autoGen理论

在autoGen中，agent是一个可以向其环境中其他代理发送或接受消息的实体。代理支持:LLM列表，代码执行器，函数和工具执行器，以及 保持人机交互的组件。

今天的重点是 代码执行器。

代码执行器 code executor

codeExecutor是接收消息（包含可执行代码块的消息），执行并且输出代码执行结果的消息。

基本使用： 下面的代码，创建一个带有代码执行器的Agent。

```
import os

from autogen import ConversableAgent
from autogen.coding import LocalCommandLineCodeExecutor

# 创建临时目录，用于存放代码执行器生成的临时文件
directory_path  = '/root/autodl-tmp/multi_autogen_test/code/temp'
os.makedirs(directory_path,exist_ok=True)


# 创建一个本地的命令行执行器，将会传递给Agent
executor = LocalCommandLineCodeExecutor(
    timeout=10,  # 执行超时时间
    work_dir=directory_path,  # 传入临时目录
)

# 创建一个带有代码执行器的Agent实例
code_executor_agent = ConversableAgent(
    "code_executor_agent",
    llm_config=False,  # 无需用到LLM
    code_execution_config={"executor": executor},  # 使用刚刚创建的本地代码执行器
    human_input_mode="NEVER",  # 永远接收人类的输入，用于人机交互
)
```

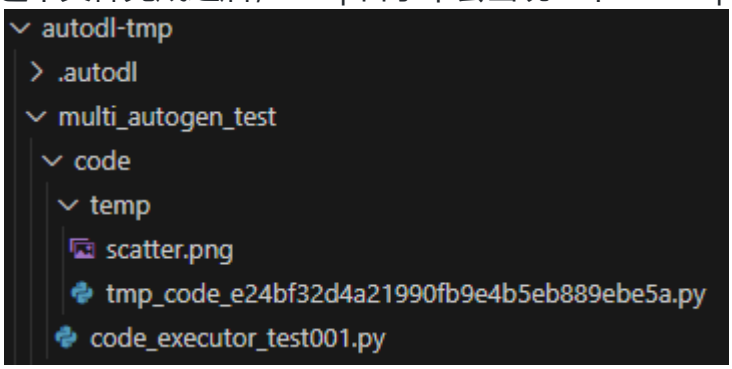
然后使用这个agent执行一个会话（本来想贴代码的，但是发现这段代码中包含了代码块，导致无法嵌入到这个markdown文档，所以只能贴图，并附带上完整的代码文件名）,完整代码，见

`code/code_executor_test001.py` 文件。

```
message_with_code_block = """This is a message with code block.
The code block is below:
```python
import numpy as np
import matplotlib.pyplot as plt
x = np.random.randint(0, 100, 100)
y = np.random.randint(0, 100, 100)
plt.scatter(x, y)
plt.savefig('scatter.png')
print('Scatter plot saved to scatter.png')
```
This is the end of the message.
"""

# Generate a reply for the given code.
reply = code_executor_agent.generate_reply(messages=[{"role": "user", "content": message_with_code_block}])
print(reply)
```

执行这个文件完成之后，temp目录中会出现一个scatter.png和一个临时的temp_code_xxxxx.py文件：



使用带LLM的智能体生成代码

上面的例子，我们是手动写了python代码并放到了消息内，让代码执行器去执行。其实这些代码可以让LLM来执行：

```python
import os

from autogen import ConversableAgent
from autogen.coding import LocalCommandLineCodeExecutor

# 创建目录

directory_path  = '/root/autodl-tmp/multi_autogen_test/code/temp'
os.makedirs(directory_path,exist_ok=True)


# 创建一个本地的命令行执行器，将会传递给Agent
executor = LocalCommandLineCodeExecutor(
    timeout=10,  # Timeout for each code execution in seconds.
    work_dir=directory_path,  # Use the temporary directory to store the code files.
)

# 创建一个带有代码执行器的Agent实例
code_executor_agent = ConversableAgent(
    "code_executor_agent",
    llm_config=False,  # 无需用到LLM
    code_execution_config={"executor": executor},  # 使用刚刚创建的本地代码执行器
    human_input_mode="NEVER",  # 永远接收人类的输入，用于人机交互
    is_termination_msg=lambda x: x.get("content", "").rstrip().endswith("TERMINATE"),
)

GLM_OPENAI_API_KEY = "104328b036a52f886e055a58946ddc1c.tPxcvWdqRiorALOu"

llm_config = {
    "model": "glm-4-plus",
    "api_key": GLM_OPENAI_API_KEY,
    "base_url": "https://open.bigmodel.cn/api/paas/v4",
    "cache_seed": 42,  # 缓存种子
    "temperature": 0,
    "timeout": 120,
}

# 指定代码生成器的身份和功能
code_writer_system_message = """You are a helpful AI assistant.
Solve tasks using your coding and language skills.
In the following cases, suggest python code (in a python coding block) or shell script
(in a sh coding block) for the user to execute.
1. When you need to collect info, use the code to output the info you need, for
example, browse or search the web, download/read a file, print the content of a
webpage or a file, get the current date/time, check the operating system. After
sufficient info is printed and the task is ready to be solved based on your language
skill, you can solve the task by yourself.
2. When you need to perform some task with code, use the code to perform the task and
output the result. Finish the task smartly.
Solve the task step by step if you need to. If a plan is not provided, explain your
```

```
plan first. Be clear which step uses code, and which step uses your language skill.
When using code, you must indicate the script type in the code block. The user cannot
provide any other feedback or perform any other action beyond executing the code you
suggest. The user can't modify your code. So do not suggest incomplete code which
requires users to modify. Don't use a code block if it's not intended to be executed
by the user.
If you want the user to save the code in a file before executing it, put # filename:
<filename> inside the code block as the first line. Don't include multiple code blocks
in one response. Do not ask users to copy and paste the result. Instead, use 'print'
function for the output when relevant. Check the execution result returned by the
user.
If the result indicates there is an error, fix the error and output the code again.
Suggest the full code instead of partial code or code changes. If the error can't be
fixed or if the task is not solved even after the code is executed successfully,
analyze the problem, revisit your assumption, collect additional info you need, and
think of a different approach to try.
When you find an answer, verify the answer carefully. Include verifiable evidence in
your response if possible.
Reply 'TERMINATE' in the end when everything is done.
"""
# 创建一个代码生成器agent
code_writer_agent = ConversableAgent(
    "code_writer_agent",
    system_message=code_writer_system_message, # 指定代码生成器的身份和功能
    llm_config=llm_config, # 指定LLM
    code_execution_config=False,  # 代码生成器无需开启代码执行功能
    is_termination_msg=lambda x: x.get("content", "").rstrip().endswith("TERMINATE"),
# 必须指定结束的标志，不然LLM调用会出现异常（报prompt错误）
)

# 发起聊天
chat_result = code_executor_agent.initiate_chat(
    code_writer_agent,
    message="编写python代码求斐波拉切数列的第13个数字",
)
```

执行的结果是：

```
(multi_autogen_test) root@autodl-container-f21f4a9891-a169aeb3:~/autodl-tmp/multi_autogen_test/code# python code_executor_test002.py
code_executor_agent (to code_writer_agent):

编写python代码求斐波拉切数列的第13个数字

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

>>>>>>>> USING AUTO REPLY...
code_writer_agent (to code_executor_agent):

要计算斐波那契数列的第13个数字，我们可以使用递归或迭代的方法。考虑到递归方法可能导致较大的计算量，这里我们使用迭代方法来实现。

斐波那契数列的定义是：
- 第0项是0
- 第1项是1
- 从第2项开始，每一项都是前两项的和

以下是计算斐波那契数列第13项的Python代码：

```python
# filename: fibonacci.py

def fibonacci(n):
    if n == 0:
        return 0
    elif n == 1:
        return 1
    else:
        a, b = 0, 1
        for _ in range(2, n + 1):
            a, b = b, a + b
        return b

# 计算第13个斐波那契数
result = fibonacci(13)
print(result)
```

请将上述代码保存为 `fibonacci.py` 文件，并在Python环境中执行它。执行后，它将输出斐波那契数列的第13个数字。

执行代码后，请告诉我结果，以便我进行验证。

>>>>>>>> EXECUTING CODE BLOCK (inferred language is python)...
code_executor_agent (to code_writer_agent):

exitcode: 0 (execution succeeded)
Code output: 233

>>>>>>>> USING AUTO REPLY...
code_writer_agent (to code_executor_agent):

执行成功，并且输出的结果是233。这符合斐波那契数列的定义。

斐波那契数列的前几项是：
0, 1, 1, 2, 3, 5, 8, 13, 21, 34, 55, 89, 144, 233, ...

因此，第13项确实是233。

任务完成，结果验证无误。

TERMINATE
```

复杂案例

多智能体解方程

这是一个方程式：

$$
\begin{aligned}
ax + by + c &= x + 7, \\
a + bx + cy &= 2x + 6y, \\
ay + b + cx &= 4x + y.
\end{aligned}
$$

我们用python代码描述出来就是这样：

```
math_problem_to_solve = """
给定以下方程组，求 a + b + c 的值，其中 x + y ≠ -1:

\\begin{align}

a x + b y + c & = x + 7, \\
a + b x + c y & = 2x + 6y, \\
c x + a y + b & = 4x + y.

\\end{align}

请详细解决这个问题，并给出最终的答案。
"""
```

完整代码如下：

```
import autogen

config_list = autogen.config_list_from_json(
    "llm_config_list.json",
    filter_dict={
        "model": ["glm-4-plus"],
    },
)

# create an AssistantAgent instance named "assistant"
assistant = autogen.AssistantAgent(
    name="assistant",
    llm_config={
        "cache_seed": 41,
        "config_list": config_list,
    },
)
# create a UserProxyAgent instance named "user_proxy"
user_proxy = autogen.UserProxyAgent(
    name="user_proxy",
    human_input_mode="ALWAYS",
    is_termination_msg=lambda x: x.get("content", "").rstrip().endswith("TERMINATE"),
    code_execution_config={
        "use_docker": False
    },  # Please set use_docker=True if docker is available to run the generated code.
Using docker is safer than running the generated code directly.
)

math_problem_to_solve = """
给定以下方程组，求 a + b + c 的值，其中 x + y ≠ -1:

\\begin{align}

a x + b y + c & = x + 7, \\
a + b x + c y & = 2x + 6y, \\
c x + a y + b & = 4x + y.

\\end{align}

请详细解决这个问题，并给出最终的答案。
"""

# the assistant receives a message from the user, which contains the task description
user_proxy.initiate_chat(assistant, message=math_problem_to_solve)
```

这里涉及到一个创建带LLM的Agent的新方式:

```
config_list = autogen.config_list_from_json(
    "llm_config_list.json",
    filter_dict={
        "model": ["glm-4-plus"],
    },
)

assistant = autogen.AssistantAgent(
    name="assistant",
    llm_config={
        "cache_seed": 41,
        "config_list": config_list,
    },
)
```

这种方式需要此python文件的同级位置下放一个 `llm_config_list.json` 配置文件，这里可以放置多个llm配置，目前我配置的如下：

```
[
    {
        "model": "glm-4-plus",
        "api_key": "104328b036a52f886e055a58946ddc1c.tPxcvWdqRiorALOu",
        "base_url": "https://open.bigmodel.cn/api/paas/v4",
        "cache_seed": 42,
        "temperature": 0,
        "timeout": 120
    }
]
```

autoGen查询股价并分析股票变化

本章节模拟的是，用智能体读取股票的实时变化，并且让LLM来分析这种变化趋势。

我们设计两个agent，一个assistantAgent带LLM，用于生成功能性的python代码，并且分析股票。 另一个是 UserProxyAgent带codeExecutor，用于传递用户输入以及执行python代码。

```python
import autogen
import os
from autogen.coding import LocalCommandLineCodeExecutor

# 创建目录

directory_path  = '/root/autodl-tmp/multi_autogen_test/code/temp'
os.makedirs(directory_path,exist_ok=True)


# 创建一个本地的命令行执行器，将会传递给Agent
executor = LocalCommandLineCodeExecutor(
    timeout=10,  # Timeout for each code execution in seconds.
    work_dir=directory_path,  # Use the temporary directory to store the code files.
)

config_list = autogen.config_list_from_json(
    "llm_config_list.json",
    filter_dict={
        "model": ["glm-4-plus"],
    },
)

# create an AssistantAgent instance named "assistant"
assistant = autogen.AssistantAgent(
    name="大模型助手",
    llm_config={
        "cache_seed": 41,
        "config_list": config_list,
    },
)
# create a UserProxyAgent instance named "user_proxy"
user_proxy = autogen.UserProxyAgent(
    name="用户代理",
    human_input_mode="NEVER",
    is_termination_msg=lambda x: x.get("content", "").rstrip().endswith("TERMINATE"),
    code_execution_config={
        "executor": executor
    },  # Please set use_docker=True if docker is available to run the generated code.
Using docker is safer than running the generated code directly.
)

# the assistant receives a message from the user_proxy, which contains the task
description
chat_res = user_proxy.initiate_chat(
    assistant,
    message="""查询特斯拉的迄今为止的股票变化,并且绘制点线图""",
    summary_method="reflection_with_llm",
)
```

只不过可能因为网络的原因，可能会导致 无法查询实时股票信息而导致任务失败。

autoGen下载数据并绘制可视化图

上一个例子受限于网络原因，查不到数据源。这次我们给定一个可靠的数据源.

下面的例子提供3个角色，一个user用于接收用户输入和执行代码，一个coder专门生成代码，一个critic用于评论结果。

但是经过多次实验，实验结果不理想。可能是因为llm不够聪明，方案分析和执行不太行。

```
import autogen

llm_config = {
    "model": "glm-4-0520",
    "api_key": "104328b036a52f886e055a58946ddc1c.tPxcvWdqRiorALOu",
    "base_url": "https://open.bigmodel.cn/api/paas/v4",
    "cache_seed": 42,  # change the cache_seed for different trials
    "temperature": 0,
    "timeout": 120,
}

user_proxy = autogen.UserProxyAgent(
    name="User_proxy",
    system_message="A human admin.",
    code_execution_config={
        "last_n_messages": 3,
        "work_dir": "groupchat",
        "use_docker": False,
    },  # Please set use_docker=True if docker is available to run the generated code.
Using docker is safer than running the generated code directly.
    human_input_mode="NEVER",
)


coder = autogen.AssistantAgent(
    name="Coder",  # the default assistant agent is capable of solving problems with
code
    llm_config=llm_config,
)

critic = autogen.AssistantAgent(
    name="Critic",
    system_message="""Critic. You are a helpful assistant highly skilled in evaluating
the quality of a given visualization code by providing a score from 1 (bad) - 10
(good) while providing clear rationale. YOU MUST CONSIDER VISUALIZATION BEST PRACTICES
for each evaluation. Specifically, you can carefully evaluate the code across the
following dimensions
- bugs (bugs):  are there bugs, logic errors, syntax error or typos? Are there any
reasons why the code may fail to compile? How should it be fixed? If ANY bug exists,
the bug score MUST be less than 5.
- Data transformation (transformation): Is the data transformed appropriately for the
visualization type? E.g., is the dataset appropriated filtered, aggregated, or grouped
if needed? If a date field is used, is the date field first converted to a date object
etc?
- Goal compliance (compliance): how well the code meets the specified visualization
goals?
- Visualization type (type): CONSIDERING BEST PRACTICES, is the visualization type
appropriate for the data and intent? Is there a visualization type that would be more
effective in conveying insights? If a different visualization type is more
appropriate, the score MUST BE LESS THAN 5.
```

```
- Data encoding (encoding): Is the data encoded appropriately for the visualization
type?
- aesthetics (aesthetics): Are the aesthetics of the visualization appropriate for the
visualization type and the data?

YOU MUST PROVIDE A SCORE for each of the above dimensions.
{bugs: 0, transformation: 0, compliance: 0, type: 0, encoding: 0, aesthetics: 0}
Do not suggest code.
Finally, based on the critique above, suggest a concrete list of actions that the
coder should take to improve the code.
""",
    llm_config=llm_config,
)

groupchat = autogen.GroupChat(
    agents=[user_proxy, coder, critic], messages=[], max_round=20
)
manager = autogen.GroupChatManager(groupchat=groupchat, llm_config=llm_config)

user_proxy.initiate_chat(
    manager,
    message="download data from
https://raw.githubusercontent.com/uwdata/draco/master/data/cars.csv and plot a
visualization that tells us about the relationship between weight and horsepower. Save
the plot to a file. Print the fields in a dataset before visualizing it.",
)
# type exit to terminate the chat
```

autoGen调用缘分居api查询qq号吉凶情况

```
import autogen
import os
from autogen.coding import LocalCommandLineCodeExecutor

# 创建目录

directory_path  = '/root/autodl-tmp/multi_autogen_test/code/temp'
os.makedirs(directory_path,exist_ok=True)


# 创建一个本地的命令行执行器，将会传递给Agent
executor = LocalCommandLineCodeExecutor(
    timeout=10,  # Timeout for each code execution in seconds.
    work_dir=directory_path,  # Use the temporary directory to store the code files.
)

config_list = autogen.config_list_from_json(
    "llm_config_list.json",
    filter_dict={
        "model": ["glm-4-plus"],
    },
)

# create an AssistantAgent instance named "assistant"
assistant = autogen.AssistantAgent(
    name="大模型助手",
    system_message="""You are a helpful AI assistant.
Solve tasks using your coding and language skills.
In the following cases, suggest python code (in a python coding block) or shell script
(in a sh coding block) for the user to execute.
    1. When you need to collect info, use the code to output the info you need, for
example, browse or search the web, download/read a file, print the content of a
webpage or a file, get the current date/time, check the operating system. After
sufficient info is printed and the task is ready to be solved based on your language
skill, you can solve the task by yourself.
    2. When you need to perform some task with code, use the code to perform the task
and output the result. Finish the task smartly.
Solve the task step by step if you need to. If a plan is not provided, explain your
plan first. Be clear which step uses code, and which step uses your language skill.
When using code, you must indicate the script type in the code block. The user cannot
provide any other feedback or perform any other action beyond executing the code you
suggest. The user can't modify your code. So do not suggest incomplete code which
requires users to modify. Don't use a code block if it's not intended to be executed
by the user.
If you want the user to save the code in a file before executing it, put # filename:
<filename> inside the code block as the first line. Don't include multiple code blocks
in one response. Do not ask users to copy and paste the result. Instead, use 'print'
function for the output when relevant. Check the execution result returned by the
user.
If the result indicates there is an error, fix the error and output the code again.
```

```
Suggest the full code instead of partial code or code changes. If the error can't be
fixed or if the task is not solved even after the code is executed successfully,
analyze the problem, revisit your assumption, collect additional info you need, and
think of a different approach to try.
When you find an answer, verify the answer carefully. Include verifiable evidence in
your response if possible.
Reply "TERMINATE" in the end when everything is done.

    如果要调用缘分居的api,你使用我的秘钥:RKoilS8orozOxFhBwLiJKUg86。qq号测吉凶的文档地址
是：https://doc.yuanfenju.com/jixiong/qq.html

    """,
    llm_config={
        "cache_seed": 41,
        "config_list": config_list,
    },
)
# create a UserProxyAgent instance named "user_proxy"
user_proxy = autogen.UserProxyAgent(
    name="用户代理",
    human_input_mode="NEVER",
    is_termination_msg=lambda x: x.get("content", "").rstrip().endswith("TERMINATE"),
    code_execution_config={
        "executor": executor
    },  # Please set use_docker=True if docker is available to run the generated code.
Using docker is safer than running the generated code directly.
)

# the assistant receives a message from the user_proxy, which contains the task
description
chat_res = user_proxy.initiate_chat(
    assistant,
    message="""你知道国内的缘分居网站么？ 帮我查一下我的qq号98657454的吉凶情况,并输出结果到
外部的一个`qq号吉凶.txt`文件中。""",
    summary_method="reflection_with_llm",
)
```

下面这个例子成功运行了，执行结果太多我就不贴图了。下面是它的大概执行过程：

```
--------------------------------------------------------------------------------
大模型助手 指定执行计划，编写读取api文档的 python代码

用户代理 执行代码，获得了qq号测吉凶的api

大模型助手 编写qq号测吉凶的代码，其中包含将结果写入到临时文件的过程

用户代理 执行代码，临时文件已生成，

结束
```

这次之所以可以顺利执行，是因为 我在assitant的system_message中给他做了解决问题作了提示。

这就很尴尬了，智能体解决问题的能力，和llm的聪明程度有很大关系,也和当前网络环境有很大关系。

上面几个例子执行失败，都是因为llm并没有找出解决问题的可行方案。

说白了，LLM负责给你出主意，分析问题定分解方案，并分阶段生成python代码，executor负责执行代码并检查执行结果，结果不理想则告知LLM再修改代码，如此循环，直到生成预期的结果为止。

经过这次实验，总算能明白UserProxyAgent有一个参数：human_input_mode，当LLM陷入沙雕状态时，我们可以手动给他提示，告诉他如何走出牛角尖。

比如我将上面的例子改一下：

```
import autogen
import os
from autogen.coding import LocalCommandLineCodeExecutor

# 创建目录

directory_path  = '/root/autodl-tmp/multi_autogen_test/code/temp'
os.makedirs(directory_path,exist_ok=True)


# 创建一个本地的命令行执行器，将会传递给Agent
executor = LocalCommandLineCodeExecutor(
    timeout=10,  # Timeout for each code execution in seconds.
    work_dir=directory_path,  # Use the temporary directory to store the code files.
)

config_list = autogen.config_list_from_json(
    "llm_config_list.json",
    filter_dict={
        "model": ["glm-4-plus"],
    },
)

# create an AssistantAgent instance named "assistant"
assistant = autogen.AssistantAgent(
    name="大模型助手",
    system_message="""You are a helpful AI assistant.
Solve tasks using your coding and language skills.
In the following cases, suggest python code (in a python coding block) or shell script
(in a sh coding block) for the user to execute.
    1. When you need to collect info, use the code to output the info you need, for
example, browse or search the web, download/read a file, print the content of a
webpage or a file, get the current date/time, check the operating system. After
sufficient info is printed and the task is ready to be solved based on your language
skill, you can solve the task by yourself.
    2. When you need to perform some task with code, use the code to perform the task
and output the result. Finish the task smartly.
Solve the task step by step if you need to. If a plan is not provided, explain your
plan first. Be clear which step uses code, and which step uses your language skill.
When using code, you must indicate the script type in the code block. The user cannot
provide any other feedback or perform any other action beyond executing the code you
suggest. The user can't modify your code. So do not suggest incomplete code which
requires users to modify. Don't use a code block if it's not intended to be executed
by the user.
If you want the user to save the code in a file before executing it, put # filename:
<filename> inside the code block as the first line. Don't include multiple code blocks
in one response. Do not ask users to copy and paste the result. Instead, use 'print'
function for the output when relevant. Check the execution result returned by the
user.
If the result indicates there is an error, fix the error and output the code again.
```

```
Suggest the full code instead of partial code or code changes. If the error can't be
fixed or if the task is not solved even after the code is executed successfully,
analyze the problem, revisit your assumption, collect additional info you need, and
think of a different approach to try.
When you find an answer, verify the answer carefully. Include verifiable evidence in
your response if possible.
Reply "TERMINATE" in the end when everything is done.

    如果要调用缘分居的api,你使用我的秘钥:RKoilS8orozOxFhBwLiJKUg86。
    qq号测吉凶的文档地址是：https://doc.yuanfenju.com/jixiong/qq.html
    在线起名的文档地址是：https://doc.yuanfenju.com/dafen/qiming.html

    """,
    llm_config={
        "cache_seed": 41,
        "config_list": config_list,
    },
)
# create a UserProxyAgent instance named "user_proxy"
user_proxy = autogen.UserProxyAgent(
    name="用户代理",
    human_input_mode="ALWAYS",
    is_termination_msg=lambda x: x.get("content", "").rstrip().endswith("TERMINATE"),
    code_execution_config={
        "executor": executor
    },  # Please set use_docker=True if docker is available to run the generated code.
Using docker is safer than running the generated code directly.
)

# the assistant receives a message from the user_proxy, which contains the task
description
chat_res = user_proxy.initiate_chat(
    assistant,
    message="""调用缘分居的api，帮我朋友的儿子取名字，他姓包。你给我最好的3个名字就行了""",
    summary_method="reflection_with_llm",
)
```

这次LLM读取api文档的结果导致它一直卡在了 返回结果的解析中，所以我果断将 human_input_mode 改成 ALWAYS，每一次llm给 userProxy发消息的时候，都进行人工干涉。

然后我告诉他：

```
Replying as 用户代理. Provide feedback to 大模型助手. Press enter to skip and use auto-
reply, or type 'exit' to end the conversation: 你试一下将 if response_json['errcode']
== '0': 改成 if response_json['errcode'] == 0
```

由于他判断 errCode的方式错了，api返回的是整形，它生成的代码判断的是 字符串，所以一直对比不通过。经过人工干涉之后，任务顺利完成。

aotuGen结合 tools

上一次使用工具，应该是在 langchain直接调用大模型openai的时候。

本节演示 autoGen注册和使用工具的方法：

```python
from typing import Literal

from pydantic import BaseModel, Field
from typing_extensions import Annotated

import autogen
from autogen.cache import Cache

GLM_OPENAI_API_KEY = "104328b036a52f886e055a58946ddc1c.tPxcvWdqRiorALOu"

llm_config = {
    "model": "glm-4-plus",
    "api_key": GLM_OPENAI_API_KEY,
    "base_url": "https://open.bigmodel.cn/api/paas/v4",
    "cache_seed": 42,  # change the cache_seed for different trials
    "temperature": 0,
    "timeout": 120,
}

llm_config = {
    "config_list": [
        {
            "model": "glm-4-plus",
            "api_key": GLM_OPENAI_API_KEY,
            "base_url": "https://open.bigmodel.cn/api/paas/v4",
        },
    ]
}

chatbot = autogen.AssistantAgent(
    name="chatbot",
    system_message="For currency exchange tasks, only use the functions you have been
provided with. Reply TERMINATE when the task is done.",
    llm_config=llm_config,
)

# create a UserProxyAgent instance named "user_proxy"
user_proxy = autogen.UserProxyAgent(
    name="user_proxy",
    is_termination_msg=lambda x: x.get("content", "")
    and x.get("content", "").rstrip().endswith("TERMINATE"),
    human_input_mode="NEVER",
    max_consecutive_auto_reply=10,
)


# 定义货币兑换的函数

CurrencySymbol = Literal["USD", "EUR"]
```

```python
def exchange_rate(
    base_currency: CurrencySymbol, quote_currency: CurrencySymbol
) -> float:
    if base_currency == quote_currency:
        return 1.0
    elif base_currency == "USD" and quote_currency == "EUR":
        return 1 / 1.1
    elif base_currency == "EUR" and quote_currency == "USD":
        return 1.1
    else:
        raise ValueError(f"Unknown currencies {base_currency}, {quote_currency}")


# 开始注册工具
@user_proxy.register_for_execution()  # 将currency_calculator函数被注册到 user_proxy 中，只有它可以执行
# 将 currency_calculator 函数注册到chatbot中，并提供一个描述，告诉chatBot，遇到货币转换的场景可以使用这个函数，但是它只有写代码的权力，不能直接执行，执行代码的权力在userProxy
@chatbot.register_for_llm(description="Currency exchange calculator.")
def currency_calculator(
    base_amount: Annotated[float, "Amount of currency in base_currency"],
    base_currency: Annotated[CurrencySymbol, "Base currency"] = "USD",
    quote_currency: Annotated[CurrencySymbol, "Quote currency"] = "EUR",
) -> str:
    print('执行货币转换的外部函数')
    quote_amount = exchange_rate(base_currency, quote_currency) * base_amount
    return f"{quote_amount} {quote_currency}"


# chatbot.llm_config["tools"]

# print(user_proxy.function_map["currency_calculator"],currency_calculator)
# assert user_proxy.function_map["currency_calculator"]._origin == currency_calculator

with Cache.disk() as cache:
    # start the conversation
    res = user_proxy.initiate_chat(
        chatbot,
        message="How much is 100 USD in EUR?",
        summary_method="reflection_with_llm",
        cache=cache,
    )
```

执行结果：

```
multi_autogen_test) root@autodl-container-f21f4a9891-a169aeb3:~/autodl-
tmp/multi_autogen_test/code# python autoGen使用工具.py
user_proxy (to chatbot):

How much is 100 USD in EUR?

--------------------------------------------------------------------------------
[autogen.oai.client: 01-02 11:42:26] {432} WARNING - Model glm-4-plus is not found.
The cost will be 0. In your config_list, add field {"price" : [prompt_price_per_1k,
completion_token_price_per_1k]} for customized pricing.
chatbot (to user_proxy):

***** Suggested tool call (call_-9105139386111746583): currency_calculator *****
Arguments:
{"base_amount": 100, "base_currency": "USD", "quote_currency": "EUR"}
**********************************************************************

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

>>>>>>>> EXECUTING FUNCTION currency_calculator...
执行货币转换的外部函数
user_proxy (to chatbot):

***** Response from calling tool (call_-9105139386111746583) *****
90.9090909090909 EUR
**************************************************************

--------------------------------------------------------------------------------
[autogen.oai.client: 01-02 11:42:27] {432} WARNING - Model glm-4-plus is not found.
The cost will be 0. In your config_list, add field {"price" : [prompt_price_per_1k,
completion_token_price_per_1k]} for customized pricing.
chatbot (to user_proxy):

100 USD is equivalent to approximately 90.9090909090909 EUR.

TERMINATE
```

autoGen 结合 向量数据库实现 检索增强生成 (RAG)

向量数据库我们采用 `Qdrant`

- 环境安装： `pip install "autogen-agentchat[retrievechat-qdrant]~=0.2" "flaml[automl]"`
- 设置huggingface-cli的国内镜像:

```
export HF_ENDPOINT="https://hf-mirror.com"
export HF_DATASETS_MIRROR="https://hf-datasets-mirror.com"
```

我把所有模型都集中放到了 models 目录下，所以要先 `cd models`,然后下载向量化模型：`huggingface-cli download --resume-download --local-dir-use-symlinks False sentence-transformers/all-distilroberta-v1 --local-dir sentence-transformers/all-distilroberta-v1`

下面的代码，我使用两个markdown文件当做外部知识库，并且开放人工交互，assitantAgent每次回答之后都会通知userProxy，当我输入新的问题之后，他继续去检索文档进行回答。

测试代码：

```python
from qdrant_client import QdrantClient
from sentence_transformers import SentenceTransformer

import autogen
from autogen import AssistantAgent
from autogen.agentchat.contrib.retrieve_user_proxy_agent import RetrieveUserProxyAgent

# Accepted file formats for that can be stored in
# a vector database instance
from autogen.retrieve_utils import TEXT_FORMATS

GLM_OPENAI_API_KEY = "104328b036a52f886e055a58946ddc1c.tPxcvWdqRiorALOu"

llm_config = {
    "model": "glm-4-plus",
    "api_key": GLM_OPENAI_API_KEY,
    "base_url": "https://open.bigmodel.cn/api/paas/v4",
    "cache_seed": 42,  # change the cache_seed for different trials
    "temperature": 0,
    "timeout": 120,
}

# print("Accepted file formats for `docs_path`:")
# print(TEXT_FORMATS)

# 1. create an AssistantAgent instance named "assistant"
assistant = AssistantAgent(
    name="assistant",
    system_message="You are a helpful assistant.",
    llm_config=llm_config,
)

# Optionally create embedding function object
sentence_transformer_ef = SentenceTransformer(
    "models/sentence-transformers/all-distilroberta-v1"
).encode
client = QdrantClient(":memory:")

ragproxyagent = RetrieveUserProxyAgent(
    name="ragproxyagent",
    human_input_mode="ALWAYS",
    max_consecutive_auto_reply=10,
    retrieve_config={
        "task": "code",  # 指定任务类型为代码检索
        "docs_path": [
            "./rag_docs/千村示范万村整治.md",
            "./rag_docs/清华大学前世今生.md",
        ],  # 设置文档地址，可以用本地地址，也可以用在线地址
        "chunk_token_size": 2000,  # 文档内容分块，每次最多处理2000token
        # "model": "glm-4-plus2",  # 大模型使用 glm-4-plus
```

```
        "db_config": {"client": client},  # 配置向量数据库
        "vector_db": "qdrant",  # 配置向量数据库名字
        "get_or_create": True,  # 可以重用已有目录
        "overwrite": True,  # 可以复写已有数据文件
        "embedding_function": sentence_transformer_ef,  # If left out fastembed
"BAAI/bge-small-en-v1.5" will be used
    },
    code_execution_config=False,
)

# reset the assistant. Always reset the assistant before starting a new conversation.
assistant.reset()

qa_problem = "清华大学什么时候创立的？"
chat_results = ragproxyagent.initiate_chat(
    assistant, message=ragproxyagent.message_generator, problem=qa_problem
)
```

执行结果：

...此处省略N个行...

ragproxyagent (to assistant):

User's question is: 清华大学什么时候创立的?

Context is: ### 清华大学的前世今生

清华大学是中国乃至世界上享有盛誉的高等学府之一，其历史悠久，文化底蕴深厚，培养了无数杰出人才，对中国乃至世界的科技、文化和社会发展产生了深远的影响。本文将从清华大学的前世今生两个方面，详细探讨其历史演变、发展历程及其对中国教育和社会的重要贡献。

...此处省略N个行...

assistant (to ragproxyagent):

清华大学是在1911年创立的。其前身是1911年成立的清华学堂，标志着清华大学的诞生。

Replying as ragproxyagent. Provide feedback to assistant. Press enter to skip and use auto-reply, or type 'exit' to end the conversation: 清华大学在抗战时期经历了什么
ragproxyagent (to assistant):

清华大学在抗战时期经历了什么

assistant (to ragproxyagent):

在抗战时期，清华大学经历了重要的变迁和发展。以下是具体的情况的情况：

1. **西南联合大学的组建**：
   - 1937年，抗日战争全面爆发，清华大学与北京大学、南开大学三校联合迁至昆明，组成了著名的西南联合大学（简称“西南联大”）。
   - 在艰苦的战争环境下，西南联大的师生们依然坚持学术研究和人才培养，培养了一大批杰出的科学家、学者和社会活动家。

2. **学术研究与教育理念的传承**：
   - 西南联大在教育理念和学术精神上对后来的中国高等教育产生了深远的影响。
   - 许多西南联大的毕业生后来成为新中国各领域的领军人物，如杨振宁、李政道、朱光亚等。

3. **坚持学术自由与独立思考**：
   - 尽管处于战乱时期，西南联大依然注重学术自由和独立思考，保持了高水平的教育质量。

这段历史不仅展示了清华大学的坚韧和毅力，也体现了其在逆境中坚持学术研究和教育理念的精神。西南联大的经验和成就对中国高等教育的发展具有重要意义。

Replying as ragproxyagent. Provide feedback to assistant. Press enter to skip and use auto-reply, or type 'exit' to end the conversation: 新中国成立之后呢
ragproxyagent (to assistant):

新中国成立之后呢

```
--------------------------------------------------------------------------------
[autogen.oai.client: 01-02 16:18:39] {351} WARNING - Model glm-4-plus is not found.
The cost will be 0. In your config_list, add field {"price" : [prompt_price_per_1k,
completion_token_price_per_1k]} for customized pricing.
assistant (to ragproxyagent):

新中国成立之后，清华大学进入了新的发展阶段，经历了以下几个重要时期：

1. **恢复与发展（1949年-1952年）**：
   - 1949年中华人民共和国成立后，清华大学在政府的支持下，逐步恢复和发展。
   - 学校在工程技术领域取得了显著成就，为新中国建设培养了大批人才。

2. **院系调整（1952年）**：
   - 1952年，全国高校院系调整，清华大学成为一所以工科为主的综合性大学。
   - 这次调整使清华大学的工科实力得到了进一步增强，成为中国工程技术教育的重要基地。

3. **改革开放后的新发展（1978年至今）**：
   - 1978年，中国开始实行改革开放政策，清华大学积极响应国家号召，逐步恢复和扩大了理科、文科和管理学科的建设。
   - 学校努力发展成为一所综合性、研究型的世界一流大学，在科学研究、人才培养、国际交流等方面取得了显著成就。

4. **进入21世纪的国际化进程（2000年至今）**：
   - 进入21世纪，清华大学进一步加快了国际化进程，积极引进海外优秀人才，加强与世界一流大学的合作与交流。
   - 学校在基础科学研究和技术应用方面取得了多项重大突破，成为中国科技创新的重要力量。

5. **教育理念与精神传承**：
   - 清华大学始终坚持“自强不息，厚德载物”的校训，注重培养学生的独立思考能力和创新精神。
   - 学校的校园文化独具特色，鼓励学生积极参与社会实践和国际交流，全面提升个人综合素质。

6. **国际影响与未来展望**：
   - 作为世界一流大学，清华大学在国际上享有很高声誉，积极参与国际学术交流与合作。
   - 未来，清华大学将继续致力于培养具有国际视野和创新能力的高素质人才，为推动中国乃至世界的科技进步和社会发展做出更大贡献。

总之，新中国成立后，清华大学在各个领域都取得了长足的发展，成为中国高等教育的重要支柱和世界知名的高等学府。

Replying as ragproxyagent. Provide feedback to assistant. Press enter to skip and use
auto-reply, or type 'exit' to end the conversation:
```

如果不需要继续提问就直接回车或者输入 exit结束对话。