失败，这时候保存正则表达式词典是成功的，该错误可忽略，后续在添加了意图和说法后，就正常了。

正则表达式定义支持多个表达式，在使用正则表达式词典时，会按照正则表达式列表的顺序进行匹配，如果匹配到词汇，就跳过其它表达式，否则继续匹配，直到遍历表达式列表。

保存成功后，还需要确认表达式是能按照预期工作，这个验证的过程通过【验证表达式】完成，该项目在表达式列表的下面。

正则表达式

操作

```
\w[-\w.+]@([A-Za-z0-9][-A-Za-z0-9]+\.)+[A-Za-z]{2,14}
```

编辑 **删除**

保存

✓ 训练成功，可进行测试

重置

验证表达式

验证

能发电子发票到我的邮箱吗? foo@bar.com

(X)

是否匹配：是

能发电子发票到我的邮箱吗? **foo@bar.com**

匹配表达式：`\w[-\w.+]@([A-Za-z0-9][-A-Za-z0-9]+\.)+[A-Za-z]{2,14}`

验证表达式过程就是输入一段测试文本，点击“验证”，验证结果将显示在下面。

常用正则表达式

正则表达式是一个易学难精的技能，一方面是介绍材料众多，另一方面是网络上有很多写好的常用的表达式。以下定义三个常用的表达式。

项目	表达式
邮箱	\w[-\w.+]@([A-Za-z0-9][-A-Za-z0-9]+\.)+[A-Za-z]{2,14}
身份证号	\d{15}(\d\d[0-9xX])?
QQ 号	[1-9][0-9]{4,}

正则表达式的基础知识参考[链接](#), 更多关于正则表达式材料在网络上自行查找, 相关 **Perl** 正则表达式或 **PCRE** 正则表达式的适用于 **Chatopera** 机器人平台的正则表达式词典。

表达式注意事项

上文提到了 Chatopera 机器人平台的正则表达式词典是基于 C++ Boost Regex 库的, 并使用 PCRE 规范, 实际上和 PCRE 有细微出入, 以下内容进行介绍。

1. 如果在[]之间需要包含-, 并且不代表区间, 而是-代表本身, 那么需要将-放在末尾或开头, 否则会报错;
2. 在表达式使用中只取第一个匹配的值, 如果一个表达式能匹配到输入语句的多个值, 其它的值被忽略。

PCRE

PCRE 是“Perl-compatible regular expressions”的缩写, 广泛的被各种编程语言支持, 虽然在不同语言支持时, 会有微小的不同, 实际使用中仍可参考 PCRE 的说明, Chatopera 机器人平台使用 C++ 开发, 底层的正则表达式接口使用[Boost Regex 库](#)

知识库问答对维护

能回答问题的前提要了解问题是什么。比如用户问：土豆多少钱一斤？如果知识库中没有这个问题，机器人就无法回复给用户，我们需要把这个用户的输入添加到知识库中，下次用户再问这个问题，机器人就可以把答案回复给用户。问题就是预期用户会发送的文字，机器人使用答案回复。

问题、相似问题和答案组成了问答对。建立问答对时，支持添加多个相似问题 - 不同的用户说法不同，但都是同一个意思，相似问也被称为扩展问，二者表达同一个概念。相似问题和自定义词典大大加强了问答对适应用户各种问法的能力，自定义词典支持维护近义词。下面进行详细介绍。

新建问答对

在知识库页面点击「+」

The screenshot shows a modal dialog for creating a question-answer pair. It includes fields for 'Question' (必填), 'Category Selection', 'Similar Questions' (with an 'Add' button), 'Answer' (with a 'Text' dropdown menu showing 'Text', 'HTML', and 'Image' options), and 'Buttons' for 'Confirm' and 'Close'.

新建问题

问题 *

选择分类

请选择

相似问题

+ 增加

答案

答1 ×

增加 ▾

文本

HTML

图文

确定

关闭

在机器人管理面板中，还有其它地方可以创建问答对，比如历史记录页面点击「+」创建问答对。这些创建或修改问答对的快捷方式提升了维护知识库的便捷性。

设定答案

问答对支持多答案

针对用户业务的特点，每个问答对中，可以包含 1-10 个答案。多答案的设定，对于业务系统集成知识库更友好。这是因为，业务系统将保持更大的灵活性，选择最合适的策略推送给来访者。

“业务系统”是一个宽泛的概念，因为知识库是通用的，比如电商、教育等行业都可以使用，在 Chatopera 机器人平台被来访者访问到前，通常会有一个集成渠道或处理业务逻辑的软件，这个软件统一的称之为“业务系统”。

比如, 电商类的问题, 有一个鞋帽类产品介绍, 包含了多个图片, 那么知识库中关联多个图文消息, 每个图文消息可以每隔 3 秒钟发送一个, 消费者收到消息时, 对缩略图都形成印象。

又比如, 客户端涉及答案 1推送给微信渠道用户, 答案 2推送给微博渠道用户, 多答案的设定, 可以满足更多复杂需求。

以上举例只是为了说明多答案给应用知识库带来的灵活性, 同时, 知识库问答对支持多种答案类型。

纯文本

以一段文本内容作为答案。步骤: 新建问题页面点击「增加」下拉框, 选择「文本」

富文本

使用富文本编辑器, 最终答案将保存为 HTML 格式。

步骤: 新建问题页面点击「增加」下拉框, 选择「HTML」

The screenshot shows the 'Create Question' interface. At the top, there's a 'New Question' button. Below it, the 'Question *' field contains '西红柿多少钱一斤'. Under 'Category Selection', there's a dropdown menu set to 'Primary Category'. In the 'Similar Questions' section, there's a '+ Add' button and a field containing '番茄多少钱一斤'. The 'Answer' section has two tabs: 'Answer 1' (selected) and 'Answer 2'. A 'More' button is next to the tabs. Below the tabs is a rich text editor toolbar with icons for bold, italic, underline, font size, alignment, and other styling options. A floating menu on the right lists 'Text', 'HTML', and 'Image'. The main content area contains the text '1. 西红柿5元一斤'. At the bottom right are 'Confirm' and 'Cancel' buttons.

图文消息

由标题、描述、图片和超链接组成的答案, 适合展示为缩略图并配合文字, 来访者点击后打开超链接。

步骤: 新建问题页面点击「增加」下拉框, 选择「图文」

新建问题

问题 *

西红柿多少钱一斤

选择分类

分类一级

相似问题

+ 增加

番茄多少钱一斤

答案

答1

答2

答3 ×

增加 ▾

标题 *

文本

HTML

图文

URL *

描述

确定

关闭

机器人置信度

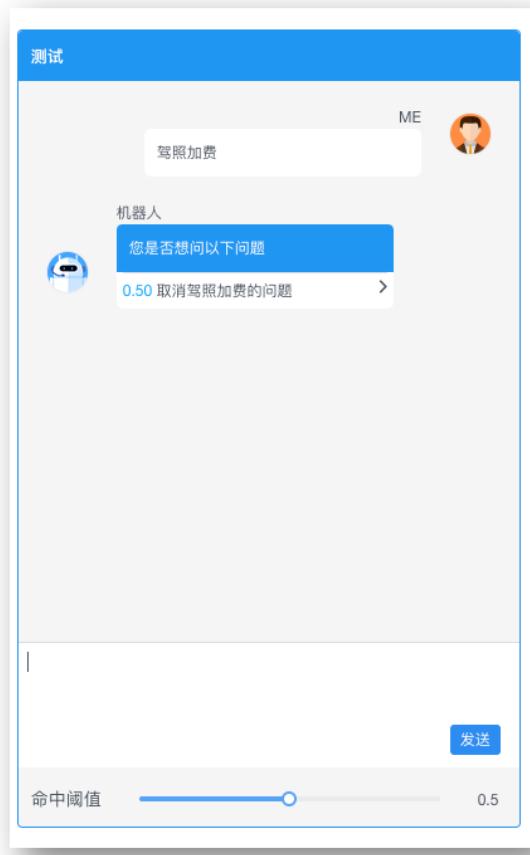
设置问答对后, 知识库能回答哪些问题了呢? 如果用户问法不一样, 机器人的“理解”能力怎么样呢?

置信度

机器人在检索知识库时, 有一个判断来访者输入的问题和知识库中哪个问答对最匹配的时候, 使用“置信度”为衡量标准。置信度score (也称为命中域值、相似度), 它的值在[0~1]区间, 值越大代表机器人越确定回复的准确性。

最佳回复和建议回复

当置信度高于某数值时, 机器人就认为置信度最高的答案匹配上了来访者的问题, 可以将该问答对的答案回复给来访者, 该数值称为最佳回复阀值; 当没有问答对的相似度高于最佳回复阀值, 但是有一些问答对可以作为建议回复发送给来访者, 展示为“您是想咨询一下问题吗”, 然后列一个列表, 请来访者选择, 那么可以进入这个列表的问答对, 需要高于某数值, 该数值称为建议回复阀值。



对知识库的评测，常常用一系列测试问题查询返回结果，进而计算机器人回答的准确率评价检索系统。调试结果可以指导合理设置最佳回复阀值和建议回复阀值，这两个值对于业务系统集成知识库具有很重要的意义。

在熟悉了置信度的概念以后，就可以开始调试知识库了。

知识库问答及调优测试

测试知识库问答

在机器人管理面板，通过测试对话页，测试知识库。

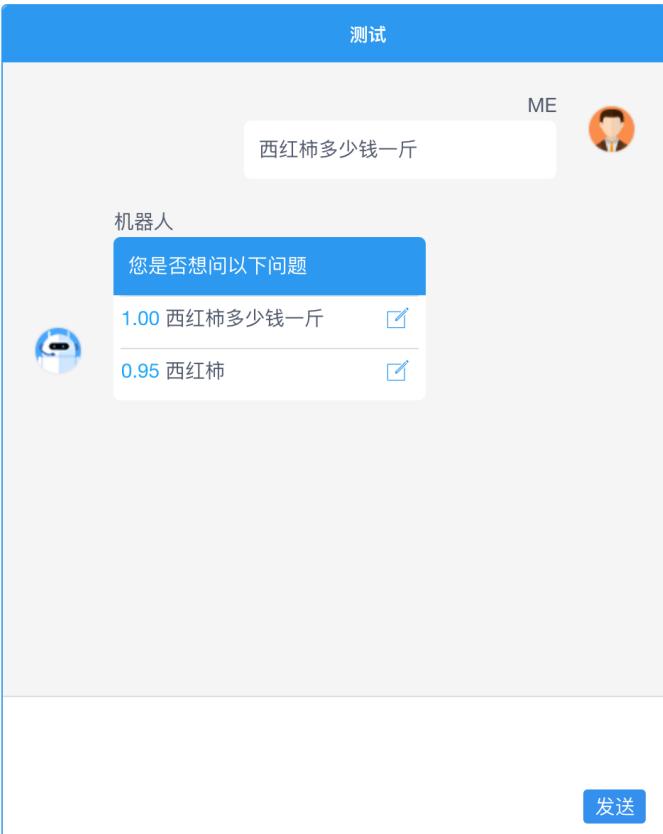
步骤：测试对话页面输入问题，查看机器人的回复

测试对话

知识库

意图识别

多轮对话



命中阈值 0.5



```
[{"id": "AXNxDVd8zLtU6n9zKj7C", "score": 1, "post": "西红柿多少钱一斤", "replies": [{"rtype": "plain", "enabled": true, "content": "5元一斤"}]}, {"id": "AXNXDdQdzLtU6n9zKj7D", "score": 0.95, "post": "西红柿", "replies": [{"rtype": "plain", "enabled": true, "content": "5元"}]}]
```

快速编辑问答对

如果机器人回复的问题命中阈值不高，可以直接点击问题，进行编辑

测试对话

知识库

意图识别

多轮对话

测试

ME



西红柿多少钱一斤

机器人

您是否想问以下问题

1.00 西红柿多少钱一斤

编辑问答对

0.95 西红柿



发送

如果有大量而频繁的评价知识库准确率的需求，可以使用 [系统集成/SDK](#) 实现。

知识库问答对导入和导出

导入和导出是对知识库的数据进行批量管理，导出的数据可以导入到其它机器人知识库中，导出的内容包括知识库分类信息、标准问、相似问和答案和启用状态。

应用场景

- 1) 备份机器人知识库； 2) 新建机器人快速导入。

本文以 Chatopera 机器人管理控制台为工具进行说明，如果想通过更自动化的方式，参考[使用 CLI 导入和导出对话语料](#)。

批量导出

步骤：知识库页面点击「导出」



The screenshot shows the 'Knowledge Base' section of the Chatopera management console. At the top, there are buttons for 'Add' (+), 'Question' (问题), 'Select Category' (选择分类: 请选择), 'Search' (搜索), 'Batch Import' (批量导入), 'Export' (导出, highlighted with a red box), 'Download Sample Data' (下载示例数据), and 'Category Management' (分类管理). Below this is a table listing three knowledge base entries:

问题	是否启用	操作
语文好的进来！ 所属分类: 分类一级	<input checked="" type="checkbox"/> 启用	[Edit] [Delete]
阜阳卫校怎么走？坐几路公交？阜阳市阜临路511号 所属分类: 分类一级	<input checked="" type="checkbox"/> 启用	[Edit] [Delete]
黑道圣徒2怎么取消武器的秘籍 所属分类: 分类一级	<input checked="" type="checkbox"/> 启用	[Edit] [Delete]

文件导出格式为 json 格式，json 是一种节省空间、灵活的、方便用文本编辑器处理和代码处理的流行格式。

批量导入

注意事项：

- 导入的知识库仅支持 json 格式，内容定义参考[示例数据](#)
- 点击「下载示例数据」之后，可以对示例数据进行编辑保存，再导入到知识库
- 导入时可设置问题的状态，问题状态为「禁用」在测试知识库时检索不到

导入

 选择文件 100条.json

请添加格式为json

状态	问题
<input checked="" type="radio"/> 启用	一次泡5根苦丁茶多吗
<input checked="" type="radio"/> 启用	阜阳卫校怎么走？坐几路公交？阜阳市阜临路511号
<input type="radio"/> 禁用	语文好的进来！
<input type="radio"/> 禁用	黑道圣徒2怎么取消武器的秘籍
<input checked="" type="radio"/> 启用	想给宝宝取个名字
<input checked="" type="radio"/> 启用	围墙围成面积为216平方米的一块矩形的土地，在正中用一堵墙隔成两块，问土地的长和宽选多大，材料最省？
<input checked="" type="radio"/> 启用	翻译英语

 上传 取消

自定义词典增强知识库

近义词是另一种增加机器人智能化水平的手段。比如：

创建知识库问答对，标准问：“土豆多少钱一斤”。不添加相似问题。此时，用户问：“马铃薯多少钱一斤”。

这个时候机器人处理“马铃薯多少钱一斤”时，知识库里匹配不到置信度很高的问题。

那么，怎样才能让机器人的回复与“马铃薯多少钱一斤”一样的答案呢？因为它们可是同一个意思啊。

这种情况下就需要配置近义词，我们找到包含「土豆」词条的自定义词典，为土豆添加近义词「马铃薯」，然后再去测试知识库，机器人的回复与“马铃薯多少钱一斤”一样的答案。

自定义词典定义

自定义词典就是用户添加自己的词典，支持创建：词汇表和正则表达式两种类型的词典。自定义词典通常用来管理近义词，新词，专有名词。

提示：知识库中应用自定词典只支持词汇表类型的自定义词典。

创建自定义词典

创建自定义词典步骤：词典管理页面点击「新建自定义词典」词典类型选择「词汇表」

新建自定义词典



词典标识名一经确定后将无法进行修改。

* 词典标识名

词典标识名为字母、数字、下划线的组合，1-32位，如：Boc



* 词典类型

词汇表



取消

确定

增加词条

添加词条步骤：自定义词典列表点击「编辑」进入添加词条页面

[← vegetables](#)

- 开发者可以根据业务需要为自己创建的自定义词典添加多个词条。
- 一个词条包含至少一个标准词和不限个数的同义词，同义词会被识别为标准词。
- 词条用于知识库和意图识别

添加词条

请使用分号；分隔词项，第一个词项将作为标准词，后面的词项将作为同义词

[添加](#) 按Enter键快速添加

词条已变更，为保证词条生效 [更新知识库索引](#) [训练意图识别模型](#)

输入词项进行搜索

标准词	同义词	操作
西红柿	番茄	编辑 删除

[保存](#)

[取消](#)

知识库同步自定义词典

在保存了自定义词典后，知识库并没有立即应用变更的自定义词典，近义词并没有立刻生效。只有在自定义词典管理页面或知识库管理页面，点击“更新知识库索引”后，知识库才会同步自定义词典，近义词才会生效。

步骤：知识库页面点击「更新知识库」



验证近义词

"番茄多少钱一斤"有结果了！

用户的问题中含有「西红柿」的同义词「番茄」，机器人可以识别到。

测试对话

知识库

意图识别

多轮对话

测试

ME

番茄多少钱一斤

人

机器人

您是否想问以下问题

1.00 西红柿多少钱一斤

0.95 西红柿

发送

命中阈值 0.5

命中阈值 0.5

0.5

```
[{"id": "AXNxDVd8zLtU6n9zKj7C", "score": 1, "post": "西红柿多少钱一斤", "replies": [{"rtype": "plain", "enabled": true, "content": "5元一斤"}]}, {"id": "AXNxDdQdzLtU6n9zKj7D", "score": 0.95, "post": "西红柿", "replies": [{"rtype": "plain", "enabled": true, "content": "5元"}]}]
```

近义词，新词，专有名词都可以通过自定义词典进行维护，维护好自定义词典对机器人智能化水平有着至关重要的作用。

知识库热门问题查看和导出

热门问题评分是知识库中的问答对的相对热门程度。问题的热门度是一个分数，分数值代表某一个问答对作为回复结果出现在客户端的次数，可能是作为最佳答案，也可能是作为建议答案。

具体来说，就是在系统集成时，使用[知识库检索接口](#)或[多轮对话检索接口](#)时，该问答对出现在结果中的次数。

查看问答对热门度

在 Chatopera 机器人管理控制台上，每个机器人的知识库管理页面，每个问答对增加【热门度】一列。

知识库

知识库已同步自定义词典信息 ✓

问题	是否启用	热门度 ①	操作
天空为什么是蓝色的 所属分类: 暂无	<input checked="" type="checkbox"/>	1	编辑 删除

下载热门问题

如何获得所有知识库问答对的按热门度排名信息？在知识库管理页面，【下载】下拉菜单中，增加【热门问题】。

控制台 Mo 创建机器人

知识库

知识库已同步自定义词典信息 ✓

问题	是否启用	热门度 ①	操作
天空为什么是蓝色的 所属分类: 暂无	<input checked="" type="checkbox"/>	1	编辑 删除

示例数据 操作
Excel 模版
热门问题 编辑 删除

点击后将下载到按热门度降序排名的 Excel 汇总信息。

使用函数获得动态答案

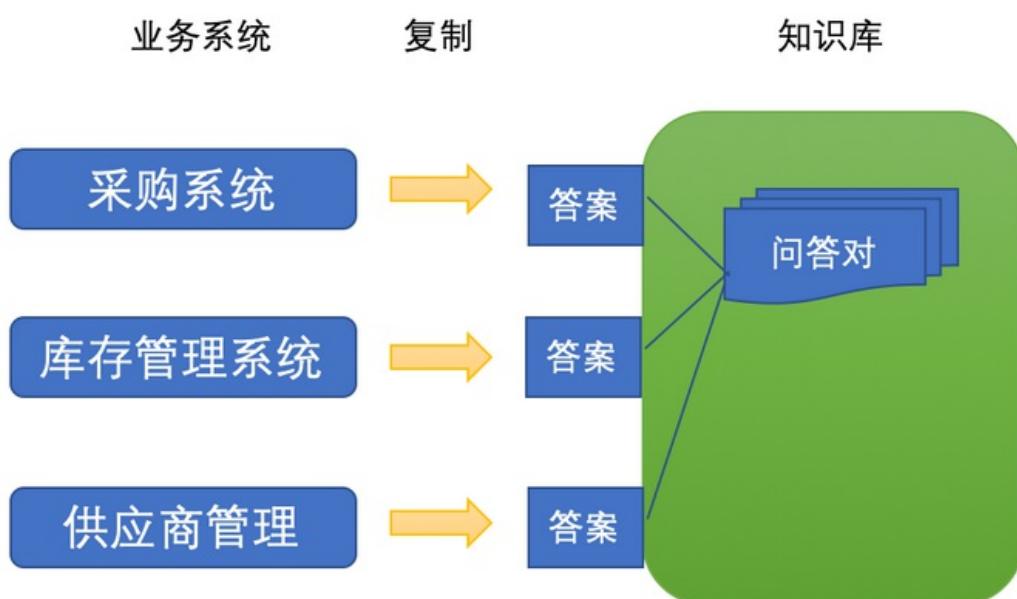
目标

通过使用动态答案，联合多轮对话与知识库的整体使用，使机器人更智能。

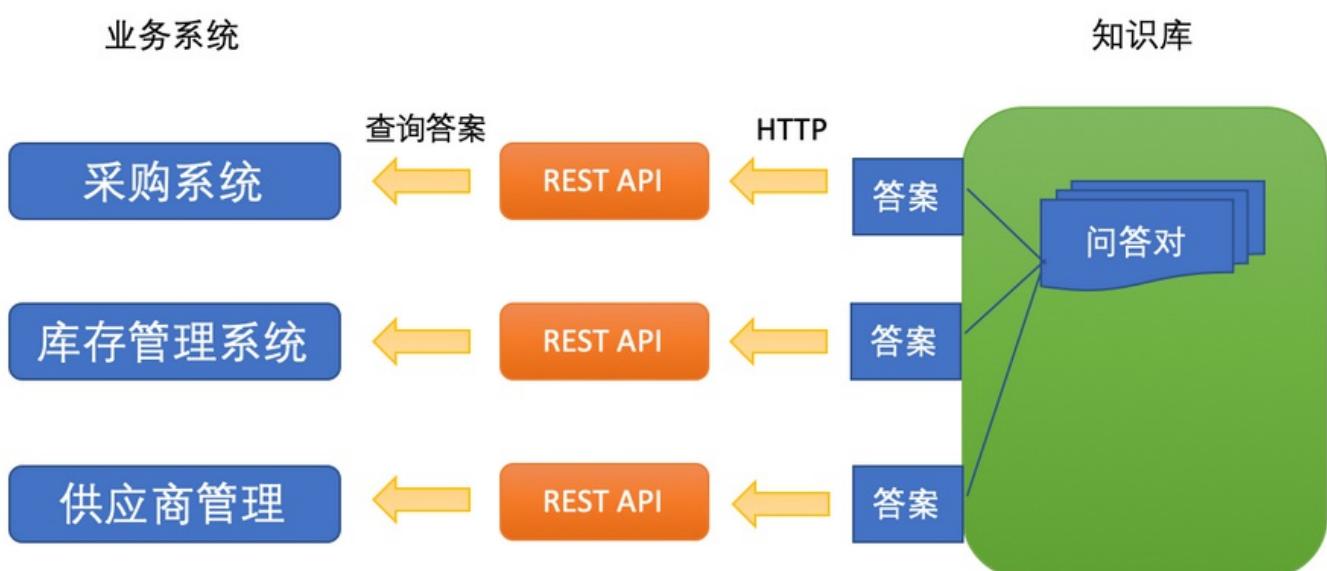
问题描述

在智能对话机器人中，知识库管理可能涉及到从业务系统获得答案，实时的同步答案一个是造成延迟，一个是成本高：在多个地方维护。比如，在业务场景中，有三个子业务系统，分别是【采购系统】/【库存管理系统】/【供应商管理】。每个系统都有一些常见问题，比如采购部门的负责人及联系方式，库存管理系统的负责人及联系方式，这些信息在不同子系统的某个网页上，或者通过 API 调用可以查询到。

在传统方案中，知识库的答案通过手动复制同步，而且这个过程几乎都需要人工完成。



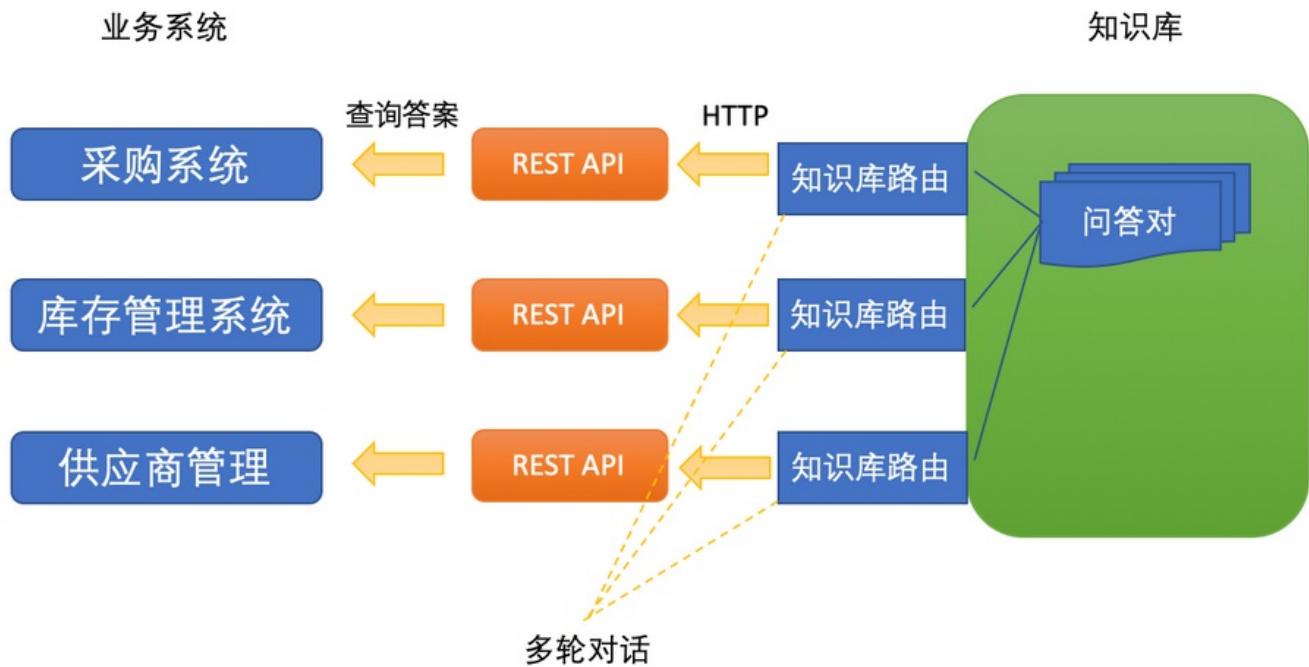
但是，因为有 HTTP API 存在，这个过程可以更简单。一个可行的方案如下：



目前，Chatopera 机器人平台支持这样的使用场景，接下来具体说明。

解决方案

在 Chatopera 机器人平台管理控制台上 BOT 开发者可以维护知识库的不同类型的答案，除了录入固定的答案，Chatopera 机器人平台知识库也支持通过调用[多轮对话的话题](#)来从多轮对话获得答案。在多轮对话中，支持自定义 JavaScript 函数，就使得知识库设置动态抓取的答案可行，称之为：知识库路由。



知识库路由

在知识库的答案或多轮对话的函数中设置回复时，可以用 **routeDirectReply** 来指定一个话题和匹配器获得答案。

语法

将知识库答案按照如下语法设置：

```
routeDirectReply#[ "TOPIC_NAME", "TOPIC_GAMBIT_ID" ]
```

TOPIC_NAME: 对话话题

TOPIC_GAMBIT_ID: 匹配器

使用示例

下面我们用一个示例介绍这个功能的使用。

1) 知识库问题 -

```
采购部门的负责人联系方式是什么？
```

2) 知识库答案 -

```
routeDirectReply#[ "__depart_purch_incharge", "__contact_info" ]
```

在控制台设置如下 -

编辑问题

问题 *

采购部门的负责人联系方式是什么?

选择分类

请选择

相似问题

+ 增加

答案

答1 ×

routeDirectReply#["__depart_purch_incharge", "__contact_info"]

3) 在多轮对话设计器中，设置机器人的回复内容

在机器人管理控制台，打开【多轮对话】版块，点击【下载多轮对话设计器】。

Chatopera Beta

控制台 Test007 创建机器人

多轮对话

- 多轮对话通过 Chatopera 聊天机器人脚本语法实现对话话题。
- 多轮对话设计器是 Chatopera 聊天机器人的集成开发环境。

下载多轮对话设计器 导入 环境变量 对话模板

话题	状态	操作
greetings	启用	查看

安装后，启动应用并通过机器人设置页面的 clientId 和 secret 导入机器人。创建新的技能: `__depart_purch_incharge` 在新的技能中，添加用户说法:

+ __contact_info
- ^get_purch_incharge_contact()

The screenshot shows the Chatopera Multi-round Conversation Designer interface. At the top, it says "话题: __depart_purch_incharge". On the right, there are buttons for "新建话题" and "环境变量". Below the topic, there's a section titled "检索多轮对话 返回值" which contains a JSON object: `{}`. To the right, there's a "对话" (Conversation) window showing a message from "Bot" to "Me": "张三(15801213166)". The "SDK" tab is selected. On the far right, there are two toggle switches: "已启用" (Enabled) and "自动上传" (Automatic Upload). Below these are tabs for "脚本" (Script) and "函数" (Function), with "函数" currently selected. The function code area shows:

```

1 + __contact_info
2 - ^get_purch_incharge_contact()

```

在函数中，添加新的函数。

```

const DATA_URL = "https://gitee.com/chatopera/chatbot-samples/raw/master/assets/demo-contact.json"

// 使用 HTTP 访问 API 动态获得答案
exports.get_purch_incharge_contact = async function() {
  debug("fetchInfo: fetch from remote url %s ...", DATA_URL);
  let resp = await http.get(DATA_URL);
  debug("fetchInfo: ", resp.data)
  return resp.data["name"] + "(" + resp.data["phone"] + ")";
}

```

在多轮对话设计器中进行测试：

The screenshot shows the Chatopera Multi-round Conversation Designer interface again. The "对话" window now displays a message from "Me": "采购部门负责人联系方式" with a paperclip icon, and a response from "Bot": "张三(15801213166)". The "SDK" tab is selected. On the right, the "已启用" (Enabled) and "自动上传" (Automatic Upload) switches are turned on. The "函数" tab is selected, showing the function code. The code has been modified to include a new function "getW". The code is as follows:

```

1 const DATA_URL = "https://gitee.com/chatope
2
3 // 使用 HTTP 访问 API 动态获得答案
4 exports.get_purch_incharge_contact = async
5   debug("fetchInfo: fetch from remote url
6   let resp = await http.get(DATA_URL);
7   debug("fetchInfo: ", resp.data)
8   return resp.data["name"] + "(" + resp.d
9 }
10
11
12 exports.getW = async function(city) {
13   return {
14     text: "helllo, " + city
15   }
16

```

4) 在机器人控制台进行测试

The screenshot shows the Chatopera platform interface. At the top, there's a navigation bar with icons for control panel, Test007, and create robot. Below the navigation bar, there are tabs: '测试对话' (Test Dialog), '知识库' (Knowledge Base), '意图识别' (Intention Recognition), and '多轮对话' (Multi-turn Dialog), with '多轮对话' highlighted by a red box. To the right of these tabs, there are buttons for '知识库路由' (Knowledge Base Routing), '多轮对话' (Multi-turn Dialog), 'SDK API' (SDK API), and '返回值' (Return Value). A large red arrow points from the '多轮对话' button to the JSON code on the right. On the left, there's a '测试' (Test) panel showing a message exchange between a user ('MF') and a robot ('机器人'). The user asks '采购部门负责人联系方式', and the robot responds with '张三(15801213166)'. On the right, a JSON object is displayed:

```
{  
  "state": "default",  
  "createdAt": 1627025951826,  
  "string": "张三(15801213166)",  
  "topicName": "__depart_purch_incharge",  
  "subReplies": [  
  ],  
  "service": {  
    "provider": "conversation"  
  },  
  "logic_is_fallback": false,  
  "botName": "Test007",  
  "faq": [  
  ],  
  "profile": {}  
}
```

下一步，就可以接入到业务系统，比如春松客服。在返回结果中，除了使用文本的形式，还可以支持自定义的数据结构，在春松客服中，返回结果支持展示为卡片、图文、按钮、列表的形式，详细参考 (<https://docs.cskefu.com/docs/work-chatbot/message-types>)。

高级进阶

使用用户输入作为触发器

在一些情况下，按照用户的输入进入多轮对话匹配器更方便，那么在设置路由答案时，使用如下方法。

当 `TOPIC_GAMBIT_ID` 的值为 `$ctx.textMessage$` 时，则使用当前对话的用户输入，在 `TOPIC_NAME` 中进行检索。

举例

1) 知识库问题 -

采购部门的负责人联系方式是什么？

2) 知识库答案 -

```
routeDirectReply#["__depart_purch_incharge", "$ctx.textMessage$"]
```

在控制台设置如下 -

The screenshot shows the Chatopera dashboard interface. On the left sidebar, there are sections for '控制台' (Console), 'Test007', and '知识库' (Knowledge Base). Under '知识库', there are two items: '西红柿的价格' (Tomato price) and '榴莲是几月份的' (When is durian in season). The main area is titled '新建问题' (Create New Question). It has fields for '问题 *' (Question *) containing '采购部门的负责人联系方式是什么?' (What is the contact information of the person in charge of the procurement department?), '选择分类' (Select Category) with a dropdown set to '请选择' (Please select), '相似问题' (Similar Questions) with a '+ 增加' (Add) button, and an '答案' (Answer) section with a '答1' (Answer 1) card containing the code 'routeDirectReply#["__depart_purch_incharge", "\$ctx.textMessage\$"]'. There are '确定' (Confirm) and '关闭' (Close) buttons at the bottom right.

那么，在多轮对话设计器中，这个规则可以更新为：

The screenshot shows the multi-round dialog designer. At the top, there are toggle switches for '已启用' (Enabled) and '自动上传' (Automatic Upload), and a '最近' (Recent) button. Below these are tabs for '脚本' (Script) and '函数' (Functions), with '脚本' currently selected. A list of rules is displayed, with the first rule highlighted. The rule details are as follows:

1	+ 采购部门负责人联系方式
2	- ^get_purch_incharge_contact()

话术助手快速检索话术

快速从机器人知识库检索的输入法！

关键词：快捷语、网络聊天快捷语、网络销售快捷语、知识库快捷、企业输入法、定制知识库、聊天助手、营销话术快捷语、营销话术助手、客服话术助手



Chatopera 知识库话术助手（简称“话术助手”）是面向企业的客户服务人员发布的一款从知识库检索建议回复的桌面软件。长期以来，客户服务占据着企业运营的重要位置，因为回复不及时、不标准和不准确都会给企业造成损失，对于常见问题集的管理，主要使用的工具是 Excel，客服人员想要快速回复客户消息非常困难，同时也有不容易维护的问题，话术助手就是专门用于解决这个问题而开发的。

产品亮点

- 易于配合其他应用，通过复制面板和粘贴面板工作；
- 话术助手工作条支持输入关键词检索，提升话术、问答对检索的便捷性；
- 悬浮窗口，占用很小空间；
- 设计简洁，和桌面背景百搭。

使用教程

使用浏览器打开链接，观看视频：<https://dwz.chatopera.com/A76i8H>

下载和安装

提示: 目前, 话术助手只支持Windows客户端。

下载地址:

版本	操作系统	下载
v1.5	Windows 7 或更高版本	链接

在下载后, 浏览器可能提示“这种类型的文件可能会损坏您的计算机”或者“Windows Defender SmartScreen 已组织启动一个未识别的应用。”，这时请点击“保留”及“更多信息”进行下一步。



配置



双击图标，启动话术助手，看到如下配置界面。



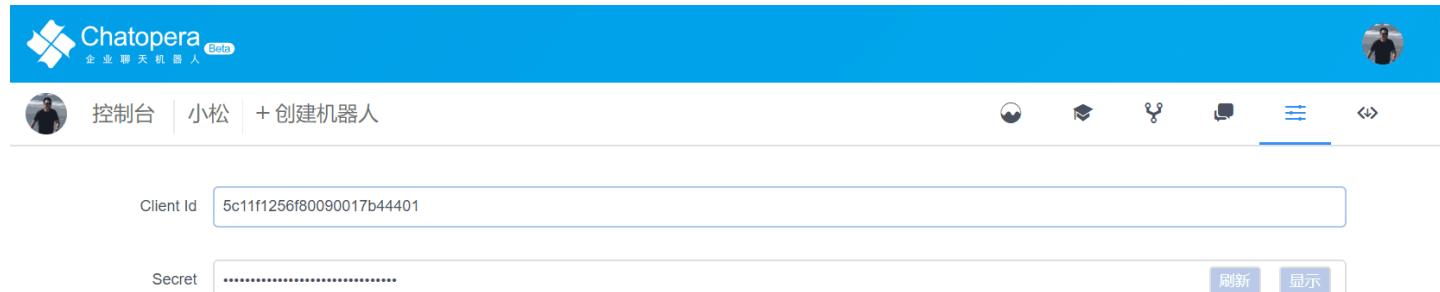
“邮箱”一栏填写有效的电子邮箱地址，该邮箱属于从该 PC 客户端使用 话术助手 的人员，比如客服坐席。客服主管可以从 Chatopera 机器人平台的对话历史中看到该邮箱访问机器人知识库的使用情况。

- 获取应用 ID 和密钥

提示：进入 Chatopera 云服务，机器人管理控制台，设置页面，获得 Client ID 和 Secret。

安装完成后，可以在启动菜单和桌面看到“话术助手”的快捷方式。

已经具备了应用 ID 和密钥的用户跳过此步。作为团队组长/企业主管等角色使用或作为个人用途，可以通过注册 Chatopera 云服务，然后创建聊天机器人获得。

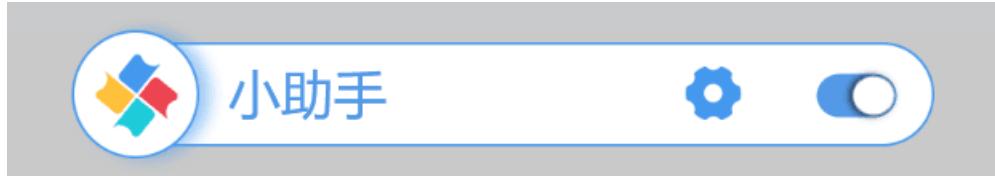


这三项都属于必填项，填写好后点击“保存”。

使用话术助手获得智能建议回复，有两种形式：1) 利用系统粘贴板；2) 在话术助手机具条里输入关键词。

复制到系统粘贴板

保存设置后，话术助手是置顶在桌面的，在话术助手的最右侧是切换启用和停用状态的按钮。



工作时，您可能处于不同的聊天软件中，甚至是文档中，都可以通过复制文本获得建议回复。比如，下面是在QQ群中，快速回复一个问题，就把这个问题复制到粘贴板，通常是通过【Ctrl + C】完成。

这时，话术助手就从机器人的知识库中查询相似问题，并按照问题相似度展开一个智能建议回复列表。点击一个相似问题，就将该问题答案复制到粘贴板，再粘贴该答案到聊天窗口，通常通过【Ctrl + V】完成。

所以，这个过程可以简单的描述为复制->点击->粘贴。

输入关键词

保存设置后，话术助手机具条内显示“请输入搜索关键字”，在此录入文本后，通过回车键或右侧放大镜都可以提交搜索。获得建议回复。



点击取得的答案，答案内容被复制找系统粘贴板，然后可以再粘贴到其它地方，通过【Ctrl + V】完成。

临时停用

话术助手处于启用状态时，每次执行文本复制操作都会做智能建议回复的查询，被复制内容会被发送到远程服务，为了保护用户隐私，您可以临时停用话术助手，这时并不执行智能建议回复，您的隐私数据也就得到保护。在任何状态，任何理由下，Chatopera 不会在未取得用户允许的情况下，获取或利用用户的隐私数据。



退出

彻底关闭话术助手，在话术助手的悬浮区域右键，在弹出的菜单中点击“退出”。



开源地址

Chatopera 知识库话术助手程序目前已经开源，项目地址：

<https://github.com/chatopera/assistant>

对该工具的任何建议和优化，在 Issues 中和我们交流。

<https://github.com/chatopera/assistant/issues>

创建和训练意图识别模型

意图和词典的关系

进入机器人意图管理页面，新建意图，意图的编辑包括用户说法和槽位。在添加意图的槽位时需要对词典进行配置，包括新建自定义词典或引用系统词典，参考[词典的管理文档](#)。

需要强调的是，在词表表词典创建后，默认是没有词条的，这种情况会导致训练失败；相似的，正则表达式词典也需要有表达式定义，否则会导致机器人训练失败。在遇到训练失败时，通过错误提示消息进行解决，在保存词典，保存意图时，会进行训练。

意图命名规则

- 意图标识名为字母、数字、下划线的组合，1-32位，如：RailTypes
- 同一个机器人下的意图标识名不能重复
- 意图标识名一经确认后无法进行修改

创建：

在意图管理页面点击新建自定义词典，在弹出的对话框里输入正确的自定义词典名称，点击确定。与新建自定义词典一样

编辑意图和槽位

添加槽位：添加槽位时可以选择应用系统词典或自定义词典，可以设置是否必填和追问，必填的话必须设置追问。



控制台 | ceshi | + 创建机器人



← eat

测试

用户说法 ①

输入用户可能的说法, 用{}使用槽位, 例如: 我想预订(CityName)的机票

添加

按Enter键可以快速添加

用户说法

操作

我想吃饭

删除

我要吃{food}

删除

我饿了

删除

槽位 ①

槽位名称 ① time

词典 ①

@TIME

必填 ①



追问 ① 什么时候

添加

槽位名称

词典

taset

taset

food

food

food

num

taset

引用系统词典

@TIME

@LOC

您可以 新建词典 或 引用系统词典

操作

删除

你想吃啥

删除

保存

取消

删除槽位: 对于不需要的槽位可以进行删除, 点击删除按钮即可。

修改槽位: 对于需要修改的槽位, 直接进行编辑和修改即可。

槽位 ①

槽位名称 ① 例如: CityName 词典 ① 请选择 必填 ①

追问 ① 例如: 请问是哪个城市

添加

槽位名称	词典	必填	追问	操作
taset	taset	<input checked="" type="checkbox"/>	有啥口味要求	<input type="button" value="删除"/>
food	food	<input checked="" type="checkbox"/>	你想吃啥	<input type="button" value="删除"/>
time	num	<input checked="" type="checkbox"/>	什么时候	<input type="button" value="删除"/>

自定义词典
taset
food
num
引用系统词典
@TIME
@LOC

红色箭头指向“词典”输入框右侧的下拉菜单。

保存 取消

添加说法: 对于用户的说法, 可以添加没有槽位的说法, 也可以用{}关联槽位, 把槽名称放到{}里面即可, 一个说法可以绑定多个槽位。

[← eat](#)[测试](#)

用户说法 [?](#)

输入用户可能的说法, 用{}使用槽位, 例如: 我想预订{CityName}的机票

[添加](#) 按Enter键可以快速添加

用户说法	操作
我想吃饭	删除
我要吃{food}	删除
我饿了	删除

槽位 [?](#)

槽位名称 [?](#) 例如: CityName 词典 [?](#) 请选择 必填 [?](#)

追问 [?](#) 例如: 请问是哪个城市

[添加](#)

槽位名称	词典	必填	追问	操作
taset	taset	<input checked="" type="checkbox"/>	有啥口味要求	删除
food	food	<input checked="" type="checkbox"/>	你想吃啥	删除
time	@TIME	<input checked="" type="checkbox"/>	什么时候	删除

[保存](#)[取消](#)

删除说法: 点击删除按钮即可。

修改说法: 直接进行编辑和修改即可。

保存意图

在编辑完所有说法和槽位之后, 需要进行保存。保存并且训练成功有提示。成功就可以到测试对话页面进行测试。

The screenshot shows the Chatopera Cloud Service interface for creating and training an intent. The top navigation bar includes the Chatopera logo, user information, and various icons for account management and integration.

The main content area is titled 'eat'. It contains two sections:

- 用户说法 (User Phrases):** A text input field for entering user phrases, with a note: "输入用户可能的说法, 用{}使用槽位, 例如: 我想预订{CityName}的机票". Below it is a "添加" (Add) button and a note: "按Enter键可以快速添加". A table lists three user phrases:

用户说法	操作
我想吃饭	删除
我要吃{food}	删除
我饿了	删除
- 槽位 (Slots):** A configuration section for slots. It includes fields for slot name (e.g., CityName), dictionary selection, and required status. A note says: "例如: 请问是哪个城市". Below is a table for slot definitions:

槽位名称	词典	必填	追问	操作
taset	taset	<input checked="" type="checkbox"/>	有啥口味要求	删除
food	food	<input checked="" type="checkbox"/>	你想吃啥	删除
time	@TIME	<input checked="" type="checkbox"/>	什么时候	删除

At the bottom, there are buttons for "保存" (Save) and "取消" (Cancel), and a success message: "训练成功, 可进行测试" with a red arrow pointing to it.

测试和发布机器人

保存机器人词典、意图等信息后，Chatopera 云服务会提示开始重新训练机器人，训练会有成功或失败两种结果，训练会持续一段时间，时间长短取决于意图、词表、槽位的数量；完成训练后，代表机器人测试版本已经用最新的数据更新，在测试页面，导航至“意图识别”，和测试版本进行对话。

发布上线意图识别生产版本

调试版本

每次保存并且训练成功后，都会自动更新调试版本。但是此时不该急于发布上线，需要多次在测试对话页面进行测试，确保无误之后，再发布上线。

测试：在测试对话页面与机器人进行对话，在右侧会把对应的意图、槽位以及槽位值之类的信息展示出来。

The screenshot shows the Chatopera testing interface. On the left, there is a conversation log window titled "测试" (Test) with the following messages:

- User: 我要吃面条
- Robot: 有啥口味要求
- User: 加辣
- Robot: 什么时候
- User: 晚上5点

Below the conversation log, a message says "会话已完成, 请建立新的会话" (Conversation completed, please start a new one).

On the right, there is a "回复信息" (Response Information) section for the "eat" intent. It shows the following table:

序号	槽位	值	是否必填	绑定词典
1	taset	加辣	是	taset
2	food	面条	是	food
3	time	晚上5点	是	@TIME

生产版本

测试无误后便可在【意图>>发布管理】模块对调试版本进行发布上线，发布后的线上版本可用于实际使用，线上的集成通过集成页面获取 SDK 和使用文档。



版本管理

- 每个机器人包含调试和生产两个版本。
- 每次机器人训练通过后，将自动更新调试版本。
- 调试版本发布上线后，将覆盖当前生产版本，生产版本用于实际的集成应用。

调试版本

版本 ID	修改时间	操作
16d0b9d3a4e	2019-09-07 20:05:17	发布上线

生产版本

版本 ID	上线时间
暂无筛选结果	

多轮对话设计器安装

多轮对话设计器 是开发聊天机器人的桌面软件，全称 Conversation Design Environment, 简称为 CDE。

安装

- 下载后，得到应用安装包，双击后根据提示进行安装
- 安装过程中遇到问题先查看本文下面的“可能遇到的问题”一节，如遇新问题[创建工作单](#)

版本	下载地址
v2.5.7	macOS Windows

提示：点击下载地址未能开始下载时，将下载地址拖拽到新的 Tab 页，启动下载。

工具介绍

多轮对话设计器支持 Chatopera 的对话机器人脚本语法描述复杂的对话逻辑，并且通过函数的形式集成企业的其它服务。

多轮对话设计器是第代码开发有强大对话能力的机器人的高级工具，不需要熟悉甚至了解自然语言处理、机器学习。IT 开发者或业务人员可以很容易的学习脚本语法，制作满足各种需求的聊天机器人。

多轮对话设计器是设计满足业务需求的聊天机器人的 PC 端应用程序，现已支持 Windows 和 macOS 平台。多轮对话能力是聊天机器人模仿人的对话能力的一大挑战，在复杂的上下文和需要很多背景知识的前提下，现有的人工智能技术是无能为力的，在 Chatopera，我们相信在企业服务中，当话术或流程固定的情况下，依赖 Chatopera 的产品可以输出用对话完成任务的服务，比如用对话完成点餐、报销、请假。这些对话可以在企业的聊天工具中，也可以通过智能音箱的等其他客户端。

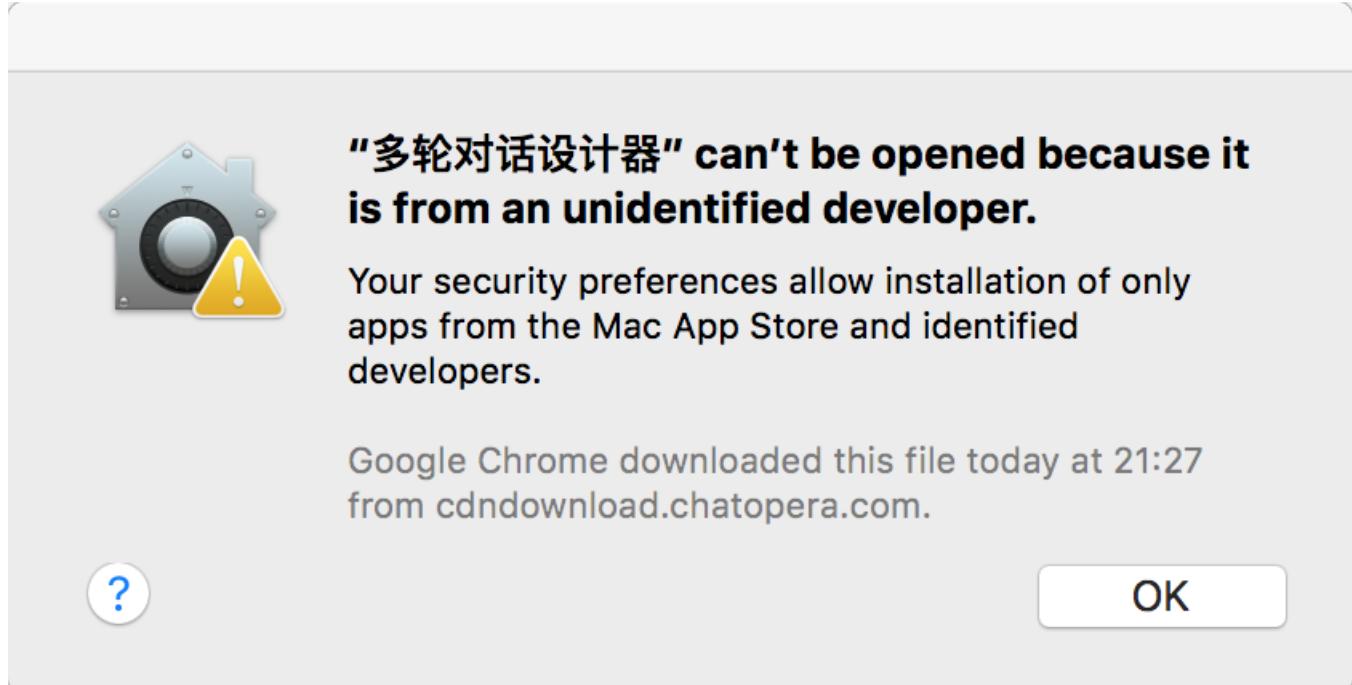
启动应用

- 安装完打开应用程序，如下图：



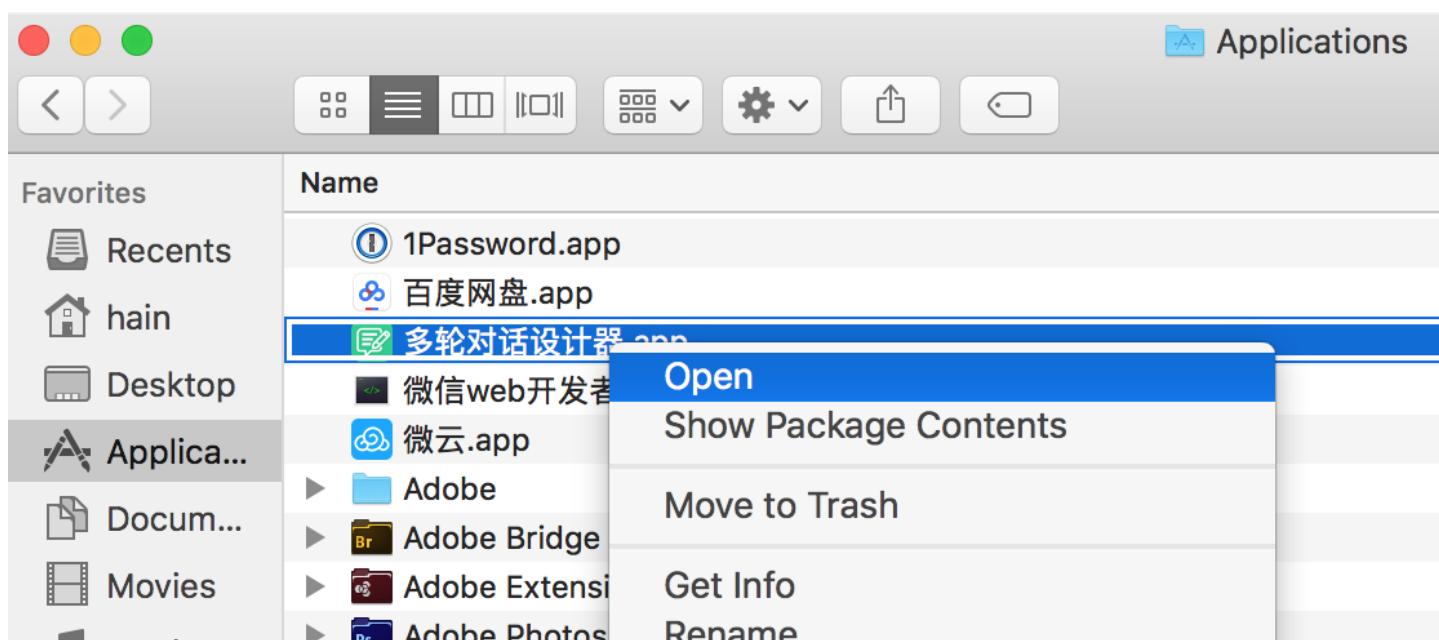
可能遇到的问题

macOS 上安装和启动的提示

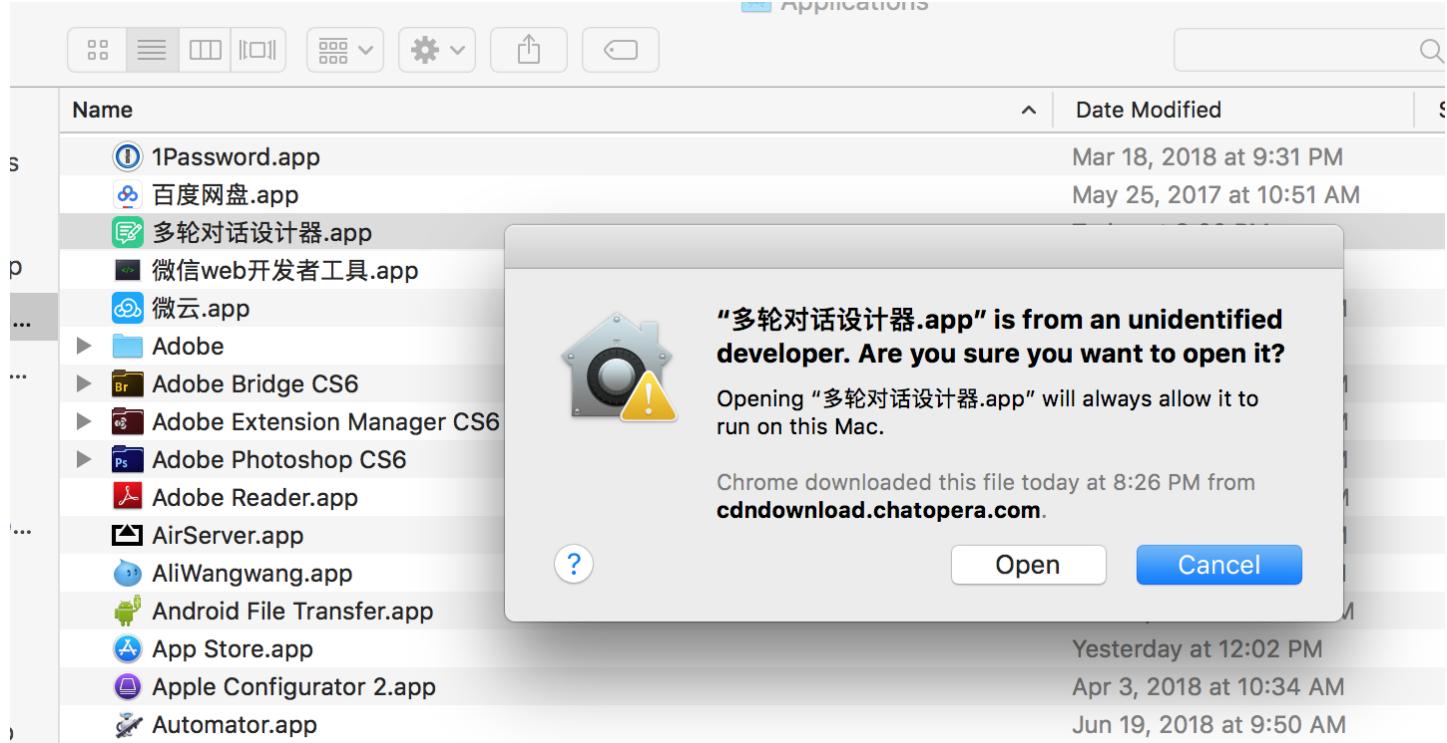


解决方案

在应用中心，找到“多轮对话设计器”：



选择“多轮对话设计器”并且右键：



点击“打开”("Open")。

通配符匹配器

- + [今天] 晚上吃什么
- 烤鸭

通配符匹配器 / Star Gambit

为了让**匹配器**能适应复杂的需求，Chatopera 机器人平台使用通配符规则，通配符既能让规则具有更好的匹配能力，也能让[回复和函数中使用不同通配符的值，即通配符取值](#)，基于通配符的匹配器称为通配符匹配器，也称为**Star Gambit**。

注意：下面的某些通配符左右带有空格，这些空格是必须的。

全能匹配

全能匹配通配符会匹配零到无穷个字符、单词。

(*)

此处的输入也会被系统捕获或者存储。

- + (*) 你好 (*)
- 欢迎光临

对话效果

匹配：客服你好
匹配：你好
匹配：你好吗

固定长度通配符

如果你知道你想要的字符长度，使用固定长度通配符。

*n

其中 n 代表长度。

此处的通配符可以被系统捕获，而且可以在回答中使用。

- + 早安 *2
- 早安

对话效果

匹配: 早安北京

不匹配: 早安乌鲁木齐

可变长度通配符

*~n

如果只想匹配一些字符, 可控长度的通配符是个不错的选择。n 代表你想匹配的最大长度。

+ 早安 *~4

- 早安

对话效果

匹配: 早安

匹配: 早安北京

匹配: 早安哈尔滨

匹配: 早安乌鲁木齐

不匹配: 早安君士坦丁堡

区间通配符

如果想匹配一个确定的区间, 比如 2 到 4 个字符之间, 区间通配符绝对可以满足需要。

*(最短-最长)

此通配符可以被系统捕获和用在回复中。

+ 早安 *(2-4)

- 早安

对话效果

匹配: 早安北京

匹配: 早安乌鲁木齐

不匹配: 早安

词性通配符

在匹配器中, 在某个位置使用词性来匹配一个词汇集合, 比如名称、动词等。被匹配到的词, 可以使用通配符取值在回复中使用。

比如:

+ <noun>是个好地方

- 嗯

+ <noun>很好吃

- ^echoYummy(<cap>)

```
exports.echoYummy = async function(cap1) {
    return "那就多吃点" + cap1
}
```

对话效果

匹配: 香港是个好地方
匹配: 蛋糕很好吃

如果在一句话中存在多个要匹配的词性, 使用 `<noun1>`, `<noun2>` 进行标记, 就是增加序号。

支持的词性

词性	词性匹配器	含义	支持语言
名词 / Noun	<code><noun>, <noun1>, <noun2> ... <nounN>, <nouns></code>	组织机构, 地名, 人名等	机器人所有语言
形容词 / Adjective	<code><adjective>, <adjective1>, <adjective2> ... <adjectiveN>, <adjectives></code>	比如: 美好, 顺利, 难忘	机器人所有语言
动词 / Verb	<code><verb>, <verb1>, <verb2> ... <verbN>, <verbs></code>	比如: 跑步, 游泳	机器人所有语言
副词 / Adverb	<code><adverb>, <adverb1>, <adverb2> ... <adverbN>, <adverbs></code>	比如: 急忙, 快速	机器人所有语言
代词 / Pronoun	<code><pronoun>, <pronoun1>, <pronoun2> ... <pronounN>, <pronouns></code>	比如: 我, 他, 她	机器人所有语言
人名 / Names	<code><name>, <name1>, <name2> ... <nameN>, <names></code>	比如: 张三, 李四	简体中文, 繁体中文, 英文

更多例子

- + `<noun>`是个好地方
 - 嗯
- + `<noun>`很好吃
 - ^echoYummy(`<cap1>`)
- + 今天是`<adjective>` (*) 一天
 - 美好!
- + `<verb>`对健康有好处
 - 喜欢`<cap1>`
- + `<adverb>`地躲避
 - 机智
- + `<pronoun>`在张望什么
 - 等人
- + `<name>`有什么爱好
 - 旅游

```
exports.echoYummy = async function(cap1) {
  return "那就多吃点" + cap1
}
```

必选项

必选项用在你有一系列可选项, 但是必须有一个被匹配。

(候选-1|候选-2|...|候选-n)

输入中的可选项会被系统捕获和用在回复中。

- + 早安(北京|上海|天津)
- 早安

匹配: 早安北京

不匹配: 早安西安

不匹配: 早安

可选项

可选项用来确定一些额外的内容。

[可能值-1|可能值-2]

- + 早安 [美丽的] 北京
- 早安

注意: 此处 [] 前后的空格不可省略。

匹配: 早安北京

匹配: 早安美丽的北京

不匹配: 早安热闹的北京

下一步

- 通配符取值: 在回复中使用匹配到的值
- 模糊匹配器: 容错能力更强和智能的匹配器
- 意图匹配器: 借助意图识别模块, 轻松实现任务型对话

使用模糊匹配器

```
+ ${商场几点开始营业|商场几点开门}  
- 营业时间 : 8:00 AM – 11:00 PM
```

模糊匹配器 / Like Gambit

也称为 Like Gambit，使用容错能力更强的语法定义规则。

- 模糊匹配器的设计初衷和背景知识

模糊匹配器的功能

模糊匹配器的主要特点是支持设定多个句子，输入和句子进行匹配，根据阀值判定是否匹配。有默认阀值，Chatopera 机器人平台可以在机器人管理控制台设置页面，调整该阀值，也可以在每个匹配器上设定单独的阀值；该阀值的取值需要介于 0 和 1 之间，越高代表越严格的匹配，越相似。

基于模糊匹配器，匹配器的容错能力更强，调试更简单；但是模糊匹配器不考虑位置的因素，这一点与通配符匹配器较为不同。

模糊匹配器和通配符匹配器各有优劣，Chatopera 机器人平台开发者在看过不同的匹配器示例后，根据需要灵活选择。

接下来介绍模糊匹配器的详细使用。

模糊匹配器的语法

形式 1

```
+ ${threshold}{句子1|句子2|...|句子N}  
- 回复
```

句子可以设定 1 或多个，多个句子时使用 " | " 进行分隔。threshold 的取值范围是 $(0, 1)$ 。

比如：

```
+ ${0.7}{商场几点开门|商场营业时间}  
- 营业时间：早上八点半至晚上十一点
```

形式 2

```
+ ${句子1|句子2|...|句子N}  
- 回复
```

比如：

- + \${商场几点开门|商场营业时间}
- 营业时间: 早上八点半至晚上十一点

在这种形式下, 省略了形式 1 里的阀值 threshold, 那么在计算输入和句子1, ..., 句子N 的是否匹配时, 使用的阀值是机器人控制台【设置页面】设定的值【多轮对话模糊匹配阀值】, 默认为 0.8。

The screenshot shows the 'Conversation Model' settings for a robot named 'BOT202108-2'. It includes fields for Client Id (611b9a96d1164f001b9c095b), Secret (redacted), Language (en_US), and Description (Please describe the robot). The 'Conversation Model' section contains fields for Welcome Message (My super power is talk.), Bottom Response (I don't understand.), Session溯最长时长 (1800), Session溯最大轮次 (10), and a highlighted 'Multi-round dialog fuzzy matching threshold' set to 0.9. A note in the interface states: 'In multi-round dialog search for replies, when using fuzzy matching (like Gambit), the default confidence, and the similarity of the input text exceeding the confidence rule is considered a match; the fuzzy matching can cover the default value, such as \${0.7}{句1|句2}'.

匹配器及回复复用

模糊匹配器受机器人记忆模型限制, 如果需要在一个记忆周期内一直生效, 可以使用 {keep} 标记, 比如:

- + \${商场几点开门|商场营业时间}
- {keep} 营业时间: 早上八点半至晚上十一点

详细介绍, 参考[文档](#)。

使用调试

对于一条模糊匹配器规则, 究竟要适应多少种问法? 容错是保守一点, 还是要非常灵活? 这需要在调试机器人时, Chatopera 机器人平台用户, 自行判断。在多轮对话设计器内, 可以增加模糊匹配器里的句子, 或者适当的调节阀值 threshold。

调试过程

- 1) 开启自动上传;
- 2) 撰写模糊匹配规则, 设定答案;
- 3) 保存;
- 4) 在对话中, 发送测试文本;
- 5) 查看日志, 得到匹配调试信息。

如下图:

The screenshot shows the Chatopera Multi-round Conversation Designer interface. On the left, there's a code editor for an 'SDK' file containing JSON configuration for a 'greetings' topic. In the center, a conversation log shows a Bot asking '开心' (Happy) and '去哪里玩' (Where to play), and a User replying '商场开门时间' (Business hours). The Bot then asks '9点半' (9:30 AM). On the right, a code editor shows a script with numbered annotations:

- 1. 启动自动上传**: Shows a toggle switch labeled '自动上传' (Automatic upload) being turned on.
- 2. 编写规则**: Shows a red box around a section of the script defining rules for the 'greetings' topic.
- 3. 测试对话**: Points to the user message '商场开门时间' in the conversation log.
- 4. 查看日志，得到调试信息**: Points to the log viewer at the bottom showing debug logs related to the query.

日志

```
[saas_611b9829d1164f001b9c095a], branch [master], requirePlugin
2021-08-17 19:27:46 INFO conversation loadPlugin chatbotID
[saas_611b9829d1164f001b9c095a], branch [master]
2021-08-17 19:27:46 INFO conversation loadPlugin chatbotID
[saas_611b9829d1164f001b9c095a], branch [master], requirePlugin customPlugins []
2021-08-17 19:27:46 DEBUG conversation [DoesMatch] like gambit(threshold 0.7): ${0.7}{商场几点开门} -matched-> 商场开门时间
2021-08-17 19:27:46 DEBUG conversation [DoesMatch] like gambit(threshold 0.7): ${0.7}{商场几点开门} -similarity-> 1. score(0.71029) : 商场几点开门
```

调试信息

关于模糊匹配器在调试时，得到的调试信息，进一步介绍。

每个模糊匹配器，都会和输入进行比较，计算相似度，打印的日志类似：

```
2021-08-18 13:52:52 DEBUG conversation [DoesMatch] like gambit(threshold 0.8): ${商场几点开门} -similarity-> 1. score(1) : 商场几点开门
```

其中，`DoesMatch` 是服务端计算该相似度的函数名，`like gambit(threshold)` 表示这个匹配器是模糊匹配，并输出本次的阀值，后面就是匹配器内容，`-similarity->` 后的输出是每个句子和输入内容的相似度，`score()` 内就是相似度，分数后面是句子的分词结果；多个句子时，使用“|”分隔。

比如：

```
2021-08-18 13:57:18 DEBUG conversation [DoesMatch] like gambit(threshold 0.8): ${商场几点开门|商场开门时间} -similarity-> 1. score(1) : 商场几点开门 | 2. score(0.71029) : 商场开门时间
```

以上句子之间增加了序号，按照分数由大到小进行排列。

如果模糊匹配器的一个句子与输入之间相似度大于阀值，那么就判定为匹配，输出日志类似如下：

```
2021-08-18 13:57:18 DEBUG conversation [DoesMatch] like gambit(threshold 0.8): ${商场几点开门|商场开门时间} -matched-> 商场几点开门
```

模糊匹配器取值

如果匹配上模糊匹配器，回复的函数中，可以得到哪些信息，进一步的让回复内容更加智能？

模糊匹配器上，函数中利用输入的自然语言处理办法：使用 `this.message` 内包含的信息进行分析。

假如有这样的脚本和函数：

```
/**  
 * 模糊匹配器示例脚本  
 */  
  
+ ${商场几点开门|商场开门时间}  
- ^getOpenTime()
```

```
exports.getOpenTime = async function() {  
    debug("getOpenTime words %j", this.message.words) // this.message.words 是 JSONArray  
    debug("getOpenTime tags %j", this.message.tags) // this.message.tags 是 JSONArray  
    return "九点半"  
}
```

那么，就可以在 this.message.words 中得到分词的信息，在 this.message.tags 得到相对应位置的词性。

比如，如果输入是“商场的开门时间”，那么相应的 words 和 tags 如下：

```
getOpenTime words: ["商场", "的", "开门", "时间"]  
getOpenTime tags: ["nis", "ude1", "vi", "n"]
```

所以，“商场”的词性是“nis”，“的”的词性是“ude1”。那么，“nis”和“ude1”又代表什么呢？它们是词性标识。

在 Chatopera 机器人平台，不同语言使用词性标识因为语言本身的原因，不是很一致。

语言	词性标注集
中文（简体中文 zh_CN，繁体中文 zh_TW）	文档
英文 / en_US / Enlgish	文档
日语 / Japanese	文档

得到匹配信息，再结合 Chatopera 机器人平台内其它的内置函数库，可以实现更为强大的对话能力！

通过提供不同形式的匹配器，适合智能对话机器人开发者在不同场景下因地制宜的选择构建规则的方法。

接下来

- [有关更多关于 this.message 的使用介绍](#)
- [下载多轮对话设计器](#)
- [理解多轮对话工作机制](#)
- [检索多轮对话：使用多轮对话接口进行系统集成](#)
- [通配符匹配器：使用语法建立规则](#)
- [意图匹配器：借助意图识别模块，轻松实现任务型对话](#)

使用意图匹配器

```
intent book_cab
- ^orderCab()
- {x} ^loseOrderCab()
```

意图识别匹配器 / Intent Gambit

也称为 Intent Gambit，在多轮对话脚本中使用 [意图识别模块](#) 实现任务型对话。

- 意图匹配器的设计初衷和背景知识

意图匹配器语法

语法（注意语法中存在的空格）：

```
intent INTENT_NAME
- ^succHandlerFn()
- {x} ^loseHandlerFn()
```

在意图匹配器中，开始必须是 "intent"，用来标志后面的内容是面向意图匹配器。

intent INTENT_NAME 就是意图匹配器，INTENT_NAME 是意图识别模块中，意图的名字，大小写不敏感，这意味着 book_cab 和 BOOK_CAB 是一样的。

意图

- 意图是用户与机器人进行对话背后的目的，是用户希望完成的任务。
- 查看 [快速开始](#)，掌握意图识别模块上线聊天机器人过程。
- 意图对话的结果是返回意图和槽位信息，开发者通过SDK集成到其它系统进一步完成任务。

新建意图

版本管理

意图识别模块调试分支已训练

意图标识名	意图中文名	更新时间	创建时间	操作
book_cab	/	2021-08-19 15:05:58	2021-08-19 15:05:58	编辑 删除

共 1 条

<

1

>

INTENT_NAME / 意图名字

然后，在下一行，使用 "-" 开始设定回复，因为需要处理意图识别信息，而该信息是传递给函数的，所以，回复设置为自定义的函数。在本文后面的部分，会详细说明在函数中取值。

在意图匹配器下，有两种类型的回复：

1) 成功函数：识别到意图，并在对话交互中得到所有的必须的槽位信息，调用“成功函数”；

2) 失败函数：识别到意图，但是追问交互中，在限定的追问次数下，没有得到必须的槽位信息，在这种情况下，会使用对话用户最后的检索，从知识库和对话脚本中，继续检索，如果得到答案，则回复该答案；如果没有得到答案，则调用“失败函数”。

“失败函数”有一个特殊的标记：`{x}`。即一对大括号内写入小写字母 `x`。失败函数是选填的，如果没有定义则会回复兜底回复。在失败函数内，也是可以取得当前意图识别的信息。

举例

```
intent book_cab
- ^orderCab()
- {x} ^loseOrderCab()
```

以上，`orderCab` 是成功函数，`loseOrderCab` 是失败函数。

成功函数和失败函数的执行时刻的更多信息，参考[多轮对话工作机制](#)。

以上就是意图匹配器的脚本语法，接下来介绍函数中，使用意图识别信息、配置意图识别分支等更高级功能点。

意图会话生命周期

当一个意图被识别到，对话进入意图识别对话，就会创建一个意图会话。意图会话，在下面四种情况下会失效：

空闲时间太久而过期

对话用户在【会话回溯最大时长】时间下未发送请求，意图会话因为空闲时间太久而过期；

即【会话回溯最大时长】，当对话用户，超过一段时间不和机器人交互，那么会话状态被清空，包括意图会话。同样，进入 BOT 管理控制台的设置页面调整该值，默认为半个小时。

新识别到的意图覆盖意图会话

对话从意图识别模块识别新的意图，就是又匹配上了新的意图，旧的会话周期被覆盖；

意图识别调试分支重新训练

意图匹配器，通过在多轮对话脚本中的环境变量 `@SYS_INTENT_BRANCH` 的值来指定集成[意图识别的版本](#)，即调试版本或生产版本，二者的值分别是 `pro` 和 `dev`，默认为 `dev`。

调试版本，顾名思义，是开发测试时使用，每次重新训练，其对应的意图会话都会被删除，在使用意图匹配器时，就会造成多轮对话在上下文处理上的失败。所以，每次意图识别重新训练，如果多轮对话集成的是调试分支，则意图会话的逻辑就失败了。

因为意图匹配器默认就是使用调试分支，所以，对于生产环境的 BOT，有如下建议：

1) 发布训练好的意图识别调试版本为生产版本，进入机器人意图管理页面，点击【版本管理】，点击【发布上线】。

The screenshot shows a 'Version Management' interface. At the top, there's a note: '每个机器人包含调试和生产两个版本。每次机器人通过后，将自动更新调试版本。调试版本发布上线后，将覆盖当前生产版本，生产版本用于实际的集成应用。' Below this, there are two sections: 'Debug Version' and 'Production Version'. In the 'Debug Version' section, a table lists a single row with ID '17b0ff8c821', modified time '2021-09-07 19:15:58', and an 'Release Online' button. A red arrow points to this button. The 'Production Version' section shows a table with no results.

2) 在多轮对话设计器或 Chatopera 机器人平台管理控制台，进入环境变量设置。

修改 `@SYS_INTENT_BRANCH` 的值为 `pro`。

```
@SYS_INTENT_BRANCH=pro
```

这时，因为使用了意图识别生产版本，不会因为调试分支的变动而影响意图会话。

函数内丢弃意图会话

在多轮对话函数中，包括成功函数、失败函数或其它的匹配到的脚本函数，使用 `this.intent.drop = true` 设置后。

对于第三条，这意味着，需要 BOT 开发者在成功的处理了意图信息后，设置 `this.intent.drop = true`，就是放弃当前意图信息。

采用这个设计的原因是，意图会话的信息处理，这项权利尽量的保留给 BOT 开发者，在函数中实现，可以满足更多业务需求，让 BOT 更智能。

使用函数完成对话

意图识别可以在多轮对话设计器 v2.2.0+ 中使用，但是基于对多轮对话设计器用户体验的升级，建议使用 v2.3.0+ 版本的多轮对话设计器，最新版本[下载地址链接](#)。

编辑意图识别对话窗口如下：

话题: greetings ▾

[新建话题](#)[环境变量](#)[绑定信](#)

搜索多轮对话 返回值

对话

已启用话题

```
{
  "state": "default",
  "string": "目的地是哪里？",
  "botName": "dev1078",
  "logic_is_unexpected": false,
  "logic_is_fallback": false,
  "service": {
    "provider": "intent",
    "intent": {
      "branch": "dev",
      "state": "proactive",
      "entities": [
        {
          "name": "originLoc",
          "val": "深圳",
          "requires": true,
          "dictName": "@LOC"
        }
      ]
    }
  }
}
```



脚本

函数

```

1 /**
2 * 意图识别对话示例
3 * https://github.com/chatopera/chatbot
4 */
5
6
7
8 intent book_cab
9 - ^orderCab()
10 - {x} ^loseOrderCab()

```

日志

```

branch [master] exist in database, but not syncd with memory, load it into memory
2021-09-01 19:42:51 INFO conversation loadPlugin chatbotID
[saas_611325bed908ba001bc659da], branch [master], plugin name: [loseOrderCab]
2021-09-01 19:42:51 INFO conversation loadPlugin chatbotID
[saas_611325bed908ba001bc659da], branch [master], plugin name: [orderCab]
2021-09-01 19:42:51 DEBUG conversation [doesGambitMatch] intent gambit(threshold-0.9)
matched: book_cab(1) | branch dev
2021-09-01 19:42:55 DEBUG conversation [query] conversation request { "fromUserId": "superadmin", "textMessage": "深圳" }

```

在脚本和函数编辑区域，使用语法匹配意图识别和处理意图识别信息；对话窗口进行测试，在日志中查看日志和 `debug` 信息；在 SDK 中查看返回值 JSON 数据。

意图匹配器取值

如上文，有一个例子：

```

intent book_cab
- ^orderCab()
- {x} ^loseOrderCab()

```

当对话用户的发送文本匹配上了“book_cab”中的说法，比如“我想打车”。多轮对话即启用【book_cab】内的对话流程，意图识别对话在成功获得该意图的必须的槽位信息，或超过了设置的最大追问次数，则会进入多轮对话函数，前者会调用【成功函数】，后者会调用【失败函数】。前文，做过相关描述。那么，成功函数和失败函数的内，如何获得意图识别对话的信息呢？

成功函数

成功函数，比如上文例子中的 `orderCab` 不需要在对话脚本中设定任何参数，函数执行的时候，会将意图识别信息，加入该函数的 `this` 命名空间下，使用 `this.intent` 读取。

`this.intent` 是一个 JSON 数据，数据格式如下：

```
{  
    "name": "{{INTENT_NAME}}",  
    "topicName": "{{TOPIC_NAME}}",  
    "branch": "{{INTENT_BRANCH}}",  
    "entities": [  
        {  
            "name": "{{ENTITY_NAME}}",  
            "val": "{{ENTITY_VALUE}}",  
            "requires": true,  
            "dictname": "{{DICT_NAME}}"  
        },  
    ],  
    "state": "{{STATE}}"  
}
```

INTENT_NAME: 意图识别的名字

TOPIC_NAME: 对话脚本话题的名字

INTENT_BRANCH: 集成意图识别的版本的分支, 默认为 `dev`, 如果使用意图识别生产版本分支, 在多轮对话中设置环境变量 `@SYS_INTENT_BRANCH` 为 `pro`

`ENTITY_NAME`, `ENTITY_VALUE`, `DICT_NAME`: 该意图下的关键的参数, 槽位信息; `requires` 为 `true` 时, 代表该槽位是必填的

`STATE`: 状态, 包括 `resovled`, `proactive`, `losed` 三个状态, 分别代表意图全部必填参数识别完成, 正在追问参数信息和追问超过最大次数

除了增加了 `this.intent`, 其它成功函数的使用和多轮对话函数是一致的。比如, 设定回复:

```
// 回复只有文本  
exports.orderCab = async function() {  
    debug("[orderCab] intent %s", JSON.stringify(this.intent));  
    // 处理下单信息  
    // ...  
    this.intent.drop = true; // 成功下单, 设置当前意图会话过期  
    return "下单成功, 订单编号: SE111."; // 设定回复  
  
}  
  
// 回复包含自定义业务字段  
exports.orderCabExt = async function() {  
    debug("[orderCab] intent %s", JSON.stringify(this.intent));  
    // 处理下单信息  
    // ...  
    this.intent.drop = true; // 成功下单, 设置当前意图会话过期  
    return {text:"下单成功, 订单编号: SE111.", // 设定回复,  
           params: { YOUR_KEY: YOUR_VALUE}} // 自定义业务字段  
}
```

`this.intent` 的一个实际数据示例:

```
{  
  "name": "book_cab",  
  "topicName": "greetings",  
  "branch": "dev",  
  "entities": [  
    {  
      "name": "originLoc",  
      "val": "天津",  
      "requires": true,  
      "dictname": "@LOC"  
    },  
    {  
      "name": "date",  
      "val": "明天下午四点",  
      "requires": true,  
      "dictname": "@TIME"  
    },  
    {  
      "name": "destLoc",  
      "val": "北京",  
      "requires": true,  
      "dictname": "@LOC"  
    }  
,  
  ],  
  "state": "resolved"  
}
```

失败函数

失败函数中, `this.intent` 的信息和使用, 和成功函数是一致的, 这是调用的实际不一致, 同时它是可选的。

快速读取 `intent.entities`

默认情况下, `intent.entities` 是一个数组, 如果需要用槽位的名称去取值, 要便利, 不方便, 使用以下方法快速使用 `intent.entities` 生成一个 JSON Object, 然后使用槽位名称取值。

```
let entities = _.keyBy(this.intent.entities, 'name');
```

比如, 有一个槽位名字为 `date`, 则接下来获得这个槽位的值:

```
let dateRawString = entities["date"]["val"];
```

转化相对时间为绝对时间

加入一个日期槽位 `dateRawString` 值为“明天下午 5 点”, 那么如何从这个字符串提取出时间“2021-09-03 17:00”呢?

```
let extractedDates = await this.maestro.extractTime(dateRawString);  
debug("extracted date", extractedDates);  
if(extractedDates.length > 0){  
  // 目标时间就是: extractedDates[0] '2021/09/03 17:00'  
  debug("Target date", extractedDates[0]);  
}
```

更多关于 `extractTime` 的介绍[参考链接](#)。

保存变量信息到意图会话

函数执行的时候, 如果有意图会话存在, 就会被加载到函数的 `this.intent` 中。在两个或多个连续的多轮对话中, 可以通过 `this.intent.extras` (JSON Object) 来保存变量。该信息会和该意图会话周期一致。比如:

```
/*
 * 提取时间实体
 */
async function extractTimeEntity(maestro, entities, property) {
  let dates = await maestro.extractTime(entities[property][`val`], "YYYY年MM月DD日 HH:mm");
  return dates.length > 0 ? dates[0] : "";
}

exports.handleAirplaneTicketOrder = async function() {
  debug("[handleAirplaneTicketOrder] this.intent", JSON.stringify(this.intent))

  let entities = _.keyBy(this.intent.entities, 'name');
  // 获得信息
  let date = await extractTimeEntity(this.maestro, entities, "date");
  // 保存到 this.intent.extras
  this.intent.extras = {
    date: date
  }

  ...
}
```

`handleAirplaneTicketOrder` 函数保存了一个变量到 `this.intent.extras` 中，稍后另外一次对话时，调用了另外一个函数 `placeAirplaneTicketOrder`，就可以直接用这个变量。

```
exports.placeAirplaneTicketOrder = async function() {
  // 直接取值
  // this.intent.extras.date

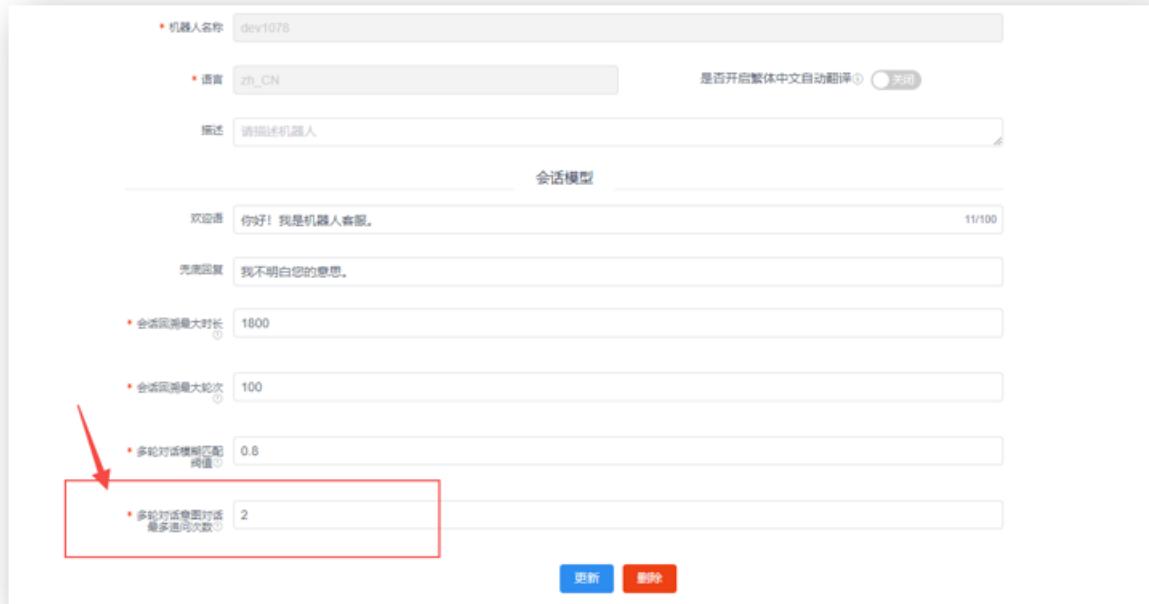
  ...
}
```

设置最大追问次数

匹配意图识别对话之后，机器人会根据槽位信息情况，必填的槽位是否都获取到，进行追问，这个追问可以设定一个最大次数，当达到最多的追问时，机器人会按照顺序检索回复内容：知识库->对话脚本->失败函数（见上文“失败函数”的定义）。

对于每个机器人，BOT 开发者可以自定义设置，进入 BOT 设置页面，找到【多轮对话意图对话最多追问次数】，默认为 2。

在后续的多轮对话中，如果意图对话还没有过期，则依旧会从意图对话中检索。如果追问的信息被匹配到，则做意图对话；否则将不会追问，而是以(知识库->对话脚本->失败函数)这个顺序检索回复。



设置集成意图识别分支

在意图识别模块，包括两个分支：调试分支；生产分支。生产分支，就是指发布到生产环境，实际上线的 BOT 服务。

设计两个分支，主要是考虑到，一个机器人的意图识别数据可以不断的优化，那么每个阶段优化好以后再选择上线到生产分支，这样在调试和优化的时候不影响生产分支的对话。

该设置为，在多轮对话设计器或 BOT 管理控制台的多轮对话页面，设置【环境变量】来调整，默认为调试分支，设置参数为：`@SYS_INTENT_BRANCH`，使用 `dev` 代表集成调试分支，使用 `pro` 代表集成生产分支；默认为调试分支。

升级生产环境的多轮对话脚本

通过多轮对话设计器，或 Chatopera 机器人平台的机器人多轮对话管理页面，上传多轮对话脚本，因为刷新的缘故，正在进行中的意图会话失效，这会给对话用户造成体验上的困扰，所以，对于生产环境的更新，建议在业务低分时间进行！

其它使用说明

SDK 返回意图识别信息

当对话用户，有匹配到意图，正在进行意图识别的对话时，使用多轮对话检索 API，返回值中，`service.provider` 的值是 `intent`，并且 `service.intent` 是当前意图信息。示例数据如下：

```
{  
    "string": "您想从哪里出发?",  
    "topicName": "greetings",  
    "subReplies": [],  
    "service": {  
        "provider": "intent",  
        "intent": {  
            "name": "book_cab",  
            "threshold": 0.9,  
            "branch": "dev",  
            "state": "proactive",  
            "entities": [  
                {  
                    "name": "originLoc",  
                    "val": "",  
                    "requires": true,  
                    "dictname": "@LOC"  
                },  
                {  
                    "name": "date",  
                    "val": "",  
                    "requires": true,  
                    "dictname": "@TIME"  
                },  
                {  
                    "name": "destLoc",  
                    "val": "",  
                    "requires": true,  
                    "dictname": "@LOC"  
                }  
            ]  
        },  
        "logic_is_unexpected": false,  
        "logic_is_fallback": false,  
        "botName": "dev1078",  
        "faq": [],  
        "profile": {}  
    }  
}
```

可见，`service.intent` 的数据与函数中的 `this.intent` 是一致的。

在回复中跳转到指定意图

假设在多轮对话脚本中，已经设定了一个意图的意图匹配器。然后在多轮对话中，还可这样开始该意图的对话：使用 `topicRedirect` 切换话题。

文本回复

```
+ ${0.5}{自定义的文本}  
- ^topicRedirect("intents", "book_airplane_ticket", true)
```

注意：此处 `topicRedirect` 的第三个参数设置为 `true`，代表目标匹配器是一个意图匹配器。

函数回复

文本中，定义了函数

```
+ ${0.5}{自定义的文本}  
- ^handleXXFn()
```

函数 `handleXXFn`：

```
exports.handleXXFn = async function() {
  // do your magic
  return "^topicRedirect(\"intents\", \"book_airplane_ticket\", true)"
}
```

接下来

- 函数: 使用 http 模块请求外部系统 API 服务
- 理解多轮对话工作机制
- 查看示例程序: 预约机票
- 通配符匹配器: 使用语法建立规则
- 模糊匹配器: 容错能力更强和智能的匹配器

设置回复

在匹配器中，我们已经学到了怎么添加一个回答。事实上你可以添加任意数量的回答。这里还有一些高级功能可以帮助你完成更多的任务。

文本形式

最直接的设置文本答案的方法。

- + 在吗
- 你好，在的

如果添加了多个回答，系统会从中随机挑选一个作为回复，然后丢掉这个回答。

- + 在吗
- 亲，在的
- 亲，有什么需要帮助
- 你好，请问遇到什么问题了吗？

所谓丢掉这个答案，是指机器人针对同一个用户，在【会话回溯最大时长】内再次匹配上该匹配器时，选择回复时，不考虑使用过的回复。

- + 在吗
- {keep} 亲，在的
- 亲，有什么需要帮助
- 你好，请问遇到什么问题了吗？

模糊匹配器受机器人记忆模型限制，如果需要在一个记忆周期内一直生效，可以使用 {keep} 标记，详细介绍，参考[文档](#)。

也可以在匹配器前添加*{keep}*，就不用在每个回答中都添加了

- + {keep} 在吗
- 亲，在的
- 亲，有什么需要帮助
- 你好，请问遇到什么问题了吗？

如果回答很长，可以通过“^”分割以方便可读性。可以通过“\n”实现换行

- + 在吗
- 你好，这里是客服中心，\n
- ^ 请问遇到什么问题了吗？

它等价于

- + 在吗
- 你好，这里是客服中心，请问遇到什么问题了吗？

引用回复

有些时候，在问答对中重用一些回复能使编写脚本效率更高，这时可以定义一个问答对，并在脚本其它位置引用它。

+ 在吗
- {@__greeting__} 请问有什么能帮助您?

+ __greeting__
- 亲，在的。
- 你好，客服小美为您服务
- 亲亲，稍等，客服马上就到

引用的方式就是 {@匹配器}，匹配器中的下划线不是必须的，上面例子中的匹配器 __greeting__ 的命名是为了增强脚本的可读性。

将一些常用的回复以约定的匹配器命名方式命名，不但方便复用回复，使脚本可读性好，方便维护，而且更方便使用[上下轮钩子](#)实现多轮对话。

+ 在吗
- {@__greeting__}

+ (*) 问路 (*)
% {@__greeting__}
- 请使用地图APP

+ (*) 打车 (*)
% {@__greeting__}
- 出租车在门口挥手示意停车

+ (*)
% {@__greeting__}
- {@__greeting__}

+ __greeting__
- 亲，在的。请问有什么能帮助您?
- 你好，客服小美为您服务
- 亲亲，稍等，客服马上就到

通配符取值

通配符取值是针对[通配符匹配器](#)的，匹配成功后，可以读取匹配到的词汇。在回答中需要使用输入中的通配符值，这时可以使用 <cap> 达到目的。

+ 我是 *~3
- 你好，<cap>

匹配： 小明比小红高
回答： 你确定小明比小红高吗？

如果用户输入，“我是张三”，那么系统将回复“你好，张三”，当有多个槽时，可以使用多个<cap>。

+ *2 比 *2 高
- 你确定<cap1>比<cap2>高吗？

在对话中，我们有时候会需要以前的通配符值，看一下下面这个例子：

+ 我叫 *~3
- 你好，<cap1>

+ 你猜我叫什么？
% 你好，<cap1>
- 你刚说了，你叫<cap1>

<pNcapM>代表了以前的通配符。其中N代表在对话中之前的问答，M代表捕获的位移。

此处，上下轮钩子，即%开头的句子，代表匹配器“你猜我叫什么？”只服务于“你好，<cap1>”作为回复时，它会被优先匹配。上一轮对话的回复通过(%)的方式指定了接下来的逻辑，形成多轮对话。

关于上下文钩子的[详细介绍文档](#)。

提示: 1) +/%/- 前的空格不是必须的，在多轮对话中，空格可以增强脚本的可读性，比如使用上下轮钩子中的段首缩进。2) 通配符取值适用于通配符匹配器，其它形式的匹配器，有不同的值传递方式，参考它们的详细说明。

函数

函数就是通过写 JavaScript 脚本，处理回复，在脚本中，可以实现各种逻辑，做外部系统集成等。

```
+ 我叫 *~3
- ^getGreetings(<cap1>)
```

那么，在多轮对话设计器内，通过 `exports.getGreetings = async function(){ // do your magic}` 来定制化回复内容。

函数的详细使用[参考文档](#)。

接下来

- [上下轮钩子](#)
- [函数](#)

使用上下轮钩子

在实际应用中，和机器人聊天时，很可能要通过上下轮钩子完成一个任务。我们用(%)来定位之前回复，声明新的匹配器，(%)后的内容是和某个回复内容一样的字符串，这里的(%)称之为“上下轮钩子”。

```
+ (*)  
- 您身高多少  
  
+ *(3-5)  
% 您身高多少  
- 我的身高也是<cap1>
```

让我们一起看看这个例子：

1. 当用户输入任何文字，我们用通用通配符触发回答，然后系统回复“您身高多少”。
2. 当用户继续输入时，系统会先从历史中查看之前的回复中是否有对应的上下文，在这里指的是“% 您身高多少”
3. 最后，如果用户输入 3 到 5 个字符，系统匹配匹配器“+ *(3-5)”，并且回复“我也是<cap>”。<cap>代表的就是用户输入的内容。

使用函数

函数是一个强大而有趣的设计。在回复中，可以使用函数来获取整条消息对象，用户对象或者其它资源，比如数据库。

Chatopera 机器人平台的函数，就是内置的 JavaScript / Node.js 运行环境，基于安全和性能的考虑，这个 JavaScript 运行时是经过特殊处理的 Node.js 执行时。

函数定义

把通配符值当做变量传给函数，例如下面这个例子：

```
+ 我的用户名是 *(2-10)
- ^getUserAccount(<cap>)
```

所以，调用函数的方式就是使用“^”。在函数的编辑窗口中，可以这样定义：

```
exports.getUserAccount = function(account, cb) {
    cb(null, "对不起，系统没有找到" + account);
}
```

函数的声明中，参数列表首先是通配符的值，可以传多个，然后最后一个参数始终是回调函数(cb)，cb 的参数列表为 `(error, text)`，`text` 作为文本添加到回复中，`error` 是指处理中发生异常。

函数的定义也同样支持 `async/await` 语法，例子如下：

```
exports.getUserAccount = async function(account) {
    return "对不起，系统没有找到" + account;
}
```

使用 `async/await` 时，需要抛出异常时，通过 `throw new Error("ERROR MESSAGE")` 完成。

在系统集成时，业务系统的需求千差万别，为了灵活的支持各种需求，在函数中也可以自定义返回值。

```
exports.getUserAccount = function(account, cb) {
    cb(null, {
        text: "对不起，系统没有找到" + account,
        params: {
            resetPasswd: "http://example.com/reset-account"
        }
    });
}
```

其中，`params` 可定义为 `[]` 或 `{}` 对象，该用法同样适用于 `async/await` 函数。

```
exports.getUserAccount = async function(account) {
    return {
        text: "对不起，系统没有找到" + account,
        params: {
            resetPasswd: "http://example.com/reset-account"
        }
    }
}
```

在 [系统集成/多轮对话检索](#) 时，返回值 `data` 内将增加 `params` 属性，与以上设定的值相同。

复合函数

在回复中，可以添加任意多的函数，比如

- + ...
- 联合 ^callFunction1() 和 ^callFunction2(<cap1>)

嵌套函数

在函数的回调函数中，函数名会被解析成对应的函数，所以放心的在回复中添加任意合法的函数，比如在脚本中这样写：

- + ...
- ^nestedAFunction()

然后，在函数中，定义如下：

```
exports.nestedAFunction = function(cb) {  
    cb(null, "张三 ^nestedBFunction()");  
}  
exports.nestedBFunction = function(cb) {  
    cb(null, "和李四");  
}
```

下一步

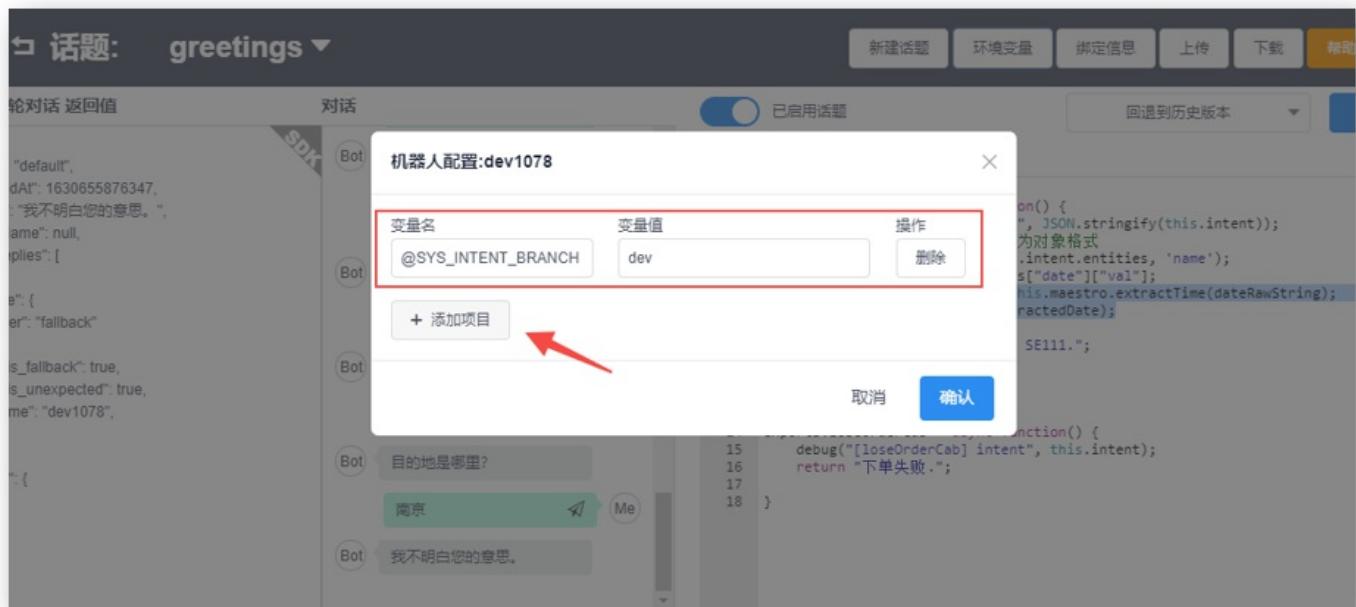
- 内置函数库
- 函数返回值

配置环境变量

多轮对话支持配置环境变量，在开发过程中，设定对话可配置的信息。

设置环境变量

在多轮对话设计器配置环境变量。

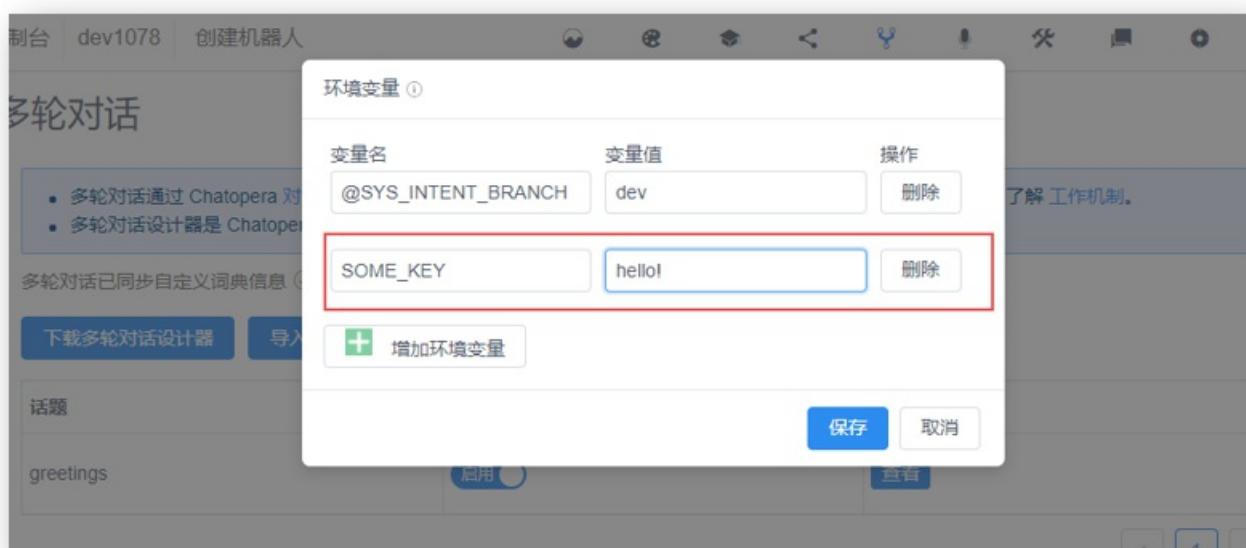


在函数中，使用全局变量 `config` 从环境变量读取值。

比如：

```
exports.someFunction = async function(){
    return config["SOME_KEY"];
}
```

这样就方便了分发多轮对话，从多轮对话设计器中导出对话应用 c66，Chatopera 机器人平台用户可以将该 c66 文件导入另外一个机器人，并使用环境变量配置属性。



系统环境变量

在多轮对话中，也存在系统环境变量，这些变量属于 Chatopera 机器人平台约定的，以 "@" 开头，其值可以被 Chatopera 机器人平台用户修改，但它们往往存在默认值和合法值的约束。

系统环境变量	默认值	取值说明	描述
@SYS_INTENT_BRANCH	dev	dev 或 pro	多轮对话集成意图识别的分支， dev 代表调试分支， pro 代表生产分支

管理对话状态

在多轮对话中，来访者和机器人对话的过程可以想像成来访者使用地图游览，每个时刻，来访者的位置都可以映射到地图上。地图有一定的行走路径，来访者下一步所能选择的方向是由当前状态提供的。

现在，来访者与机器人对话的状态会保持 30 分钟，比如来访者想要点餐，但是说了一句“我要点晚餐”，然后半个小时没有继续发送信息，那么等来访者再次发送信息时，机器人已经不记得上文了。

上下文状态和机器人记忆，都是让机器人有更强的对话处理能力：通过上下轮钩子和通配符取值，设计对话流程，满足各种定制化业务。

在实际应用中，对话状态按照匹配器、上下轮钩子和时间作用改变外，还有哪些办法影响对话状态呢？以下介绍几种方法，影响对话状态。

匹配器及回复复用

在机器人检索多轮对话时，会考虑到 30 分钟内的 100 轮对话（此处 30 分钟和 100 轮分别对应机器人设置界面参数：会话回溯最大时长，会话回溯最大轮次），我们称这样的一个限定条件是机器人的记忆。假如一个对话如下：

```
+ 你好  
- 你好！  
- 欢迎光临！
```

它使用了通配符匹配器，有两个回复：“你好！”和“欢迎光临！”。假设在记忆周期内，发送两次“你好”给机器人，得到的回复分别是“你好！”和“欢迎光临！”。这时，再次发送“你好”，机器人则会回复兜底回复。

这是因为，机器人对“你好”的回复都消耗尽了。如果想让机器人在记忆周期内，保证可以重复的使用某一个回复，就需要加上 {keep} 标记，使用方法如下。

通配符匹配器

```
+ 你好  
- {keep} 你好！  
- 欢迎光临！
```

也可以在匹配器前，使用 {keep}，这样就相当于在每个回复前都加了 {keep}，比如：

```
+ {keep} 你好  
- 你好！  
- 欢迎光临！
```

当匹配器的有多个回复时，会按照记忆过滤，再从后选中随机选择一个作为回复，如果在回复上使用了 {keep} 则起到了：保证该匹配器在记忆中重复使用，标记该回复作为默认回复。

注意：此处 { 和 } 是半角符号。

模糊匹配器

模糊匹配器中，{keep} 的使用也有两种形式，形式与通配符匹配器一致。

```
+ {keep} ${0.6}{喜欢夏天还是冬天}  
- 都喜欢
```

或者

```
+ ${0.6}{喜欢夏天还是冬天}  
- {keep} 都喜欢
```

意图匹配器在 `intent` 后添加 `{keep}`，注意 `{keep}` 前后有空格。

```
// 预约机票
intent {keep} book_airplane_ticket
- ^handleAirplaneTicketOrder()
```

意图匹配器中，使用 `{keep}` 只有一种形式。

清除状态

所谓清除状态，就是在处理完某次回复，不再需要用户处于当前对话状态，而恢复默认对话状态。

解决办法：设置回复的文本以 `{CLEAR}` 开头。比如：

```
+ 再见
- {CLEAR} 感谢您选择我们，期待再次光临！
```

`{CLEAR}` 可以添加在 **回复** 或 **函数返回值** 中。在 Chatopera 机器人平台返回给来访者时，文本内容会去掉这个前缀。

这个方法很实用，尤其是在**全能匹配器**，对话可能进入死循环，因为所有的输入都被全能匹配拦截，那么在对应的回复中使用 `{CLEAR}` 就达到了下一次对话进入默认状态，不优先匹配**全能匹配器**的目的。

切换话题

在多轮对话中，开发者可以定义多个对话主题，对话主题的名字是字母组成的字符串，使用回复在不同对话主题间跳转。

[跳转到通配符匹配器或模糊匹配器](#)

```
+ 你好
- ^topicRedirect("greetings","你好")
```

`topicRedirect` 是内置的函数，第一个参数是目标主题名字，第二个参数是目标匹配器。

[跳转到意图匹配器](#)

在函数中，跳转到指定的意图匹配器有一点特殊，参考[使用说明](#)。

知识库路由

在知识库的答案或多轮对话的函数中设置回复时，可以用 **routeDirectReply** 来检索一个指定的话题和匹配器，直接路由到多轮对话的主题和匹配器。

```
routeDirectReply#["TOPIC_NAME", "TOPIC_GAMBIT_ID" [,INHERIT_PARAMS]]
```

`TOPIC_NAME`: 话题名称

`TOPIC_GAMBIT_ID`: 匹配器

其中，`INHERIT_PARAMS` 是可选参数，决定当前对话取得的 `params` 是否覆盖接下来对话的 `params`，值为 `[true|false]`，默认为 `false`。

另外，当 `TOPIC_GAMBIT_ID` 的值为 `$ctx.textMessage$` 时，则使用当前对话的用户输入，在 `TOPIC_NAME` 中进行检索。

比如

```
routeDirectReply#["class_001_pre", "__C1PRE_GAMBIT_003",true]
```

The screenshot shows the Chatopera dashboard interface. On the left, there's a sidebar with '控制台' (Control Panel) and '知识库' (Knowledge Base). The main area is titled '新建问题' (Create New Question). It has fields for '问题 *' (Question *) containing 'how to get rewards', '选择分类' (Select Category) with '请选择' (Please Select), and '相似问题' (Similar Questions) with a '+ 增加' (Add) button. Below these is a '答案' (Answer) section with a red box around the first answer entry. This entry has a title '答1' (Answer 1) and a text area containing the JSON code: 'routeDirectReply#[{"reward": "promote_tasks", "true"}].'. At the bottom are '确定' (Confirm) and '关闭' (Close) buttons.

提示: **routeDirectReply**需要设定为知识库问答对里的第一个答案, 答案类型为纯文本 plain。

设置定时任务

应用场景

在多轮对话中，执行了 `函数 A`，然后想在若干时间后执行 `函数 B`。那么则可以使用 `setTimeout` 函数实现。

使用说明

比如有如下脚本：

```
+ 你好
- {keep} ^fnA()
```

`fnA` 是一个函数，在 `函数` 中定义如下：

```
async function B() {
  debug("Exec B")
}

exports.fnA = async () => {

  setTimeout(async () => {
    debug("Run fnB");
    await B()
  }, 5000)

  return "hello"
}
```

其中，`setTimeout` 的使用就是执行计时任务，`5000` 代表 5 秒后执行函数体：

```
async () => {
  debug("Run fnB");
  await B()
}
```

执行结果：

```
2022-07-09 11:05:51 DEBUG conversation call fnA result: hello
2022-07-09 11:05:51 DEBUG conversation call fnA:
2022-07-09 11:05:55 DEBUG conversation Run fnB
2022-07-09 11:05:55 DEBUG conversation Exec B
```

关于 `setTimeout` 函数的进一步介绍，参考[文档](#)。

调用语音识别服务

Chatopera 机器人平台支持语音识别服务，将语音转化为文本内容，即 ASR (Automatic Speech Recognition)。

支持语言

目前，Chatopera 机器人平台只支持中文普通话（zh_CN）的语音识别，当机器人语言设置为 zh_CN 时可以使用。

调用接口

SDK

使用 SDK 发送语音识别请求，[参考文档](#)。

CLI

基于 Chatopera CLI 命令行工具调用，[参考文档](#)。

查看语音识别历史

进入语音识别模块，查看语音识别请求历史。

The screenshot shows a search interface for ASR history. It includes a search bar for '按用户ID进行查询' and a blue '查询' button. Below the search area is a summary box containing two bullet points: '自动语音识别(ASR) 服务，让开发人员能够轻松地为其应用程序添加语音转文本功能。' and '通过 API 或 SDK 使用该服务参考 [文档中心](#)，计费情况请查看 [说明](#)。' The main content area displays a table with one row of data. The table columns are: '发生时间' (Time), '来访者ID' (Visitor ID), '识别结果' (Recognition Result), '语音时长' (Duration), '播放' (Play), and '操作' (Operations). The data in the table is as follows:

发生时间	来访者ID	识别结果	语音时长	播放	操作
2021-09-05 10:26	commandline	上海 浦东机场 入境房 输入 全 闭 环 管理	6250ms	<div style="width: 100px; position: relative;"><div style="width: 10px; height: 10px; background-color: blue; border-radius: 50%; position: absolute; left: 0; top: 0;"></div><div style="width: 100%; height: 100%; position: absolute; left: 0; top: 0; background: linear-gradient(to right, transparent 45%, #ccc 45%, #ccc 55%, transparent 55%);"></div><div style="width: 10px; height: 10px; background-color: blue; border-radius: 50%; position: absolute; left: 50%; top: 0;"></div><div style="width: 100%; height: 100%; position: absolute; left: 50%; top: 0; background: linear-gradient(to right, transparent 45%, #ccc 45%, #ccc 55%, transparent 55%);"></div></div> 00:00 00:06	下载语音 结果详情

At the bottom, there is a pagination indicator '共 1 条' with page numbers <, 1, >.

计费及发票

语音识别模块的服务费用参考[计费及发票](#)。

对话测试

测试页面

对于不同方式实现对话，都需要验证机器人的对话是否符合预期，Chatopera 机器人平台提供统一个模块完成这个任务，导航进入【机器人详情>>测试页面】。

The screenshot shows the Chatopera platform's testing interface. On the left, a conversation log is displayed between a user ('ME') and a robot. The user asks about face recognition, and the robot responds that it doesn't understand. The user then asks if they can ask a question, and the robot replies that it failed to recognize faces with a score of 0.73. On the right, a JSON response is shown with a threshold of 0.5. The JSON data lists two items, each representing a failed face recognition attempt with a low score.

```
[{"id": "AW70N5Q907BxWDoZ_hDM", "score": 0.7317734512205795, "post": "人脸识别失败", "reply": "1111", "reply_plain_text": "1111"}, {"id": "AW70N71307BxWDoZ_hDN", "score": 0.34730097313826713, "post": "身份证识别", "reply": "2222", "reply_plain_text": "2222"}]
```

在该页面下，也需要选择对话实现方式，根据对话方式不同，对话过程中，右侧的信息展示形式也是不同的，使用测试对话页面，也提升用户实现机器人的开发效率。

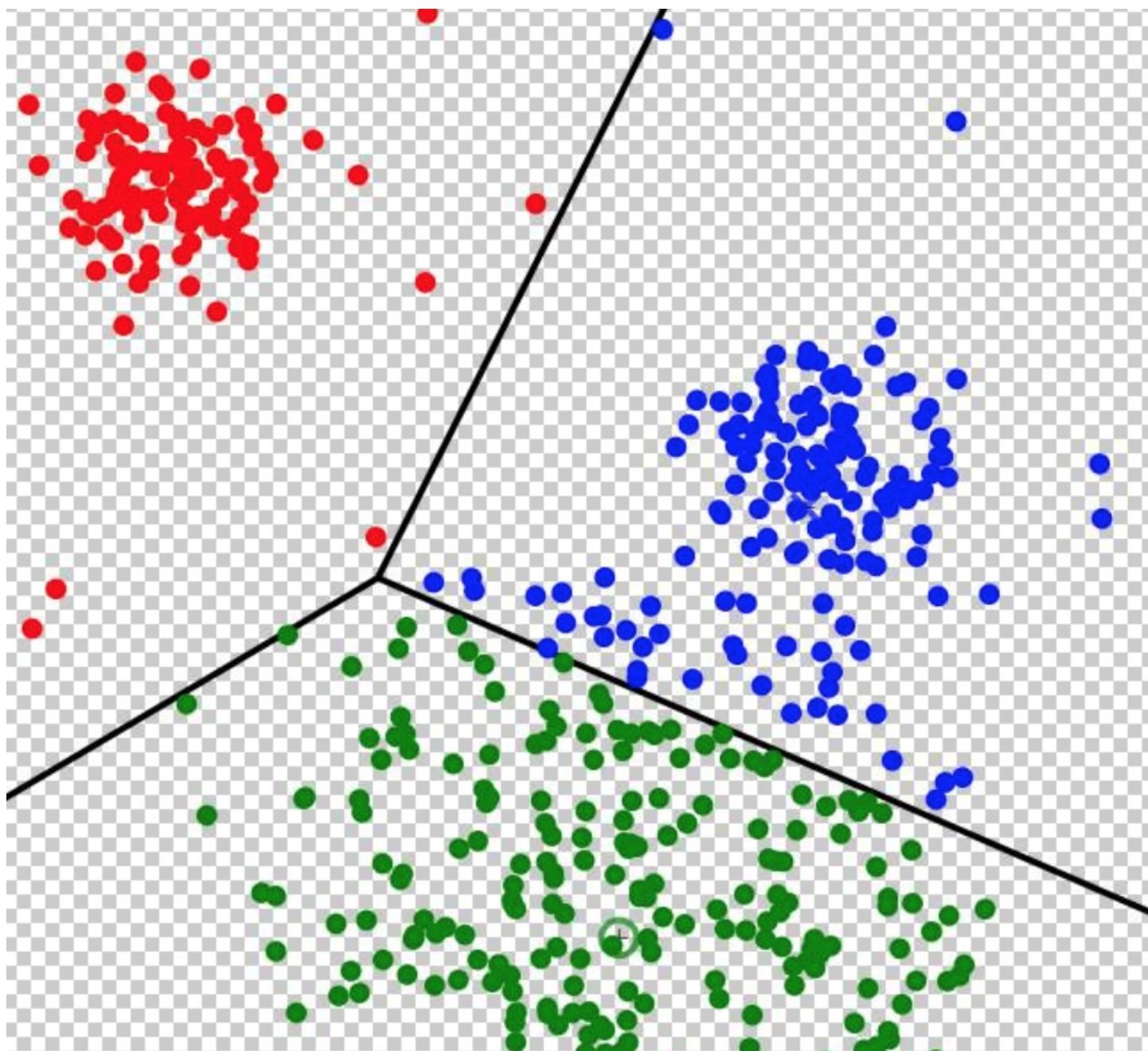
对话历史分析

概述

使用机器学习分析对话历史。**Chatopera** 机器人平台聚类分析是将多个文本按照相似度分类，分成一个个聚簇。每个聚簇中任何两个问题是相似的。聚类分析在运营聊天机器人、提升智能化水平起着关键作用。聚类分析的结果可以用来：

- 1) 优化知识库：发现新知识，新问题或已有问题的相似问；
- 2) 优化多轮对话脚本：发现对话流程，话术，业务流程需求；
- 3) 优化意图识别：新的意图，意图中的参数；
- 4) 优化商业智能：发现新需求，访客痛点，业务流程改造；
- 5) 衡量用户活跃度，分析用户激活、留存、客户服务等环节的问题。

在聊天机器人上线后，可能未必达到及格的程度。比如知识库命中率比较低，这时候，就要经常的使用聚类分析功能，优化知识库，常用方案是每天上午执行过去 24 小时或 3 天的对话历史，然后下载聚类结果，进行机器人优化。



对话历史分析

更新日期: 5/4/2023

聚类分析, 有如上图, 不同颜色的零散的点, 使用线段尽量将相同颜色的点汇聚在一个区域, 聚类分析算法就是寻找最佳的线段。聚类分析算法使用非监督的机器学习算法, 不保证聚类结果的准确无误, 通常是聚类后方便进一步的人工标注工作, 此处恕不赘述细节。

使用须知

- 1) 聚类分析只针对对话历史记录产生的数据, 包括知识库检索 API 和多轮对话检索 API;
- 2) 执行聚类分析任务是免费的。

提交聚类任务

在机器人详情页面, 在导航菜单中打开“聚类分析”页面。点击“创建任务”。



在弹出的对话窗口中选择聚类针对的时间范围: 开始日期距离现在远, 结束日期距离现在近。聚类任务分析的数据就是这个时间段内机器人和访客的对话历史数据。

使用快捷方式创建聚类时间范围: 今天, 过去 24 小时, 过去 3 天等。

下载聚类结果

提交聚类结果后, 自动回到聚类分析页面, 聚类分析页面的主要内容是聚类任务的列表, 包括每个聚类任务的创建日期, 聚类时间范围, 状态。

- 聚类分析是将多个文本按照相似度归类，分成一个个聚簇。每个聚簇中任何两个问题是相似的。
- 聚类分析常用来发现新知识，以及衡量用户在客服中的咨询情况分析，更多内容参考 [文档中心](#)。

创建任务

创建时间	数据开始日期	数据截止日期	状态	操作
2020-10-28 15:17:36	2020-07-28 00:00:00	2020-10-28 23:59:00	执行中	<button>刷新</button>
2020-10-28 11:05:12	2020-07-28 00:00:00	2020-10-28 23:59:00	已完成	<button>下载结果</button>
2020-10-28 10:25:05	2020-10-27 10:24:00	2020-10-28 10:24:00	异常结束	<button>下载结果</button>

状态分为：执行中、异常结束、执行完成。

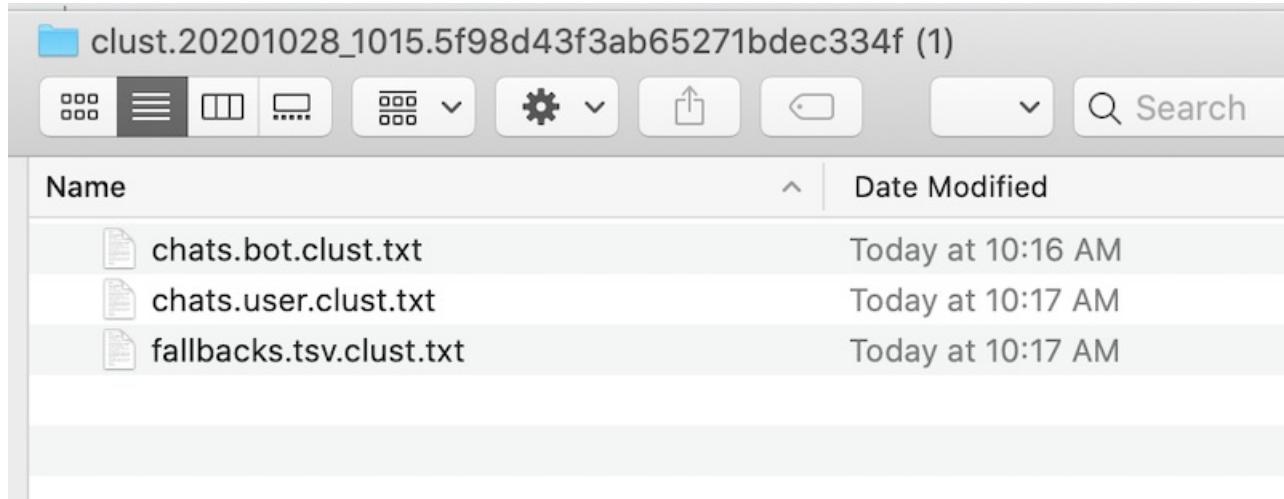
针对执行完成的聚类任务，在操作中可以下载结果，点击后浏览器将下载 Zip 压缩包，建议用户安装 [7-Zip 软件](#)。

执行中的聚类任务，可以通过点击操作中的刷新按钮更新状态信息。

对于异常结束，会有提示，通常是由于聚类时间范围内，没有发生对话历史记录的原因。

查看聚类结果

将上一步得到的压缩包解压，解压文件内包含如下文件：



Chatopera 机器人平台提供聚类输出包括：

- 访客发送内容的聚类结果，文件：chats.user.tsv.clust.txt；
- BOT 发送内容的聚类结果，文件：chats.bot.tsv.clust.txt；
- BOT 回复为兜底回复时，用户的发送内容的聚类结果，文件：fallbacks.tsv.clust.txt；
- 访客点击没有帮助时，发送的问题，聚类结果，文件：negative.tsv.clust.txt。

每个文件中，包含了聚簇、该聚簇关键词和聚簇内成员等信息。

如果所选聚类时间范围包含数据量太少，也会在聚类结果中提示，因为在这种情况下，没有聚类的价值，通过肉眼就可以完成对话分析。

聚类通常针对的是大规模数据，成千上万、乃至数十万数百万的对话历史记录。

CLI 连接多轮对话

Chatopera CLI 是命令行终端工具, [参考安装文档](#)。在测试 Chatopera BOT 多轮对话时, 可以使用 CLI 连接到机器人的多轮对话服务, 进行测试验证, 使用 CLI 可以:

- 提升自动化任务
- 查看接口融合的返回值
- 快速验证对话技能

以下的代码是在命令行终端中运行, 比如 Windows CMD、PowerShell、Git Bash、Linux Bash 等 Shell 环境。

以下的代码使用 CLI 版本是 [@chatopera/sdk 2.8.12](#)。

- 检查 Chatopera CLI 版本: `bot --version`
- 升级执行命令 `npm install -g @chatopera/sdk`

初始化

```
mkdir YOUR_FOLDER_NAME # 自定义一个目录  
cd YOUR_FOLDER_NAME  
bot env
```

运行后的日志如下:

```
L $ bot env  
.env file[/mnt/c/Users/Administrator/chatopera/cli/_ /ts/.env] is generated.  
ra.com/products/chatbot-platform/howto-guides/cli-install-config.html
```

此时在目录下生成了配置文件 `.env`:

```
L $ cat .env  
# Chatopera Cloud Service  
BOT_PROVIDER=https://bot.chatopera.com  
BOT_CLIENT_ID=  
BOT_CLIENT_SECRET=  
BOT_THRESHOLD_FAQ_BEST_REPLY=0.8  
BOT_THRESHOLD_FAQ_SUGG_REPLY=0.6  
BOT_ACCESS_TOKEN=
```

设置 `.env` 里的值:

值	描述
BOT_PROVIDER	Chatopera BOT 平台地址, 默认为 http://bot.chatopera.com/
BOT_CLIENT_ID	机器人的 ClientID, 在机器人控制台的设置界面获得
BOT_CLIENT_SECRET	机器人的 secret, 在机器人控制台的设置界面获得
BOT_THRESHOLD_FAQ_BEST_REPLY	机器人知识库最佳回复阈值, 默认为 0.8

值	描述
BOT_THRESHOLD_FAQ_SUGG_REPLY	机器人知识库建议回复阈值, 默认为 0

机器人控制台的设置界面:

设置

基本信息

Client Id: [REDACTED] Secret: [REDACTED]

* 机器人名称: 入门教程测试

* 语言: zh_CN 是否开启繁体中文自动翻译

描述: 请描述机器人

连接

在完成配置后, 在 `.env` 文件所在文件夹下, 执行下面命令, 进行对话:

```
bot connect
```

发送问题:

```
[REDACTED] /c/Users/Administrator/chatopera/cli [REDACTED] ] Fri Ap
$ bot connect
dirpath /mnt/c/Users/Administrator/chatopera/cli/saas_62e756889fb70... / .env ...
>> load env file /mnt/c/Users/Administrator/chatopera/cli/saas_62e756889fb70... / .env ...
2023-04-21 12:48:12 INFO >> connect to https://bot.chatopera.com, clientId 62e756889fb70..., secret ...
2023-04-21 12:48:12 INFO [connect] FAQ Best Reply Threshold 0.8, Suggest Reply Threshold 0
? Text 服务
2023-04-21 12:48:15 INFO {
  "rc": 0,
  "data": {
    "state": "default",
    "createdAt": 1682052492200,
    "string": "我不明白您的意思。",
    "topicName": null,
    "subReplies": [],
    "service": {
      "provider": "fallback",
      "faqBestReplyThreshold": 0.8,
      "faqSuggReplyThreshold": 0
    },
    "logic_is_fallback": true,
    ...
  }
}
```

在回复中, 展示返回值 JSON 数据。

H5 聊天控件

Chatopera 云服务支持将聊天机器人立即发布到网页，这项功能就是 H5 聊天控件，具体功能和使用见下。



支持网页聊天窗口

网页聊天窗口是 Chatopera 云服务为每个机器人提供的一个 URL 地址，打开该地址可以直接作为访客和机器人进行对话。

- 支持通过 URL 超链接打开一个网页聊天窗口，与机器人进行会话
- 打开的网页聊天窗口支持手机客户端和电脑客户端，自适应布局
- 打开的网页聊天窗口支持自定义形象、欢迎语、品牌标识
- 支持电脑客户端的网页中，自定义介绍信息、广告图片

支持网页聊天控件

H5 聊天控件另外一项能力是支持在已有的网页中添加浮动的按钮，点击按钮即跳转到网页聊天窗口。

- 支持自定义聊天控件在网页中的位置
- 支持延时加载聊天控件
- 支持自定义聊天控件中的文本
- 支持配置聊天控件的颜色
- 支持邀请功能及自定义邀请展示的信息



网页聊天控件



主动邀请

管理入口

以上信息，可通过【机器人管理控制台-系统集成】找到 H5 聊天控件的信息及配置。

企业应用

企业应用是已经集成了 Chatopera 机器人平台的企业软件，用户只需要配置即可使用。

 H5 聊天控件	 春松客服 chatopera.com	 Facebook Messenger
接入设置 使用说明	使用说明 私有部署 定制开发	演示示例 私有部署 定制开发

文字链代码

将文字链代码 URL 发送给访客：

- 通过 IM 工具发送，比如微信
- 将该 URL 使用工具生成二维码，然后发送该二维码
- 将该 URL 作为超链接，比如网页 HTML 超链接、微信公众号菜单的按钮的超链接、微信公众号文章阅读原文的地址

使用 H5 聊天控件的形式有很多，只要访客在浏览器中打开【文字链代码】的 URL 即可与你的聊天机器人对话。

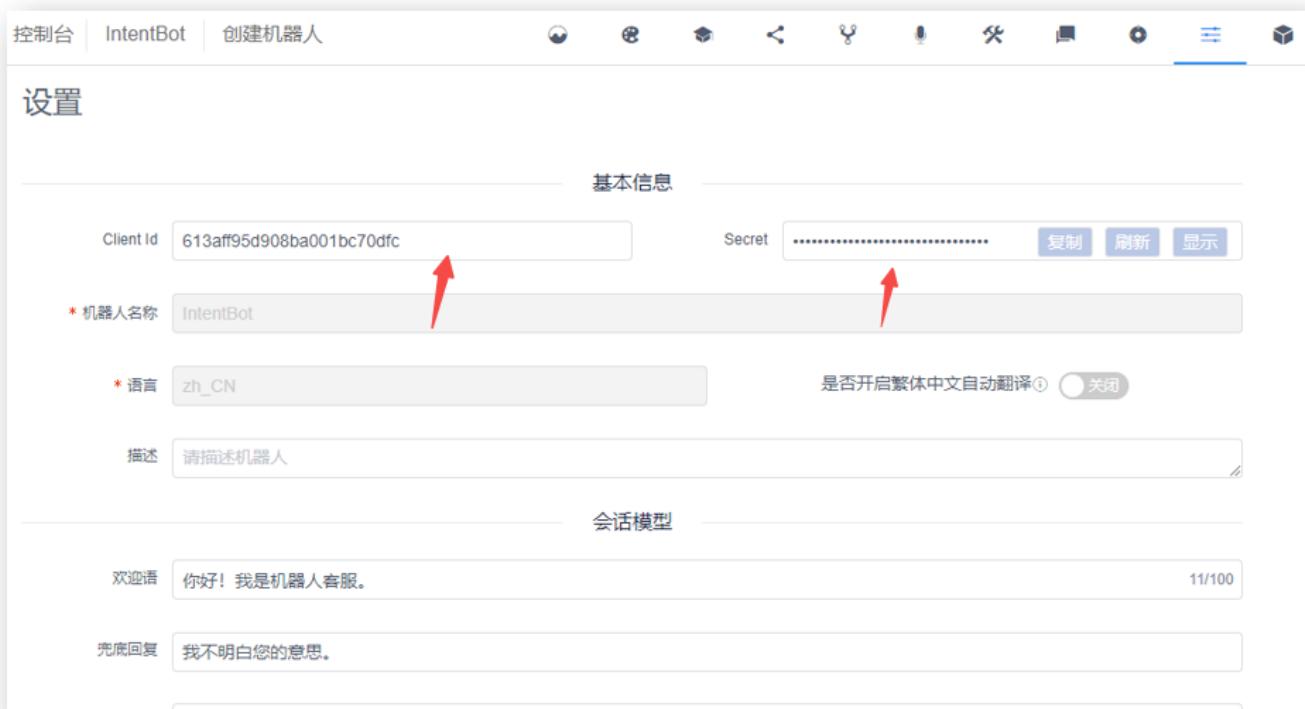
本节以[Nodejs SDK](#)为例子演示使用流程，更多 SDK 下载参考[链接](#)。

基础知识

[Nodejs](#) 是 JavaScript 运行时环境，面向服务器端应用开发，底层使用 Google V8 引擎。[Nodejs](#) 尤其被前端开发者偏爱，因为它让前端开发者以“熟悉”的方式开发后端应用。[Nodejs](#) 的出现一度降低了开发成本，并且成为“快应用”开发趋势出现，[Nodejs](#) 包管理工具 [Npm 站点](#) 是开源领域最大的包管理服务。不同语言的 SDK 使用细节大同小异，因为它们都是调用[Chatopera](#) 机器人平台的[RestAPIs](#)，这些 RestAPIs 是标准一致的。每种语言的 SDK 使用流程按照顺序包括：下载 SDK，实例化[Chatbot](#)类为对象，请求接口和处理返回结果。

获取 *ClientId* 和 *Secret*

SDK 中每个机器人实例需要通过*ClientId*和*Secret*初始化，这两个字段是认证和授权用途。打开机器人【设置】页面，拷贝*ClientId*和*Secret*。



安装 SDK

```
npm install @chatopera/sdk --save
```

实例化[Chatbot](#)类为对象

```
var Chatbot = require("@chatopera/sdk").Chatbot;
var chatbot = new Chatbot(clientId, secret [, serviceProvider]);
```

参数说明

NAME	TYPE	REQUIRED	DESCRIPTION
clientId	string	□	在 机器人控制台/机器人/设置 中获取
secret	string	□	获取办法同上
serviceProvider	string	□	Chatopera 机器人平台地址, 当使用 Chatopera 云服务时, 该值为 https://bot.chatopera.com , 也是默认值

提示: 参数列表中, 写在 `[]` 内的部分是选填参数, 如果不填写使用默认值, 下同。

调用接口示例

得到 `Chatbot` 实例后, 怎么样请求接口服务呢? 假设对该机器人的基本信息感兴趣, 获取基本信息方式如下:

```
var response = await chatbot.command("POST", "/faq/query", {
  query: "不锈钢板现在是什么价格",
  fromUserId: "sdktest1",
});
console.log("response: ", response)
```

或者获取 `Promise` 返回

```
chatbot.command("GET", "/").then(
  (response) => {
    console.log("机器人名称: ", response.data.name);
  },
  (err) => {}
);
```

此处, 不深入探讨 `await` 和 `Promise` 的相关知识, 它们是和 JavaScript 语言相关的内容。在这个例子中, 我们请读者注意, 给定一个机器人类的实例, 再请求 API 服务是多么的简单, `Chatbot#command` 接口提供了一系列的方法, 也是下文给您详细介绍的重点。

示例程序

示例程序代码库: <https://github.com/chatopera/webchat>

通过系统集成的示例程序, 快速掌握 SDK 使用, 尤其是对话检索 API; 系统集成示例程序也可以用来调试和检查机器人, 在和业务系统集成前测试相关接口。与 Chatopera 机器人平台的测试页面不同, 系统集成示例程序更侧重上线前, 对 SDK 相关接口的测试。

在系统集成示例程序中, 使用 Chatopera 机器人平台地址, `clientId` 和 `secret` 立刻连接聊天机器人, 开始对话。

功能:

- 提供对话页面, 方便系统集成测试
- 使用 Bot Provider 地址, `clientId` 和 `secret` 连接机器人
- 实现 Dialogue Management: 融合意图识别检索、多轮对话检索和知识库检索
- `app.js` 使用 [Chatopera Node.js SDK](#), 可作为系统集成参考

启动应用

本示例程序提供 Docker 容器镜像, 使用 Docker 启动服务进行体验

```
docker run -it --rm -p 8668:8668 chatopera/webchat:develop
```

Chatopera Test Client

Service Provider**Client ID****Client Secret****Username****Chat**

CLI 安装和配置

Chatopera CLI 是连接 Chatopera 机器人平台，管理和维护资源的工具，包括一些常用的命令，辅助开发者实现和管理对话机器人。尤其是在有自动化或批量管理的需要时。

安装

Bash

因为使用过程中，依赖于命令终端交界面，在 Linux 或 Mac 上，有 Bash（或兼容）环境。

对于 Windows 用户，请安装 [Git Bash for Windows](#)，安装完成后，在桌面右键，就可以点击 【Git Bash】进入命令终端交界面。

本文以下内容，涉及 Shell 命令，都假设命令终端交界面是 Bash Shell 环境。

Node.js

Chatopera CLI 依赖于 [Node.js v10+](#) 环境，使用 `npm` 进行安装（`npm` 是 [Node.js](#) 安装完成后得到的 CLI 工具）。

- Windows 上安装 Node.js 及配置环境变量 [参考文档](#)
- Linux 上安装 Node.js [参考文档](#)

Chatopera CLI

有了 Node.js 环境，在命令行终端，执行如下命令：

```
npm install -g @chatopera/sdk
```

在 Windows 上，安装过程输出类似的日志：

```
C:\Users\Administrator\AppData\Roaming\npm\bot -> C:\Users\Administrator\AppData\Roaming\npm\node_modules\@chatopera\ sdk\bin\bot.js
+ @chatopera/sdk@2.7.1
added 147 packages from 104 contributors in 36.389s
```

检查安装是否正确，执行：

```
$ bot --version
2.x # 得到类似输出，代表安装正确，2.x 为当时最新的 Chatopera CLI 版本
```

如果上述命令 `bot --version` 执行提示错误，检查环境变量 `PATH` 路径，比如 `C:\Users\Administrator\AppData\Roaming\npm` 是否在 `PATH` 内。

配置参数

Chatopera CLI 命令行工具支持读取文件配置变量，比如 `provider`, `clientid` 等常用的变量。

参数设定优先级: 命令行参数 > `.env` 文件 > 环境变量

映射配置项	命令行参数	环境变量	备注
clientId, 机器人 ID	<code>-c, --clientid [value]</code>	<code>BOT_CLIENT_ID</code>	无默认值，必填
secret, 机器人密钥	<code>-s, --clientsecret [value]</code>	<code>BOT_CLIENT_SECRET</code>	无默认值，必填

映射配置项	命令行参数	环境变量	备注
provider, Chatopera 机器人平台地址	-p, --provider [value]	BOT_PROVIDER	默认值, https://bot.chatopera.com
accessToken, 访客设置密钥	无	BOT_ACCESS_TOKEN	保留值, 无默认值
faqBest, FAQ best reply threshold	-fb, --faq-best [value]	BOT_THRESHOLD_FAQ_BEST_REPLY	FAQ 知识库最佳回复阈值, 默认值 0.8
faqSugg, FAQ suggest reply threshold	-fs, --faq-sugg [value]	BOT_THRESHOLD_FAQ_SUGG_REPLY	FAQ 建议回复阈值, 默认值 0.6

其中, `.env` 文件例子如:

```
# Chatopera Cloud Service
BOT_PROVIDER=https://bot.chatopera.com
BOT_CLIENT_ID=
BOT_CLIENT_SECRET=
BOT_ACCESS_TOKEN=
```

`.env` 文件存储的也是环境变量值。`bot` 命令会沿当前执行命令的路径(`pwd`), 寻找 `.env` 文件。

比如, 在 `/Users/chatopera/chatopera-nodejs-sdk` 下执行 `bot` 命令, 那么, `.env` 文件按照以下顺序进行查找, 一旦查找到就加载为配置, 并退出查找。

```
/Users/chatopera/chatopera-nodejs-sdk/.env
/Users/chatopera/.env
/Users/.env
/.env
```

下一步

- [命令行界面 \(CLI\) 参考手册: 帮助、使用命令参考手册](#)
- [使用 CLI 导入和导出对话语料](#)

CLI 导入和导出对话语料

Chatopera CLI 支持的功能就是对不同[对话语料](#)的数据导入和导出；外加一些其他命令，辅助自动化管理机器人。

准备工作

- 安装 Chatopera CLI，参考文档 [Chatopera CLI](#)；
- 配置 Chatopera CLI，使用 `.env` 文件，参考文档 [Chatopera CLI](#)。

为简化说明，以下各示例配置使用了 `.env` 文件，因为略去了从命令行传入的一些参数。

导入和导出命令规范

针对于不同模块的导入和导出命令，CLI 命令是遵循同一个规范的。

```
bot [dicts|faq|intents|conversation] --action [import|export] --filepath {{FILE_PATH}}
```



如果 `dicts` 后导入，则需要执行 `'bot dicts -a sync'` 同步到其它模块。

如果是导出 (`export`)，没有加入 `--filepath` 部分，则会默认保存文件到当前工作目录 (`cwd` 目录)。

导出对话语料

对话语料指词典、知识库、意图识别和多轮对话的内容，它们包括了机器人的所有对话技能。

删除文件

```
rm -rf bot.dicts.json bot.faqs.json bot.intents.json bot.conversations.c66
```

执行导出，以下命令并无顺序依赖关系

```
bot dicts -a export -f bot.dicts.json && \
bot faq -a export -f bot.faqs.json && \
bot intents -a export -f bot.intents.json && \
bot conversation -a export -f bot.conversations.c66
```

导入对话语料

将到处的对话语料上传给指定的机器人（比如通过 `.env` 设置）。

以下命令有顺序依赖关系

```
bot dicts -a import -f bot_dicts.json && \
bot faq -a import -f bot_faqs.json && \
bot intents -a import -f bot_intents.json && \
bot conversation -a import -f bot_conversations.c66
```

这样，目标机器人就具备从之前导出的机器人的技能。

CLI 命令详细介绍

更多命令介绍，参考文档 [Chatopera CLI](#)。

参考手册

- 术语
- 通配符匹配器
- 内置函数库
- 函数返回值
- 命令行工具 (CLI)
- SDK
- 常见问题

术语

在多轮对话实现的过程中，解决的是一个复杂问题：能够让机器能分析和执行的自然语言的聊天过程。为此，需要定义一些专用的术语和概念。

多轮对话

在多轮对话实现的过程中，解决的是一个复杂问题：能够让机器能分析和执行的自然语言的聊天过程。为此，不得不定义了一些专用的术语和概念。

多轮对话设计器

Conversation Designer，根据需求撰写对话脚本，对话函数的软件工具。

对话、话题

Topic / Topic Name，这几个词代表同一个概念：由一些脚本组成的、满足预期对话能力的单元。机器人的对话能力由多个对话主题组成，对话主题是对话脚本模块化管理的形式。

输入

Input，用户向聊天机器人发送的消息的文字形式。

匹配器

Gambit，匹配用户输入文字的规则，包括以下三种：

- 通配符匹配器 / Star Gambit，写法规则，约束字符串集合
- 模糊匹配器 / Like Gambit，使用机器学习匹配用户输入和匹配器包括的成员之间的相似度，如果相似度大于阀值则认为匹配，在机器人控制台或语法规则内设置阀值
- 意图匹配器] / Intent Gambit，调用意图识别，在成功获得槽位信息或超过最大追问次数后，调用回复

回复

Reply，机器人回复用户输入的文字。

上次回复

Last Reply，作为机器人给来访者回复的最近一次的内容。

上下轮钩子

Conversation Hooks，通过上次回复，链接匹配器，形成多轮对话。

函数

Function，可以从脚本中接受输入，并通过 JavaScript 执行任务返回结果的代码。

对话状态

Conversation State，多轮对话是按照状态机的模型设计的，对话状态就是当前对话处于状态机的什么状态。

对话应用

Conversational App，c66 文件，从多轮对话设计器导出的文件。

对话模板

Chatbot Sample，聊天机器人示例程序，开源地址，<https://github.com/chatopera/chatbot-samples>。

The screenshot shows a table with columns: 对话名称 (Conversation Name), 是否启用 (Enabled), 修改时间 (Last Modified), and 操作 (Operations). The rows contain:

对话名称	是否启用	修改时间	操作
chatopera	已启用	7/24/2018, 11:15:37 AM	<button>编辑</button> <button>删除</button>
profile	已启用	7/24/2018, 11:15:37 AM	<button>编辑</button> <button>删除</button>
weather	已启用	7/25/2018, 7:15:23 AM	<button>编辑</button> <button>删除</button>

对话列表

意图识别

词典

机器人的词汇表

- 自定义词典: 通过词汇表, 正则表达式来设定词汇集合, 表达实体概念
- 系统词典: Chatopera 机器人平台内置地名、组织机构名、人名和时间等

意图

一个对话任务的最小单元

- 说法: 表达同样意图的不同说法, 可包含槽位标识
- 槽位: 在意图中, 完成用户任务需要的信息

调试

从设计到实现机器人的技能

- 训练机器人: 建立词汇表, 意图, 为意图添加槽位和说法
- 测试对话: 在测试对话页面, 测试意图识别, 优化训练内容

版本

每次训练都会得到新的意图识别模型, 系统创建新的版本

- 调试版本: 调试中的机器人版本
- 生产版本: 训练好的机器人模型, 满足上线使用需求, 从调试版本发布而成

会话

也称为 session, 和用户的单个意图关联的对话; 在系统集成时, 会话生命周期, 部分阶段由开发者管理。

- 请求对话, 需要先创建会话, 会话会绑定 0-1 个任务: 刚开始不知道用户意图, 当确定用户意图后, 该 session 就只和这个意图相关。

聊天

发送访客的请求, 获得机器人的回复

- 通过访客的 ID, session id, 文本消息和机器人进行聊天

会话 和 聊天 都是与上线集成相关, 详细参考 [系统集成](#)。

其它术语

对话语料

指机器人的词典内容、意图识别内容、知识库问答对和多轮对话脚本、函数和环境变量，以上内容的集合。对话语料就是机器人的对话能力依赖的所有原料。

内置函数库

在开发中，进一步提供系统灵活性，在`functions`定义中，系统预置了以下几个工具对象，方便聊天机器人开发者实现各种功能的聊天机器人。以下内容假设读者已经掌握了`JavaScript`编程语言。

要点提示

注意 `this` 的使用

一些`builtin`的函数只在`exports`的函数内可用，注意在这些函数前的`this`，它们是必须的。使用`this`前缀的变量代表该变量依赖于在函数内部的`this`对象。比如：

```
exports.customFn = async function(){
  debug(this.user.id); // this.user is available here inside an exports function
}
```

JavaScript 运行时

目前，Chatopera 机器人平台为函数提供的 JavaScript 运行时是 `node:10.21.0`，发布于 2020-06-02。一些微小的最新的`Node.js`的 LTS 上的功能可能并不支持，`v10.21.0` 目前是应用最多的 LTS 版本，更稳定和兼容广泛。

函数库

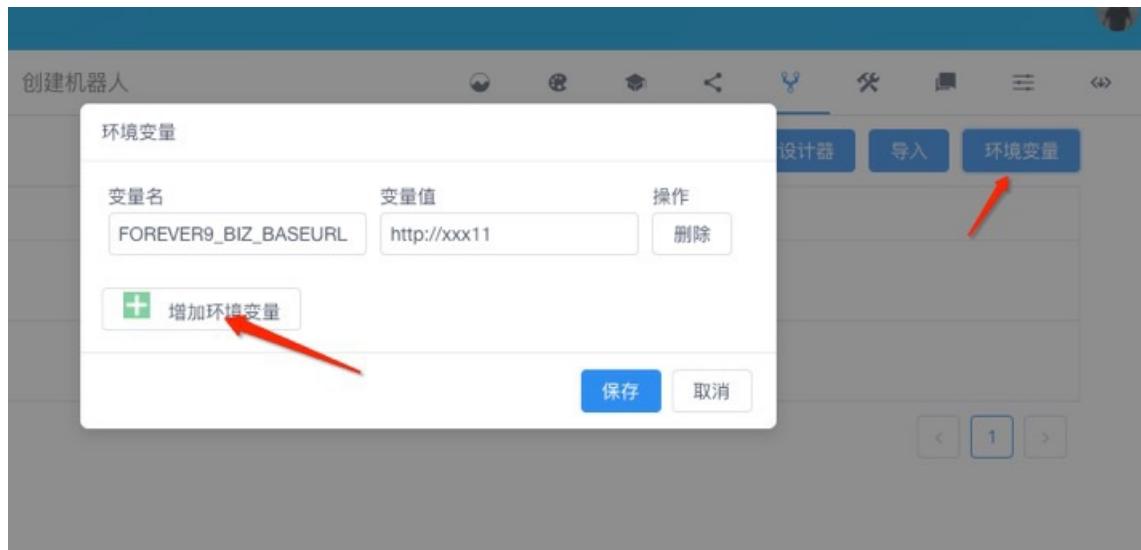
- [basics](#)
- [message](#)
- [user](#)
- [maestro](#)
- [3rd-party](#)

basics

基本函数库。

config

获取环境变量，环境变量在多轮对话设计器和聊天机器人多轮对话控制台都可以定义，目的是在设计阶段和运行阶段，多轮对话使用不同的配置。



`config` 作为函数中的全局常量，不需要用户定义，不支持改变该值，只能作为读取用途，并且 `config` 只是键值对，值只是 `string` 类型。

```
exports.printKeyValue = function(key, cb){  
    // 通过通配符获得key，查看其在环境变量中对应对值  
    cb(null, "Ok, value is " + config[key]);  
}
```

- config 更多使用示例代码: <https://github.com/chatopera/chatbot-samples/search?q=config>

`环境变量` 常用来配置一些生产环境对应的信息。

http

用于在函数内部，通过 HTTP 协议集成外部系统。

`http` 作为函数中的全局常量，不需要用户定义。

```
http.get(url[, config])  
http.delete(url[, config])  
http.head(url[, config])  
http.options(url[, config])  
http.post(url[, data[, config]])  
http.put(url[, data[, config]])  
http.patch(url[, data[, config]])
```

Chatopera 机器人平台中，函数内置 `http` 常量使用 `axios` 包实例化，`http` 即 `axios`。

- http 更多使用示例代码: <https://github.com/chatopera/chatbot-samples/search?q=http>
- axios 详细使用文档: <https://github.com/axios/axios>

debug

打印调试日志。

`debug` 作为函数中的全局常量，不需要用户定义。

```
debug("hello")
debug("hello %s", stringVar)
debug("hello %s, %j", stringVar, jsonVar)
debug("hello %s, %o", stringVar, objVar)
```

- debug 更多使用示例代码: <https://github.com/chatopera/chatbot-samples/search?q=debug>

-

`lodash` 是一个一致性、模块化、高性能的 JavaScript 实用工具库。Lodash 通过降低 array、number、objects、string 等等的使用难度从而让 JavaScript 变得更简单。Lodash 的模块化方法 非常适用于：

- 遍历 array、object 和 string
- 对值进行操作和检测
- 创建符合功能的函数

鉴于这些特点，Chatopera 内置函数库增加了全局变量 `_` 来使用 `lodash v4.17.15` 的接口。

比如，可以在函数中这样写将 JSON Array 转化为 JSON Object。

```
var array = [
  { 'dir': 'left', 'code': 97 },
  { 'dir': 'right', 'code': 100 }
];

_.keyBy(array, function(o) {
  return String.fromCharCode(o.code);
});
// => { 'a': { 'dir': 'left', 'code': 97 }, 'd': { 'dir': 'right', 'code': 100 } }

_.keyBy(array, 'dir');
// => { 'left': { 'dir': 'left', 'code': 97 }, 'right': { 'dir': 'right', 'code': 100 } }
```

更多 `lodash` 方法，参考 <https://lodash.com/docs/4.17.15>。

message

`this.message` 是每次用户输入文本经过自然语言处理后的对象，它并不是全局对象，必须在 `exports` 的函数中，使用 `this.message` 来引用。

请求文本

即对话用户发送的文本内容。

```
this.message.original // String, 输入文本的原始内容  
this.message.clean // String, 输入文本的改写, 经过词根转化的输入字符串
```

词性信息

```
this.message.words // JSONArray, 分词结果  
this.message.tags // JSONArray, 对应位置分词的词性  
this.message.entities // JSONArray, 输入文本包含的命名实体  
this.message.names // JSONArray, 输入文本包含的人名  
this.message.nouns // JSONArray, 输入文本包含的名词  
this.message.adverbs // JSONArray, 输入文本包含的副词  
this.message.verbs // JSONArray, 输入文本包含的动词  
this.message.adjectives // JSONArray, 输入文本包含的形容词  
this.message.pronouns // JSONArray, 输入文本包含的指示代词
```

关于 `this.message.tags` 包含了所有的分词对应的词性，可以用来做更多的判断，不同语言的词性集是不同的，参考 [Chatopera 多轮对话 Message 语言词性标注](#)。

自定义信息

在多轮对话检索接口中，传入自定义的信息 `extras`。

```
this.message.extras
```

传入方法参考，[多轮对话检索接口](#)。

user

在函数中获得对话用户的信息。

this.user.id

当前对话用户标识，在[多轮对话检索](#)传入的 `userId` 信息。

在函数中，获取该唯一标识信息：

```
this.user.id
```

比如：

```
exports.myUserId = function(cb){
  cb(null, {
    text: "Your id is " + this.user.id
  })
}
```

this.user.history

当前用户与机器人对话的历史

```
this.user.history // 类型：数组
```

其中，每个元素内容如下：

```
{
  "input": {
    "timestamp": ISODate("2020-07-28T17:06:01.458Z"),
    "original": "hello",
    "extras": {}
  },
  "reply": {
    "createdAt": 1595955961672.0,
    "string": "#in-params#",
    "topicName": "greetings",
    "clearConversation": false,
    "props": {
      "params": [
        {
          "hyperlink": "http://",
          "thumbnail": "http://xx.png",
          "summary": "描述",
          "title": "标题",
          "type": "card"
        }
      ]
    },
    "extras": {}
  }
}
```

其中，`original`就是来访者的输入，`reply`是上一轮对话中机器人的回复。`extras`是通过多轮对话接口传入的自定义信息，传入方法参考，[多轮对话检索接口](#)。

历史对话按照降序排列，即最近发生的对话在上面，最多存储 100 轮历史对话数据。

maestro

`maestro` 是管理对话状态存储和自然语言处理的高级接口，它并不是全局对象，必须在 `exports` 的函数中，使用 `this.maestro` 来引用。

注意事项：

1. 对于返回值是 `Promise` 的接口，使用 `try catch`, `await` 进行调用可读性更好。
2. 该对象不是全局函数，作用域在函数内部，使用时依赖于函数内 `this` 变量。

User

对话用户信息相关，此部分为 `maestro` 内实现的，和对话用户相关的还有 `user` 库。

`profile`

设置用户画像

```
await this.maestro.profile(userId, property, value) 返回值 Promise
```

`userId`: 用户唯一标识

`property`: 用户的属性

`value`: 属性值

该接口用于持久化一个用户的信息，此处 `userId` 可以使用当前用户的信息 `this.user.id`；`property` 和 `value` 是根据业务灵活定义。

该信息被持久化到数据库里，该接口的目的是设置用户画像，比如用户对一个产品是否有兴趣。

用户画像可以通过[系统集成/用户管理/获取用户画像信息 API](#)获得。

KeyValue Pairs

用于管理持久化的信息，通过键值对存取。

`set`

存储键值对，支持过期时间，过期时间以秒为单位。VALUE 可以是时间，字符串，数字。

该命令也可以用于更新键值对，或者更新 EXPIRES 时间，让该信息不过期。

```
this.maestro.set(KEY, VALUE [, EXPIRES]) 返回值: Promise
```

`ttl`

获得键值对有效时间，键值对是存储在 Redis 服务中，如果设定键值对时同时对 `EXPIRES` 赋值，那么到达过期时刻，该键值对将被删除。使用 `ttl` 接口获得一个键值对有效存在时间，返回值是有效的秒数，如果返回值为 `-1` 则代表该值不存在。

```
this.maestro.ttl(KEY) 返回值: Promise(number)
```

`incrby`

增加键的值，对于 Number 类型的键，增加一定值

```
this.maestro.incrby(KEY, NUMBER) 返回值: Promise
```

maestro**get**

获得一个键的值

```
this.maestro.get(KEY) 返回值: Promise
```

del

删除一个键和其值

```
this.maestro.del(KEY) 返回值: Promise
```

Extract information

信息提取

extractNumber

获得一句文本中的阿拉伯数字，比如来访者提到“一二三”，“123”等数字，会被提取为一个转化为阿拉伯数字的数组。

```
let numbers = await this.maestro.extractNumber(cap1);
```

返回值: `numbers` 是处理后的数字数组，识别的文本中，可以包含多个数字；未获得数字时，数组长度为 0。比如

```
let numbers = await this.maestro.extractNumber("一二三和123？");
// numbers [ '123', '123' ]
```

extractTime

获得绝对时间，比如来访者提到“明天”，“后天”，“下周一”等相对时间，会被计算出正确的绝对时间，时区为北京时间。

```
let dates = await this.maestro.extractTime(cap1[, format, timezone]);
```

其中，`format` 格式参考[momentjs#format](#)，如果设定 `format` 的值为 `long`，返回值数组中数据格式为数字，类似于使用 JavaScript `(new Date()).getTime()` 获得的数据。

`timezone` 值设定参考[momentjs#timezone](#)。

`format` 默认值为 `YYYY/MM/DD HH:mm`，`timezone` 默认值为 `Asia/Shanghai`。

返回值: `dates` 是处理后的时间数组，识别的文本中，可以包含多个时间；可以具体到分钟，格式化时，未得到的时间被设置为 `0`。

比如

```
let dates = await this.maestro.extractTime("明天和后天是几月几号？", "YYYY年MM月DD日", "Asia/Shanghai");
// dates [ '2021年09月02日', '2021年09月03日' ]
```

digest

自动生成文本的摘要，假设文本有“。！？”等分隔句子的符号。那么 `digest` 将会获得其中最关键的句子作为摘要。

```
let texts = await this.maestro.digest("冬天来了。春天还会远吗？", 5);
```

第一个参数文本内容，第二个参数是返回摘要文本的字数长度。

返回值 `texts` 是一个数组，是输入文本中的若干句子，按照重要程度排序。

keywords

提取文本关键词。

```
let words = await this.maestro.keywords("冬天来了。春天还会远吗?", 5);
```

第一个参数文本内容，第二个参数是返回关键词语的数量。返回值 `words` 是一个数组，是输入文本中的若干词语，按照重要程度排序。

Notifications

通知服务。

sendMail

通过邮件服务器发送邮件，使用 `nodemailer` 实现。

```
exports.sendMail = async function(argvs) {  
  
    let mailSettings = {  
        service: config['MAIL_SERVICE'],  
        auth: {  
            user: config['MAIL_ACCOUNT'],  
            pass: config['MAIL_PASSWORD']  
        }  
    };  
    let transporter = this.maestro.nodemailer.createTransport(mailSettings);  
    // setup e-mail data with unicode symbols  
    let mailOptions = {  
        from: `HR<${config['MAIL_ACCOUNT']}>`, // sender address  
        to: config['MAIL_RECEP'], // list of receivers  
        subject: `【应聘】小主，新增候选人了，岗位${openning}`, // Subject line  
        text: '', // plaintext body  
        html: content, // html body  
    };  
    // send mail with defined transport object  
    transporter.sendMail(mailOptions, function(error, info) {  
        if (error) {  
            debug(error);  
        }  
    });  
};
```

以上 SMTP 邮件服务的配置需要定义在环境变量中，支持的邮箱服务参考[nodemailer](#)，可配置 QQ 企业邮箱、163 邮箱等。

3rd-party

其他第三方函数库。

Octokit

Octokit 是 GitHub 官方推出的 RestAPIs Toolkit，可以方便的使用 JavaScript 管理 GitHub 资源，请求服务。

在 Chatopera 机器人平台内置函数库中，包含了 Octokit 类，可以直接在函数中，完成 GitHub Issue 创建，查询等任务，使用举例如下：

```
const octokit = new Octokit({
  auth: "YOUR_GITHUB_PERSONAL_ACCESS_TOKEN"
});
exports.handleOpenGithubIssue = async function() {
  await octokit.request(`POST /repos/chatopera/cskefu/issues`, {
    owner: "chatopera",
    repo: "cskefu",
    title: "YOUR TITLE",
    body: "YOUR BODY",
    labels: ["label1" "label2"]
  });
}
```

其中，`YOUR_GITHUB_PERSONAL_ACCESS_TOKEN` 是每位 GitHub 注册用户自己创建的 Personal Access Token，字符串类似于：

`ghp_i75Hmkglxxx`。

更多关于 `Octokit` 以及 `Octokit#request` 的介绍，[参考链接](#)，更多 `GitHub RestAPIs` 的介绍，[参考链接](#)。

- 在函数中使用 Octokit 更多使用示例代码：<https://github.com/chatopera/chatbot-samples/search?q=Octokit>

函数返回值

函数的返回值，可以用于自定回复消息内容或状态跳转等。在多轮对话中，回复除纯文本外，还可以支持多媒体消息，这样用户交互的体验更佳。不同消息的类型还需要渠道能够兼容，或在业务系统中进行适配，以下各消息类型在 [春松客服](#) 和 [多轮对话设计器](#) 中已经支持。

列表消息

函数返回值中，`params` 数组中每个元素约定如下：

```
// 问候语中关联常见问题
exports.get_greetings = async function() {
  return {
    text: "请问有什么可以帮到您？",
    params: [
      {
        label: "1. 产品列表",
        type: "qlist",
        text: "产品列表"
      },
      {
        label: "2. 当季热销产品",
        type: "qlist",
        text: "当季热销产品"
      },
      {
        label: "3. 退换货咨询",
        type: "qlist",
        text: "退换货咨询"
      }
    ]
  };
}
```

其中，`label` 是显示文字，`type` 为固定值 `qlist`，`text` 是点击该问题时发送给机器人的文本内容。

按钮消息

```
// 按钮选择消息
exports.get_products = async function() {
  return {
    text: "您对下面哪个产品感兴趣",
    params: [
      {
        label: "上衣",
        type: "button",
        text: "介绍一下上衣"
      },
      {
        label: "服装",
        type: "button",
        text: "介绍一下鞋帽"
      }
    ]
  };
}
```

类似图文消息，不同的是：1) `type` 值为 `button`；2) 业务上一般会设定一组按钮消息只能有一个按钮被点击一次。

图文消息

// 图文消息

```
exports.get_shangyi = async function() {
    return [
        {
            text: "{CLEAR} 图文消息",
            params: [
                {
                    type: 'card',
                    title: "秋冬上衣优选",
                    thumbnail: "https://ss0.bdstatic.com/xx.jpg",
                    summary: "秋冬上衣优选秋冬上衣优选秋",
                    hyperlink: "https://www.1688.com/B6AC.html"
                }
            ]
        }
}
```

此处，因为显示图文消息以包含文字，`text`需要使用固定值 `{CLEAR} 图文消息`，`type`为固定值 `card`；`title`，`thumbnail`，`summary` 和 `hyperlink` 分别是图文消息的标题，缩略图，描述和超链接地址。

提示： `{CLEAR}` 作为前缀代表机器人本轮清除多轮对话状态。

自定义业务字段

从以上的消息形式的实现，就是借助函数返回值中 `params` 这个属性，在实现聊天机器人时，函数中可在 `params` 自定义的业务字段，都会被 SDK/API 返回。

```
// 自定义业务字段
exports.get_shangyi = async function() {
    return [
        {
            text: "以帮助您下单了。",
            params: {
                orderId: "BIL-001"
            }
        }
}
```

通常是在业务系统内，使用业务字段完成更多工作。

命令行界面 (CLI)

Chatopera CLI 是连接 Chatopera 机器人平台，管理和维护资源的工具，包括一些常用的命令，辅助开发者实现和管理对话机器人。尤其是在有自动化或批量管理的需要时。CLI 完全基于 [Chatopera Node.js SDK](#)。

- [安装和配置 Chatopera CLI](#)
- [使用 CLI 导入和导出对话语料](#)

命令帮助

打印 CLI 可用命令。

```
bot --help
```

得到类似输出:

```
Usage: bot [options] [command]

Options:
  -V, --version           output the version number
  -h, --help              display help for command

Commands:
  details [options]       get a bot's detail info, such as name, primaryLanguage
  connect [options]        chat with bot via bot#conversation interface
  conversation [options]  import or export a bot's conversations data
  trace [options]          tail a bot's conversations logging info
  asr [options]            request Chatopera ASR API
  faq [options]            import or export a bot's faqs data
  dicts [options]          sync, import or export a bot's dicts data
  intents [options]        train, import or export a bot's intents data
  help [command]          display help for command
```

也可以针对一个命令，获得更多帮助提示，比如:

```
bot connect --help
bot trace --help
bot conversation --action export --help
```

获得类似输出:

```
Usage: bot connect [options]

Options:
  -c, --clientid [value]    ClientId of the bot
  -s, --clientsecret [value] Client Secret of the bot, optional, default null
  -u, --username [value]     Username to chat with bot, default: commandline
  -p, --provider [value]     Chatopera Bot Service URL, optional, default
                            https://bot.chatopera.com
  -fb, --faq-best [value]   FAQ best reply threshold, optional, default 0.8
  -fs, --faq-sugg [value]   FAQ suggest reply threshold, optional, default
                            0.6
  -h, --help                display help for command
```

配置文件

env

生成配置文件，对生成的配置文件，需要修改，设置键值对。

```
bot env [-fp FOLDER]
```

默认为当前命令执行所在目录。

比如:

```
bot env -fp /c
```

将生成配置文件 `/c/.env`。

词典

词典在知识库、多轮对话、意图识别中都有使用和依赖，在导入知识库文件、多轮对话文件或意图识别文件之前，最好是先导入词典文件，以免为使用带来影响。

`export`

导出引用的系统词典、所有自定义词典（词汇表词典和正则表达式词典）。

举例:

```
bot dicts --action export --filepath /tmp/bot_dicts.json
```

`import`

导入引用的系统词典、所有自定义词典（词汇表词典和正则表达式词典）。

举例:

```
bot dicts --action import --filepath /tmp/bot_dicts.json
```

`sync`

触发同步命令，知识库、意图识别和多轮对话同步最新的近义词词典；此步骤将引起数据改写，生产环境宜业务低峰时间段进行。

```
bot dicts --action sync
```

多轮对话

`connect`

连接聊天机器人，在命令行终端连接 BOT 并进行对话。

示例:

```
bot connect -c xxx -s xxx -u zhangsan
```

其中，`clientid` 和 `clientsecret` 从每个机器人的设置页面获取，`username` 代表用户名，是一个不含空格或特殊符号的字符串，每个用户的唯一标识，`provider` 是 [Chatopera 机器人平台](#) 地址，默认为 [Chatopera 云服务](#)。

在对话中，可以使用快捷方式，快速输入。

快捷方式	MAC OSX / WINDOWS
回溯历史	↑ 上箭头；↓ 下箭头
打印历史	Shift + → 右箭头

快捷方式	MAC OSX / WINDOWS
使用索引输入历史，索引根据打印历史获得	输入索引，然后 Ctrl + Shift + Shift + → 右箭头

export

导出多轮对话为 c66 文件。

示例:

```
bot conversation --action export --filepath /tmp/bot.conversations.c66
```

import

导入多轮对话脚本，在命令行终端发布脚本文件到[多轮对话](#)中。

示例:

```
bot conversation --action import --filepath /tmp/bot.conversations.c66
```

其中 `filepath` 为 `xx.c66` 文件，支持相对路径和绝对路径。

因为多轮对话可能使用了词典，所以宜先导入词典，再导入多轮对话。

trace

打印聊天机器人日志：方便调试多轮对话脚本，实时跟踪服务器端日志，排查问题。

示例:

```
bot trace --log-level DEBUG
```

Log level 可以是 `[DEBUG|INFO|WARN|ERROR]`。

知识库

export

导出知识库问答对、扩展问和分类信息等，目前 CLI 导入和导出只支持 JSON 格式，欲使用 Excel 从机器人平台浏览器管理界面完成。

举例:

```
bot faq --action export --filepath /tmp/bot.faqs.json
```

import

导入知识库问答对、扩展问和分类信息等。

举例:

```
bot faq --action import --filepath /tmp/bot.faqs.json
```

因为知识库可能使用了词典，所以宜先导入词典，再导入知识库。

意图识别

export

导出意图识别说法、槽位等信息。

```
bot intents --action export --filepath /tmp/bot.intents.json
```

import

导入意图识别说法、槽位等信息。

```
bot intents --action import --filepath /tmp/bot.intents.json
```

因为意图识别可能使用了词典，所以宜先导入词典，再导入意图识别。

导入命令也会自动执行训练意图调试分支，训练完成后，命令退出。

意图识别的导入和导出不会处理生产版本上线信息和操作，需要 Chatopera 机器人平台用户自行维护意图识别模块的生产分支。

train

训练意图识别的调试分支。

```
bot intents --action train
```

语音识别

语音识别 SDK 接口使用[参考文档](#)。

asr

```
Usage: bot asr [options]
```

Options:

-c, --clientid [value]	ClientId of the bot
-u, --username [value]	Username to chat with bot
-s, --clientsecret [value]	Client Secret of the bot, optional, default null
-p, --provider [value]	Chatopera Bot Service URL, optional, default https://bot.chatopera.com
-f, --file <value>	Target file to recognize, *required.
-h, --help	display help for command

示例：

```
bot asr -c clientId \
-s secret \
-u username \
-f ./asr.sample.001.wav
```

示例语音文件下载，[asr.sample.001.wav](#)。

当前工作路径如果有 `.env` 文件时，也可以简化为命令：

```
bot asr -f ./asr.sample.001.wav
```

返回结果

```
{  
  "rc": 0,  
  "data": {  
    "duration": 6250,  
    "predicts": [  
      {  
        "confidence": 0.960783,  
        "text": "上海 浦东机场 入境 房 输入 全 闭 环 管理"  
      },  
      {  
        "confidence": 0.960767,  
        "text": "上海 浦东机场 入境 防 输入 全 闭 环 管理"  
      },  
      {  
        "confidence": 0.960736,  
        "text": "上海 浦东机场 入境 坊 输入 全 闭 环 管理"  
      }  
    ]  
  }  
}
```

已知问题

在 Windows WSL 内调用 SDK，返回异常 invalid timestamp

错误原因：Windows WSL 的时间同步有 BUG，未能和互联网时间同步。

解决方案：执行手动的时间同步。

假设使用的 WSL 发行版本基于 Ubuntu，那么可以执行下面的命令解决。

```
sudo apt-get install ntpdate  
sudo ntpdate time.windows.com
```

开源项目

Chatopera CLI 工具是[开源的](#)，更多使用示例参考：

<https://github.com/chatopera/chatopera-nodejs-sdk/tree/master/scripts>

下一步

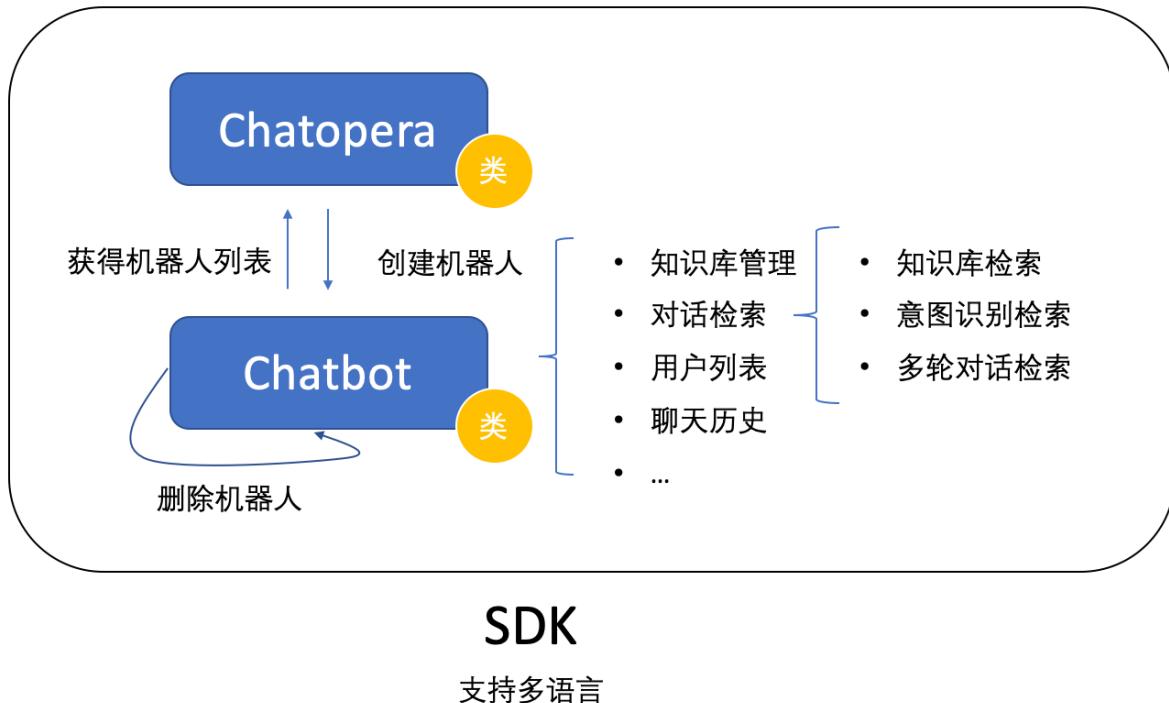
- 使用 CLI 导入和导出对话语料

Chatopera 机器人平台用户可以在不同的软件系统、程序中接入 Chatopera 机器人。SDK 是面向不同编程语言和 Chatopera 机器人平台集成的工具库，包括 Node.js、Go、PHP、Python、Java 等。

能力概述

SDK 能力概述：使用 `chatopera` 类创建机器人、获得机器人信息；使用 `Chatbot` 类管理某一机器人的资源；`Chatopera` 类是系统账户级别，`Chatbot` 是单个机器人级别。

`Chatopera` 类使用控制台中【访问设置】的 `Personal Access Token` 实例化，`Chatbot` 类使用每个机器人的 `clientId` 和 `secret` 初始化。



下载安装

SDK 简化了集成的复杂度，某些语言可以通过包管理工具安装。SDK 下载地址参考列表：

语言	下载地址	使用指南
Python	chatopera-py-sdk	示例程序 / 技术支持
Java	chatopera-java-sdk	示例程序 / 技术支持 / JavaDocs
Go	chatopera-go-sdk	示例程序 / 技术支持
PHP	chatopera-php-sdk	示例程序 / 技术支持
Node.js	chatopera-nodejs-sdk	示例程序 / 技术支持

为了方便开发者调用 SDK，每个 SDK 内均有示例程序或测试程序作为参考。

提示：以上 SDK 同时支持私有部署的 Chatopera 机器人平台。

在聊天机器人的一级菜单，可以进入集成页面查看。

SDK

SDK是将聊天机器人以服务的方式接入您的系统的方式。若您在配置中遇到问题，请通过info@chatopera.com或[查看帮助文档](#)。

 Java	 PHP	 golang	 node.js
使用说明 示例程序 接口文档	使用说明 示例程序 接口文档	使用说明 示例程序 接口文档	使用说明 示例程序 接口文档
 Python			
使用说明 示例程序 接口文档			

如不能满足您使用环境或者语言的 SDK，请创建工单进行描述，Chatopera 将尽快满足您的需求。

核心类

在每个语言的 SDK 中，均实现两个类：`Chatopera` 类和 `Chatbot` 类。

`Chatopera` 类

`Chatopera` 类是与 Chatopera 机器人平台集成的一个高级类，因为 Chatopera 云服务为开发者提供聊天机器人服务，`Chatopera` 类的对象就是 Chatopera 云服务中一个注册账户的代理。

通过左侧导航菜单了解使用详情。

`Chatbot` 类

`Chatbot` 类是与 Chatopera 云服务集成的一个核心类，因为 Chatopera 云服务为开发者提供聊天机器人服务，`Chatbot` 类的对象就是 Chatopera 云服务中一个聊天机器人的代理。

通过左侧导航菜单了解使用详情。

已知问题

在 Windows WSL 内调用 SDK，返回异常 `invalid timestamp`

错误原因：Windows WSL 的时间同步有 BUG，未能和互联网时间同步。

解决方案：执行手动的时间同步。

假设使用的 WSL 发行版本基于 Ubuntu，那么可以执行下面的命令解决。

```
sudo apt-get install ntpdate  
sudo ntpdate time.windows.com
```

接下来

- [Chatopera](#) 类详细说明

- [Chatbot](#) 类详细说明: 对话检索、机器人信息及更新, etc.

- 遇到问题? 常看“获得帮助”

Chatopera 类构造函数

构造函数

构造函数

```
Chatopera(accessToken [, botProvider])
```

参数说明

NAME	TYPE	REQUIRED	DESCRIPTION
accessToken	string	□	在 机器人控制台/访问设置 中获取, 即 Personal Access Token
botProvider	string	□	Chatopera 机器人平台地址, 当使用 Chatopera 云服务时, 该值为 https://bot.chatopera.com , 也是默认值

在机器人控制台, 打开【访问设置】:



刚刚注册的账户需要点击[生成](#)进行初始化; 点击[复制](#)将 Token 值复制到粘贴板。

注意: **Personal Access Token**(简称: **Access Token**) 是您访问 Chatopera 机器人平台的密钥, 具有管理平台资源的权限, 请您务必妥善保管! 不要以任何方式公开 **Access Token** 到外部渠道(例如 **Github**), 避免被他人利用造成安全威胁。

更多实例化例子

不同语言下, [Chatopera](#) 类的包名或引用方式不同。

Node.js

```
const { Chatopera } = require('@chatopera/sdk');
...
const chatopera = new Chatopera(accessToken[, botProvider]);
```

Java

```
import com.chatopera.bot.sdk.Chatopera;
...
Chatopera chatopera = new Chatopera(accessToken[, botProvider]);
```

Python

```
from chatopera import Chatopera
co = Chatopera(accessToken[, botProvider])
```

PHP

假设使用[composer](#)作为包管理工具, 其它安装方式参考[chatopera-php-sdk](#)。

```
<?php

include_once '**DIR** . "/vendor/autoload.php";
$chatopera = new Chatopera\SDK\Chatopera($accessToken[, $botProvider]);
```

Go

```
import (
    "github.com/chatopera/chatopera-go-sdk"
)
...
var admin = chatopera.NewChatopera(accessToken[, botProvider])
```

Chatopera 类接口规范

发送请求

Chatopera 实例的核心接口是 `command`，以下也使用 `Chatopera#command` 来指这个接口，该接口是对 RestAPI Request 的高级封装，内部完成签名认证，`RequestHeaders` 和 `RequestBody` 等处理。

接口规范

```
result = chatopera.command(method, path [, body])
```

提示： result 返回在 Node.js 中使用 `await` 或 `Promise`，参考 [快速开始](#)；其它语言直接用 `=` 便可获取。

参数说明

NAME	TYPE	REQUIRED	DESCRIPTION
method	string	□	对于资源的具体操作类型，由 HTTP 动词表示。有效值包括 <code>GET</code> , <code>POST</code> , <code>PUT</code> , <code>DELETE</code> 和 <code>HEAD</code> 等
path	string	□	资源的执行路径，通常包含资源实体名称或唯一标识，也可能在 <code>path</code> 中使用 <code>queryString</code> 传递参数
body	JSON 数据结构	□	<code>body</code> 是请求中的数据，对应 RestAPI 中的 Http Body

`method` 不同动词代表的含义一般如下：

- GET - 从服务器取出一项或多项资源；
- POST - 在服务器创建一个资源；
- PUT - 在服务器更新一个资源；
- DELETE - 在服务器删除一个资源。

还有更多类型的 `method`，没有上述几种常用，在此不进行赘述。

`queryString` 是 URL 的一部分。典型的 URL 看起来像这样: <http://server/resource?foo=A&bar=B>。其中，`foo=A&bar=B` 就是 `queryString`，通常用来传递参数，这个例子中包含两个参数：`foo` 值为 A，`bar` 值为 B。在下文中，`path` 参数中可能包含 `queryString`，形式如 `foo={{var1}}&bar={{var2}}`，需要把 `{{var1}}` 和 `{{var2}}` 替换为实际值。

`body` 数据是 JSON 格式的，不同语言对于 JSON 格式支持方式不同。[JSON](#) 是一种轻量级的数据交换格式，描述了使用键值对、数组、字符串、数字、日期和布尔类型等值存储对象。[JSON](#) 在不同语言下，等价数据结构如下。

语言	JSON OBJECT	JSON ARRAY
JavaScript	<code>{...}</code>	<code>[...]</code>
Java	<code>org.json.JSONObject</code>	<code>org.json.JSONArray</code>
PHP	基本类型 <code>array</code>	基本类型 <code>array</code>
Python	基本类型 <code>dict</code>	基本类型 <code>list</code>
Go	<code>map[string]interface{}</code>	<code>[]map[string]interface{}</code>

下文表述时, 统一使用 `JSON`, `JSON Object` 和 `JSON Array` 代表 `JSON` 数据结构和其不同语言下的等价数据结构。

提示: 相对而言, `JSON` 等价的数据结构, 在获取 `JSON Object` 的键值或 `JSON Array` 的长度和成员时, 语法不同, 但都易于掌握。在使用时, 参考不同 SDK 的[示例程序](#)。

`body` 是否必填以及是 `JSON Object` 还是 `JSON Array`, 取决于 `method` 和 `path` 的值, 不同 `method` 和 `path` 的组合对应了不同的接口功能, 满足不同需求, 下文将介绍满足各种需求的 `method` 和 `path`, 并各个说明 `body` 参数。

返回值

返回值即请求结果, 针对接口定义, `Chatopera#command` 的返回值 `result` 是 `JSON Object`, 并有以下属性。

KEY	TYPE	DESCRIPTION
<code>rc</code>	int	response code, 返回码, 大于等于 0 的正整型。 <code>0</code> 代表服务器端按照请求描述, 正常返回结果; <code>rc</code> 不等于 <code>0</code> 是代表异常返回。
<code>data</code>	JSON	数据资源。正常返回时, 服务器端执行逻辑成功, 比如查询时, <code>data</code> 就是查询结果。
<code>msg</code>	string	消息, 当服务器端执行请求成功, 并且不需要返回数据资源时, 通过 <code>msg</code> 代表文本信息, 比如提示信息。
<code>error</code>	string	异常消息, 当服务器端返回异常时, 具体出错信息包含在 <code>error</code> 中。
<code>total</code>	int	分页, 所有数据记录条数。
<code>current_page</code>	int	分页, 当前页码, (分页从 1 开始)。
<code>total_page</code>	int	分页, 所有页数。

每次请求结果中, `rc` 是必含有的属性, 其它属性为可能含有。不同 `rc` 的正整数形代表不同的异常, `data`、`status` 以及分页信息, 则因 `method` 和 `path` 而异, 以下进行详细介绍。

提示: 不同语言对返回值可能进行了封装, 但是不离其宗, 都是基于以上定义, 比如 Java SDK 中, 定义 `com.chatopera.bot.sdk.Response` 作为 `Chatopera#command` 接口返回值, `Response` 类提供 `getRc`、`getData` 和 `toJSON` 等方法, 提升代码可读性。在使用时, 参考不同 SDK 的[示例程序](#)。

下文中使用的 `method`, `path`, `body` 和 `result` 等均代表以上介绍的概念。

机器人管理

创建聊天机器人

```
Chatopera#command("POST", "/chatbot", body)
```

示例代码: [Node.js](#) | [Java](#) | [PHP](#) | [Python](#) | [Go](#)

body / JSON Object

```
{
  "name": "小巴",
  "primaryLanguage": "zh_CN",
  "fallback": "请联系客服。",
  "description": "我的超级能力是对话",
  "welcome": "你好，我是机器人小巴巴",
  "trans_zhCN_ZhTw2ZhCn": false
}
```

KEY	TYPE	REQUIRED	DESCRIPTION
<code>name</code>	string	□	机器人名字。
<code>primaryLanguage</code>	string	□	聊天机器人语言，目前支持简体中文(<code>zh_CN</code>)、繁体中文(<code>zh_TW</code>)、英文(<code>en_US</code>)、日语(<code>ja</code>)、泰语(<code>th</code>)。默认为 <code>zh_CN</code> 。当使用 <code>zh_CN</code> 时可开启自动识别繁体并翻译(<code>trans_zhCN_ZhTw2ZhCn</code> : <code>true</code>)，默认为 <code>false</code> 。
<code>fallback</code>	string	□	兜底回复，当请求机器人对话时，没有得到来自多轮对话、知识库或意图识别回复时，回复此内容。
<code>welcome</code>	string	□	机器人问候语。
<code>description</code>	string	□	机器人描述。

result / JSON Object

```
{
  "rc": 0,
  "data": {
    "clientId": "{{clientId}}",
    "secret": "{{secret}}",
    "name": "小巴",
    "description": "Test",
    "primaryLanguage": "zh_CN",
    "createdAt": "Mon Aug 02 2021 20:35:23 GMT+0800 (CST)"
  }
}
```

`clientId`: 初始化 [Chatbot](#) 类 的信息

`secret`: 初始化 [Chatbot](#) 类 的信息

获得聊天机器人列表

```
Chatopera#command("GET", "/chatbot?limit={{limit}}&page={{page}}")
```

path

KEY	TYPE	DEFAULT	DESCRIPTION
limit	int	默认值 20, 选填	返回数据条数
page	int	默认值 1, 选填	返回数据页面索引

result/ JSON Object

```
{  
    "rc": 0,  
    "total": 3,  
    "total_page": 3,  
    "data": [  
        {  
            "clientId": "{{clientId}}",  
            "name": "TestBot1627889023922",  
            "description": "",  
            "primaryLanguage": "zh_CN",  
            "createdAt": "Mon Aug 02 2021 15:23:44 GMT+0800 (CST)",  
            "secret": "{{secret}}",  
        },  
        ...  
    ]  
}
```

total: 所有数据条数

total_page: 所有页面数

类构造函数

构造函数

构造函数

```
Chatbot(clientId, secret [, botProvider])
```

参数说明

NAME	TYPE	REQUIRED	DESCRIPTION
clientId	string	□	在 机器人控制台/机器人/设置 中获取
secret	string	□	获取办法同上
botProvider	string	□	Chatopera 机器人平台地址, 当使用 Chatopera 云服务时, 该值为 https://bot.chatopera.com , 也是默认值

更多实例化例子

不同语言下, `Chatbot` 类的包名或引用方式不同, Node.js SDK 的实例化上文已经表述, 以下再介绍其它语言。

Java

```
import com.chatopera.bot.sdk.Chatbot;  
...  
Chatbot chatbot = new Chatbot(clientId, secret);
```

Python

```
from chatopera import Chatbot  
bot = Chatbot(clientId, secret)
```

PHP

假设使用[composer](#)作为包管理工具, 其它安装方式参考[chatopera-php-sdk](#)。

```
<?php  
  
include_once '**DIR** . "/vendor/autoload.php";  
$chatbot = new Chatopera\SDK\Chatbot($appId, \$secret);
```

Go

```
import (  
    "github.com/chatopera/chatopera-go-sdk"  
)  
...  
var chatbot = chatopera.Chatbot(clientId, secret)
```

Chatbot 类接口规范

发送请求

Chatbot 实例的核心接口是 `command`, 以下也使用 `Chatbot#command` 来指这个接口, 该接口是对 RestAPIs 的高级封装, 内部完成签名认证, `RequestHeaders` 和 `RequestBody` 等处理。

接口规范

```
result = chatbot.command(method, path [, body])
```

提示: `result` 返回在 Node.js 中使用 `await` 或 `Promise`, 参考 [快速开始](#); 其它语言直接用 `=` 便可获取。

参数说明

NAME	TYPE	REQUIRED	DESCRIPTION
method	string	□	对于资源的具体操作类型, 由 HTTP 动词表示。有效值包括 <code>GET</code> , <code>POST</code> , <code>PUT</code> , <code>DELETE</code> 和 <code>HEAD</code> 等
path	string	□	资源的执行路径, 通常包含资源实体名称或唯一标识, 也可能在 <code>path</code> 中使用 <code>queryString</code> 传递参数
body	JSON 数据结构	□	<code>body</code> 是请求中的数据, 对应 RestAPI 中的 Http Body

`method` 不同动词代表的含义一般如下:

- GET - 从服务器取出一项或多项资源;
- POST - 在服务器创建一个资源;
- PUT - 在服务器更新一个资源;
- DELETE - 在服务器删除一个资源。

还有更多类型的 `method`, 没有上述几种常用, 在此不进行赘述。

`queryString` 是 URL 的一部分。典型的 URL 看起来像这样: <http://server/resource?foo=A&bar=B>。其中, `foo=A&bar=B` 就是 `queryString`, 通常用来传递参数, 这个例子中包含两个参数: `foo` 值为 A; `bar` 值为 B。在下文中, `path` 参数中可能包含 `queryString`, 形式如 `foo={{var1}}&bar={{var2}}`, 需要把 `{{var1}}` 和 `{{var2}}` 替换为实际值。

`body` 数据是 JSON 格式的, 不同语言对于 JSON 格式支持方式不同。`JSON` 是一种轻量级的数据交换格式, 描述了使用键值对、数组、字符串、数字、日期和布尔类型等值存储对象。`JSON` 在不同语言下, 等价数据结构如下。

语言	JSON OBJECT	JSON ARRAY
JavaScript	<code>{...}</code>	<code>[...]</code>
Java	<code>org.json.JSONObject</code>	<code>org.json.JSONArray</code>
PHP	基本类型 <code>array</code>	基本类型 <code>array</code>
Python	基本类型 <code>dict</code>	基本类型 <code>list</code>
Go	<code>map[string]interface{}</code>	<code>[]map[string]interface{}</code>

下文表述时, 统一使用 `JSON`, `JSON Object` 和 `JSON Array` 代表 JSON 数据结构和其不同语言下的等价数据结构。

提示: 相对而言, JSON 等价的数据结构, 在获取 JSON Object 的键值或 JSON Array 的长度和成员时, 语法不同, 但都易于掌握。在使用时, 参考不同 SDK 的示例程序。

`body` 是否必填以及是 JSON Object 还是 JSON Array, 取决于 `method` 和 `path` 的值, 不同 `method` 和 `path` 的组合对应了不同的接口功能, 满足不同需求, 下文将介绍满足各种需求的 `method` 和 `path`, 并各个说明 `body` 参数。

返回值

返回值即请求结果, 针对接口定义, `Chatbot#command` 的返回值 `result` 是 JSON Object, 并有以下属性。

KEY	TYPE	DESCRIPTION
<code>rc</code>	int	response code, 返回码, 大于等于 0 的正整型。 <code>0</code> 代表服务器端按照请求描述, 正常返回结果; <code>rc</code> 不等于 0 是代表异常返回。
<code>data</code>	JSON	数据资源。正常返回时, 服务器端执行逻辑成功, 比如查询时, <code>data</code> 就是查询结果。
<code>msg</code>	string	消息, 当服务器端执行请求成功, 并且不需要返回数据资源时, 通过 <code>msg</code> 代表文本信息, 比如提示信息。
<code>error</code>	string	异常消息, 当服务器端返回异常时, 具体出错信息包含在 <code>error</code> 中。
<code>status</code>	JSON	全局任务的状态信息。
<code>total</code>	int	分页, 所有数据记录条数。
<code>current_page</code>	int	分页, 当前页码, (分页从 1 开始)。
<code>total_page</code>	int	分页, 所有页数。

每次请求结果中, `rc` 是必含有的属性, 其它属性为可能含有。不同 `rc` 的正整数形代表不同的异常, `data`、`status` 以及分页信息, 则因 `method` 和 `path` 而异, 以下进行详细介绍。

提示: 不同语言对返回值可能进行了封装, 但是不离其宗, 都是基于以上定义, 比如 Java SDK 中, 定义 `com.chatopera.bot.sdk.Response` 作为 `Chatbot#command` 接口返回值, `Response` 类提供 `getRc`、`getData` 和 `toJSON` 等方法, 提升代码可读性。在使用时, 参考不同 SDK 的示例程序。

下文中使用的 `method`, `path`, `body` 和 `result` 等均代表以上介绍的概念。

常用 APIs 介绍

在 [Chatopera 云服务文档中心](#), 仅介绍了关键、常用的 APIs 作为示范, 这些 APIs 可以在左侧的子级菜单中获得详细介绍。

全部 APIs 文档

对于高级用户, Chatopera 提供了一个浏览全部 RestAPIs 的站点, 这些 APIs 一样可以通过 `Chatbot#command` 接口调用, 因为 `Chatbot#command` 就是封装了的 RestAPIs。

<https://api-docs.chatopera.com/>

机器人基本管理

获取机器人画像

```
Chatbot#command("GET", "/")
```

result / JSON Object

```
{
    "rc": 0,
    "data": {
        "name": "小巴巴",
        "fallback": "请联系客服。",
        "description": "Performs Tasks or retrieves FAQ.",
        "welcome": "你好，我是机器人小巴巴",
        "primaryLanguage": "zh_CN",
        "status": {
            "reindex": 0,
            "retrain": 0
        }
    }
}
```

KEY	TYPE	DESCRIPTION
name	string	机器人名字。
fallback	string	兜底回复，当请求机器人对话时，没有得到来自多轮对话、知识库或意图识别回复时，回复此内容。
welcome	string	机器人问候语。
description	string	机器人描述。
primaryLanguage	string	机器人语言。
status	JSON Object	全局任务的执行状态，reindex 代表知识库同步自定义词典的状态；retrain 代表意图识别同步自定义词典的状态。

更新机器人画像

```
Chatbot#command("PUT", "/", body)
```

body / JSON Object

```
{
    "fallback": "请联系客服。",
    "description": "我的超级能力是对话",
    "welcome": "你好，我是机器人小巴巴"
}
```

result / JSON Object

```
{  
  "rc": 0,  
  "data": {  
    "name": "小巴巴",  
    "fallback": "请联系客服。",  
    "description": "Performs Tasks or retrieves FAQ.",  
    "welcome": "你好，我是机器人小巴巴"  
}
```

获取全局任务状态

```
Chatbot#command("GET", "/status")
```

result / JSON Object

```
{  
  "rc": 0,  
  "data": {  
    "status": {  
      "reindex": 0,  
      "retrain": 0  
    }  
  }  
}
```

删除聊天机器人

删除聊天机器人，该接口销毁机器人资源，并且不可逆。使用时需格外谨慎！

```
Chatbot#command("DELETE", "/")
```

result / JSON Object

```
{  
  "rc": 0,  
  "msg": "done"  
}
```

对话检索

对话检索接口，就是将对话用户的请求发送给机器人，获得机器人的回复。

- [检索多轮对话](#): 从多轮对话获得回复
- [检索知识库](#): 从知识库获得回复
- [检索意图识别](#): 从意图识别模块获得回复

检索多轮对话，也同时会从知识库、意图识别、对话脚本中获得答案并按照算法回复最佳答案，也是 Chatopera 官方最推荐的集成形式，使用检索多轮对话接口，可以定制出更为智能的对话机器人。了解详情，请阅读[《多轮对话的工作机制》](#)。

检索多轮对话

多轮对话是通过脚本规则、函数编程实现问答服务，在检索多轮对话接口中，同时融合了知识库参与回复决策，返回结果，尤其是通过知识库答案路由到指定话题的指定触发器，非常实用。为了方便使用，宜先理解[多轮对话的工作机制和工作原理](#)，熟悉多轮对话机制可以真正将 Chatopera 机器人平台的能量发挥到最大。

```
Chatbot#command("POST", "/conversation/query", body)
```

body / JSON Object

```
{
  "fromUserId": "{{userId}}",
  "textMessage": "想要说些什么",
  "faqBestReplyThreshold": 0.6,
  "faqSuggReplyThreshold": 0.35,
  "extras": {},
  "isDebug": false
}
```

KEY	TYPE	REQUIRED	DESCRIPTION
fromUserId	string	□	用户唯一 ID，用户 ID 由业务系统传递或生成，保证每个用户用唯一字符串
textMessage	string	□	用户输入的对话文字
faqBestReplyThreshold	number	□	知识库最佳回复阈值，知识库中置信度超过该值通过返回值 <code>string</code> 和 <code>params</code> 返回；可以在机器人平台管理控制台的设置页面设置默认值，使用 API 传递参数覆盖默认值
faqSuggReplyThreshold	number	□	知识库建议回复阈值，知识库中置信度超过该值的问答对通过返回值 <code>faq</code> 属性返回；可以在机器人平台管理控制台的设置页面设置默认值，使用 API 传递参数覆盖默认值
isDebug	boolean	□	是否返回调试信息，调试信息包括匹配信息等
extras	JSONObject 或 JSONArray	□	在消息中，添加自定义的信息，然后在多轮对话脚本的函数 <code>this.message.extras</code> 和 <code>this.user.history</code> 中使用

其中，`extras` 用以支持更灵活，自定义的场景，使用[参考](#)。

result/ JSON Object

```
{
  "rc": 0,
  "data": {
    "state": "default",
    "string": "方法",
    "logic_is_unexpected": false,
    "logic_is_fallback": false,
    "service": {
      "provider": "faq",
      "docId": "{{doctId}}",
      "score": 0.3781,
      "faqBestReplyThreshold": 0.37,
      "faqSuggReplyThreshold": 0.1,
      "categories": ["x", "y"]
    },
    "botName": "小巴巴",
    "faq": [
      {
        "id": "{{doctId}}",
        "score": 0.3781,
        "post": "查看相似问题可能的",
        "categories": ["x", "y"]
      }
    ]
  }
}
```

state: 业务字段, 可以在多轮对话脚本中设置

string: 机器人回复的文本内容

topicName: 机器人会话主题

logic_is_fallback: 是否是兜底回复

botName: 机器人的名字

faq: 知识库中匹配 textMessage 的相似度超过 **faqSuggReplyThreshold** 的记录, 数组类型

service 代表返回的数据来源, **provider:conversation** 指多轮对话, **provider:faq** 指知识库, **provider:intent** 指意图识别; 不同数据来源也会提供相应信息。

PROVIDER	KEY	解释
faq		
	docId	文档 ID
	post	标准问
	score	分数
intent	意图识别	更多描述参考 意图匹配器
	intent.name	意图名称
	intent.state	意图会话状态
	intent.entities	意图中的命名实体

PROVIDER	KEY	解释
conversation	多轮对话	
fallback	兜底回复	
mute	该用户被该机器人屏蔽	

检索知识库

```
Chatbot#command("POST", "/faq/query", body)
```

body / JSON Object

```
{  
  "query": "查找相似的问题",  
  "fromUserId": "{{userId}}",  
  "faqBestReplyThreshold": 0.5,  
  "faqSuggReplyThreshold": 0.1  
}
```

查询匹配是根据阈值设置的，也就是 `faqBestReplyThreshold` 和 `faqSuggReplyThreshold`，前者是最佳回复阈值，后者是建议回复阈值，前者高于后者，都在 [0-1] 区间，是问题相似度，值越大，越从知识库查询相似度高的记录，1 代表问题和查询完全一样。

result/ JSON Object

```
{  
    "rc": 0,  
    "data": [  
        {  
            "id": "{{docId}}",  
            "score": 0.48534,  
            "post": "查看相似问题不可能的",  
            "replies": [  
                {  
                    "rtype": "plain",  
                    "enabled": true,  
                    "content": "方法"  
                }  
            ],  
            "categories": [  
                "节日",  
                "农历"  
            ]  
        },  
        {  
            "id": "{{docId}}",  
            "score": 0.32699,  
            "post": "聊天",  
            "replies": [  
                {  
                    "rtype": "plain",  
                    "content": "foo",  
                    "enabled": true  
                },  
                {  
                    "rtype": "plain",  
                    "content": "bar",  
                    "enabled": true  
                }  
            ],  
            "categories": []  
        }  
    ]  
}
```

`categories` 是分类信息。分类是树状结构，返回值是按照树状结构顺序的数组。

检索意图识别

意图识别是基于请求者的文本内容分析意图，然后基于意图追问意图槽位信息的对话，这部分的详细介绍参考<https://docs.chatopera.com/products/chatbot-platform/intent.html>，下面重点介绍在系统集成中，通过意图识别服务提供智能问答。

什么是“会话”

“会话”(session)在代表一个用户对话的周期，认为用户在这个周期内是为了完成某个任务的。从确定任务，到得到和这个任务相关的信息，这个 session 就正常结束了，但是如果用户变化了任务，这个 session 就不能正常结束。开发者选择什么时候创建新的 session，但是服务器端决定什么时候完成这个 session，session 的管理涉及：意图的确定，意图参数的确定，会话最大空闲时间，会话是否解决(resolved)。

- 训练完成后请求对话，需要先创建会话，会话会绑定 0-1 个任务：刚开始不知道用户意图，当确定用户意图后，该 session 就只和这个意图相关；
- 会话有最大空闲日期，如果在半个小时内没有更新，会被服务器删除；
- 会话可以任意创建，只要没有超过最大空闲日期都是有效的；
- 不同的用户使用不同的会话，同一个用户可以同时有多个会话，但是为了实际效果，用户最好同时只使用一个会话；
- 当用户的意图和槽位信息被全部确认，会话包含的 resolved 字段会被设置为 true，这时开发者可以再次创建一个新的会话。

创建会话

```
Chatbot#command("POST", "/clause/prover/session", body)
```

body / JSON Object

```
{  
    "uid": "{{userId}}",  
    "channel": "{{channelId}}"  
}
```

KEY	TYPE	REQUIRED	DESCRIPTION
userID	string	□	用户标识，由字母和数字组成的字符串。开发者自定义，保证每个用户唯一
channelId	string	□	用户来源的渠道标识，由字母和数字组成的字符串。由开发者自定义，保证每个渠道唯一

result/ JSON Object

```
{  
    "rc": 0,  
    "data": {  
        "intent_name": null,  
        "uid": "{{userId}}",  
        "channel": "{{channelId}}",  
        "resolved": null,  
        "id": "{{sessionId}}",  
        "entities": null,  
        "createdate": "2019-08-28 18:08:51",  
        "updatedate": "2019-08-28 18:08:51"  
        "ttl": 3600  
    },  
    "error": null  
}
```

intent_name: 意图名字

id: 会话 ID

resolved: 该会话是否完成收集参数

entities: 参数列表，完成填槽或待填槽

ttl: 该会话信息在多少秒后过期，每个会话默认是 1 小时的空闲周期，在该时间内没有跟进的对话，则会话过期

检索意图识别

```
Chatbot#command("POST", "/clause/prover/chat", body)
```

body / JSON Object

```
{
  "fromUserId": "{{userId}}",
  "session": {
    "id": "{{sessionId}}"
  },
  "message": {
    "textMessage": "我想购买明天火车票"
  }
}
```

KEY	TYPE	REQUIRED	DESCRIPTION
userId	string	□	用户唯一 ID, 用户 ID 由业务系统传递或生成, 保证每个用户用唯一字符串
sessionId	string	□	使用创建会话接口创建
textMessage	string	□	用户输入的对话文字

result/ JSON Object

```
{
  "rc": 0,
  "data": {
    "session": {
      "intent_name": "{{intentName}}",
      "uid": "{{userId}}",
      "channel": "{{channelId}}",
      "resolved": false,
      "id": "{{sessionId}}",
      "entities": [
        {
          "name": "cityName",
          "val": "中国首都"
        }
      ],
      "createdate": "2019-08-28 18:15:24",
      "updatedate": "2019-08-28 18:15:24",
      "ttl": 3595
    },
    "message": {
      "textMessage": "你想做什么工具",
      "is_fallback": null,
      "is_proactive": true
    }
  },
  "error": null
}
```

查看会话详情

```
Chatbot#command("GET", "/clause/prover/session/{{sessionId}}")
```

result/ JSON Object

```
{  
    "rc": 0,  
    "data": {  
        "intent_name": "{{intentName}}",  
        "uid": "{{userId}}",  
        "channel": "{{channelId}}",  
        "resolved": false,  
        "id": "{{sessionId}}",  
        "entities": null,  
        "createdate": "2019-08-28 18:41:56",  
        "updatedate": "2019-08-28 18:41:56",  
        "ttl": 3600  
    },  
    "error": null  
}
```

词典管理

创建自定义词典

```
Chatbot#command("POST", "/clause/customdicts", body)
```

body / JSON Object

```
{
  "name": "{{customDictName}}",
  "type": "vocab"
}
```

KEY	TYPE	REQUIRED	DESCRIPTION
name	string	<input type="checkbox"/>	自定词典名称, 使用小写字母和数据组成的字符串

result/ JSON Object

```
{
  "rc": 0,
  "data": {
    "name": "{{customDictName}}",
    "description": "",
    "samples": null,
    "createdate": "2019-08-07 19:59:14",
    "updatedate": "2019-08-07 19:59:14"
  }
}
```

获取自定义词典列表

```
Chatbot#command("GET", "/clause/customdicts?limit={{limit}}&page={{page}}")
```

result/ JSON Object

```
{
  "rc": 0,
  "total": 3,
  "current_page": 1,
  "total_page": 3,
  "data": [
    {
      "name": "{{customDictName}}",
      "description": "",
      "samples": null,
      "createdate": "2019-08-07 19:58:08",
      "updatedate": "2019-08-07 19:58:08"
    }
  ]
}
```

更新自定义词典

```
Chatbot#command("PUT", "/clause/customdicts/{{customDictName}}", body)
```

path

KEY	TYPE	DEFAULT	DESCRIPTION
customDictName	string	无默认值, 必填	自定义词典标识

body / JSON Object

```
{
  "description": "高级轿车品牌"
}
```

result/ JSON Object

```
{
  "rc": 0,
  "data": {
    "name": "pizza",
    "description": "",
    "samples": null,
    "createdate": "2020-07-20 20:52:00",
    "updatedate": "2020-07-20 20:51:59",
    "type": "vocab",
  }
}
```

删除自定义词典

```
Chatbot#command("DELETE", "/clause/customdicts/{{customDictName}}")
```

path

KEY	TYPE	DEFAULT	DESCRIPTION
customDictName	string	无默认值, 必填	自定义词典标识

result/ JSON Object

```
{
  "rc": 0,
  "msg": "success",
  "error": null,
  "data": {
    "status": {
      "needReindex": 2,
      "needRetrain": 2
    }
  }
}
```

创建知识库分类

```
Chatbot#command("POST", "/faq/categories", body)
```

body / JSON Object

```
{  
    "label": "{{categoryText}}"  
}
```

result/ JSON Object

```
{  
    "rc": 0,  
    "data": {  
        "value": "{{categoryId}}",  
        "categories": [  
            {  
                "value": "{{categoryId}}",  
                "label": "{{categoryText}}",  
                "children": [  
                    {  
                        "value": "I7vfx47i5I",  
                        "label": "二级分类名"  
                    }  
                ]  
            },  
            {  
                "value": "{{categoryId}}",  
                "label": "x2"  
            }  
        ]  
    }  
}
```

获取知识库分类信息

```
Chatbot#command("GET", "/faq/categories")
```

result / JSON Object

```
{  
    "rc": 0,  
    "data": [  
        {  
            "value": "{{categoryId}}",  
            "label": "{{categoryText}}",  
            "children": [  
                {  
                    "value": "{{categoryId}}",  
                    "label": "{{categoryText}}"  
                }  
            ]  
        }  
    ]  
}
```

更新知识库分类

```
Chatbot#command("", "/faq/categories", body)
```

```
{  
    "value": "{{categoryId}}",  
    "label": "新的名字"  
}
```

result/ JSON Object

```
{  
    "rc": 0,  
    "data": [  
        {  
            "value": "wwQyjS310",  
            "label": "一级分类名",  
            "children": [  
                {  
                    "value": "{{categoryId}}",  
                    "label": "新的名字"  
                }  
            ]  
        }  
    ]  
}
```

删除知识库分类

```
Chatbot#command("DELETE", "/faq/categories/{{categoryId}}")
```

path

KEY	TYPE	DEFAULT	DESCRIPTION
categoryId	string	无默认值, 必填	分类唯一标识

result/ JSON Object

```
{  
    "rc": 0,  
    "data": [  
        {  
            "value": "TSDD-W6T9",  
            "label": "x2"  
        }  
    ]  
}
```

创建问答对

```
Chatbot#command("post", "/faq/database", body)
```

body / JSON Object

```
{  
  "post": "如何查看快递单号",  
  "replies": [  
    {  
      "rtype": "plain",  
      "content": "foo",  
      "enabled": true  
    },  
    {  
      "rtype": "plain",  
      "content": "bar",  
      "enabled": true  
    }  
  ],  
  "enabled": true,  
  "categoryTexts": [  
    "一级分类名",  
    "二级分类名"  
  ]  
}
```

result / JSON Object

```
{  
  "rc": 0,  
  "data": {  
    "id": "{docId}",  
    "replyLastUpdate": "{{replyLastUpdate}}"  
  }  
}
```

更新知识库问答对

```
Chatbot#command("PUT", "/faq/database/{{docId}}", body)
```

path

KEY	TYPE	DEFAULT	DESCRIPTION
docId	string	无默认值, 必填	问答对标识

body / JSON Object

```
{  
  "post": "怎么开通微信支付?",  
  "replyLastUpdate": "{{replyLastUpdate}}",  
  "replies": [  
    {  
      "rtype": "plain",  
      "content": "foo2",  
      "enabled": true  
    },  
    {  
      "rtype": "plain",  
      "content": "bar2",  
      "enabled": true  
    }  
  ],  
  "enabled": true  
}
```

result / JSON Object

```
{
  "rc": 0,
  "data": {
    "id": "{{docId}}",
    "replyLastUpdate": "{{replyLastUpdate}}"
  }
}
```

获取问答对列表

```
Chatbot#command("GET", "/faq/database?limit={{limit}}&page={{page}}&q={{q}}")
```

path

KEY	TYPE	DEFAULT	DESCRIPTION
limit	int	20	返回最多多少条数据
page	int	1	返回第多少页
q	string	空	问答对匹配时，问题应包含的关键字

result / JSON Object

```
{
  "total": 3,
  "current_page": 1,
  "total_page": 1,
  "data": [
    {
      "post": "如何查看快递单号",
      "categories": [
        "wwQyjS310",
        "I7vfx47i5I"
      ],
      "enabled": true,
      "id": "{{docId}}"
    }
  ],
  "rc": 0,
  "status": {
    "reindex": 0,
    "retrain": 0
  }
}
```

创建问答对相似问

```
Chatbot#command("POST", "/faq/database/{{docId}}/extend", body)
```

path

KEY	TYPE	DEFAULT	DESCRIPTION
docId	string	无默认值, 必填	问答对标识

body / JSON Object

```
{
  "post": "怎样支持微信支付?"
}
```

```
{  
    "rc": 0,  
    "data": {  
        "id": "{{extendId}}"  
    }  
}
```

获取问答对相似问列表

```
Chatbot#command("GET", "/faq/database/{{docId}}/extend")
```

path

KEY	TYPE	DEFAULT	DESCRIPTION
docId	string	无默认值, 必填	问答对标识

result / JSON Object

```
{  
    "total": 1,  
    "current_page": 1,  
    "total_page": 1,  
    "data": [  
        {  
            "post": "怎样支持微信支付?",  
            "postId": "{{docId}}",  
            "enabled": true,  
            "id": "{{extendId}}"  
        }  
    ],  
    "rc": 0  
}
```

更新问答对相似问

```
Chatbot#command("PUT", "/faq/database/{{docId}}/extend/{{extendId}}", body)
```

path

KEY	TYPE	DEFAULT	DESCRIPTION
docId	string	无默认值, 必填	问答对标识
extendId	string	无默认值, 必填	扩展问标识

body / JSON Object

```
{  
    "post": "怎样支持微信支付?"  
}
```

result / JSON Object

```
{  
    "rc": 0,  
    "data": {  
        "id": "{{extendId}}"  
    }  
}
```

删除问答对相似问

```
Chatbot#command("DELETE", "/faq/database/{{docId}}/extend/{{extendId}}")
```

path

KEY	TYPE	DEFAULT	DESCRIPTION
docId	string	无默认值, 必填	问答对标识
extendId	string	无默认值, 必填	扩展问标识

result / JSON Object

```
{  
    "rc": 0,  
    "msg": "done"  
}
```

删除问答对

```
Chatbot#command("DELETE", "/faq/database/{{docId}}")
```

result / JSON Object

```
{  
    "rc": 0,  
    "msg": "done"  
}
```

获取知识库热门问题

```
Chatbot#command("GET", "/faq/database/inquiryrank?topN={{topN}}")
```

path

KEY	TYPE	DEFAULT	DESCRIPTION
topN	int	10	获得热门问题的条数, 即导出数据的条数

result / JSON Object

```
{  
  "rc": 0,  
  "data": [  
    {  
      "docId": "{{docId}}",  
      "inquiryscore": "13",  
      "post": "{{post}}",  
      "enabled": true,  
      "categories": [  
        "s1",  
        "s2"  
      ]  
    },  
    {  
      "docId": "{{docId}}",  
      "inquiryscore": "0",  
      "post": "{{post}}",  
      "enabled": true,  
      "categories": []  
    },  
    ...  
  ]  
}
```

其中，data 内元素按照 `inquiryscore` 的值，降序排列，即该问题越热门，越靠上。

用户和对话历史

获取用户列表

```
Chatbot#command("GET", "/users")
```

path

KEY	TYPE	DEFAULT	DESCRIPTION
limit	int	1	返回最多多少条数据
page	int	20	返回第多少页

result / JSON Object

```
{
    "rc": 0,
    "total": 5,
    "current_page": 1,
    "total_page": 1,
    "data": [
        {
            "userId": "{{userId}}",
            "lasttime": "2020-07-19T14:12:13.690Z",
            "created": "2020-07-19T13:48:02.225Z"
        }
    ]
}
```

userId: 和机器人对话的用户标识

lasttime: 最后沟通时间

created: 第一次沟通时间

屏蔽用户

```
Chatbot#command("POST", "/users/{{userId}}/mute")
```

path

KEY	TYPE	DEFAULT	DESCRIPTION
userId	string	无默认值, 必填	用户唯一标识

result / JSON Object

```
{
    "rc": 0,
    "data": {}
}
```

取消屏蔽

```
Chatbot#command("POST", "/users/{{userId}}/unmute")
```

path

KEY	TYPE	DEFAULT	DESCRIPTION
userId	string	无默认值, 必填	用户唯一标识

result / JSON Object

```
{
  "rc": 0,
  "data": {}
}
```

是否被屏蔽

```
Chatbot#command("POST", "/users/{{userId}}/ismute")
```

result / JSON Object

```
{
  "rc": 0,
  "data": {
    "mute": false
  }
}
```

`data.mute` 返回 boolean 类型值。

获取用户画像信息

```
Chatbot#command("GET", "/users/{{userId}}/profile")
```

path

KEY	TYPE	DEFAULT	DESCRIPTION
userId	string	无默认值, 必填	用户唯一标识

result/ JSON Object

```
{
  "rc": 0,
  "data": {
    "userId": "postman9",
    "name": null,
    "lasttime": "2020-07-19T14:12:13.690Z",
    "created": "2020-07-19T13:48:02.225Z",
    "profile": {},
    "mute": false
  }
}
```

获取聊天历史

```
Chatbot#command("GET", "/users/{{userId}}/chats?limit={{limit}}&page={{page}}")
```

path

KEY	TYPE	DEFAULT	DESCRIPTION
userId	string	无默认值, 必填	用户唯一标识
limit	int	1	返回最多多少条数据
page	int	20	返回第多少页

result / JSON Object

```
{
  "rc": 0,
  "total": 16,
  "current_page": 1,
  "total_page": 1,
  "data": [
    {
      "userId": "postman9",
      "textMessage": "方法",
      "direction": "outbound",
      "service": "faq",
      "confidence": 0.3781,
      "docId": "AXNCspoufx0JhfysI3-Z",
      "created": "2020-07-19T14:12:13.802Z"
    }
  ]
}
```

total: 该用户和机器人之间对话总数

current_page: 当前页

total_page: 总页数

userId: 用户标识

textMessage: 文本内容

direction: 消息传递方向, 【inbound】为消费者发送, 【outbound】为机器人发送

service: 提供回复的服务

confidence: 置信度

created: 消息创建时间

语音识别

目前, Chatopera 机器人平台只支持中文普通话 (zh_CN) 机器人做中文语音识别。

语音文件格式: 16k 采样率, 单通道, PCM。

```
Channels      : 1
Sample Rate   : 16000
Precision     : 16-bit
Sample Encoding: 16-bit Signed Integer PCM
```

[下载音频示例](#)。

语音识别接口可使用两种形式提交语音文件: 1) 文件路径; 2) 语音文件 base64 格式字符串。

提交文件路径识别

```
Chatbot#command("POST", "/asr/recognize", body)
```

body / JSON Object

```
{
  "filepath": "{{WAV_FILE_ABS_PATH}}",
  "nbest": 5,
  "pos": true
}
```

KEY	TYPE	REQUIRED	DESCRIPTION
filepath	string	□	语音文件绝对路径, 或当前应用启动路径(CWD) 的相对路径
nbest	int	□	语音识别可返回多个结果, 方便查询关键词, 默认 5
pos	boolean	□	返回值是否分词, 默认 false

result/ JSON Object

```
{
  "rc": 0,
  "data": {
    "duration": 6250,
    "predicts": [
      {
        "confidence": 0.960783,
        "text": "上海浦东机场入境房输入全闭环管理"
      },
      {
        "confidence": 0.960767,
        "text": "上海浦东机场入境防输入全闭环管理"
      },
      {
        "confidence": 0.960736,
        "text": "上海浦东机场入境坊输入全闭环管理"
      }
    ]
  }
}
```

KEY	TYPE	DESCRIPTION
duration	int	语音文件时间长度, 单位 毫秒, 比如 6250 代表 6.25 秒
predicts	JSONArray	识别结果
text	string	识别得到的文本
confidence	double	置信度, [0-1], 值越大越有可能, <code>predicts</code> 按 <code>confidence</code> 降序

提交 base64 字符串识别

```
Chatbot#command("POST", "/asr/recognize", body)
```

body / JSON Object

```
{
  "type": "base64",
  "data": "data:audio/wav;base64,{BASE64_STRING}",
  "nbest": 5,
  "pos": true
}
```

KEY	TYPE	REQUIRED	DESCRIPTION
type	string	□	固定值 <code>base64</code>
data	string	□	语音文件使用 base 编码的字符串, 并且必须以 <code>data:audio/wav;base64,</code> 作为前缀, 比如 <code>data:audio/wav;base64,xyz...</code>
nbest	int	□	语音识别可返回多个结果, 方便查询关键词, 默认 5
pos	boolean	□	返回值是否分词, 默认 false

此处, `nbest` 和 `pos` 与 提交文件路径识别 API 一致。

result/ JSON Object

返回值与 提交文件路径识别 API 一致。

```
{  
    "rc": 0,  
    "data": {  
        "duration": 6250,  
        "predicts": [  
            {  
                "confidence": 0.960783,  
                "text": "上海浦东机场入境房输入全闭环管理"  
            },  
            {  
                "confidence": 0.960767,  
                "text": "上海浦东机场入境防输入全闭环管理"  
            },  
            {  
                "confidence": 0.960736,  
                "text": "上海浦东机场入境坊输入全闭环管理"  
            }  
        ]  
    }  
}
```

KEY	TYPE	DESCRIPTION
duration	int	语音文件时间长度, 单位 毫秒, 比如 6250 代表 6.25 秒
predicts	JSONArray	识别结果
text	string	识别得到的文本
confidence	double	置信度, [0-1], 值越大越有可能, predicts 按 confidence 降序

常见问题

SDK

返回错误 invalid appId

```
{"rc":1,"error":"invalid appId."}
```

这是因为clientId 和 secret 配置不正确。

返回错误 invalid timestamp

```
{"rc":1,"error":"invalid timestamp."}
```

操作系统需要设置的时间同步为互联网时间，参考

<https://www.sysgeek.cn/manage-time-server-windows-10/>

也可以设置为阿里云时间同步器

https://blog.csdn.net/qq_35448976/article/details/78977164

Java SDK / Maven 执行抛出异常

`mvn install` 抛出异常，SunCertPathBuilderException

```
sun.security.provider.certpath.SunCertPathBuilderException: unable to find valid certification path to requested target
```

解决方案

计费及保障

Chatopera 云服务是企业级的服务，依托于 Chatopera 的 Platform as a Service 服务上，使用监控、告警、负载均衡、集群等功能保证服务的高可靠性。

计费及发票

Chatopera 云服务计费及发票，参考[文档](#)。

私有部署

Chatopera 云服务私有部署，即 Chatopera 机器人平台，参考[文档](#)。

服务条款

Chatopera 云服务用户 & 私有部署客户使用规范和须知等，参考[文档](#)。

服务水平协议

Chatopera 承诺的可靠性保证等，参考[文档](#)。

计费及发票

Chatopera 云服务的费用中心可以查询计费情况、资源包使用和购买、开具发票等信息。

<https://bot.chatopera.com/billing>

以下是对使用该模块的进一步的介绍。

API 计费预估

Chatopera 云服务的机器人平台消费情况，按照现有用户统计，90% 的用户的年使用费用在 ¥ 400元 ~ ¥ 2000元之间，这个区间可以帮助您做出适合的决策，因行业和用途的差异较大，不做进一步的分析。

您可以通过以下两个方式更准确的预估费用：

- 购买 [¥ 100 元](#) 的资源包，评测使用的天数；
- 通过下文的内容详细了解计费标准，预估您的业务量，计算费用。

在您获得了价格费用之后，只需要衡量产品是否满足了您的需求。您是不需要与之对比其它聊天机器人厂商的价格的，因为以下两点：

- Chatopera 云服务的机器人平台在产品定位、功能设计、技术实现等方面具有独创性，不同厂商之间有产品差异，不具备比较的标准；
- Chatopera 云服务的机器人平台具备完善的知识库、多轮对话等能力。只有某些厂商抄袭 Chatopera，但是 Chatopera 没有抄袭过其它厂商。

API 计费服务接口

Chatopera 云服务的计费接口包括：

1) 对话检索 API，按照 API 调用次数进行计费

- 检索知识库对话，[说明文档](#)
- 检索多轮对话，[说明文档](#)
- 检索意图识别，[说明文档](#)

2) 语音识别 API，每秒折合为 1 次，按照 API 调用次数进行计费

- 语音识别-中文普通话，[说明文档](#)

换算时，剩余的末尾时间不足一秒，则剩余时间按照 1 秒算。

API 计费标准

购买的资源包数额越大，则 API 调用配额次数越大、资源包周期越长、换算为单次请求越便宜。

资源包包括次数额度，折合每次 API 请求的价格如下：

服务范围	所有计费接口			
计费接口	检索知识库对话	检索多轮对话	检索意图识别	语音识别-中文普通话
资源包内容				
通用 100 元资源包 5000 次对话检索 API 请求，购买日期起 3 个月有效；语音识别 ASR 请求，每秒折合 1 次。	5,000 次 0.020 元/次			
通用 1000 元资源包 10 万次对话检索 API 请求，购买日期起 6 个月有效；语音识别 ASR 请求，每秒折合 1 次。		100,000 次 0.010 元/次		
通用 10000 元资源包 200 万次对话检索 API 请求，购买日期起 12 个月有效；语音识别 ASR 请求，每秒折合 1 次。	2,000,000 次 0.005 元/次			

资源包购买地址 -

<https://bot.chatopera.com/billing/purchase>

最终价格以【资源包购买地址】为准。

发票

发票管理地址 -

<https://bot.chatopera.com/billing/open-invoice>

在控制台右上角的导航栏进入【费用中心-开具发票】进行申请，点击【发票管理】查看开具状态，在开票过程中，可能通过短信、邮件和电话方式联系，请保证填写正确信息和联系畅通。

提交申请发票如下图：

开具发票 [回到费用中心](#)

- 开通文字识别服务后，您将拥有对应额度的免费资源包，详情请参考[免费额度规则](#)
- 预付费资源包有效期均为1年，1年内若资源包次数未使用完，则过期作废；预付费资源包使用后不支持剩余次数冻结理由退款。
- 调用量的扣费顺序为“免费资源包->付费资源包->后付费”。当您的免费资源包耗尽时，服务将面临不可用风险，为保证预付费资源包或前往设置页开通后付费模式。
- 当该服务仅剩一个可用资源包，且余量小于20%和余量为0时，系统都会通过微信/短信/邮件/站内信向您推送预警消息（[查看提醒设置](#)）

* 发票抬头

比如：北京华夏春松科技有限公司

* 纳税人识别号

纳税人办理各类涉税事项的唯一代码标识

发票类型

增值税普通发票

分类

研发和技术服务

其它使用须知

1. 资源包

资源包有有效期和配额，用户在使用服务过程中，按照 API 调用次数或时长（ASR 语音识别）换算为计数，从配额中扣除。

2. 配额扣除

API 调用的计数定期从配额中扣除，记录查看【配额扣除】。配额扣除的顺序为“赠送资源包 > 购买资源包”。

<https://bot.chatopera.com/billing/quota>

配额扣除的记录如下图：

开始时间	结束时间	配额计数	资源包类型
2023-02-10 14:27	2023-02-10 15:07	-5	通用 100 元资源包
2023-02-10 14:07	2023-02-10 14:27	-25	通用 100 元资源包
2023-02-10 13:47	2023-02-10 14:07	-15	通用 100 元资源包
2023-02-10 13:27	2023-02-10 13:47	-1	通用 100 元资源包
2023-02-10 13:06	2023-02-10 13:27	-1	通用 100 元资源包
2023-02-10 12:26	2023-02-10 13:06	-4	通用 100 元资源包
2023-02-10 11:46	2023-02-10 12:26	-3	通用 100 元资源包

3. 注册送资源包

注册账户成功后，您将获得赠送资源包一个，请在其有效期内尽量使用：体验产品和上线服务。

4. 无资源包停服

当没有可用配额时，用户服务请求被拒绝，服务不可用，直至购买资源包后恢复。为保证业务不受影响，前往【购买资源包】页面购买。

5. 配额不足告警

资源包从下单支付后生效。若在有效期内配额未用尽，余量作废。仅剩一个可用资源包，且其余量小于 20% 时，平台会通过短信/邮件向您推送预警消息，消息通知可能有数小时延迟，收到通知后请尽快购买资源包。

6. 获得赠送资源包

反馈建议：

- 开发环境搭建、功能咨询和使用问题；
- 提交软件缺陷；

- 描述新需求、反馈建议；
- 瓶颈分析、性能优化建议和安全漏洞等。

打开链接 -

<https://github.com/chatopera/docs/issues/new/choose>

请在 issue 中填写您在 Chatopera 云服务的用户名，然后联系平台客服 info@chatopera.com 说明您的 Issue 地址和平台账户信息（用户名，手机号）。

根据反馈的内容的严重程度，合理程度等，给予 【通用 100 元资源包】，【通用 1000 元资源包】的赠送资源包。

私有部署

对于企业客户，Chatopera 提供 **Chatopera** 机器人平台的私有部署支持，**Chatopera** 机器人平台除计费和支付模块、账号注册和管理模块与**Chatopera** 云服务不同外，核心的智能问答服务模块和操作均一致，在产品层面是同一套代码，集成 SDK，SDK 接口等均适用，具体的产品功能介绍，参考[文档](#)。

企业用户获得 Chatopera 机器人平台实例，是安装到企业的指定的设备上，私有部署收取软件的商业授权证书费用，不按 API 请求数量计费，具体资费见下。

价格

-	专业版	加强版	全家桶
画像设置	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>
使用统计	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>
词典管理	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>
知识库	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>
多轮对话	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>
测试对话	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>
对话历史	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>
系统集成(SDK)	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>
意图识别	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>
聚类分析	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>
语音识别	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>
第1套 实例价格	RMB 150,000	RMB 250,000	RMB 350,000
使用年限	永久	永久	永久

补充说明：

- 春松客服兼容 Chatopera 机器人平台私有部署实例，使用方法参考[文档](#)
- 因为没有意图识别模块，专业版中，不支持[意图匹配器](#)
- 每套实例，代表一个独立的软件服务；软件运行在一个固定的操作系统提供访问地址，为一个单独的实例
- 第2套实例价格（及以后每套）：该版本第一套价格的 10%
- 升配或降配：从**专业版**可以升级到**加强版**和**全家桶**，也可以从**加强版**升级到**全家桶**，升级办法为补差价；不支持降配，比如从**全家桶**降至**加强版**
- 质保和升级服务单独付费；第一年免费，第二年及以后每年价格为私有部署总价的 15%；质保和升级累积原则，即第二年没有购买质保和升级服务，第三年需要时，要一并支付第二年的费用；累积原则上限为 5 年，超过 5 年的，按 5 年计算；

质保和升级服务说明见后文

- 该价格会根据市场和季节进行调整，在下单时，合同商定价格如果与上述表格不一致，本公司具有最终解释权

部署硬件和软件配置要求

部署方案使用 Docker 和 Docker Compose；Docker 部署可实现跨平台和操作系统的目的。

以下以单机部署为例，最低配置要求：

-	专业版	加强版	全家桶
内存	16GB	32GB	64GB
CPU 颗数	4	8	16
硬盘	125GB	125GB	125GB
架构体系	x86_64	x86_64	x86_64
支持并发	10 qps	20 qps	20 qps

补充说明：

- Chatopera 机器人平台，在密集计算和机器学习方面，使用 C++，因此具备良好的性能，对操作系统的资源要求不高
- 操作系统：Linux，兼容不同发布版本(Ubuntu (*推荐), CentOS, Redhat, Debian, etc.)
- 并发：10 qps 代表每秒处理 10 个 API 请求，在 1 秒内的请求未得到处理，会延迟到下一秒，从 Chatopera 机器人平台处理上，每个请求在 100ms 内，具体请求响应时间，依赖于网络情况，以上数据为预估；qps, Query Per Second 的缩写

检查 CPU

Intel 处理器

支持目前常见 Intel, AMD 处理器，如：

- Intel(R) Core(TM) i5-6500 CPU @ 3.20GHz 及更高
- AMD Ryzen 7 2700X Eight-Core Processor 及更高

AVX 指令集

检查一台机器的CPU是否支持 AVX 指令集，执行命令：

```
grep flags /proc/cpuinfo |grep avx
```

如果控制台没有打印任何输出，代表该机器CPU不支持 AVX 指令集。该指令集在 2012 年以后的机器上，普遍支持，比较常见。

质保和升级服务说明

- 每次 Chatopera 机器人平台有新功能，推送更新内容，由企业客户自行决定是否升级，升级操作为 Chatopera 团队远程执行
- 运行过程出现网络延迟大等情况， Chatopera 团队远程联机解决
- 备份和恢复等咨询

采购下单

采购下单、商务合作和售后技术支持等，查看[联系方式](https://www.chatopera.com/mail.html) (<https://www.chatopera.com/mail.html>)。

Chatopera 对公银行账户信息：

Bank 银行	招商银行 支行：北京上地支行
Name 账户名	北京华夏春松科技有限公司
Acct. 账户	110934004410902
Tax ID 税号	91110108MA01CN41XA

交付准备

付款后，由甲方准备好机器，支持连接互联网。Chatopera 团队远程连接，执行部署，部署过程小于 1 天；如果需要现场执行，差旅费用由甲方额外支付；部署环境不支持互联网情况下，每套部署额外支付 RMB 6,000

法律条款

- [服务条款](#)
- [服务水平协议](#)

服务附加条款

- 不允许将 Chatopera 私有部署上线为 SaaS 服务

Chatopera 机器人平台的私有部署版本，不允许将 Chatopera 上线为 SaaS 服务，即将 Chatopera 机器人管理控制台 Dashboard 发布到互联网，供互联网用户注册使用和计费。

在 Chatopera 机器人平台私有部署服务上，机器人平台账户管理使用配置用户名和密码的方式，不支持使用邮箱注册。

- 私有部署不包括源码

企业客户因为安全、合规审计等要查看源码，Chatopera 提供浏览和查看代码的支持，在 Chatopera 的电脑上，仅供查看，不支持拷贝、录像等形式；查看代码需要额外支付费用，具体办法另行协商。

企业客户在私有部署后，对 Chatopera 服务的源码进行拟协议、反编译，甚至修改源码等，是不被允许的，Chatopera 有权利在此种情况下停止服务并不予退款，如果给 Chatopera 造成经济和商业损失，Chatopera 有权利进行法律维权。

免责声明

对于购买 Chatopera 私有部署版本的企业客户，需保证对本公司提供的材料合法、合规。如发生下列情形之一，本公司有权随时中断或终止向用户提供软件或服务而无需通知用户，并拒绝用户于现在和未来使用本公司所提供之全部或任何部分：（1）用户提供任何错误、不实、过时、不完整等具有误导性的资料，或者本公司有理由怀疑前述资料为错误、不实、过时、不完整等具误导性的；（2）用户违反本法律声明或相关使用协议中规定的使用规则。

法律适用与司法管辖

本法律声明的制定、执行和解释及争议的解决均应适用中华人民共和国法律。双方因使用本公司所产生的争议，协商不成的，任何一方可以向本公司所在地人民法院提起诉讼。

服务水平协议

可靠性保证

- 服务可靠性 99.1%，即每年（365 天）有不超过 3 天的累积服务中断时间
- 由第三方造成的服务间断，数据丢失并不由 Chatopera 承担责任，比如第三方机房意外断电、断网
- 由用户操作失误造成的异常，数据丢失并不由 Chatopera 承担责任
- 由自然不可抗拒因素造成的服务间断，数据丢失并不由 Chatopera 承担责任
- 因 Chatopera 运维事故造成的数据丢失、无法恢复，服务中断时间超过承诺，由 Chatopera 及用户协商赔偿用户损失，赔偿手段包括代金券或退款

工单提交及处理

【工单处理】发送描述至邮箱，info@chatopera.com，16 小时内 Chatopera 运营人员反馈。

【紧急联系方式】需要快速获得支持，应对服务宕机等，紧急联系方式为企业微信

专属顾问

有疑问？



需要帮助？

私有部署？

扫一扫 加我的企业微信/支持微信&企业微信

投诉

联系方式：见 <https://www.chatopera.com/mail.html>

微信或企业微信扫一扫，告知工作人员、服务等问题。



服务条款

"Chatopera 云服务"（以下简称"本服务"）是由北京华夏春松科技有限公司（以下简称本公司）在向用户提供的聊天机器人云服务。本服务主要提供定制化聊天机器人的工具、API 接口、PC 软件等。

您对本协议的接受即自愿接受全部条款的约束，包括接受本公司对任一服务条款随时所做的任何修改。本协议可由本公司随时更新，更新后的协议条款一旦公布即代替原来的协议条款，恕不再另行通知，用户可在本网站查阅最新版协议条款。在本公司修改本协议相关条款之后，如果用户不接受修改后的条款，请立即停止使用本公司提供的服务，用户继续使用本公司提供的服务将被视为已接受了修改后的协议。

除非您接受本协议所有条款，否则您无权注册、登录或使用本协议所涉相关服务。您的注册、登录、使用等行为将视为对本协议的接受，并同意接受本协议各项条款的约束。非云服务用户不受以下条款约束，具体内容以双方签订合作协议为准。

1、用户使用规则

- 1.1、用户须是具备完全民事权利能力和完全民事行为能力的自然人、法人或其他组织，并保证所提供的个人或单位身份资料信息真实、完整、有效。
- 1.2、用户须同意并接受本公司通过电子邮件，手机号向用户登记的联系方式发送相关商业信息。
- 1.3、用户不得为了任何非法目的而使用本公司上的服务。
- 1.4、用户须遵守所有与本公司服务有关的网络协议、规定和程序。
- 1.5、用户不得恶意占用网络带宽等本公司服务资源。
- 1.6、用户不得使用爬虫等手段收集本服务网站、API 信息。
- 1.7、用户不得在请求 API、脚本中注入 SQL 等破坏条件、代码。
- 1.8、不得逆向工程、反编译或试图以其他方式发现本公司提供的软件的源代码。
- 1.9、不得以任何方式登录或者尝试登录提供服务的主机。
- 1.10、不得以任何方式获取或者尝试获取提供服务的主机的管理权限。
- 1.11、不得利用账户进行任何可能对互联网的正常运转造成不利影响的行为；
- 1.12、不得利用本公司服务系统发布任何骚扰性的、中伤他人的、辱骂性的、恐吓性的、庸俗淫秽的或其他任何非法的信息；
- 1.13、不得利用本公司服务系统进行任何不利于本公司的行为；
- 1.14、就本公司及合作商业伙伴的服务、产品、业务咨询应采取相应机构提供的沟通渠道，不得在公众场合发布有关本公司及相关服务的负面宣传；
- 1.15、如发现任何非法使用用户账号或账号出现安全漏洞的情况，应立即通告本公司。

2、用户权利及义务

- 2.1、默认情况下，在服务协议终止后或者在用户主动删除数据或用户服务期满后，用户使用过程中产生的数据会立即删除。
- 2.2、本公司承诺用户能够控制数据的迁移，保证启用或弃用该服务时，数据能有偿迁入和迁出。
- 2.3、用户私自登录节点进行操作，导致的数据丢失，本公司不提供恢复服务。
- 2.4、本公司不会将用户数据、个人信息等资料泄露给任何第三方，除非政府监管部门监管审计需要。

- 2.5、本公司承诺用户在必要的条件下，由于合规或是安全取证调查等原因，可以提供相关的信息。本公司承诺遵守国家相应的法律法规，配合政府监管部门的监管审查。提供的相关信息包括关键组件运行日志、运维人员操作记录等。
- 2.6、为了便于运维，本公司保留查看用户操作记录的权利，登录主机进行运维操作的权利。

3、免责条款

- 3.1、用户实施违反包括但不限于本协议的行为，本公司有权拒绝向用户提供服务，终止本协议，用户已支付的“服务费”预付款不予退还，同时本公司有进一步追究用户法律责任的权利。
- 3.2、本公司通过内容部检测程序发现或经其他机构或用户举报而发现用户有发布违法信息、严重违背社会公德以及其他违反法律禁止性规定或本协议约定的行为时，本公司有权立即终止对用户提供服务并不退还任何款项。
- 3.3、不论在何种情况下，本公司均不对由于互联网正常的设备维护，互联网网络联接故障，电脑、通讯或其他系统的故障，电力故障，黑客攻击、计算机病毒侵入或发作、电信部门技术调整导致的影响、因政府管制而造成的暂时性关闭、由于第三方原因(包括但不限于不可抗力，例如火灾、水灾、雷击、地震、洪水、台风、龙卷风、火山爆发、瘟疫和传染病流行、罢工、战争或暴力行为或类似事件等，政府行为，司法行政机关的命令或第三方的不作为等不可抗力而造成的不能服务或延迟服务、用户数据丢失等承担责任，但本公司应及时通知用户，并及时采取措施防止损失扩大。
- 3.4、用户了解并同意，本公司不对因下述任一情况而导致用户的任何损害赔偿承担责任，包括但不限于利润、商誉、使用、数据等方面损失或其它无形损失的损害赔偿（无论本公司是否已被告知该等损害赔偿的可能性）：使用或未能使用本公司服务；第三方未经批准的使用用户的账户或更改用户的数据；用户对本公司服务的误解；任何非因本公司的原因而引起的与本公司服务有关的其它损失。
- 3.5、特别提示：在受限于本协议其他规定的前提下，为更好的提供服务，本公司将定期或不定期的对本公司平台进行功能及（或）页面上的改造及（或）升级。本公司会通过本公司平台或其他有效方式将相关事项及用户注意事项告知用户，经上述告知，本公司不对因用户未能遵守相关注意事项而造成的损失承担任何责任，本公司经改造及（或）升级后，对于用户因操作习惯性而引起的损失，本公司不承担任何责任。
- 3.6、受限于本协议其他规定的前提下，因本公司平台中接入的其他方的支付渠道与用户就支付关系发生任何争议的，与本公司无关，本公司不承担任何责任。
- 3.7、当出现来自或针对客户网站的互联网攻击行为或者接到政府相关部门的监管要求时，本公司将通知客户立即进行处理，对未按照要求及时处理的，本公司有权采取相应措施，以避免网络或内容安全事件的进一步扩大；当出现紧急网络或内容安全事件(例如内容涉及国家监管部门禁止范围，被国家监管部门如网监勒令封闭 IP 等事件)时，为保护广大用户的合法权益，本公司有权在事先不通知对方的情况下采取相应措施。本公司对上述紧急网络或内容安全事件的处理措施免于承担责任。
- 3.8、用户接入第三方服务，比如百度统计等，并不是本公司负责安全保护，如果造成用户财产损失，与本公司无关，本公司不承担任何责任。

2020 年 10 月 [北京华夏春松科技有限公司](#)

背景知识

Chatopera 机器人平台的定位是低代码上线智能对话机器人的工具和服务。本篇的主要目的是帮助 Chatopera 机器人平台用户从系统的角度理解 Chatopera 的多轮对话的工作机制，尤其是刚刚开始认识 Chatopera 服务的新用户，待阅读背景知识后，相信你可以更好的使用 Chatopera 机器人平台定制出满足各种需求的聊天机器人。

多轮对话的定义

Chatopera 的智能对话机器人解决方案主要面向企业在客户服务、营销和企业内部协作中使用自然语言交互完成信息查询、任务和自动化流程。目前，市场中比较常见的智能对话机器人解决方案主要是一问一答：设定问答对，使用信息检索形式，提供信息查询功能，在 Chatopera 看来，这不是智能对话机器人的体验。

在 Chatopera，我们认为多轮对话是在一定时间内，一定交互次数内，考虑对话的上下文，每次人发送聊天内容时，机器人具备从上下文情境中分析最合理的回答，准确的为人提供回复。

什么场景下使用多轮对话

Chatopera 的智能对话机器人解决方案是工具，用于设计、实现和发布聊天机器人。从价值提供上，Chatopera 所面向的是封闭域聊天，更关注智能对话机器人在生产、消费和制造等领域的聊天机器人应用；在开放域聊天，尤其是以闲聊为主的应用，不是 Chatopera 智能对话机器人的关注点，虽然我们提供了标准的通用的工具，但是作为开放域聊天，在目前还没有好的技术方案，强人工智能还没有成熟，对于实际应用上，还不能大规模的提供价值。

Chatopera 的多轮对话服务，可以面向不同行业，比如教育、电商、游戏和生产制造等。尤其是客服和企业内部协作中，将以前通过表单、人工等的服务使用自然语言对话的形式实现。应用场景比如智能客服、智能招聘面试、智能提交工单、智能提交请假、智能分析潜在客户意向等。

实现原理

Chatopera 将问答技术中的不同组成部分构建为基础模块，提供每个模块的管理工具、APIs，Chatopera 机器人平台用户可以单独使用。在基础模块之上，再融合为一个多轮对话方案。

基础模块包括：

- 深度学习: 实现语义理解、回答检索、聚类分析等
- 词典: 机器人能理解和捕捉的概念
- 知识库: 以搜索引擎技术为主
- 意图识别: 以机器学习为主，实现任务型对话
- 对话脚本: 以脚本规则为主，设定对话规则同时融合 FAQ 和意图识别

以对话脚本为中心融合知识库和意图识别，形成多轮对话，实现定制聊天机器人的标准，是 Chatopera 机器人平台的主要特色：

- 多轮对话的检索: Chatopera 多轮对话如何融合基础模块为一个标准技术
- 模糊匹配器
- 意图匹配器

其它资料

- 视频教程

深度学习

智能问答技术及开源项目

Chatopera 云服务使用深度神经网络。深度学习技术应用于以下任务：

- 近义词匹配, Chatopera 开放相关技术参考 [中文近义词: 聊天机器人, 智能问答工具包](#)
- 句子相似度, 词向量距离预训练模型
- 依存关系分析, Chatopera 开放相关技术参考 [text-dependency-parser](#)
- 词性标注和命名实体识别
- 意图识别, Chatopera 开放相关技术参考 [Clause 中文语义理解引擎](#)
- 聚类分析
- Chatopera 语音识别服务

出版物

2018 年底, Chatopera 会同行业内其他工作于智能问答与深度学习领域的专家、研究员, 与电子工业出版社合作, 出版《智能问答与深度学习》一书, 该书部分内容是 Chatopera 探索智能问答领域的分享。目前在京东等电商平台图书中有售。



快速查看本书链接: [《智能问答与深度学习》](#)

内容概述

对于需要了解 Chatopera 云服务实现原理的用户, 该书将会大有帮助, 本书是智能问答的知识介绍, 也可以作为入门、工具书。该书介绍的内容相对丰富, 目前在京东售出逾 7,000 册, 好评 98%。

智能问答与深度学习(博文视点出品)

深度学习人工智能参考书 介绍近年来自然语言处理和机器阅读的成果，配有翔实示例，助力实际应用，现
下载，业内大咖力荐 团购电话4006186622

王海良, 李卓桓, 林旭鸣 著

京东价 ￥34.50 [5折] [定价 ￥69.00] 降价通知

增值业务 助力环保, 传递知识, 旧书换新

排行榜 入选『通俗易懂的人工智能书TOP』 详情>>

配送至 北京朝阳区八里庄街道 可配货，预计下单后5-10天发货

京东物流 京准达 | 211限时达 | 京尊达

由 京东 发货，并提供售后服务。

服务支持 放心购 7天价保 | 上门换新 | 破损包退换

可配送全球49元免基础运费

京东服务 季度意外换新 ￥1.50

白条分期 不分期 ￥11.88 × 3期 ￥6.13 × 6期 ￥3.16 × 12期 ￥1.76 × 24期

1 + - 加入购物车 购买电子书 ￥15.99

开源代码

该书中介绍的示例程序的源码，已经已开源项目的形式发布：

<https://github.com/chatopera/book-of-qna-code>

开放语料

随着 Chatopera 云服务的不断优化，在研发过程中，形成了一些语料，Chatopera 也以开放数据的形式开放，这增加了我们的工作量，而得到的回报是不多的。这个工作的主要目的是回馈给客户、开源社区的同仁们，以及期待更多的研究者可以用这些数据做出技术上的突破、创新，Chatopera 竭诚与业内同仁们一起做好智能问答，早日实现聊天机器人的普及，让科技使我们的生活更美好。

- [保险行业中文语料库](#)
- [心理咨询问答语料库](#)

更多开源项目

- [DeepQA](#): Chatopera 于生成式智能问答的探索、实践
- [docsbot](#): 将使用 Markdown 的文档转化为问答服务

更多开源项目，敬请通过 Chatopera GitHub 主页了解：

<https://github.com/chatopera>

词典

概述

词典就是机器人在对话中所掌握的概念，机器人凭借词典区分不同实体，词典通过不同形式约定了一个词汇的集合。

什么是实体呢？以一个例子说明，比如订一份外卖需要知道用户想定什么外卖，什么时候定，定多少，以及口味要求和地址。那么此时需要的实体就包括：food（食物）、time（时间）、num（数量）、taste（口味）、address（地址）。

Chatopera 机器人平台提供多种形式的词典。

词典类型

词典分为：**系统词典** 和 **自定义词典** 两种类型。

系统词典是 Chatopera 机器人平台预置的词典，词典标识以 "@" 为前缀开头。自定义词典，是用户创建和维护的词典，包括 **词汇表词典** 和 **正则表达式词典** 两种类型。

知识库

在智能问答需求中，有一些问题可以提前设置好答案，不涉及上下文环境，是一问一答。对于这类问答，就可以使用知识库模块。

- 录入问题和答案
- 设计标准问、相似问和近义词
- 使用搜索和排序，根据相似度返回结果，同搜索引擎

产品特色

- 内置的检索模型基于大数据训练
- 完善的问答对管理和词典管理

实现原理

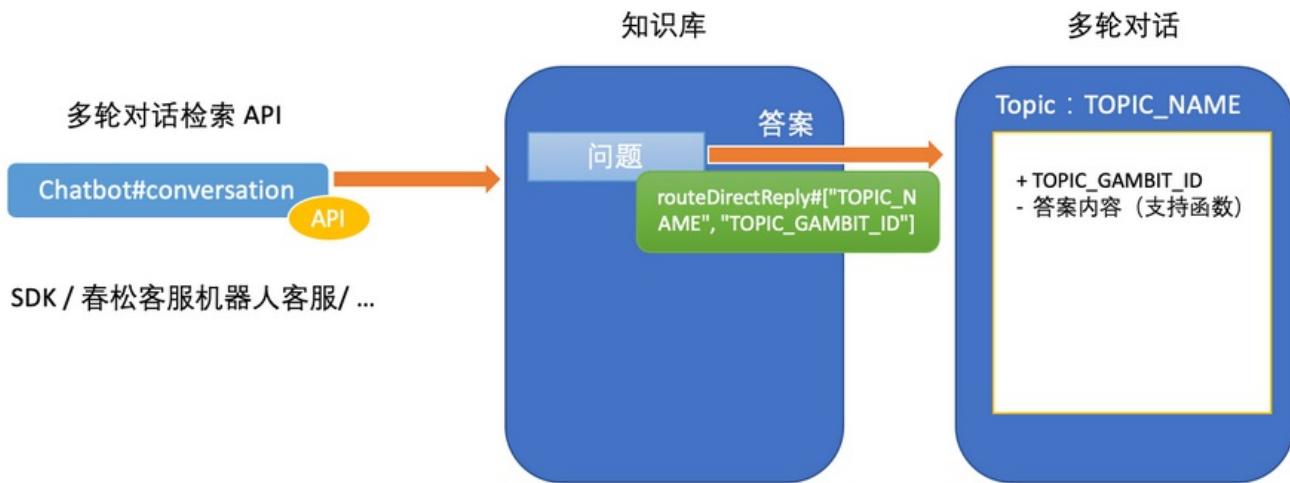
知识库通过标准问、相似问、自定义词典和答案维护对话内容，为 Chatopera 机器人平台用户提供标准的、快速的对话检索服务。对话用户的请求文本和问答对里的标准问或扩展问比较，计算相似度，当相似度高于阀值时，即认定该问答对的答案是对话用户需要的回复。



编辑知识库问答对

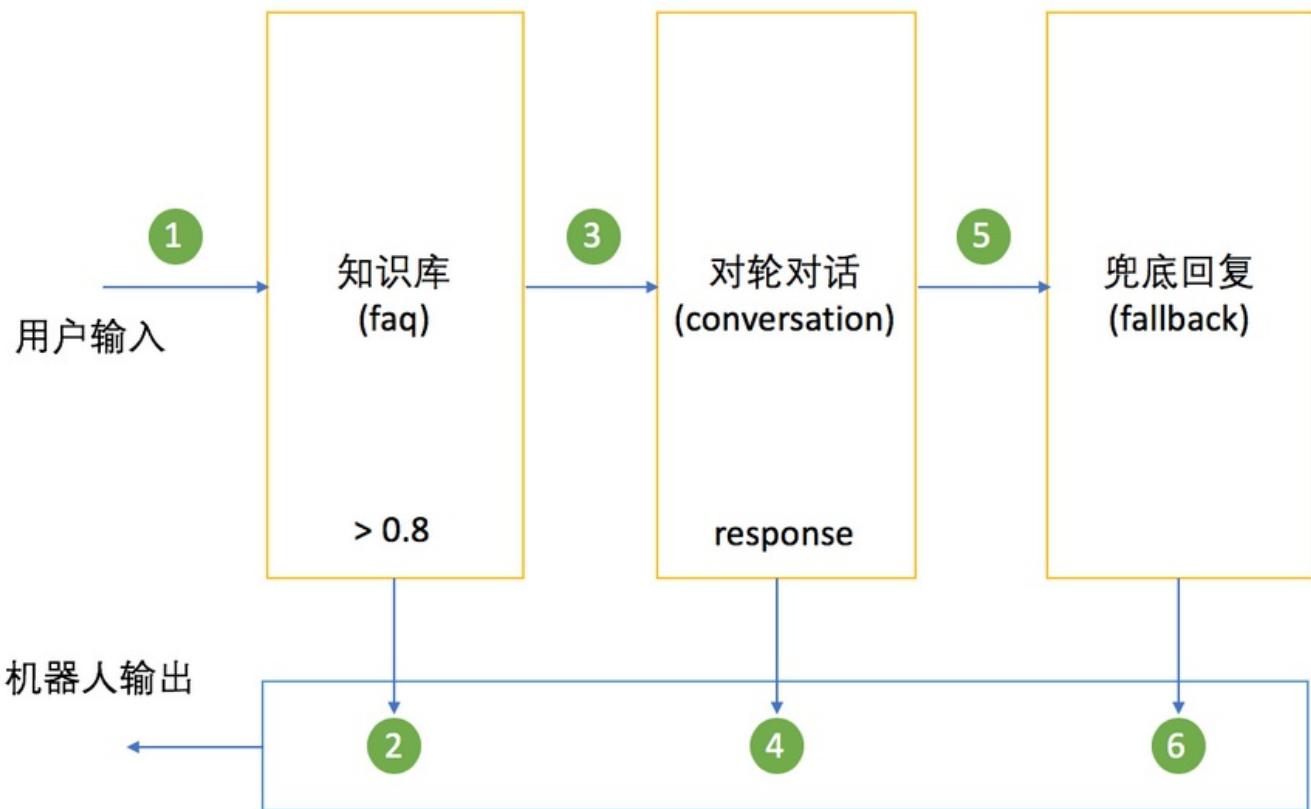
动态答案

将知识库答案路由到多轮对话的话题，然后从多轮对话获得答案。



目前，知识库路由（routeDirectReply答案）只在[多轮对话检索](#)上支持，多轮对话检索中，自动优先检索知识库。

Chatopera 多轮对话检索过程，优先级：知识库 > 脚本 > 兜底回复。



应用场景

尤其是在客服场景中，来访者有大量的问题是重复的一问一答的问题，答案相对来说固定，或者在一段时间内固定，这样最适合通过完善知识库提升客服工作效率。

提示：一问一答是指一个问题对应固定的答案。比如“世界上有几个大洲？”就属于这类问题；但是如果问“今天股市大盘走势如何？”，答案不是固定的，该类对话使用[意图识别模块](#)和[多轮对话脚本](#)解决。

使用过程

学习使用知识库的过程很简单，历史数据也不是必需的。通常先由业务人员整理一些常见问题，并在知识库管理页面添加问答对，在对话测试页面进行简单的验证就可以集成上线了。

进阶优化知识库包括：

- 1) 设置自定义词典，增强知识库检索时处理近义词的能力；
- 2) 为标准问增加扩展问；
- 3) 在对话历史记录页面查看沉寂问题和兜底回复，创建新的问答对；
- 4) 使用聚类分析服务，进行大规模的对话历史的自动机器人学习分析。

增强知识库的智能水平，知识库的优化，知识库的优化也是长期的过程，也是企业重要的很有价值的资产。

意图识别

在应用聊天机器人的场景中，有一大类是任务驱动型的问答，比如来访者说“我想购买车票”，那么接下来机器人围绕订票这个需求，询问出行时间、目的地等信息，并将收集到的信息返回给聊天机器人开发者，完成订单确认、支付和下单订票等。这就是一个典型的任务驱动的问答：以识别意图并根据意图收集相关信息为目的。类似的对话需求，在 Chatopera 云服务中，是通过意图识别模块支持。

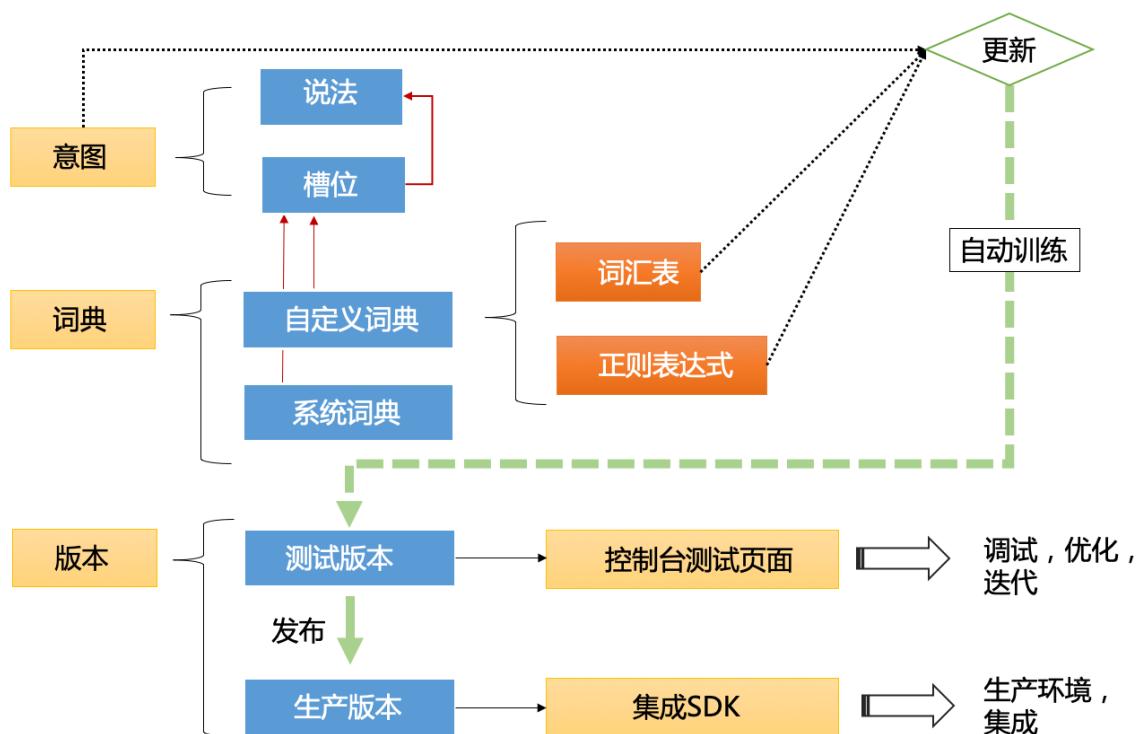
- 使用说法定义意图分类模型
- 利用序列标注识别槽位信息

产品特色

- 高性能算法
- 支持小规模数据量训练
- 在线标注和训练

实现原理

意图识别是语义理解的一个重要话题：与机器人对话时，是需要机器首先理解人的意图的；然后，根据这个意图，机器人继续与人进行问答；得到了这个意图的相关信息，机器才去执行这个意图代表的任务。



使用过程

意图：通过为每个意图添加说法和槽位，训练机器学习模型。对话用户的文本被分析为某一个意图，如果识别了意图，则计算槽位信息，并且对于没有识别到的槽位进行追问。

槽位：和一个意图相关的关键信息，比如时间、地点和专有名词等。槽位可以绑定到某个词典。

Chatopera 机器人平台提供自定义词典和系统词典：

- * 自定义词典包括词汇表词典和正则表达式词典；
- * 词汇表词典主要用于处理业务上的关键字段、同义词和专有名词；
- * 正则表达式词典则用于识别手机号、身份证号、订单号、邮箱等一些具有规则的信息；
- * 系统词典是用机器学习训练的命名实体提取判定的信息，比如人名、地名、组织机构和时间等。

The screenshot shows the Chatopera platform's intent recognition configuration screen. At the top, there is a search bar with placeholder text: "输入用户可能的说法, 用 () 使用槽位。例如: 我想预订(CityName)的机票". Below the search bar are two buttons: "添加" (Add) and "按Enter键可以快速添加" (Press Enter to quickly add). A table titled "用户说法" (User Sayings) lists a single entry: "我想打车". To the right of the table is a "操作" (Operation) column with a "删除" (Delete) button. Below the user sayings table is a section titled "槽位" (Slots). It contains a form for defining a slot: "槽位名称" (Slot Name) is set to "TIME", "词典" (Dictionary) is set to "names", and "必填" (Required) is checked. A dropdown menu next to "词典" is open, showing "names". Below this form is a text input field with placeholder text: "什么时间出发?". A "添加" (Add) button is located below the slot definition form. At the bottom of the interface, there is a table header with columns: "槽位名称", "词典", "必填", "追问", and "操作". The table body is currently empty, displaying the message "暂无数据" (No data available).

标注意图识别

意图匹配器

在 Chatopera 机器人平台，意图识别模块也是集成进入了多轮对话模块，参考使用[意图匹配器](#)。

对话脚本

- 以语法规则定义对话
- 融合知识库问答和意图识别
- 可执行自定义的 JavaScript 函数

产品特色

- 规范、灵活的脚本语法
- 语法支持编程语言，内置 NLP 函数
- 对话脚本可共享分发，Chatopera 提供了[对话模板](#)供快速定制对话服务

实现原理

对话脚本是通过书写 Chatopera 多轮对话脚本语法建立的规则实现问答。

- 对话脚本将对话分为**匹配器**，即和对话用户的输入进行比较的规则
- **回复**，匹配规则后的回复内容，支持文本和函数
- **上下轮钩子**，将规则与规则进行关联
- 对话脚本中的**函数**，使用 JavaScript 方式低代码编程，内置函数库，即方便了系统集成，同时也提供大量自然语言处理帮助函数，不需要掌握自然语言处理 NLP 的知识，开发者可以专注的实现业务逻辑

使用过程

匹配器是对话的基础，当用户向聊天机器人发送一条消息时，Chatopera 机器人平台会从所有定义的**匹配器**中找到匹配的规则。**匹配器**用半角字符加号+开始，对应的回复是紧邻的下行，用半角字符减号-开始。

例如，我们可以这样定义一个对话：

```
+ 晚饭吃什么  
- 烤鸭
```

注意：这里+和-和文字之间需要隔一个空格。

匹配器有多种，分别通过不同形式建立匹配规则，以适应不同场景的灵活使用。

- [通配符匹配器：使用语法建立规则](#)
- [模糊匹配器：容错能力更强和智能的匹配器](#)
- [意图匹配器：借助意图识别模块，轻松实现任务型对话](#)

在脚本匹配器语法和函数中，融合知识库和意图识别，所以，对话脚本是多轮对话的中心，围绕脚本实现不同问答技术的融合。所以，很多时候，文档和产品中，也用【多轮对话】表述对话脚本，就是这个原因。对话脚本独立构成了一个模块，在 Chatopera 机器人平台内部，实现了对话脚本引擎。

多轮对话

● 多轮对话通过 Chatopera 聊天机器人脚本语法实现对话话题。
● 多轮对话设计器是 Chatopera 聊天机器人的集成开发环境。

多轮对话已同步自定义词典信息 

[下载多轮对话设计器](#) [导入](#) [环境变量](#) [对话模板](#)

话题	状态	操作
greetings	 启用	 查看

管理多轮对话

模糊匹配器

通配符匹配器的不足

以前，在多轮对话中，使用通配符匹配器存在一个问题：匹配器规则的书写要尽量和输入的对话文本一致。使用通配符，用一些特殊符号，比如 `*n` 去定义一些目标词汇的集合，然而这依然在构建对话流程时显得不够智能，对于目标词汇的识别以及按照位置匹配，是通配符匹配器的优势。

举一个例子：

The screenshot shows a developer interface for a chatbot. On the left, there's a '对话' (Conversation) pane with a history of messages between a user ('Me') and a bot. A red box highlights a section of the conversation where the user says '你好' (Hello) and the bot responds with '你好！' (Hello!). Below this, another red box highlights the user saying '你好吗' (How are you?) and the bot responding with '我不明白您的意思。' (I don't understand your meaning.). At the bottom of the conversation pane, there are input fields labeled '请在这里输入进行问答测试' (Please enter here for question-and-answer testing) and '发送' (Send).

On the right, there's a '脚本' (Script) tab open in the editor. The script content is as follows:

```
1 /**
2 * 获得示例程序，快入开始！
3 * https://github.com/chatopera/chatbot-samples
4 */
5
6 +
7 + (你好|您好|你好啊)
8 - 你好！
9
10
11
12
13
14
15
16
17
18
19
20
```

A red arrow points to the '+' symbol in line 6 of the script, indicating the regex pattern being discussed.

在上面的脚本中，定义了一个通配符匹配器 "+ (你好|您好|你好啊)"，它支持三种问法，但是用户可能用更为多变的说法，为了支持这些说法，可以选择使用更宽松的通配符，但很多时候并不好把握：

- 1) 使用枚举的思路，当要支持所有相关意图的说法时，耗时费力；
- 2) 使用通用通配符，太过灵活，有的时候匹配到了期望以外的词语。

为了解决“容错”和表达能力不足的问题，需要提供更多形式的匹配器来解决。

接下来

- [在多轮对话中使用模糊匹配器](#)

意图匹配器

设计初衷

Chatopera 机器人平台 2019 年就发布了意图识别模块。意图识别模块使用机器学习技术，通过少量的数据标注，就可以匹配很多种用户说法，实现智能的任务型对话机器人，意图识别模块基于大规模的通用语料，内置多个预训练模型，以适应不同行业应用。在过去两年，Chatopera 团队一直在探索，将对话脚本和意图识别结合，促成更为强大的多轮对话定制能力。现在，一系列最佳实践通过意图匹配器的形式实现并发布了。

在没有意图匹配器以前，对话脚本，是按照一个个固定的流程进行的，通过上下轮钩子，比如用户说：我想打车，写好了开始的规则，并匹配上以后，按照脚本顺序问打车的数据：时间、出发地、目的地等。如果机器人问“您想从哪里出发？”，而对话用户发送了时间“今天下午 5 点”，那么在脚本中，处理起来费事，还需要增加学习成本。而针对于类似的场景，机器人需要能识别时间，然后继续追问出发地。智能的处理任务中的关键信息，是普遍存在的需求。而 Chatopera 意图识别对话，就是解决这个问题的。

The screenshot shows the Chatopera platform's 'Test Conversation' feature. On the left, a conversation log is displayed between a 'Robot' (represented by a blue car icon) and 'ME' (represented by a person icon). The conversation steps are:

- Robot: 请问您需要什么帮助
- ME: 我想打车从武汉出发
- Robot: 什么时间出发?
- ME: 到南京
- Robot: 什么时间出发?
- ME: 明天早上
- Robot: 会话已完成，请建立新的会话
- Robot: 请问您需要什么帮助

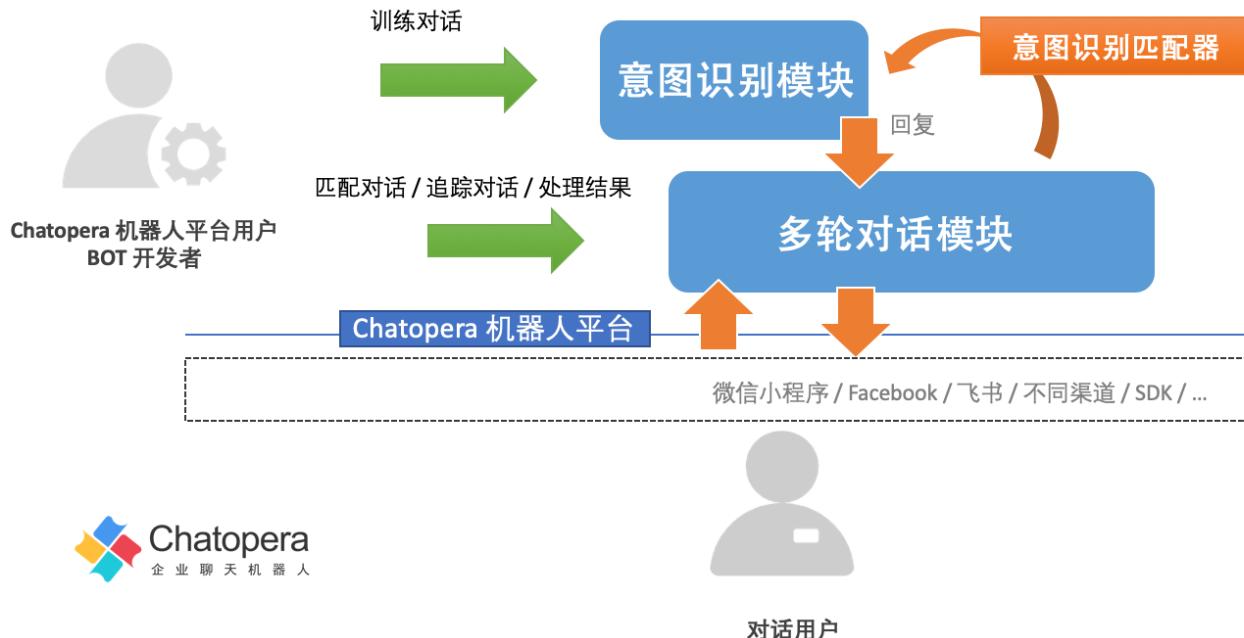
On the right, a 'Reply Information' panel is open, showing the details of the identified intent:

意图名称: book_cab

序号	槽位	值	是否必填	绑定词典
1	originLoc	武汉	是	@LOC
2	date	明天	是	@TIME
3	destLoc	南京	是	@LOC

意图识别模块，简单说就是识别并分析意图信息，根据意图需要的关键信息进行追问，直到获取到所有必须的关键信息或超过追问限制次数。

以前，在 Chatopera 机器人平台内，意图识别模块和多轮对话模块，是两个独立模块，Chatopera 提供了[开源项目](#)的形式提供最佳实践：在业务端，BOT 开发者用 SDK 灵活的使用不同模块。这样的形式，学习成本加大，怎么样通过低代码的方式上线有意图识别能力的机器人呢？现在，这个答案就是意图匹配器。在多轮对话脚本中，借助意图匹配器很好的融合了脚本、函数和意图识别。Chatopera 机器人平台用户，可以用更为简单和快捷的方式，上线更为强大的智能对话机器人。



对话脚本和意图识别二者的结合，至少获得了如下几点好处：

- 1) 会话周期在多轮对话中管理，以前意图识别需要 BOT 开发者自行维护会话周期，机器人管理控制台提供参数进行配置；
- 2) 增加对话灵活性，识别意图后，转入意图识别对话，但是对于一个意图中关键信息，没有获取到，则追问；对一个关键信息连续追问一定设置次数，则继续查询知识库和脚本，提升了处理的能力（这个继续查找行为也称为“穿透”）；
- 3) 意图识别得到的意图信息，在多轮对话脚本的函数可以读取，可处理意图识别成功或者意图识别槽位追问次数超过设定最大值，两种情况的信息；
- 4) 利用多轮对话函数中，内置的高级 NLP 函数，继续分析意图信息，比如使用 `extractTime` 将相对时间转化为绝对时间，也就是“今天”识别为具体的日期 2021-08-31；
- 5) 利用多轮对话函数中，`http` 工具类，和业务系统、互联网服务等集成。

以上的这些好处，让 Chatopera 机器人平台可以诞生更多的创新的对话机器人，帮助 BOT 开发者尽情的释放创造力！

欲了解更多 Chatopera 多轮对话工作机制，查看[详情链接](#)，您也可以阅读完本篇后再阅读多轮对话工作机制。

意图匹配器的功能

在对话机器人中，一个常见的应用场景是通过自然语言形式，自动化的完成流程，比如：“我想打车”、“我想看病挂号”和“我想点外卖”等。对话用户说了这些后，就是要明确的希望机器人帮助完成一个任务：查询信息、下单或管理智能家居等。这个任务和这个任务里的关键信息，比如时间、地点和专有名词，就是机器人需要捕捉的，机器人通过追问的形式交互。

在 Chatopera 机器人平台上，也使用如下的术语表达这些概念。

概念	术语
任务	意图
任务的说法	说法
关键信息	槽位
概念；词语	词典

这些对话内容的管理，就在意图识别模块中：开发者或业务人员，提供说法；在说法中可以写槽位；槽位对应的是不同类型的词典。

那么对话用户的哪些说法对应了意图呢？哪些词对应了槽位呢？在 Chatopera 意图识别模块中，提供维护意图、说法、槽位和词典的页面。

The screenshot shows the 'book_cab' intent configuration page. At the top, there is a back arrow and the intent name 'book_cab'. Below this, there is a section titled '用户说法' (User Sayings) with a count of 1. A text input field says '输入用户可能的说法, 用{}使用槽位, 例如: 我想预订{CityName}的机票'. Below the input field are two buttons: '添加' (Add) and '按Enter键可以快速添加' (Press Enter to quickly add). Under the heading '用户说法' (User Sayings), there are three examples listed in separate boxes: '我想打车', '我想打车, 从{originPlace}出发', and '我想打车到{destPlace}'.

词典建设，是意图识别中很重要的方面，在 Chatopera 机器人平台上，词典管理非常完善。词典包括系统词典，自定义词典。系统词典是 Chatopera 机器人平台上开箱即用的词典，包括人名、组织机构名、时间和地点的。自定义词典有词汇表词典和正则表达式词典，是 Chatopera 机器人平台用户自行维护的。

训练意图识别对话机器人的工作，不需要开发技能，无代码工作；不管是开发者，还是业务人员，只要熟悉上述的几个概念就可以训练。

那么，训练好了意图识别对话，那么要上线就还存在两个问题：

1) 该如何处理识别结果，比如“我想打车”，得到了对话用户的出发地、目的地和时间等信息，如何发布订单呢？

2) 如果对话要被集成到一个客服机器人里，这个客服机器人也可以回答常见问题，在 Chatopera 知识库里设定了，如何让不同模块的对话融合起来呢？

上述两个问题的解决方案，就是使用意图匹配器。

接下来

- [在多轮对话中使用意图匹配器](#)

多轮对话的检索

在 Chatopera 多轮对话中，理解检索顺序是熟悉多轮对话原理的关键知识点。首先是基础模块间的检索顺序，然后是对话脚本中的话题的检索顺序。

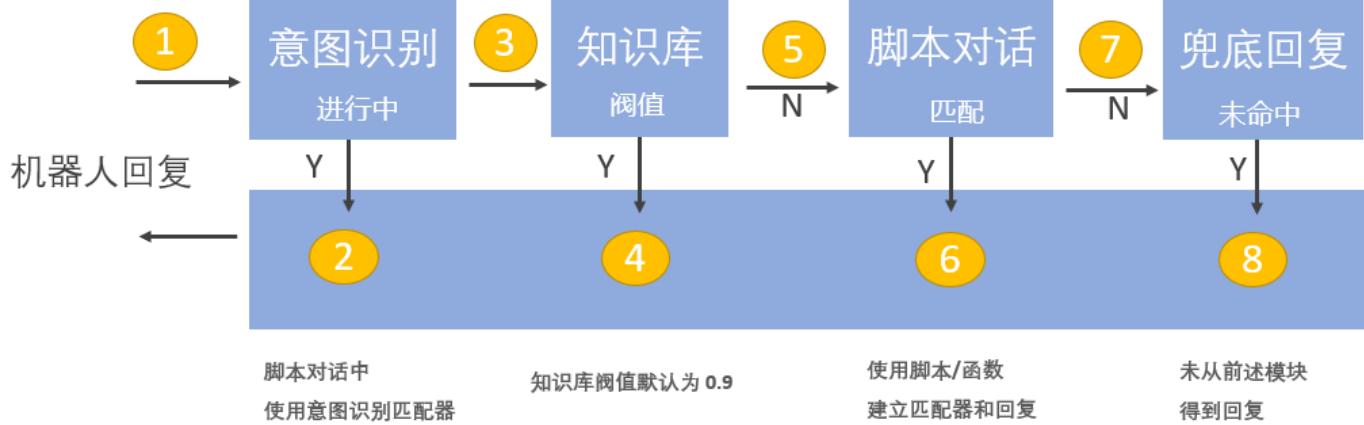
模块间检索

对话用户请求时，多轮对话会先检查是否有进行中的意图识别对话，然后是知识库检索，匹配知识库问答对，当有问答对高于知识库阀值时，机器人回复问答对中的答案内容；未匹配知识库，进入对话脚本，从话题中匹配，匹配上则回复内容；否则则回复兜底回复。从概念上有下图关系。

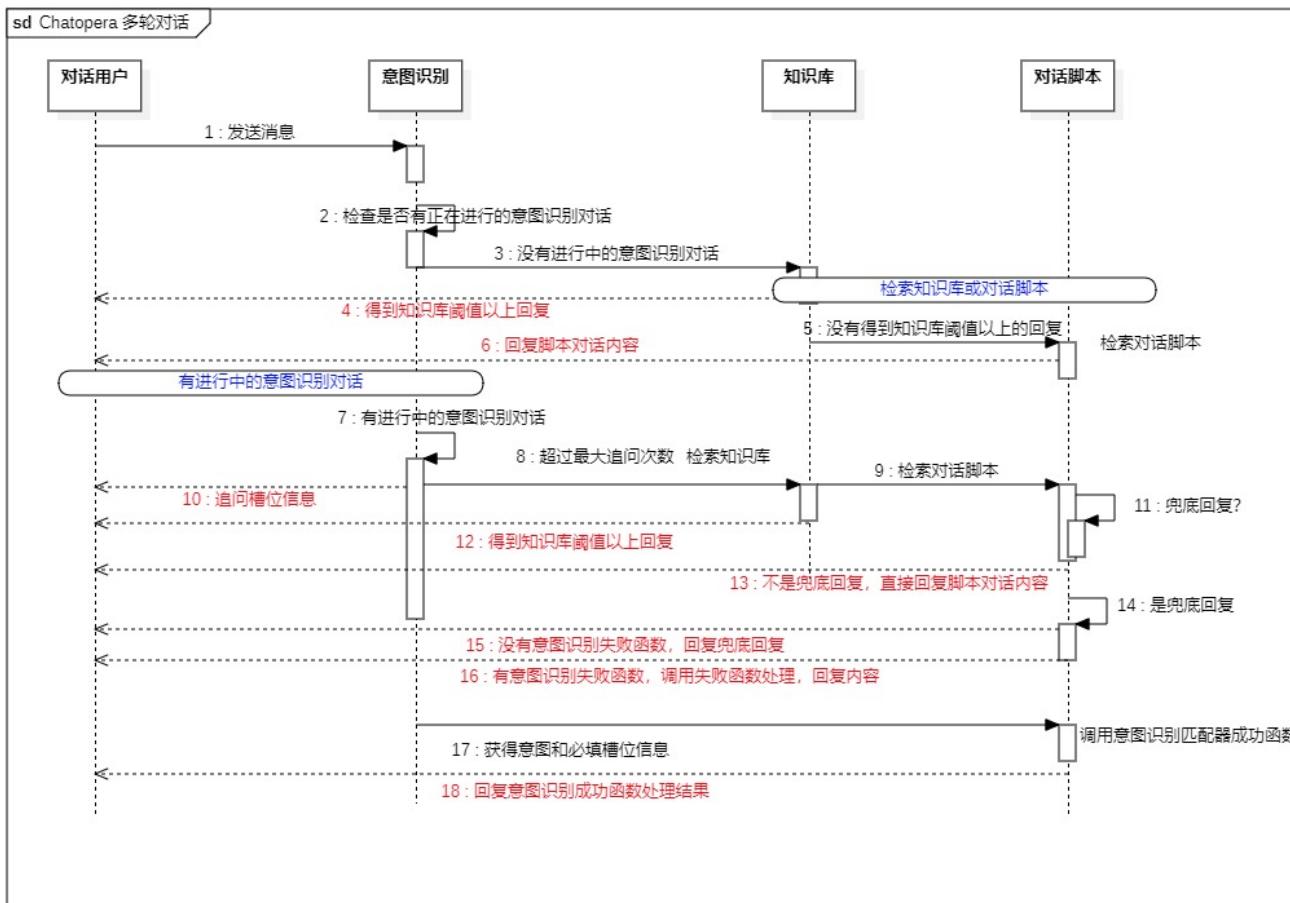
Y: 匹配 / N: 未匹配

对话用户

输入



之所以说是从概念上，是因为整个检索过程更为复杂，不方便理解，先用概念图从直观上快速理解 Chatopera 多轮对话的框架。一个更为详细的检索机制说明见下图。



详细说明图（查看大图） 虽然更为复杂，但和概念图含义基本一致，其中的要点是模块间有更多的状态检查和穿透行为（从一个模块进入另外一个模块）。在检索的过程中，涉及到一些参数，这些参数可以在 Chatopera 机器人平台对话机器人设置页面设定或者在 SDK 中传入参数。比如知识库阀值默认为 `0.9`，该阀值可以通过在请求中设定参数来调整，[介绍链接](#)。

关于知识库、对话脚本和意图识别的相互之间的调用关系，后文会有更多介绍。

创建对话脚本的话题

聊天机器人的多轮对话主要就是很多对话规则的组合，可以设想这些对话规则组成了一个个的话题，在对话时，可以聊一个话题，可以跳跃到其它的话题。那么，每次获得了聊天用户的文本，机器人进行答案的检索就是选择最合适的话题并找到匹配的规则，计算回复内容。

在创建好机器人后，下载多轮对话设计器，在多轮对话设计器中，创建话题。



话题里使用脚本实现对话逻辑，用一个简单的例子说明脚本语法非常容易掌握。

+ <noun>是个好地方
- 我很喜欢<cap>

+ \${0.5}{上一次去是什么时候}
% 我很喜欢<cap>
- 去年10月份

+ 开始的是匹配器，目前 Chatopera 多轮对话支持通配符匹配器和模糊匹配器； - 开始的是回复，目前支持文本和函数，函数是 JavaScript 脚本； % 开始的是上下轮钩子，用以关联规则。

使用多轮对话设计器撰写对话脚本的用户体验，经过了多年的打磨，已经非常成熟和稳定。



在多轮对话设计内调试对话，现在对于多轮对话设计器还不需要详细了解，本节内容旨在介绍原理，而话题作为重要的概念，需要先介绍如何创建以及在哪里编辑。

话题检索顺序

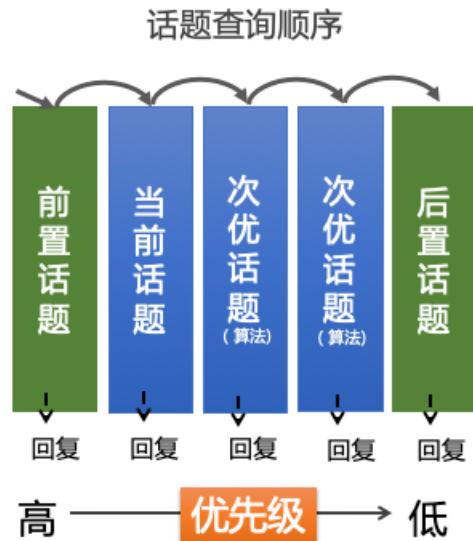
每个话题包含名字和一些规则，每个规则包括了匹配器和回复，业务上有明确的上下轮依赖的规则使用上下轮钩子进行关联。

话题检索也是有顺序的，匹配到了某一个话题的规则，后续的话题就被跳过。匹配从高优先级到低优先级进行，最高优先级是【前置话题】，话题名称为系统约定：`_pre_`；最低优先级是【后置话题】，话题名称为系统约定：`_post_`。优先级仅低于前置话题的话题是当前话题，就是上一次对话用户匹配到的规则所在的话题；其余的话题根据算法动态的排序。

话题检索栈

前置话题 : pre

后置话题 : post

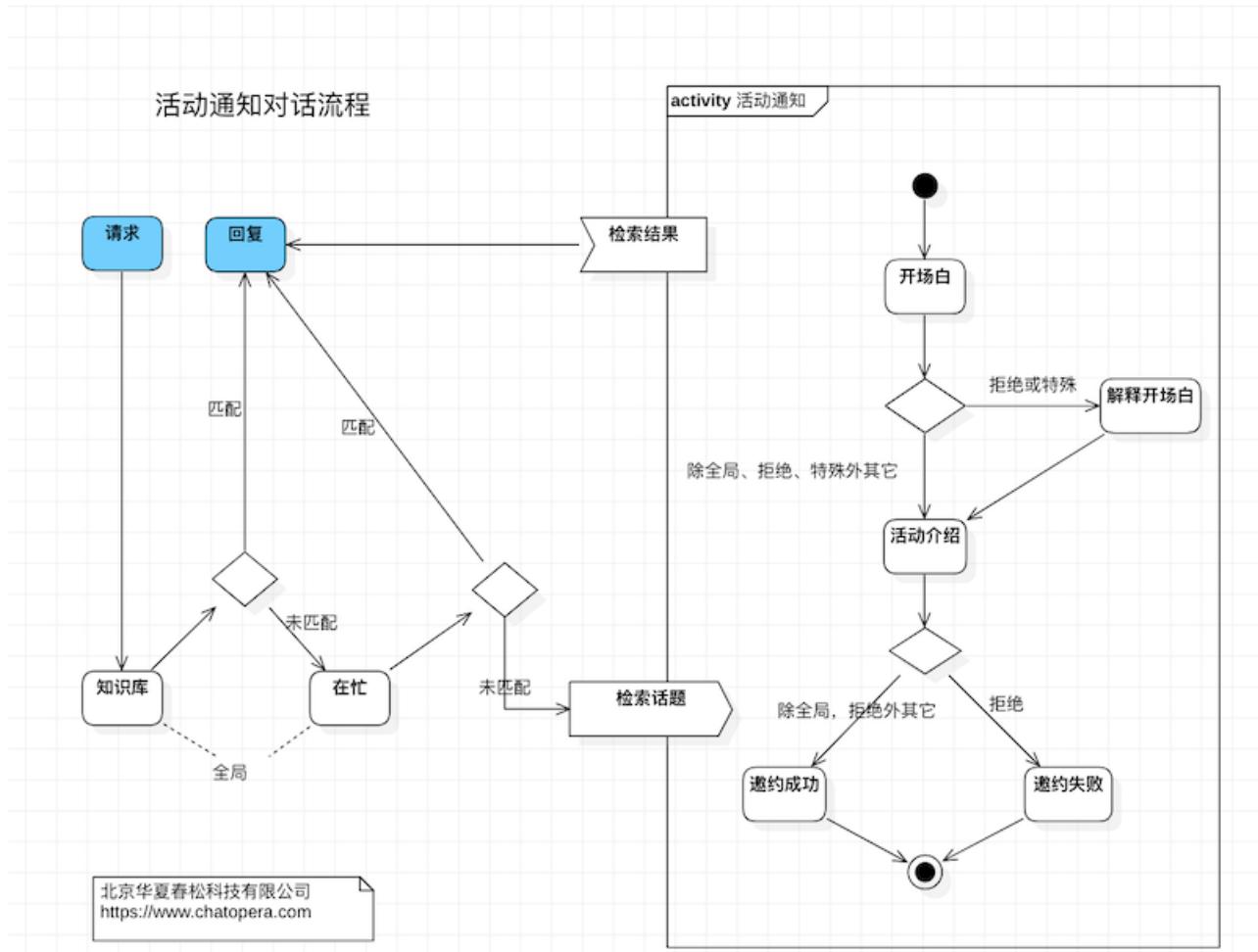


如果使用了上下文钩子，则最先匹配携带有上下文钩子（就是 % 上次回复内容）的规则。

整体上，对话脚本在检索时，检索栈是动态变化的。

对话状态机

现在，我们从另一个角度，状态机的角度思考多轮对话，因为是在多个连续的交互中，完成一个对话目标，那么就存在一个状态的问题，状态机是状态可以转移的图，两个状态之间的关系通过状态机约束。比如，某个活动通知的对话状态机如下：



这是个图示，仅为了说明原理。“请求”节点代表每次对话用户发送了文本，“回复”节点代表机器人处理结果，回复文本。因为知

多轮对话的检索

识库在检索中最先发生，可以放入一些一问一答的问答对，而一些全局的关键词放入前置话题【`_pre_`】中，图中右侧方框内，则是由其他话题组成的对话脚本，整个对话构成了状态机。

这个问题在对话脚本中，尤其需要注意：Chatopera 对话脚本引擎会考虑过去一段时间内，一定对话轮次的历史，机器人会回看这些记录来分析最合理的回复。这个时间长度和轮次的约束，在 Chatopera 机器人平台管理控制台内可以设定，是每个机器人的属性：`会话回溯最大时长` 和 `会话回溯最大轮次`。



当对话用户的输入匹配到对话脚本的规则时，即是对话到达了一个状态。

擦除状态

开发者在多轮对话脚本的函数中，也可以擦除这个状态，这样到下次对话用户再请求时，话题检索的栈回到初始状态，状态机回到原始，这个擦除的方法是在回复或函数中添加 `{CLEAR}` 前缀，关于这个知识点的使用说明见[文档](#)。

使用函数切换状态

如果想从一个话题，切换到另外一个话题获得回复。那么可以在函数中使用 `"topicRedirect(TOPIC_NAME, TOPIC_GAMBIT_ID)"`。将 `TOPIC_NAME` 替换为话题名字，`TOPIC_GAMBIT_ID` 替换为匹配器。就可以切换到该规则下获得回复。

`topicRedirect` 函数的更多介绍，[参考文档](#)。

知识库路由

对于状态机的状态跳转，Chatopera 多轮对话方案中，还有一个高级方法，通过知识库路由对话到对话状态机任意状态。

```
routeDirectReply#["TOPIC_NAME", "TOPIC_GAMBIT_ID"]
```

设置知识库问答对

设定知识库的问答对中的答案，内容使用上述格式，将 `TOPIC_NAME` 替换为话题名字，`TOPIC_GAMBIT_ID` 替换为匹配器。就可以切换到该规则下获得回复。

关于知识库路由的更多介绍，[参考文档](#)。

视频教程

本系列视频帮助企业在 Chatopera 机器人平台上开发聊天机器人，通过自然语言交互的形式，定制开发聊天机器人，提升业务流程自动化。Chatopera 机器人平台包括知识库、多轮对话、意图识别和语音识别等组件，标准化聊天机器人开发，支持企业 OA 智能问答、HR 智能问答、智能客服和网络营销等场景。

课程内容分为两大部分：

- 【初级入门】《Chatopera 机器人平台使用指南》
- 【高级深入】《Chatopera 机器人平台 Deep Dive》

内容大纲(部分)

1.1 自然语言交互的当下与未来

- 自动化、就业和生产力
- 聊天机器人的商业应用
- Chatopera 机器人平台概览
- 从入门到精通

1.2 聊天机器人应用场景及前提准备

- 需求、场景设定、系统组成
- 对话流程设计
- 建模工具介绍
- 设计“聊天机器人实现工具”的思考

1.3 实现对话应用：活动通知

- 知识库用途和管理
- 脚本实现话题
- 多轮对话的检索顺序
- 动手完成活动通知话题
- 更多对话模板及总结

1.4 深入理解聊天机器人实现和更多实用技巧

- 信息检索系统概述
- 话题脚本引擎
- 多轮对话：从知识库路由到脚本引擎
- Chatopera 文档中心的使用

1.5 系统集成

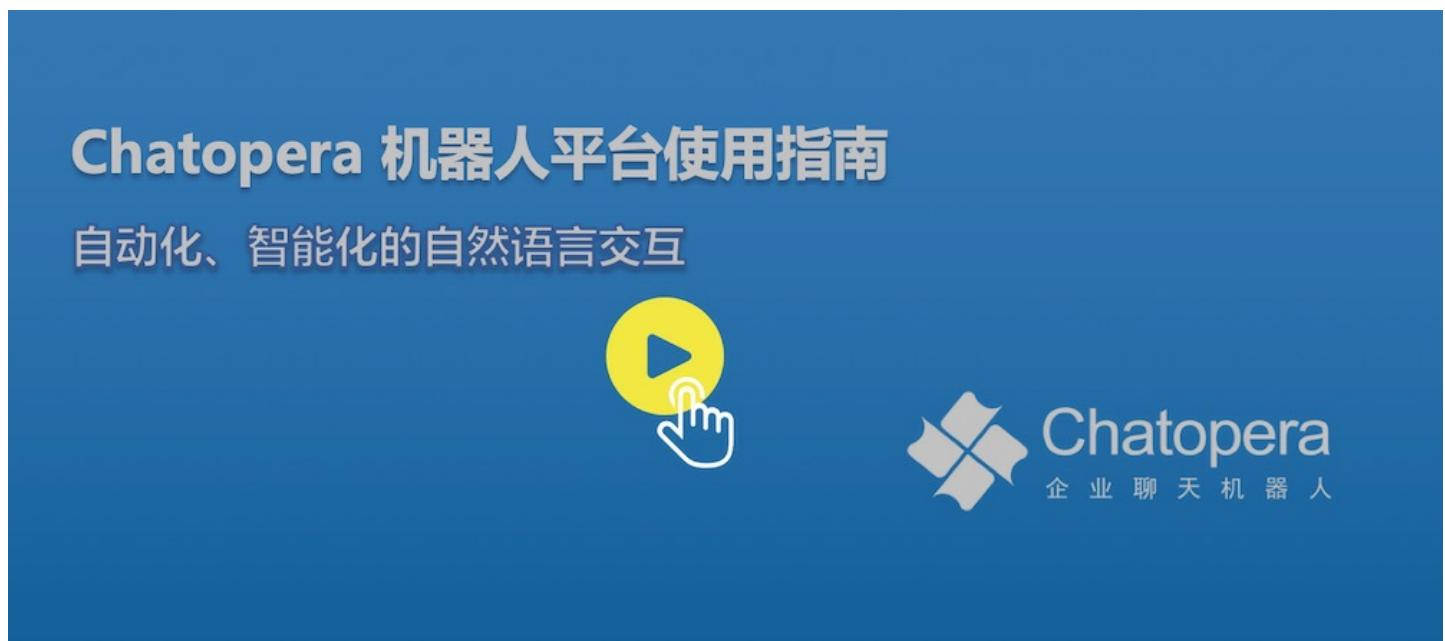
- SDK 语言及安装
- 实例化 Chatbot

- Command 接口介绍
- 对话检索: 知识库、多轮对话

1.6 聊天机器人上线后优化

- 分析投资回报率 (ROI)
- 优化沉寂问题和错误回复
- 总结
- 后续分享计划

课程地址: <https://www.bilibili.com/read/cv7526530>



附录

通过左侧导航菜单了解不同的附录内容。

词性标注

词性标注 (Part-of-Speech tagging 或 POS tagging), 又称词类标注或者简称标注, 是指为分词结果中的每个单词标注一个正确的词性的程序, 也即确定每个词是名词、动词、形容词或其他词性的过程。

使用位置

在 Chatopera 机器人平台内, 较多的使用词性标注, 为了方便 BOT 开发者实现更为强大的对话能力的机器人, 部分模块开放词性标注的结果, 供灵活的支持更多场景。

多轮对话内置函数

在 Chatopera 机器人平台中, 多轮对话模块, 使用[函数](#)处理请求和回复时, 支持在函数中, 使用 `this.message.words` 和 `this.message.tags` 访问经过自然语言处理的输入文本信息。

比如输入: 商场的开门时间。那么, 则有

```
this.message.words = ["商场", "的", "开门", "时间"];
this.message.tags = ["nis", "ude1", "vi", "n"]
```

针对, 不同的语言, tags 所使用的定义集合是不一样的。以下正是说明不同语言, Chatopera 机器人平台使用的词性标注集合。

中文

包括简体中文和繁体中文, 使用的词性标注集。在汉语中, 词性标注比较简单, 因为汉语词汇词性多变的情况比较少见, 大多词语只有一个词性, 或者出现频次最高的词性远远高于第二位的词性。

TAG	词性
a	形容词
ad	副形词
ag	形容词性语素
al	形容词性惯用语
an	名形词
b	区别词
begin	仅用于始##始
bg	区别语素
bl	区别词性惯用语
c	连词
cc	并列连词
d	副词

TAG	词性
dg	辄,俱,复之类的副词
dl	连语
e	叹词
end	仅用于终##终
f	方位词
g	学术词汇
gb	生物相关词汇; 或 Chatopera Built-in 词汇, 在 Chatopera 意图识别模块使用该标签作为自定义词汇
gbc	生物类别
gc	化学相关词汇
gg	地理地质相关词汇
gi	计算机相关词汇
gm	数学相关词汇
gp	物理相关词汇
h	前缀
i	成语
j	简称略语
k	后缀
l	习用语
m	数词
mg	数语素
Mg	甲乙丙丁之类的数词
mq	数量词
n	名词
nb	生物名
nba	动物名

TAG	词性
nbc	动物纲目
nbp	植物名
nf	食品, 比如“薯片”
ng	名词性语素
nh	医药疾病等健康相关名词
nhd	疾病
nhm	药品
ni	机构相关(不是独立机构名)
nic	下属机构
nis	机构后缀
nit	教育相关机构
nl	名词性惯用语
nm	物品名
nmc	化学品名
nn	工作相关名词
nnd	职业
nnt	职务职称
nr	人名
nr1	复姓
nr2	蒙古姓名
nrf	音译人名
nrj	日语人名
ns	地名
nsf	音译地名
nt	机构团体名

TAG	词性
ntc	公司名
ntcb	银行
ntcf	工厂
ntch	酒店宾馆
nth	医院
nto	政府机构
nts	中小学
ntu	大学
nx	字母专名
nz	其他专名
o	拟声词
p	介词
pba	介词“把”
pbei	介词“被”
q	量词
qg	量词语素
qt	时量词
qv	动量词
r	代词
rg	代词性语素
Rg	古汉语代词性语素
rr	人称代词
ry	疑问代词
rys	处所疑问代词
ryt	时间疑问代词

TAG	词性
ryv	谓词性疑问代词
rz	指示代词
rzs	处所指示代词
rzt	时间指示代词
rzv	谓词性指示代词
s	处所词
st	停用词, stopword 缩写
t	时间词
tg	时间词性语素
u	助词
ud	助词
ude1	的 底
ude2	地
ude3	得
udeng	等 等等 云云
udh	的话
ug	过
uguo	过
uj	助词
ul	连词
ule	了 哟
ulian	连 ("连小学生都会")
uls	来讲 來說 而言 说来
usuo	所
uv	连词

TAG	词性
uyy	一样 一般 似的 般
uz	着
uzhe	着
uzhi	之
v	动词
vd	副动词
vf	趋向动词
vg	动词性语素
vi	不及物动词（内动词）
vl	动词性惯用语
vn	名动词
vshi	动词“是”
vx	形式动词
vyou	动词“有”
w	标点符号
wb	百分号千分号, 全角: % %o 半角: %
wd	逗号, 全角: , 半角: ,
wf	分号, 全角: ; 半角: ;
wh	单位符号, 全角: ¥ \$ £ °C 半角: \$
wj	句号, 全角: 。
wky	右括号, 全角:))] } »] » 半角:)] { >
wkz	左括号, 全角: (([{ « 【 < 半角: ([{ <
wm	冒号, 全角: : 半角: :
wn	顿号, 全角: 、
wp	破折号, 全角: —— —— —— 半角: ——

TAG	词性
ws	省略号, 全角:
wt	叹号, 全角: !
ww	问号, 全角: ?
wyy	右引号, 全角: " "』
wyz	左引号, 全角: " "『
x	字符串
xu	网址 URL
xx	非语素字
y	语气词(delete yg)
yg	语气语素
z	状态词
zg	状态词

English / 英语

TAG	DESCRIPTION	EXAMPLES
CC	Coord Conjuncn	and,but,or
CD	Cardinal number	one,two
DT	Determiner	the,some
EX	Existential there	there
FW	Foreign Word	mon dieu
IN	Preposition	of,in,by
JJ	Adjective	big
JJR	Adj., comparative	bigger
JJS	Adj., superlative	biggest
LS	List item marker	1,One
MD	Modal	can,should

TAG	DESCRIPTION	EXAMPLES
NN	Noun, sing. or mass	dog
NNP	Proper noun, sing.	Edinburgh
NNPS	Proper noun, plural	Smiths
NNS	Noun, plural	dogs
POS	Possessive ending	's
PDT	Predeterminer	all, both
PRP\$	Possessive pronoun	my,one's
PRP	Personal pronoun	I,you,she
RB	Adverb	quickly
RBR	Adverb, comparative	faster
RBS	Adverb, superlative	fastest
RP	Particle	up,off
SYM	Symbol	+,%,&
TO	'to'	to
UH	Interjection	oh, oops
VB	verb, base form	eat
VBD	verb, past tense	ate
VBG	verb, gerund	eating
VBN	verb, past part	eaten
VBP	Verb, present	eat
VBZ	Verb, present	eats
WDT	Wh-determiner	which,that
WP	Wh pronoun	who,what
WP\$	Possessive-Wh	whose
WRB	Wh-adverb	how,where

TAG	DESCRIPTION	EXAMPLES
,	Comma	,
.	Sent-final punct	. ! ?
:	Mid-sent punct.	: ; Ñ
\$	Dollar sign	\$
#	Pound sign	#
"	quote	"
(Left paren	(
)	Right paren)

日语 / Japanese

TAG	説明
N	名詞 # Noun
PRO	代名詞 # Pronoun
DT	連体詞 # Adjectival determiner
V	動詞 # Verb
ADJ	形容詞 # Adjective
ADJV	形状詞 # Adjectival verb
ADV	副詞 # Adverb
PRT	助詞 # Particle
AUXV	助動詞 # Auxiliary verb
PUNCT	補助記号 # Punctuation
SYM	記号 # Symbol
SUF	接尾辞 # Suffix
PRE	接頭辞 # Prefix
TAIL	語尾 # Word tail (conjugation)
CC	接続詞 # Conjunction

TAG	説明
PRP	代名詞 # Pronoun
URL	URL # URL
ENG	英単語 # English word
FIL	言いよどみ # Filler
MSP	web 誤脱 # Misspelling
INT	感動詞 # Interjection
KYTEA_JA_DEFAULT_TAG	新規未知語 # Unclassified unknown word