

2024 年 7 月 26 日

## 目录

<b>1</b>	<b>Text_To_Speech 函数</b>	<b>3</b>
1.1	函数概述 . . . . .	3
1.2	参数说明 . . . . .	3
1.3	功能描述 . . . . .	4
1.4	注意事项 . . . . .	4
1.5	执行过程的没理解的地方 . . . . .	5
<b>2</b>	<b>listenuser</b>	<b>5</b>
2.1	函数描述 . . . . .	5
2.2	参数说明 . . . . .	6
2.3	不确定的地方 . . . . .	6
<b>3</b>	<b>Pardon</b>	<b>6</b>
3.1	参数说明 . . . . .	6
3.2	功能描述 . . . . .	6
3.3	注意事项 . . . . .	7
3.4	<u>不理解的地方</u> . . . . .	7
<b>4</b>	<b>QAClass</b>	<b>7</b>
4.1	构造函数 . . . . .	7
4.2	QAClassInit . . . . .	7
4.3	answer_question . . . . .	8
4.4	<u>我不理解的地方</u> . . . . .	8

<b>5</b>	<b>Interruption Class</b>	<b>8</b>
5.1	continue_rotate . . . . .	8
5.2	do_action_rely_instruction . . . . .	9
5.3	handle_interrupt . . . . .	11

# 1 Text\_To\_Speech 函数

## 1.1 函数概述

`text2speech` 函数用于将文本转换为语音，并播放或保存为 WAV 文件。该函数支持中断检测、异步播放和音量调整等功能。

## 1.2 参数说明

- `text=''`: 待转换的文本，默认为空字符串。
- `index=0`:  
Index=1000 表示同步播放，先文本转语音，然后在运行到播放语音的那一行代码时，会阻塞在那一行代码，直到语音全部播完，才会继续运行后面的代码  
Index=0 表示异步播放，先文本转语音，然后在运行到播放语音的那一行代码时，会新开一个进程用来播音，而主程序可以继续推进，去对下一句话进行文本转语音，因此上一句话的播音和下一句话的文本转语音是同时进行的，节省时间  
值得注意的是，上一句话播音的进程会被记录到 `STATUS.LastPlayProcessor` 变量，一旦检测到打断，就会杀死上一句话播音的进程，防止用户打断之后，机器人还一直播音不停下
- `is_beep=False`:  
`is_beep` 表示现在要播音的这句话是不是过渡句过渡句例如：“我在”、“大家有什么问题吗”在播音的同时，会把正在播音的这句话记录到 `STATUS.LAST_BROAD_WORDS` 变量，日志中会记录这句话，以及用户听到这句话之后，用户说的下一句话，用途是测试语音识别的准确性  
例如日志中可能会记录：  
时间：yyyy.mm.dd, hh:mm:ss  
上一句播音的话：玉兔号是我国的月球探测车语音识别结果：玉兔号是什么时候登上月球的？由于过渡句中不包含信息，记录日志时不能仅仅记录过渡句，还应该把上上句播音的话也记录下来例如，当最后一句播音的话是过渡句，日志中可能会记录：  
时间：yyyy.mm.dd, hh:mm:ss

上一句播音的话：这就是我对航天机械臂的介绍。

大家有什么问题吗

语音识别结果：航天机械臂有多长？

此处“这就是我对航天机械臂的介绍。”就是上上句播音的话，由于最后一句播音的话是过渡句，所以上上句播音的话也会被记录在日志里

- **wavfile=None**: 指定 WAV 文件路径，若提供，则直接播放该文件而不进行文本到语音的转换。
- **ignore\_interrupt=False**: 是否忽略中断，默认为 False。若为 True，则即使检测到中断也继续播放。

### 1.3 功能描述

1. 检查音频输出设备是否存在，若不存在则打印提示信息并返回。
2. 若未指定 **wavfile**，则调用外部脚本进行文本到语音的转换，并保存为 WAV 文件。
3. 检查是否有上一次的播放进程，若存在且未完成，则根据 **ignore\_interrupt** 参数决定是否中断上一次的播放。
4. 调整生成的 WAV 文件的音量，并保存为新的文件。
5. 根据音频卡的 ID，选择合适的播放方式进行播放。
6. 根据 **index** 参数，决定是进行同步播放还是异步播放。

### 1.4 注意事项

- 该函数依赖外部脚本和 **ffmpeg** 工具进行文本到语音的转换和音量调整。
- 要用进程自带的 **kill** 方法而非直接 **kill** 命令来终止正在进行的播放，防止播音卡死。
- 异步播放时，需要留出短暂的时间间隔，以避免文件读写冲突。

## 1.5 执行过程的没理解的地方

- ```
ttsproc=subprocess.Popen(["python3", "/home/kuavo/catkin_dt/src/voice_pkg/scripts/kedaxunfei_tts/test_host_3090_tts.py",  
    ↪ text, savepath])
```

这个文件运行起来什么功能?

答:tts 是 text\_to\_speech 的缩写, 不必深究内容。

- ttsproc.poll() 有什么功能 (已解决)?  
答: 在 Python 中, poll() 方法是 subprocess.Popen 对象的一个方法, 用于检查子进程是否已经结束。如果子进程已经结束, poll() 会返回子进程的退出码, 这通常是一个整数值; 如果子进程尚未结束, poll() 会返回 None。

- playproc, ttsproc, STATUS.Last\_Play\_Processor 都是干什么的, 什么时候要注意'在连续播放多次时, 需要注意资源的释放和进程的管理, 避免服务卡死。'

答: 假设 text2speech 函数正要播放一句话 text: ttsproc 是用来把 text 文本转为语音 (即合成音频文件) 的进程 playproc 是用来把音频文件播放出来的进程

STATUS.Last\_Play\_Processor 是上一句话播音的进程为什么要记录这些进程? 因为一旦识别到用户的打断信号, 此时正在进行的一切文本转语音和播音进程都应该被杀死。只有把这些进程全部都记录下来, 在识别到用户的打断信号时, 才能把它们全都杀死

## 2 listenuser

### 2.1 函数描述

listenuser 函数用于根据用户的声音输入或键盘输入来获取用户的指令或信息。该函数支持通过声音输入和文本输入两种方式。当系统检测到声音输入设备时, 会通过声音识别接口获取用户的指令; 若未检测到声音输入设备, 则会提示用户通过键盘输入信息。

## 2.2 参数说明

- **text** (默认值: 'ding'): 预设的文本输入, 用于声音识别接口的默认输入。
- **iter** (默认值: 1): 指定声音识别过程中的迭代次数, 用于控制识别的精度或尝试次数。

## 2.3 不确定的地方

- **iat\_web\_api** 是做什么的, 参数什么意思?  
这个 iat 是用于语音识别的, 就是语音转文本 (Intelligent Audio Transcription)
- **ding** 有什么特殊的含义吗?  
ding 是提示音, 提醒用户录音开始了。

# 3 Pardon

**pardon** 函数的目的是反复录制用户语音, 直到获取到非空的语音输入为止, 最多重复指定的轮数 **pardon\_round**。该函数在录制过程中提供了用户交互的反馈, 并处理了用户可能的打断行为。

## 3.1 参数说明

- **pardon\_round** (默认值: 1): 指定在放弃之前尝试录制用户语音的最大轮数。

## 3.2 功能描述

1. 函数开始时, 会打印一条消息提示开始录制用户语音。
2. 通过 **listenuser** 函数反复录制用户语音, 直到录制到非空的语音输入或达到最大重试次数。
3. 如果录制到的语音为特定标记 ('####'), 或者是第一次提示用户提问, 则通过 **text2speech** 函数反馈给用户, 提示他们现在可以提问, 并记录提示次数。

4. 如果用户在 6 秒内没有说话，则会再次使用 `text2speech` 函数提示用户可以提问。
5. 在用户提问过程中，如果检测到打断行为，会暂停 QA (Question Answering) 状态，并通过 `text2speech` 函数处理打断后的用户交互。
6. 最后，无论是否录到有效语音，都会重置 QA 状态和打断状态，准备下一次录音或交互。

### 3.3 注意事项

- 该函数依赖于全局状态对象 `STATUS` 来管理 QA 状态和打断状态。
- 使用了 `listenuser` 和 `text2speech` 两个外部函数来分别处理语音输入和输出。
- 特殊标记 '####' 用于特定逻辑处理，例如用户超时未回应时的处理。

### 3.4 不理解的地方

- 用户超过 6 秒没有说话就询问一遍在代码里哪里体现？  
`listener` 那个函数的接口里有录音 10 秒左右，也就是等待时间了。
- #### 是代填充的唤醒词吗？  
可以理解为唤醒词的标志
- `QAING` 这个状态有什么作用？  
`qaing` 的意思是正在 `qa` (问答)，如果 `set` 为 `True`，那么将不会处理打断的信号。可以理解为一个 `linux` 里面的信号锁
- 最后一个 `else` 是处理什么情况的？  
就是没录到东西并且 `repeat count` 不是 0 的时候才会进入第三个分支，等待被打断，机器人说完提示词 10 秒内不回答，机器人就会卡在这等待人再用唤醒词 (打断词) 唤醒它

## 4 QAClass

`QAClass` 是一个负责处理问答流程的类，它通过多线程初始化问答模型，并处理用户的提问，返回相应的答案。

## 4.1 构造函数

构造函数初始化了一个线程来启动问答模型的初始化过程，并设置了一些基本属性，如中断流的标志和任务类型标签列表。

## 4.2 QAClassInit

`QAClassInit` 方法用于初始化问答模型。它通过调用 `get_llm_answer` 函数获取模型实例，并将其存储在类的属性中。

## 4.3 answer\_question

`answer_question` 方法是类的核心功能，负责处理用户的提问并返回答案。它首先等待问答模型初始化完成，然后根据用户的提问和其他相关信息，通过问答模型生成答案。此方法还处理了流式返回的答案，并在适当的时候通过语音合成技术将答案转换为语音输出。

## 4.4 我不理解的地方

- `STATUS` 和 `get_llm_answer` 函数的具体实现和作用。(还包括 `llmclass` 的 `if_document_searched`, `process_query` 这些)
- `self.llmclass.process_query` 这个函数的返回结果是什么格式，有什么样例吗？流式和非流式的方式中，回答都是一次产生的，但是流式返回是按半句话来读，还能用非阻塞的方式节省时间，非流式是一次读完全部的回答是这个意思吗？
- 816 行，`text2speech("?")`，这个问号怎么也能转语音  
是为了保证打断，因为我们的打断是上一句。前面都是异步播放，本句后面没有下一句的话，播放过程不会被打断因此得手动加一句话。
- `interrupt_stream` 和 `STATUS` 里的 `interrupt` 什么区别，为什么要加上这个？
  - `STATUS.is_interrupted`  
`STATUS.is_interrupted` 就是指用户有没有用唤醒词打断机器人，这个变量在整个流程中都非常重要。



– `self.interrupt_stream`

我们假设用户提出了问题，大模型正在流式生成回答并且播音。

1. 如果此时用户打断了机器人，那么就会立刻切断对大模型流式生成回答的播音，然后播放提示语“我在”或“大家有什么问题吗”。
2. 如果用户从始至终都没有打断，那么就会把大模型生成的回答全部播放完毕。

由于我们让大模型在回答问题时，最后几句话要引导用户继续下达指令，所以效果是这样的：比如用户提出问题，大模型的回答可能是：“..... 以上就是天宫二号的发射过程，您还有其他关于空间实验室的问题吗？”或者“..... 长征二号的故事我讲完了，如果您还对其他火箭感兴趣，可以让我带您参观运载火箭展区”，如此一来，就不需要再播放“我在”或“大家有什么问题吗”这类提示语。

因此 `self.interrupt_stream` 的含义是大模型生成的回答是否被完整播放完。

而 `self.interrupt_stream` 的作用是控制提示语的播放：如果大模型生成的回答没有被播放完，而是被打断了，那么 `self.interrupt_stream` 为 `True`，需要播放提示语；如果大模型生成的回答被完整播完了，没有被打断，那么大模型会自己生成一些引导用户提问的话，其作用相当于提示语，此时 `self.interrupt_stream` 为 `False`，不需要播放我们写死的提示语。

- 853 行 `if self.interrupt_stream:`

```
sleep(0.05)
```

```
STATUS.set_is_Interrupted(False)
```

依旧没看明白，为什么等了 0.05 秒就给这个状态消除了，受到打断产生 `interrupt_stream` 这个信号为什么只需要维持 0.05 秒？

## 5 Interruption Class

### 5.1 continue\_rotate

该函数用于控制机器人进行旋转以寻找人脸或手持麦克风的目標。函数接受一个参数 'rotate'，用于指定旋转方向。

- 参数:
  - **rotate** (str): 指定旋转方向，默认为'left'。可以是'left'、'right'或'middle'。
- 功能:
  - 如果 **rotate** 为'middle'，则打印” 此处不需要转向寻找人脸。”，并且不进行旋转。
  - 否则，打印” 转向寻找人脸 ing...” ，并开始旋转。
- 实现步骤:
  1. 设置发布频率为 30Hz。
  2. 创建一个 **Twist** 消息实例 **move\_cmd**。
  3. 初始化 **move\_cmd** 的角速度 **angular.z** 为 0.0，并发布该消息。
  4. 休眠一段时间以确保消息发布。
  5. 根据 **rotate** 参数设置旋转速度。如果 **rotate** 为'left'，速度为 0.5；否则速度为 -0.5。
  6. 将大人脸区域设置为 **None**。
  7. 进入一个循环，持续发布旋转命令，直到检测到手持麦克风并且手持麦克风标志被设置。
  8. 打印” 转向找到人脸”。
- 异常处理:
  - 在 **finally** 块中，停止旋转，将 **move\_cmd** 的角速度 **angular.z** 设置为 0.0，并发布该消息。重置状态
- 一些不理解的地方:

- Twist 的 z 的角速度是往哪里转向的?(左右还是抬头低头)STATUS.HANDHELD\_DETECT 和 STATUS.HANDHELD\_DETECT\_FLAG, 这两个变量是谁设置的, 什么含义?
- STATUS.set\_Big\_Face\_Area(None) 什么叫 Big\_Face\_Area  
答:Big\_Face\_Area:str = "LEFT", # robot 检测到的人脸区域
- 手持麦克风检测是怎么回事? 哪里有手持麦克风? 是检测到手持麦克风就是转到正对观众的方向了吗?

## 5.2 do\_action\_rely\_instruction

该函数用于根据输入的问题执行相应的动作指令。函数接受一个参数 question, 用于指定动作指令的描述。

- 参数:
  - question (str): 动作指令的描述, 默认为空字符串。
- 功能:
  - 根据输入的问题, 确定动作类型和对应的动作索引。
  - 如果无法确定动作类型, 打印错误信息。
  - 如果确定了动作类型, 并且低计算机存在, 发布动作指令并等待动作完成。
  - 如果低计算机不存在, 仅打印动作信息。
- 实现步骤:
  1. 调用 get\_action\_type 函数获取动作类型。
  2. 初始化 action\_index 为 None。
  3. 遍历动作类型和对应的索引 (前进、后退、左转、右转分别对应 101、102、103、104), 如果动作类型匹配, 设置 action\_type 和 action\_index。
  4. 如果 action\_index 仍为 None, 打印错误信息。
  5. 否则, 检查 STATUS.LOW\_COMPUTER\_EXIST 是否为 True。

- 如果为 `True`，打印执行动作信息，发布动作指令，并等待 `STATUS.NAVI_END_FLAG` 变为 'success'。
- 如果为 `False`，仅打印执行动作信息。

- 一些不懂的地方:

- 428 行:

```
while True:
    if STATUS.NAVI_END_FLAG == 'success':
        STATUS.set_NAVI_END_FLAG(None)
        break
```

这里的 `NAVI_END_FLAG` 看定义应该为 `bool` 类型，为什么这里还能和字符串比较？

### 5.3 `handle_interrupt`

该函数用于处理所有情况下用户打断的操作。函数没有参数。

- 功能:

- 获取调用该方法的类的名称。
- 设置问答状态为正在进行。
- 如果检测到手持设备并且下位机存在，根据不同的类名称执行相应的旋转操作。
- 初始化任务类型为 `qa`，并清空触摸屏目标点。
- 进入循环处理用户的指令，直到任务类型不再是 `qa`。

- 实现步骤:

1. 获取调用该方法的类的名称，并打印处理打断的信息。
2. 设置问答状态为正在进行。
3. 如果检测到手持设备并且低计算机存在，根据类名称执行相应的旋转操作。

4. 初始化任务类型为 `qa`，清空触摸屏目标点，并设置清除展品名称标志为 `True`。
5. 进入循环处理用户的指令，直到任务类型不再是 `qa`。
  - 如果满足条件，播放提示音“大家有什么问题吗？”或“我在”。
  - 设置打断状态为 `False`，并重置相关标志。
  - 录制用户的指令。
  - 如果用户没有点击触控屏，使用任务分类模型进行任务分类，并记录任务分类结果。
  - 如果用户点击了触控屏，设置触摸屏目标点，并将任务类型设置为 `visit`。
  - 如果启用了姿态检测，并且问题中没有明确指出展品名称，进行关键词识别，并根据用户回答更新问题。
  - 如果任务类型仍为 `qa`，生成答案并播音。如果启用了动作执行，启动动作执行线程并等待其完成。

- 不理解的地方

- `name = self.get_self_name()` 什么时候能返回 `InterruptClass` 以及多个 `class`?
- `self.pre_next_situation = False` 这个属性是什么?
- `STATUS.Touchpad_Area == 'NowIsInterrupted'`  
`STATUS.set_Touchpad_Area('NowIgnoreTouchpad')` 这两种状态什么含义
- 什么情况下需要把 `QAing` 状态设置为 `true`，为什么?
- 为什么这里的 `clear` 都是 `true`，没见到修改 `clear` 的操作