**import** json  
**import** logging  
**import** os  
**import** pickle  
**import** copy  
**import** signal  
**import** sys  
**import** time  
**import** openai  
**import** anthropic  
**from** bot.bot **import** Bot  
**from** bot.openai.open\_ai\_image **import** OpenAIImage  
**from** bot.chatgpt.chat\_gpt\_session **import** ChatGPTSession  
**from** bot.gemini.google\_gemini\_bot **import** GoogleGeminiBot  
**from** bot.session\_manager **import** SessionManager  
**from** bridge.context **import** ContextType  
**from** bridge.reply **import** Reply, ReplyType  
**from** common.log **import** logger  
**from** config **import** conf  
**from** channel **import** channel\_factory  
**from** common **import** const  
**from** config **import** load\_config  
**from** plugins **import** \*  
**import** threading  
**from** common.log **import** logger  
**class** Bridge(object):  
 **def** \_\_init\_\_(self):  
 self.btype = {  
 "chat": const.CHATGPT,  
 "voice\_to\_text": conf().get("voice\_to\_text", "openai"),  
 "text\_to\_voice": conf().get("text\_to\_voice", "google"),  
 "translate": conf().get("translate", "baidu"),  
 }  
 *# 这边取配置的模型*  
 model\_type = conf().get("model") **or** const.GPT35  
 **if** model\_type **in** ["text-davinci-003"]:  
 self.btype["chat"] = const.OPEN\_AI  
 **if** conf().get("use\_azure\_chatgpt", False):  
 self.btype["chat"] = const.CHATGPTONAZURE  
 **if** model\_type **in** ["wenxin", "wenxin-4"]:  
 self.btype["chat"] = const.BAIDU  
 **if** model\_type **in** ["xunfei"]:  
 self.btype["chat"] = const.XUNFEI  
 **if** model\_type **in** [const.QWEN]:  
 self.btype["chat"] = const.QWEN  
 **if** model\_type **in** [const.QWEN\_TURBO, const.QWEN\_PLUS, const.QWEN\_MAX]:  
 self.btype["chat"] = const.QWEN\_DASHSCOPE  
 **if** model\_type **in** [const.GEMINI]:  
 self.btype["chat"] = const.GEMINI  
 **if** model\_type **in** [const.ZHIPU\_AI]:  
 self.btype["chat"] = const.ZHIPU\_AI  
 **if** model\_type **and** model\_type.startswith("claude-3"):  
 self.btype["chat"] = const.CLAUDEAPI  
  
 **if** model\_type **in** ["claude"]:  
 self.btype["chat"] = const.CLAUDEAI  
  
 **if** model\_type **in** ["moonshot-v1-8k", "moonshot-v1-32k", "moonshot-v1-128k"]:  
 self.btype["chat"] = const.MOONSHOT  
  
 **if** conf().get("use\_linkai") **and** conf().get("linkai\_api\_key"):  
 self.btype["chat"] = const.LINKAI  
 **if** **not** conf().get("voice\_to\_text") **or** conf().get("voice\_to\_text") **in** ["openai"]:  
 self.btype["voice\_to\_text"] = const.LINKAI  
 **if** **not** conf().get("text\_to\_voice") **or** conf().get("text\_to\_voice") **in** ["openai", const.TTS\_1, const.TTS\_1\_HD]:  
 self.btype["text\_to\_voice"] = const.LINKAI  
  
 self.bots = {}  
 self.chat\_bots = {}  
  
 **def** fetch\_reply\_content(self, query, context: Context) -> Reply:  
 **return** self.get\_bot("chat").reply(query, context)  
  
 **def** fetch\_voice\_to\_text(self, voiceFile) -> Reply:  
 **return** self.get\_bot("voice\_to\_text").voiceToText(voiceFile)  
  
 **def** fetch\_text\_to\_voice(self, text) -> Reply:  
 **return** self.get\_bot("text\_to\_voice").textToVoice(text)  
  
 **def** fetch\_translate(self, text, from\_lang="", to\_lang="en") -> Reply:  
 **return** self.get\_bot("translate").translate(text, from\_lang, to\_lang)  
  
 **def** find\_chat\_bot(self, bot\_type: str):  
 **if** self.chat\_bots.get(bot\_type) **is** None:  
 self.chat\_bots[bot\_type] = create\_bot(bot\_type)  
 **return** self.chat\_bots.get(bot\_type)  
  
 **def** reset\_bot(self):  
 self.\_\_init\_\_()

**class** Config(dict):  
 **def** \_\_init\_\_(self, d=None):  
 super().\_\_init\_\_()  
 **if** d **is** None:  
 d = {}  
 **for** k, v **in** d.items():  
 self[k] = v  
 self.user\_datas = {}  
  
 **def** \_\_getitem\_\_(self, key):  
 **if** key **not** **in** available\_setting:  
 **raise** Exception("key {} not in available\_setting".format(key))  
 **return** super().\_\_getitem\_\_(key)  
  
 **def** \_\_setitem\_\_(self, key, value):  
 **if** key **not** **in** available\_setting:  
 **raise** Exception("key {} not in available\_setting".format(key))  
 **return** super().\_\_setitem\_\_(key, value)  
  
 **def** get(self, key, default=None):  
 **try**:  
 **return** self[key]  
 **except** KeyError **as** e:  
 **return** default  
 **except** Exception **as** e:  
 **raise** e  
  
   
 **def** get\_user\_data(self, user) -> dict:  
 **if** self.user\_datas.get(user) **is** None:  
 self.user\_datas[user] = {}  
 **return** self.user\_datas[user]  
  
 **def** load\_user\_datas(self):  
 **try**:  
 with open(os.path.join(get\_appdata\_dir(), "user\_datas.pkl"), "rb") **as** f:  
 self.user\_datas = pickle.load(f)  
 logger.info("[Config] User datas loaded.")  
 **except** FileNotFoundError **as** e:  
 logger.info("[Config] User datas file not found, ignore.")  
 **except** Exception **as** e:  
 logger.info("[Config] User datas error: {}".format(e))  
 self.user\_datas = {}  
  
 **def** save\_user\_datas(self):  
 **try**:  
 with open(os.path.join(get\_appdata\_dir(), "user\_datas.pkl"), "wb") **as** f:  
 pickle.dump(self.user\_datas, f)  
 logger.info("[Config] User datas saved.")  
 **except** Exception **as** e:  
 logger.info("[Config] User datas error: {}".format(e))  
  
  
config = Config()  
  
  
**def** drag\_sensitive(config):  
 **try**:  
 **if** isinstance(config, str):  
 conf\_dict: dict = json.loads(config)  
 conf\_dict\_copy = copy.deepcopy(conf\_dict)  
 **for** key **in** conf\_dict\_copy:  
 **if** "key" **in** key **or** "secret" **in** key:  
 **if** isinstance(key, str):  
 conf\_dict\_copy[key] = conf\_dict\_copy[key][0:3] + "\*" \* 5 + conf\_dict\_copy[key][-3:]  
 **return** json.dumps(conf\_dict\_copy, indent=4)  
  
 **elif** isinstance(config, dict):  
 config\_copy = copy.deepcopy(config)  
 **for** key **in** config:  
 **if** "key" **in** key **or** "secret" **in** key:  
 **if** isinstance(key, str):  
 config\_copy[key] = config\_copy[key][0:3] + "\*" \* 5 + config\_copy[key][-3:]  
 **return** config\_copy  
 **except** Exception **as** e:  
 logger.exception(e)  
 **return** config  
 **return** config  
  
  
**def** load\_config():  
 **global** config  
 config\_path = "./config.json"  
 **if** **not** os.path.exists(config\_path):  
 logger.info("配置文件不存在，将使用config-template.json模板")  
 config\_path = "./config-template.json"  
  
 config\_str = read\_file(config\_path)  
 logger.debug("[INIT] config str: {}".format(drag\_sensitive(config\_str)))  
  
 config = Config(json.loads(config\_str))  
  
 **for** name, value **in** os.environ.items():  
 name = name.lower()  
 **if** name **in** available\_setting:  
 logger.info("[INIT] override config by environ args: {}={}".format(name, value))  
 **try**:  
 config[name] = eval(value)  
 **except**:  
 **if** value == "false":  
 config[name] = False  
 **elif** value == "true":  
 config[name] = True  
 **else**:  
 config[name] = value  
  
 **if** config.get("debug", False):  
 logger.setLevel(logging.DEBUG)  
 logger.debug("[INIT] set log level to DEBUG")  
  
 logger.info("[INIT] load config: {}".format(drag\_sensitive(config)))  
  
 config.load\_user\_datas()  
  
  
**def** get\_root():  
 **return** os.path.dirname(os.path.abspath(\_\_file\_\_))  
  
  
**def** read\_file(path):  
 with open(path, mode="r", encoding="utf-8") **as** f:  
 **return** f.read()  
  
  
**def** conf():  
 **return** config  
  
  
**def** get\_appdata\_dir():  
 data\_path = os.path.join(get\_root(), conf().get("appdata\_dir", ""))  
 **if** **not** os.path.exists(data\_path):  
 logger.info("[INIT] data path not exists, create it: {}".format(data\_path))  
 os.makedirs(data\_path)  
 **return** data\_path  
  
  
**def** subscribe\_msg():  
 trigger\_prefix = conf().get("single\_chat\_prefix", [""])[0]  
 msg = conf().get("subscribe\_msg", "")  
 **return** msg.format(trigger\_prefix=trigger\_prefix)  
  
  
  
plugin\_config = {}  
  
  
**def** write\_plugin\_config(pconf: dict):  
  
 **global** plugin\_config  
 **for** k **in** pconf:  
 plugin\_config[k.lower()] = pconf[k]  
  
  
**def** pconf(plugin\_name: str) -> dict:  
 **return** plugin\_config.get(plugin\_name.lower())  
  
  
  
global\_config = {  
 "admin\_users": []  
}

**def** sigterm\_handler\_wrap(\_signo):  
 old\_handler = signal.getsignal(\_signo)  
  
 **def** func(\_signo, \_stack\_frame):  
 logger.info("signal {} received, exiting...".format(\_signo))  
 conf().save\_user\_datas()  
 **if** callable(old\_handler): *# check old\_handler*  
 **return** old\_handler(\_signo, \_stack\_frame)  
 sys.exit(0)  
  
 signal.signal(\_signo, func)  
  
  
**def** start\_channel(channel\_name: str):  
 channel = channel\_factory.create\_channel(channel\_name)  
 **if** channel\_name **in** ["wx", "wxy", "terminal", "wechatmp", "wechatmp\_service", "wechatcom\_app", "wework",  
 const.FEISHU, const.DINGTALK]:  
 PluginManager().load\_plugins()  
  
 **if** conf().get("use\_linkai"):  
 **try**:  
 **from** common **import** linkai\_client  
 threading.Thread(target=linkai\_client.start, args=(channel,)).start()  
 **except** Exception **as** e:  
 **pass**  
 channel.startup()  
  
  
**def** run():  
 **try**:  
 load\_config()  
 sigterm\_handler\_wrap(signal.SIGINT)  
 sigterm\_handler\_wrap(signal.SIGTERM)  
 channel\_name = conf().get("channel\_type", "wx")  
  
 **if** "--cmd" **in** sys.argv:  
 channel\_name =

**from** bridge.context **import** Context  
**from** bridge.reply **import** Reply  
  
  
**class** Bot(object):  
 **def** reply(self, query, context: Context = None) -> Reply:  
 **raise** NotImplementedError  
**from** common **import** const  
**from** common.log **import** logger  
  
 **raise** RuntimeError

**from** common.expired\_dict **import** ExpiredDict  
**from** common.log **import** logger  
**from** config **import** conf  
  
  
**class** Session(object):  
 **def** \_\_init\_\_(self, session\_id, system\_prompt=None):  
 self.session\_id = session\_id  
 self.messages = []  
 **if** system\_prompt **is** None:  
 self.system\_prompt = conf().get("character\_desc", "")  
 **else**:  
 self.system\_prompt = system\_prompt  
  
 **def** reset(self):  
 system\_item = {"role": "system", "content": self.system\_prompt}  
 self.messages = [system\_item]  
  
 **def** set\_system\_prompt(self, system\_prompt):  
 self.system\_prompt = system\_prompt  
 self.reset()  
  
 **def** add\_query(self, query):  
 user\_item = {"role": "user", "content": query}  
 self.messages.append(user\_item)  
  
 **def** add\_reply(self, reply):  
 assistant\_item = {"role": "assistant", "content": reply}  
 self.messages.append(assistant\_item)  
  
 **if** session\_id **not** **in** self.sessions:  
 self.sessions[session\_id] = self.sessioncls(session\_id, system\_prompt, \*\*self.session\_args)  
 **elif** system\_prompt **is** **not** None: *# 如果有新的system\_prompt，更新并重置session*  
 self.sessions[session\_id].set\_system\_prompt(system\_prompt)  
 session = self.sessions[session\_id]  
 **return** session  
  
 **def** session\_query(self, query, session\_id):  
 session = self.build\_session(session\_id)  
 session.add\_query(query)  
 **try**:  
 max\_tokens = conf().get("conversation\_max\_tokens", 1000)  
 total\_tokens = session.discard\_exceeding(max\_tokens, None)  
 logger.debug("prompt tokens used={}".format(total\_tokens))  
 **except** Exception **as** e:  
 logger.warning("Exception when counting tokens precisely for prompt: {}".format(str(e)))  
 **return** session  
  
 **def** session\_reply(self, reply, session\_id, total\_tokens=None):  
 session = self.build\_session(session\_id)  
 session.add\_reply(reply)  
 **try**:  
 max\_tokens = conf().get("conversation\_max\_tokens", 1000)  
 tokens\_cnt = session.discard\_exceeding(max\_tokens, total\_tokens)  
 logger.debug("raw total\_tokens={}, savesession tokens={}".format(total\_tokens, tokens\_cnt))  
 **except** Exception **as** e:  
 logger.warning("Exception when counting tokens precisely for session: {}".format(str(e)))  
 **return** session  
  
 **def** clear\_session(self, session\_id):  
 **if** session\_id **in** self.sessions:  
 **del** self.sessions[session\_id]  
  
 **def** clear\_all\_session(self):  
 self.sessions.clear()  
user\_session = dict()  
  
 **def** reply(self, query, context=None):  
 **if** context **and** context.type:  
 **if** context.type == ContextType.TEXT:  
 logger.info("[API] query={}".format(query))  
 session\_id = context["session\_id"]  
 reply = None  
 **if** query == "#清空记录":  
 self.sessions.clear\_session(session\_id)  
 reply = Reply(ReplyType.INFO, "记录已清除")  
 **elif** query == "#清除所有":  
 self.sessions.clear\_all\_session()  
 reply = Reply(ReplyType.INFO, "全部记忆已清除")  
 **else**:  
 session = self.sessions.session\_query(query, session\_id)  
 result = self.reply\_text(session)  
 logger.info(result)  
 total\_tokens, completion\_tokens, reply\_content = (  
 result["total\_tokens"],  
 result["completion\_tokens"],  
 result["content"],  
 )  
 logger.debug(  
 "[CLAUDE\_API] new\_query={}, session\_id={}, reply\_cont={}, completion\_tokens={}".format(str(session), session\_id, reply\_content, completion\_tokens)  
 )  
  
 **if** total\_tokens == 0:  
 reply = Reply(ReplyType.ERROR, reply\_content)  
 **else**:  
 self.sessions.session\_reply(reply\_content, session\_id, total\_tokens)  
 reply = Reply(ReplyType.TEXT, reply\_content)  
 **return** reply  
 **elif** context.type == ContextType.IMAGE\_CREATE:  
 ok, retstring = self.create\_img(query, 0)  
 reply = None  
 **if** ok:  
 reply = Reply(ReplyType.IMAGE\_URL, retstring)  
 **else**:  
 reply = Reply(ReplyType.ERROR, retstring)  
 **return** reply  
  
 **def** reply\_text(self, session: ChatGPTSession, retry\_count=0):  
 **try**:  
 actual\_model = self.\_model\_mapping(conf().get("model"))  
 response = self.claudeClient.messages.create(  
 model=actual\_model,  
 max\_tokens=1024,  
 messages=GoogleGeminiBot.filter\_messages(session.messages)  
 )  
 res\_content = response.content[0].text.strip().replace("<|endoftext|>", "")  
 total\_tokens = response.usage.input\_tokens+response.usage.output\_tokens  
 completion\_tokens = response.usage.output\_tokens  
 logger.info("[CLAUDE\_API] reply={}".format(res\_content))  
 **return** {  
 "total\_tokens": total\_tokens,  
 "completion\_tokens": completion\_tokens,  
 "content": res\_content,  
 }  
 **except** Exception **as** e:  
 need\_retry = retry\_count < 2  
 result = {"completion\_tokens": 0, "content": "我现在有点累了，等会再来吧"}  
 **if** isinstance(e, openai.error.RateLimitError):  
 logger.warn("[CLAUDE\_API] RateLimitError: {}".format(e))  
 result["content"] = "提问太快啦，请休息一下再问我吧"  
 **if** need\_retry:  
 time.sleep(20)  
 **elif** isinstance(e, openai.error.Timeout):  
 logger.warn("[CLAUDE\_API] Timeout: {}".format(e))  
 result["content"] = "我没有收到你的消息"  
 **if** need\_retry:  
 time.sleep(5)  
 **elif** isinstance(e, openai.error.APIConnectionError):  
 logger.warn("[CLAUDE\_API] APIConnectionError: {}".format(e))  
 need\_retry = False  
 result["content"] = "我连接不到你的网络"  
 **else**:  
 logger.warn("[CLAUDE\_API] Exception: {}".format(e))  
 need\_retry = False  
 self.sessions.clear\_session(session.session\_id)  
  
 **if** need\_retry:  
 logger.warn("[CLAUDE\_API] 第{}次重试".format(retry\_count + 1))  
 **return** self.reply\_text(session, retry\_count + 1)  
 **else**:  
 **return** result  
  
 **def** \_model\_mapping(self, model) -> str:  
 **if** model == "claude-3-opus":  
 **return** "claude-3-opus-20240229"  
 **elif** model == "claude-3-sonnet":  
 **return** "claude-3-sonnet-20240229"  
 **elif** model == "claude-3-haiku":  
 **return** "claude-3-haiku-20240307"  
 **return** model  
  
**import** json  
**import** time  
**from** typing **import** List, Tuple  
  
**import** openai  
**import** openai.error  
**import** broadscope\_bailian  
**from** broadscope\_bailian **import** ChatQaMessage  
  
**from** bot.bot **import** Bot  
**from** bot.ali.ali\_qwen\_session **import** AliQwenSession  
**from** bot.session\_manager **import** SessionManager  
**from** bridge.context **import** ContextType  
**from** bridge.reply **import** Reply, ReplyType  
**from** common.log **import** logger  
**from** common **import** const  
**from** config **import** conf, load\_config  
  
**class** AliQwenBot(Bot):  
 **def** \_\_init\_\_(self):  
 super().\_\_init\_\_()  
 self.api\_key\_expired\_time = self.set\_api\_key()  
 self.sessions = SessionManager(AliQwenSession, model=conf().get("model", const.QWEN))  
  
 **def** api\_key\_client(self):  
 **return** broadscope\_bailian.AccessTokenClient(access\_key\_id=self.access\_key\_id(), access\_key\_secret=self.access\_key\_secret())  
  
 **def** access\_key\_id(self):  
 **return** conf().get("qwen\_access\_key\_id")  
  
 **def** access\_key\_secret(self):  
 **return** conf().get("qwen\_access\_key\_secret")  
  
 **def** agent\_key(self):  
 **return** conf().get("qwen\_agent\_key")  
  
 **def** app\_id(self):  
 **return** conf().get("qwen\_app\_id")  
  
 **def** node\_id(self):  
 **return** conf().get("qwen\_node\_id", "")  
  
 **def** temperature(self):  
 **return** conf().get("temperature", 0.2 )  
  
 **def** top\_p(self):  
 **return** conf().get("top\_p", 1)  
  
 **def** reply(self, query, context=None):  
  
 **if** context.type == ContextType.TEXT:  
 logger.info("[QWEN] query={}".format(query))  
  
 session\_id = context["session\_id"]  
 reply = None  
 clear\_memory\_commands = conf().get("clear\_memory\_commands", ["#清除记忆"])  
 **if** query **in** clear\_memory\_commands:  
 self.sessions.clear\_session(session\_id)  
 reply = Reply(ReplyType.INFO, "记忆已清除")  
 **elif** query == "#清除所有":  
 self.sessions.clear\_all\_session()  
 reply = Reply(ReplyType.INFO, "所有人记忆已清除")  
 **elif** query == "#更新配置":  
 load\_config()  
 reply = Reply(ReplyType.INFO, "配置已更新")  
 **elif** reply\_content["completion\_tokens"] > 0:  
 self.sessions.session\_reply(reply\_content["content"], session\_id, reply\_content["total\_tokens"])  
 reply = Reply(ReplyType.TEXT, reply\_content["content"])  
   
 **if** need\_retry:  
 logger.warn("[QWEN] 第{}次重试".format(retry\_count + 1))  
 **return** self.reply\_text(session, retry\_count + 1)  
 **else**:  
 **return** result  
  
 **def** set\_api\_key(self):  
 api\_key, expired\_time = self.api\_key\_client().create\_token(agent\_key=self.agent\_key())  
 broadscope\_bailian.api\_key = api\_key  
 **return** expired\_time  
  
 **def** update\_api\_key\_if\_expired(self):  
 **if** time.time() > self.api\_key\_expired\_time:  
 self.api\_key\_expired\_time = self.set\_api\_key()  
  
 **def** convert\_messages\_format(self, messages) -> Tuple[str, List[ChatQaMessage]]:  
 history = []  
 user\_content = ''  
 assistant\_content = ''  
 system\_content = ''  
 **for** message **in** messages:  
 role = message.get('role')  
 **if** role == 'user':  
 user\_content += message.get('content')  
 **elif** role == 'assistant':  
 assistant\_content = message.get('content')  
 history.append(ChatQaMessage(user\_content, assistant\_content))  
 user\_content = ''  
 assistant\_content = ''  
 **elif** role =='system':  
 system\_content += message.get('content')  
 **if** user\_content == '':  
 **raise** Exception('no user message')  
 **if** system\_content != '':  
 system\_qa = ChatQaMessage(system\_content, '好的，我会严格按照你的设定回答问题')  
 history.insert(0, system\_qa)  
 logger.debug("[QWEN] converted qa messages: {}".format([item.to\_dict() **for** item **in** history]))  
 logger.debug("[QWEN] user content as prompt: {}".format(user\_content))  
 **return** user\_content, history  
  
 **def** get\_completion\_content(self, response, node\_id):  
 **if** **not** response['Success']:  
 **return** f"[ERROR]\n{response['Code']}:{response['Message']}"  
 text = response['Data']['Text']  
 **if** node\_id == '':  
 **return** text  
  
 text\_dict = json.loads(text)  
 completion\_content = text\_dict['finalResult'][node\_id]['response']['text']  
 **return** completion\_content  
  
 **def** calc\_tokens(self, messages, completion\_content):  
 completion\_tokens = len(completion\_content)  
 prompt\_tokens = 0  
 **for** message **in** messages:  
 prompt\_tokens += len(message["content"])  
 **return** completion\_tokens, prompt\_tokens + completion\_tokens

**from** bot.session\_manager **import** Session  
**from** common.log **import** logger  
**class** AliQwenSession(Session):  
 **def** \_\_init\_\_(self, session\_id, system\_prompt=None, model="qianwen"):  
 super().\_\_init\_\_(session\_id, system\_prompt)  
 self.model = model  
 self.reset()  
  
 **def** discard\_exceeding(self, max\_tokens, cur\_tokens=None):  
 precise = True  
 **try**:  
 cur\_tokens = self.calc\_tokens()  
 **except** Exception **as** e:  
 precise = False  
 **if** cur\_tokens **is** None:  
 **raise** e  
 logger.debug("Exception when counting tokens precisely for query: {}".format(e))  
 **while** cur\_tokens > max\_tokens:  
 **if** len(self.messages) > 2:  
 self.messages.pop(1)  
 **elif** len(self.messages) == 2 **and** self.messages[1]["role"] == "assistant":  
 self.messages.pop(1)  
 **if** precise:  
 cur\_tokens = self.calc\_tokens()  
 **else**:  
 cur\_tokens = cur\_tokens - max\_tokens  
 **break**  
 **elif** len(self.messages) == 2 **and** self.messages[1]["role"] == "user":  
 logger.warn("user message exceed max\_tokens. total\_tokens={}".format(cur\_tokens))  
 **break**  
 **else**:  
 logger.debug("max\_tokens={}, total\_tokens={}, len(messages)={}".format(max\_tokens, cur\_tokens, len(self.messages)))  
 **break**  
 **if** precise:  
 cur\_tokens = self.calc\_tokens()  
 **else**:  
 cur\_tokens = cur\_tokens - max\_tokens  
 **return** cur\_tokens  
  
 **def** calc\_tokens(self):  
 **return** num\_tokens\_from\_messages(self.messages, self.model)  
  
**def** num\_tokens\_from\_messages(messages, model):  
 tokens = 0  
 **for** msg **in** messages:  
 tokens += len(msg["content"])  
 **return** tokens

**import** os  
**import** json  
**from** config **import** pconf, plugin\_config, conf  
**from** common.log **import** logger  
  
  
**class** Plugin:  
 **def** \_\_init\_\_(self):  
 self.handlers = {}  
  
 **def** load\_config(self) -> dict:  
 plugin\_conf = pconf(self.name)  
 **if** **not** plugin\_conf:  
 plugin\_config\_path = os.path.join(self.path, "config.json")  
 **if** os.path.exists(plugin\_config\_path):  
 with open(plugin\_config\_path, "r", encoding="utf-8") **as** f:  
 plugin\_conf = json.load(f)  
 plugin\_config[self.name] = plugin\_conf  
 logger.debug(f"loading plugin config, plugin\_name={self.name}, conf={plugin\_conf}")  
 **return** plugin\_conf  
  
 **def** save\_config(self, config: dict):  
 **try**:  
 plugin\_config[self.name] = config  
 global\_config\_path = "./plugins/config.json"  
 **if** os.path.exists(global\_config\_path):  
 with open(global\_config\_path, "w", encoding='utf-8') **as** f:  
 json.dump(plugin\_config, f, indent=4, ensure\_ascii=False)  
 plugin\_config\_path = os.path.join(self.path, "config.json")  
 **if** os.path.exists(plugin\_config\_path):  
 with open(plugin\_config\_path, "w", encoding='utf-8') **as** f:  
 json.dump(config, f, indent=4, ensure\_ascii=False)  
  
 **except** Exception **as** e:  
 logger.warn("save plugin config failed: {}".format(e))  
  
 **def** get\_help\_text(self, \*\*kwargs):  
 **return** "暂无帮助信息"  
  
 **def** reload(self):  
 **pass**

**class** PluginManager:  
 **def** \_\_init\_\_(self):  
 self.plugins = SortedDict(**lambda** k, v: v.priority, reverse=True)  
 self.listening\_plugins = {}  
 self.instances = {}  
 self.pconf = {}  
 self.current\_plugin\_path = None  
 self.loaded = {}  
  
 **def** register(self, name: str, desire\_priority: **int** = 0, \*\*kwargs):  
 **def** wrapper(plugincls):  
 plugincls.name = name  
 plugincls.priority = desire\_priority  
 plugincls.desc = kwargs.get("desc")  
 plugincls.author = kwargs.get("author")  
 plugincls.path = self.current\_plugin\_path  
 plugincls.version = kwargs.get("version") **if** kwargs.get("version") != None **else** "1.0"  
 plugincls.namecn = kwargs.get("namecn") **if** kwargs.get("namecn") != None **else** name  
 plugincls.hidden = kwargs.get("hidden") **if** kwargs.get("hidden") != None **else** False  
 plugincls.enabled = True  
 **if** self.current\_plugin\_path == None:  
 **raise** Exception("Plugin path not set")  
 self.plugins[name.upper()] = plugincls  
 logger.info("Plugin %s\_v%s registered, path=%s" % (name, plugincls.version, plugincls.path))  
  
 **return** wrapper  
  
 **def** save\_config(self):  
 with open("./plugins/plugins.json", "w", encoding="utf-8") **as** f:  
 json.dump(self.pconf, f, indent=4, ensure\_ascii=False)  
  
 **def** load\_config(self):  
 logger.info("Loading plugins config...")  
  
 modified = False  
 **if** os.path.exists("./plugins/plugins.json"):  
 with open("./plugins/plugins.json", "r", encoding="utf-8") **as** f:  
 pconf = json.load(f)  
 pconf["plugins"] = SortedDict(**lambda** k, v: v["priority"], pconf["plugins"], reverse=True)  
 **else**:  
 modified = True  
 pconf = {"plugins": SortedDict(**lambda** k, v: v["priority"], reverse=True)}  
 self.pconf = pconf  
 **if** modified:  
 self.save\_config()  
 **return** pconf  
  
 @staticmethod  
 **def** \_load\_all\_config():  
 all\_config\_path = "./plugins/config.json"  
 **try**:  
 **if** os.path.exists(all\_config\_path):  
 *# read from all plugins config*  
 with open(all\_config\_path, "r", encoding="utf-8") **as** f:  
 all\_conf = json.load(f)  
 logger.info(f"load all config from plugins/config.json: {all\_conf}")  
  
 *# write to global config*  
 write\_plugin\_config(all\_conf)  
 **except** Exception **as** e:  
 logger.error(e)  
  
 **def** scan\_plugins(self):  
 logger.info("Scaning plugins ...")  
 plugins\_dir = "./plugins"  
 raws = [self.plugins[name] **for** name **in** self.plugins]  
 **for** plugin\_name **in** os.listdir(plugins\_dir):  
 plugin\_path = os.path.join(plugins\_dir, plugin\_name)  
 **if** os.path.isdir(plugin\_path):  
 *# 判断插件是否包含同名\_\_init\_\_.py文件*  
 main\_module\_path = os.path.join(plugin\_path, "\_\_init\_\_.py")  
 **if** os.path.isfile(main\_module\_path):  
 *# 导入插件*  
 import\_path = "plugins.{}".format(plugin\_name)  
 **try**:  
 self.current\_plugin\_path = plugin\_path  
 **if** plugin\_path **in** self.loaded:  
 **if** self.loaded[plugin\_path] == None:  
 logger.info("reload module %s" % plugin\_name)  
 self.loaded[plugin\_path] = importlib.reload(sys.modules[import\_path])  
 dependent\_module\_names = [name **for** name **in** sys.modules.keys() **if** name.startswith(import\_path + ".")]  
 **for** name **in** dependent\_module\_names:  
 logger.info("reload module %s" % name)  
 importlib.reload(sys.modules[name])  
 **else**:  
 self.loaded[plugin\_path] = importlib.import\_module(import\_path)  
 self.current\_plugin\_path = None  
 **def** activate\_plugins(self): *# 生成新开启的插件实例*  
 failed\_plugins = []  
 **for** name, plugincls **in** self.plugins.items():  
 **if** plugincls.enabled:  
 **if** name **not** **in** self.instances:  
 **try**:  
 instance = plugincls()  
 **except** Exception **as** e:  
 logger.warn("Failed to init %s, diabled. %s" % (name, e))  
 self.disable\_plugin(name)  
 failed\_plugins.append(name)  
 **continue**  
 self.instances[name] = instance  
 **for** event **in** instance.handlers:  
 **if** event **not** **in** self.listening\_plugins:  
 self.listening\_plugins[event] = []  
 self.listening\_plugins[event].append(name)  
 self.refresh\_order()  
 **return** failed\_plugins  
  
 **def** reload\_plugin(self, name: str):  
 name = name.upper()  
 **if** name **in** self.instances:  
 **for** event **in** self.listening\_plugins:  
 **if** name **in** self.listening\_plugins[event]:  
 self.listening\_plugins[event].remove(name)  
 **del** self.instances[name]  
 self.activate\_plugins()  
 **return** True  
 **return** False  
  
 **def** load\_plugins(self):  
 self.load\_config()  
 self.scan\_plugins()  
 *# 加载全量插件配置*  
 self.\_load\_all\_config()  
 pconf = self.pconf  
 logger.debug("plugins.json config={}".format(pconf))  
 **for** name, plugin **in** pconf["plugins"].items():  
 **if** name.upper() **not** **in** self.plugins:  
 logger.error("Plugin %s not found, but found in plugins.json" % name)  
 self.activate\_plugins()  
  
 **def** emit\_event(self, e\_context: EventContext, \*args, \*\*kwargs):  
 **if** e\_context.event **in** self.listening\_plugins:  
 **for** name **in** self.listening\_plugins[e\_context.event]:  
 **if** self.plugins[name].enabled **and** e\_context.action == EventAction.CONTINUE:  
 logger.debug("Plugin %s triggered by event %s" % (name, e\_context.event))  
 instance = self.instances[name]  
 instance.handlers[e\_context.event](e\_context, \*args, \*\*kwargs)  
 **if** e\_context.is\_break():  
 e\_context["breaked\_by"] = name  
 logger.debug("Plugin %s breaked event %s" % (name, e\_context.event))  
 **return** e\_context  
  
 **def** set\_plugin\_priority(self, name: str, priority: **int**):  
 name = name.upper()  
 **if** name **not** **in** self.plugins:  
 **return** False  
 **if** self.plugins[name].priority == priority:  
 **return** True  
 self.plugins[name].priority = priority  
 self.plugins.\_update\_heap(name)  
 rawname = self.plugins[name].name  
 self.pconf["plugins"][rawname]["priority"] = priority  
 self.pconf["plugins"].\_update\_heap(rawname)  
 self.save\_config()  
 self.refresh\_order()  
 **return** True  
  
 **def** enable\_plugin(self, name: str):  
 name = name.upper()  
 **if** name **not** **in** self.plugins:  
 **return** False, "插件不存在"  
 **if** **not** self.plugins[name].enabled:  
 self.plugins[name].enabled = True  
 rawname = self.plugins[name].name  
 self.pconf["plugins"][rawname]["enabled"] = True  
 self.save\_config()  
 failed\_plugins = self.activate\_plugins()  
 **if** name **in** failed\_plugins:  
 **return** False, "插件开启失败"  
 **return** True, "插件已开启"  
 **return** True, "插件已开启"  
  
 **def** disable\_plugin(self, name: str):  
 name = name.upper()  
 **if** name **not** **in** self.plugins:  
 **return** False  
 **if** self.plugins[name].enabled:  
 self.plugins[name].enabled = False  
 rawname = self.plugins[name].name  
 self.pconf["plugins"][rawname]["enabled"] = False  
 self.save\_config()  
 **return** True  
 **return** True  
  
 **def** list\_plugins(self):  
 **return** self.plugins  
  
 **def** update\_plugin(self, name: str):  
 **try**:  
 **import** common.package\_manager **as** pkgmgr  
  
 pkgmgr.check\_dulwich()  
 **except** Exception **as** e:  
 logger.error("Failed to install plugin, {}".format(e))  
 **return** False, "无法导入dulwich，更新插件失败"  
 **from** dulwich **import** porcelain  
  
 name = name.upper()  
 **if** name **not** **in** self.plugins:  
 **return** False, "插件不存在"  
 **if** name **in** [  
 "HELLO",  
 "GODCMD",  
 "ROLE",  
 "TOOL",  
 "BDUNIT",  
 "BANWORDS",  
 "FINISH",  
 "DUNGEON",  
 ]:  
 **return** False, "预置插件无法更新，请更新主程序仓库"  
 dirname = self.plugins[name].path  
 **try**:  
 porcelain.pull(dirname, "origin")  
 **if** os.path.exists(os.path.join(dirname, "requirements.txt")):  
 logger.info("detect requirements.txt，installing...")  
 pkgmgr.install\_requirements(os.path.join(dirname, "requirements.txt"))  
 **return** True, "更新插件成功，请重新运行程序"  
 **except** Exception **as** e:  
 logger.error("Failed to update plugin, {}".format(e))  
 **return** False, "更新插件失败，" + str(e)  
  
 **def** uninstall\_plugin(self, name: str):  
 name = name.upper()  
 **if** name **not** **in** self.plugins:  
 **return** False, "插件不存在"  
 **if** name **in** self.instances:  
 self.disable\_plugin(name)  
 dirname = self.plugins[name].path  
 **try**:  
 **import** shutil  
  
 shutil.rmtree(dirname)  
 rawname = self.plugins[name].name  
 **for** event **in** self.listening\_plugins:  
 **if** name **in** self.listening\_plugins[event]:  
 self.listening\_plugins[event].remove(name)  
 **del** self.plugins[name]  
 **del** self.pconf["plugins"][rawname]  
 self.loaded[dirname] = None  
 self.save\_config()  
 **return** True, "卸载插件成功"  
 **except** Exception **as** e:  
 logger.error("Failed to uninstall plugin, {}".format(e))  
 **return** False, "卸载插件失败，请手动删除文件夹完成卸载，" + str(e)

@plugins.register(  
 name="Apilot",  
 desire\_priority=88,  
 hidden=False,  
 desc="A plugin to handle specific keywords",  
 version="0.2",  
 author="vision",  
)  
**class** Apilot(Plugin):  
 **def** \_\_init\_\_(self):  
 super().\_\_init\_\_()  
 **try**:  
 self.conf = super().load\_config()  
 self.condition\_2\_and\_3\_cities = None *# 天气查询，存储重复城市信息，Initially set to None*  
 **if** **not** self.conf:  
 logger.warn("[Apilot] inited but alapi\_token not found in config")  
 self.alapi\_token = None *# Setting a default value for alapi\_token*  
 self.morning\_news\_text\_enabled = False  
 **else**:  
 logger.info("[Apilot] inited and alapi\_token loaded successfully")  
 self.alapi\_token = self.conf["alapi\_token"]  
 **try**:  
 self.morning\_news\_text\_enabled = self.conf["morning\_news\_text\_enabled"]  
 **except**:  
 self.morning\_news\_text\_enabled = False  
 self.handlers[Event.ON\_HANDLE\_CONTEXT] = self.on\_handle\_context  
 **except** Exception **as** e:  
 **raise** self.handle\_error(e, "[Apiot] init failed, ignore ")  
  
 **def** on\_handle\_context(self, e\_context: EventContext):  
 **if** e\_context["context"].type **not** **in** [  
 ContextType.TEXT  
 ]:  
 **return**  
 content = e\_context["context"].content.strip()  
 logger.debug("[Apilot] on\_handle\_context. content: %s" % content)  
  
 **if** content == "早报":  
 news = self.get\_morning\_news(self.alapi\_token, self.morning\_news\_text\_enabled)  
 reply\_type = ReplyType.IMAGE\_URL **if** self.is\_valid\_url(news) **else** ReplyType.TEXT  
 reply = self.create\_reply(reply\_type, news)  
 e\_context["reply"] = reply  
 e\_context.action = EventAction.BREAK\_PASS *# 事件结束，并跳过处理context的默认逻辑*  
 **return**  
 **if** content == "摸鱼":  
 moyu = self.get\_moyu\_calendar()  
 reply\_type = ReplyType.IMAGE\_URL **if** self.is\_valid\_url(moyu) **else** ReplyType.TEXT  
 reply = self.create\_reply(reply\_type, moyu)  
 e\_context["reply"] = reply  
 e\_context.action = EventAction.BREAK\_PASS *# 事件结束，并跳过处理context的默认逻辑*  
 **return**  
  
 **if** content == "摸鱼视频":  
 moyu = self.get\_moyu\_calendar\_video()  
 reply\_type = ReplyType.VIDEO\_URL **if** self.is\_valid\_url(moyu) **else** ReplyType.TEXT  
 reply = self.create\_reply(reply\_type, moyu)  
 e\_context["reply"] = reply  
 e\_context.action = EventAction.BREAK\_PASS *# 事件结束，并跳过处理context的默认逻辑*  
 **return**  
  
 **if** content == "八卦":  
 bagua = self.get\_mx\_bagua()  
 reply\_type = ReplyType.IMAGE\_URL **if** self.is\_valid\_url(bagua) **else** ReplyType.TEXT  
 reply = self.create\_reply(reply\_type, bagua)  
 e\_context["reply"] = reply  
 e\_context.action = EventAction.BREAK\_PASS *# 事件结束，并跳过处理context的默认逻辑*  
 **return**  
  
 **if** content.startswith("快递"):  
 *# Extract the part after "快递"*  
 tracking\_number = content[2:].strip()  
  
 tracking\_number = tracking\_number.replace('：', ':') *# 替换可能出现的中文符号*  
 *# Check if alapi\_token is available before calling the function*  
 **if** **not** self.alapi\_token:  
 self.handle\_error("alapi\_token not configured", "快递请求失败")  
 reply = self.create\_reply(ReplyType.TEXT, "请先配置alapi的token")  
 **else**:  
 *# Check if the tracking\_number starts with "SF" for Shunfeng (顺丰) Express*  
 **if** tracking\_number.startswith("SF"):  
 *# Check if the user has included the last four digits of the phone number*  
 **if** ':' **not** **in** tracking\_number:  
 reply = self.create\_reply(ReplyType.TEXT, "顺丰快递需要补充寄/收件人手机号后四位，格式：SF12345:0000")  
 e\_context["reply"] = reply  
 e\_context.action = EventAction.BREAK\_PASS *# 事件结束，并跳过处理context的默认逻辑*  
 **return** *# End the function here*  
  
 *# Call query\_express\_info function with the extracted tracking\_number and the alapi\_token from config*  
 content = self.query\_express\_info(self.alapi\_token, tracking\_number)  
 reply = self.create\_reply(ReplyType.TEXT, content)  
 e\_context["reply"] = reply  
 e\_context.action = EventAction.BREAK\_PASS *# 事件结束，并跳过处理context的默认逻辑*  
 **return**  
  
 horoscope\_match = re.match(r'^([\u4e00-\u9fa5]{2}座)$', content)  
 **if** horoscope\_match:  
 **if** content **in** ZODIAC\_MAPPING:  
 zodiac\_english = ZODIAC\_MAPPING[content]  
 content = self.get\_horoscope(self.alapi\_token, zodiac\_english)  
 reply = self.create\_reply(ReplyType.TEXT, content)  
 **else**:  
 reply = self.create\_reply(ReplyType.TEXT, "请重新输入星座名称")  
 e\_context["reply"] = reply  
 e\_context.action = EventAction.BREAK\_PASS *# 事件结束，并跳过处理context的默认逻辑*  
 **return**  
  
 hot\_trend\_match = re.search(r'(.{1,6})热榜$', content)  
 **if** hot\_trend\_match:  
 hot\_trends\_type = hot\_trend\_match.group(1).strip() *# 提取匹配的组并去掉可能的空格*  
 content = self.get\_hot\_trends(hot\_trends\_type)  
 reply = self.create\_reply(ReplyType.TEXT, content)  
 e\_context["reply"] = reply  
 e\_context.action = EventAction.BREAK\_PASS *# 事件结束，并跳过处理context的默认逻辑*  
 **return**  
  
  
 *# 天气查询*  
 weather\_match = re.match(r'^(?:(.{2,7}?)(?:市|县|区|镇)?|(\d{7,9}))(?:的)?天气$', content)  
 **if** weather\_match:  
 *# 如果匹配成功，提取第一个捕获组*  
 city\_or\_id = weather\_match.group(1) **or** weather\_match.group(2)  
 **if** **not** self.alapi\_token:  
 self.handle\_error("alapi\_token not configured", "天气请求失败")  
 reply = self.create\_reply(ReplyType.TEXT, "请先配置alapi的token")  
 **else**:  
 content = self.get\_weather(self.alapi\_token, city\_or\_id, content)  
 reply = self.create\_reply(ReplyType.TEXT, content)  
 e\_context["reply"] = reply  
 e\_context.action = EventAction.BREAK\_PASS *# 事件结束，并跳过处理context的默认逻辑*  
 **return**  
  
 **def** get\_help\_text(self, verbose=False, \*\*kwargs):  
 short\_help\_text = " 发送特定指令以获取早报、热榜、查询天气、星座运势、快递信息等！"  
  
 **if** **not** verbose:  
 **return** short\_help\_text  
  
 help\_text = "📚 发送关键词获取特定信息！\n"  
  
 *# 娱乐和信息类*  
 help\_text += "\n🎉 娱乐与资讯：\n"  
 help\_text += " 🌅 早报: 发送“早报”获取早报。\n"  
 help\_text += " 🐟 摸鱼: 发送“摸鱼”获取摸鱼人日历。\n"  
 help\_text += " 🔥 热榜: 发送“xx热榜”查看支持的热榜。\n"  
 help\_text += " 🔥 八卦: 发送“八卦”获取明星八卦。\n"  
  
 *# 查询类*  
 help\_text += "\n🔍 查询工具：\n"  
 help\_text += " 🌦️ 天气: 发送“城市+天气”查天气，如“北京天气”。\n"  
 help\_text += " 📦 快递: 发送“快递+单号”查询快递状态。如“快递112345655”\n"  
 help\_text += " 🌌 星座: 发送星座名称查看今日运势，如“白羊座”。\n"  
  
 **return** help\_text  
  
 **def** get\_morning\_news(self, alapi\_token, morning\_news\_text\_enabled):  
 **if** **not** alapi\_token:  
 url = BASE\_URL\_VVHAN + "60s?type=json"  
 payload = "format=json"  
 headers = {'Content-Type': "application/x-www-form-urlencoded"}  
 **try**:  
 morning\_news\_info = self.make\_request(url, method="POST", headers=headers, data=payload)  
 **if** isinstance(morning\_news\_info, dict) **and** morning\_news\_info['success']:  
 **if** morning\_news\_text\_enabled:  
 *# 提取并格式化新闻*  
 news\_list = ["{}. {}".format(idx, news) **for** idx, news **in** enumerate(morning\_news\_info["data"][:-1], 1)]  
 formatted\_news = f"☕ {morning\_news\_info['data']['date']} 今日早报\n"  
 formatted\_news = formatted\_news + "\n".join(news\_list)  
 weiyu = morning\_news\_info["data"][-1].strip()  
 **return** f"{formatted\_news}\n\n{weiyu}\n\n 图片url：{morning\_news\_info['imgUrl']}"  
 **else**:  
 **return** morning\_news\_info['imgUrl']  
 **else**:  
 **return** self.handle\_error(morning\_news\_info, '早报信息获取失败，可配置"alapi token"切换至 Alapi 服务，或者稍后再试')  
 **except** Exception **as** e:  
 **return** self.handle\_error(e, "出错啦，稍后再试")  
 **else**:  
 url = BASE\_URL\_ALAPI + "zaobao"  
 data = {  
 "token": alapi\_token,  
 "format": "json"  
 }  
 headers = {'Content-Type': "application/x-www-form-urlencoded"}  
 **try**:  
 morning\_news\_info = self.make\_request(url, method="POST", headers=headers, data=data)  
 **if** isinstance(morning\_news\_info, dict) **and** morning\_news\_info.get('code') == 200:  
 img\_url = morning\_news\_info['data']['image']  
 **if** morning\_news\_text\_enabled:  
 news\_list = morning\_news\_info['data']['news']  
 weiyu = morning\_news\_info['data']['weiyu']  
  
 *# 整理新闻为有序列表*  
 formatted\_news = f"☕ {morning\_news\_info['data']['date']} 今日早报\n"  
 formatted\_news = formatted\_news + "\n".join(news\_list)  
 *# 组合新闻和微语*  
 **return** f"{formatted\_news}\n\n{weiyu}\n\n 图片url：{img\_url}"  
 **else**:  
 **return** img\_url  
 **else**:  
 **return** self.handle\_error(morning\_news\_info, "早报获取失败，请检查 token 是否有误")  
 **except** Exception **as** e:  
 **return** self.handle\_error(e, "早报获取失败")  
  
 **def** get\_moyu\_calendar(self):  
 url = BASE\_URL\_VVHAN + "moyu?type=json"  
 payload = "format=json"  
 headers = {'Content-Type': "application/x-www-form-urlencoded"}  
 moyu\_calendar\_info = self.make\_request(url, method="POST", headers=headers, data=payload)  
 *# 验证请求是否成功*  
 **if** isinstance(moyu\_calendar\_info, dict) **and** moyu\_calendar\_info['success']:  
 **return** moyu\_calendar\_info['url']  
 **else**:  
 url = "https://dayu.qqsuu.cn/moyuribao/apis.php?type=json"  
 payload = "format=json"  
 headers = {'Content-Type': "application/x-www-form-urlencoded"}  
 moyu\_calendar\_info = self.make\_request(url, method="POST", headers=headers, data=payload)  
 **if** isinstance(moyu\_calendar\_info, dict) **and** moyu\_calendar\_info['code'] == 200:  
 moyu\_pic\_url = moyu\_calendar\_info['data']  
 **if** self.is\_valid\_image\_url(moyu\_pic\_url):  
 **return** moyu\_pic\_url  
 **else**:  
 **return** "周末无需摸鱼，愉快玩耍吧"  
 **else**:  
 **return** "暂无可用“摸鱼”服务，认真上班"  
  
 **def** get\_moyu\_calendar\_video(self):  
 url = "https://dayu.qqsuu.cn/moyuribaoshipin/apis.php?type=json"  
 payload = "format=json"  
 headers = {'Content-Type': "application/x-www-form-urlencoded"}  
 moyu\_calendar\_info = self.make\_request(url, method="POST", headers=headers, data=payload)  
 *# 验证请求是否成功*  
 **if** isinstance(moyu\_calendar\_info, dict) **and** moyu\_calendar\_info['code'] == 200:  
 moyu\_video\_url = moyu\_calendar\_info['data']  
 **if** self.is\_valid\_image\_url(moyu\_video\_url):  
 **return** moyu\_video\_url  
 **else**:  
 **return** "视频版没了，看看文字版吧"  
  
 **def** get\_horoscope(self, alapi\_token, astro\_sign: str, time\_period: str = "today"):  
 **if** **not** alapi\_token:  
 url = BASE\_URL\_VVHAN + "horoscope"  
 params = {  
 'type': astro\_sign,  
 'time': time\_period  
 }  
 **try**:  
 horoscope\_data = self.make\_request(url, "GET", params=params)  
 **if** isinstance(horoscope\_data, dict) **and** horoscope\_data['success']:  
 data = horoscope\_data['data']  
  
 result = (  
 f"{data['title']} ({data['time']}):\n\n"  
 f"💡【每日建议】\n宜：{data['todo']['yi']}\n忌：{data['todo']['ji']}\n\n"  
 f"📊【运势指数】\n"  
 f"总运势：{data['index']['all']}\n"  
 f"爱情：{data['index']['love']}\n"  
 f"工作：{data['index']['work']}\n"  
 f"财运：{data['index']['money']}\n"  
 f"健康：{data['index']['health']}\n\n"  
 f"🍀【幸运提示】\n数字：{data['luckynumber']}\n"  
 f"颜色：{data['luckycolor']}\n"  
 f"星座：{data['luckyconstellation']}\n\n"  
 f"✍【简评】\n{data['shortcomment']}\n\n"  
 f"📜【详细运势】\n"  
 f"总运：{data['fortunetext']['all']}\n"  
 f"爱情：{data['fortunetext']['love']}\n"  
 f"工作：{data['fortunetext']['work']}\n"  
 f"财运：{data['fortunetext']['money']}\n"  
 f"健康：{data['fortunetext']['health']}\n"  
 )  
  
 **return** result  
  
 **else**:  
 **return** self.handle\_error(horoscope\_data, '星座信息获取失败，可配置"alapi token"切换至 Alapi 服务，或者稍后再试')  
  
 **except** Exception **as** e:  
 **return** self.handle\_error(e, "出错啦，稍后再试")  
 **else**:  
 *# 使用 ALAPI 的 URL 和提供的 token*  
 url = BASE\_URL\_ALAPI + "star"  
 payload = f"token={alapi\_token}&star={astro\_sign}"  
 headers = {'Content-Type': "application/x-www-form-urlencoded"}  
 **try**:  
 horoscope\_data = self.make\_request(url, method="POST", headers=headers, data=payload)  
 **if** isinstance(horoscope\_data, dict) **and** horoscope\_data.get('code') == 200:  
 data = horoscope\_data['data']['day']  
  
 *# 格式化并返回 ALAPI 提供的星座信息*  
 result = (  
 f"📅 日期：{data['date']}\n\n"  
 f"💡【每日建议】\n宜：{data['yi']}\n忌：{data['ji']}\n\n"  
 f"📊【运势指数】\n"  
 f"总运势：{data['all']}\n"  
 f"爱情：{data['love']}\n"  
 f"工作：{data['work']}\n"  
 f"财运：{data['money']}\n"  
 f"健康：{data['health']}\n\n"  
 f"🔔【提醒】：{data['notice']}\n\n"  
 f"🍀【幸运提示】\n数字：{data['lucky\_number']}\n"  
 f"颜色：{data['lucky\_color']}\n"  
 f"星座：{data['lucky\_star']}\n\n"  
 f"✍【简评】\n总运：{data['all\_text']}\n"  
 f"爱情：{data['love\_text']}\n"  
 f"工作：{data['work\_text']}\n"  
 f"财运：{data['money\_text']}\n"  
 f"健康：{data['health\_text']}\n"  
 )  
 **return** result  
 **else**:  
 **return** self.handle\_error(horoscope\_data, "星座获取信息获取失败，请检查 token 是否有误")  
 **except** Exception **as** e:  
 **return** self.handle\_error(e, "出错啦，稍后再试")  
  
 **def** get\_hot\_trends(self, hot\_trends\_type):  
 *# 查找映射字典以获取API参数*  
 hot\_trends\_type\_en = hot\_trend\_types.get(hot\_trends\_type, None)  
 **if** hot\_trends\_type\_en **is** **not** None:  
 url = BASE\_URL\_VVHAN + "hotlist?type=" + hot\_trends\_type\_en  
 **try**:  
 data = self.make\_request(url, "GET")  
 **if** isinstance(data, dict) **and** data['success'] == True:  
 output = []  
 topics = data['data']  
 output.append(f'更新时间：{data["update\_time"]}\n')  
 **for** i, topic **in** enumerate(topics[:15], 1):  
 hot = topic.get('hot', '无热度参数, 0')  
 formatted\_str = f"{i}. {topic['title']} ({hot} 浏览)\nURL: {topic['url']}\n"  
 output.append(formatted\_str)  
 **return** "\n".join(output)  
 **else**:  
 **return** self.handle\_error(data, "热榜获取失败，请稍后再试")  
 **except** Exception **as** e:  
 **return** self.handle\_error(e, "出错啦，稍后再试")  
 **else**:  
 supported\_types = "/".join(hot\_trend\_types.keys())  
 final\_output = (  
 f"👉 已支持的类型有：\n\n {supported\_types}\n"  
 f"\n📝 请按照以下格式发送：\n 类型+热榜 例如：微博热榜"  
 )  
 **return** final\_output  
  
 **def** query\_express\_info(self, alapi\_token, tracking\_number, com="", order="asc"):  
 url = BASE\_URL\_ALAPI + "kd"  
 payload = f"token={alapi\_token}&number={tracking\_number}&com={com}&order={order}"  
 headers = {'Content-Type': "application/x-www-form-urlencoded"}  
  
 **try**:  
 response\_json = self.make\_request(url, method="POST", headers=headers, data=payload)  
  
 **if** **not** isinstance(response\_json, dict) **or** response\_json **is** None:  
 **return** f"查询失败：api响应为空"  
 code = response\_json.get("code", None)  
 **if** code != 200:  
 msg = response\_json.get("msg", "未知错误")  
 self.handle\_error(msg, f"错误码{code}")  
 **return** f"查询失败，{msg}"  
 data = response\_json.get("data", None)  
 formatted\_result = [  
 f"快递编号：{data.get('nu')}",  
 f"快递公司：{data.get('com')}",  
 f"状态：{data.get('status\_desc')}",  
 "状态信息："  
 ]  
 **for** info **in** data.get("info"):  
 time\_str = info.get('time')[5:-3]  
 formatted\_result.append(f"{time\_str} - {info.get('status\_desc')}\n {info.get('content')}")  
  
 **return** "\n".join(formatted\_result)  
  
 **except** Exception **as** e:  
 **return** self.handle\_error(e, "快递查询失败")  
  
 **def** get\_weather(self, alapi\_token, city\_or\_id: str, content):  
 url = BASE\_URL\_ALAPI + 'tianqi'  
 *# 判断使用id还是city请求api*  
 **if** city\_or\_id.isnumeric(): *# 判断是否为纯数字，也即是否为 city\_id*  
 params = {  
 'city\_id': city\_or\_id,  
 'token': f'{alapi\_token}'  
 }  
 **else**:  
 city\_info = self.check\_multiple\_city\_ids(city\_or\_id)  
 **if** city\_info:  
 data = city\_info['data']  
 formatted\_city\_info = "\n".join(  
 [f"{idx + 1}) {entry['province']}--{entry['leader']}, ID: {entry['city\_id']}"  
 **for** idx, entry **in** enumerate(data)]  
 )  
 **return** f"查询 <{city\_or\_id}> 具有多条数据：\n{formatted\_city\_info}\n请使用id查询，发送“id天气”"  
  
 params = {  
 'city': city\_or\_id,  
 'token': f'{alapi\_token}'  
 }  
 **try**:  
 weather\_data = self.make\_request(url, "GET", params=params)  
 **if** isinstance(weather\_data, dict) **and** weather\_data.get('code') == 200:  
 data = weather\_data['data']  
 update\_time = data['update\_time']  
 dt\_object = datetime.strptime(update\_time, "%Y-%m-%d %H:%M:%S")  
 formatted\_update\_time = dt\_object.strftime("%m-%d %H:%M")  
 *# Basic Info*  
 **if** **not** city\_or\_id.isnumeric() **and** data['city'] **not** **in** content: *# 如果返回城市信息不是所查询的城市，重新输入*  
 **return** "输入不规范，请输<国内城市+天气>，比如 '成都天气'"  
 formatted\_output = []  
 basic\_info = (  
 f"🏙️ 城市: {data['city']} ({data['province']})\n"  
 f"🕒 更新: {formatted\_update\_time}\n"  
 f"🌦️ 天气: {data['weather']}\n"  
 f"🌡️ 温度: ↓{data['min\_temp']}℃| 现{data['temp']}℃| ↑{data['max\_temp']}℃\n"  
 f"🌬️ 风向: {data['wind']}\n"  
 f"💦 湿度: {data['humidity']}\n"  
 f"🌅 日出/日落: {data['sunrise']} / {data['sunset']}\n"  
 )  
 formatted\_output.append(basic\_info)  
  
  
 *# Clothing Index,处理部分县区穿衣指数返回null*  
 chuangyi\_data = data.get('index', {}).get('chuangyi', {})  
 **if** chuangyi\_data:  
 chuangyi\_level = chuangyi\_data.get('level', '未知')  
 chuangyi\_content = chuangyi\_data.get('content', '未知')  
 **else**:  
 chuangyi\_level = '未知'  
 chuangyi\_content = '未知'  
  
 chuangyi\_info = f"👚 穿衣指数: {chuangyi\_level} - {chuangyi\_content}\n"  
 formatted\_output.append(chuangyi\_info)  
 *# Next 7 hours weather*  
 ten\_hours\_later = dt\_object + timedelta(hours=10)  
  
 future\_weather = []  
 **for** hour\_data **in** data['hour']:  
 forecast\_time\_str = hour\_data['time']  
 forecast\_time = datetime.strptime(forecast\_time\_str, "%Y-%m-%d %H:%M:%S")  
  
 **if** dt\_object < forecast\_time <= ten\_hours\_later:  
 future\_weather.append(f" {forecast\_time.hour:02d}:00 - {hour\_data['wea']} - {hour\_data['temp']}°C")  
  
 future\_weather\_info = "⏳ 未来10小时的天气预报:\n" + "\n".join(future\_weather)  
 formatted\_output.append(future\_weather\_info)  
  
 *# Alarm Info*  
 **if** data.get('alarm'):  
 alarm\_info = "⚠️ 预警信息:\n"  
 **for** alarm **in** data['alarm']:  
 alarm\_info += (  
 f"🔴 标题: {alarm['title']}\n"  
 f"🟠 等级: {alarm['level']}\n"  
 f"🟡 类型: {alarm['type']}\n"  
 f"🟢 提示: \n{alarm['tips']}\n"  
 f"🔵 内容: \n{alarm['content']}\n\n"  
 )  
 formatted\_output.append(alarm\_info)  
  
 **return** "\n".join(formatted\_output)  
 **else**:  
 **return** self.handle\_error(weather\_data, "获取失败，请查看服务器log")  
  
 **except** Exception **as** e:  
 **return** self.handle\_error(e, "获取天气信息失败")  
  
 **def** get\_mx\_bagua(self):  
 url = "https://dayu.qqsuu.cn/mingxingbagua/apis.php?type=json"  
 payload = "format=json"  
 headers = {'Content-Type': "application/x-www-form-urlencoded"}  
 bagua\_info = self.make\_request(url, method="POST", headers=headers, data=payload)  
 **if** isinstance(bagua\_info, dict) **and** bagua\_info['code'] == 200:  
 bagua\_pic\_url = bagua\_info["data"]  
 **if** self.is\_valid\_image\_url(bagua\_pic\_url):  
 **return** bagua\_pic\_url  
 **else**:  
 **return** "周末不更新，请微博吃瓜"  
 **else**:  
 logger.error(f"错误信息：{bagua\_info}")  
 **return** "暂无明星八卦，吃瓜莫急"  
  
 **def** make\_request(self, url, method="GET", headers=None, params=None, data=None, json\_data=None):  
 **try**:  
 **if** method.upper() == "GET":  
 response = requests.request(method, url, headers=headers, params=params)  
 **elif** method.upper() == "POST":  
 response = requests.request(method, url, headers=headers, data=data, json=json\_data)  
 **else**:  
 **return** {"success": False, "message": "Unsupported HTTP method"}  
  
 **return** response.json()  
 **except** Exception **as** e:  
 **return** e  
  
  
 **def** create\_reply(self, reply\_type, content):  
 reply = Reply()  
 reply.type = reply\_type  
 reply.content = content  
 **return** reply  
  
 **def** handle\_error(self, error, message):  
 logger.error(f"{message}，错误信息：{error}")  
 **return** message  
  
 **def** is\_valid\_url(self, url):  
 **try**:  
 result = urlparse(url)  
 **return** all([result.scheme, result.netloc])  
 **except** ValueError:  
 **return** False  
  
 **def** is\_valid\_image\_url(self, url):  
 **try**:  
 response = requests.head(url)   
 **return** response.status\_code == 200  
 **except** requests.RequestException **as** e:  
 **return** False  
  
 **def** load\_city\_conditions(self):  
 **if** self.condition\_2\_and\_3\_cities **is** None:  
 **try**:  
 json\_file\_path = os.path.join(os.path.dirname(\_\_file\_\_), 'duplicate-citys.json')  
 with open(json\_file\_path, 'r', encoding='utf-8') **as** f:  
 self.condition\_2\_and\_3\_cities = json.load(f)  
 **except** Exception **as** e:  
 **return** self.handle\_error(e, "加载condition\_2\_and\_3\_cities.json失败")  
  
  
 **def** check\_multiple\_city\_ids(self, city):  
 self.load\_city\_conditions()  
 city\_info = self.condition\_2\_and\_3\_cities.get(city, None)  
 **if** city\_info:  
 **return** city\_info  
 **return** None  
  
  
ZODIAC\_MAPPING = {  
 '白羊座': 'aries',  
 '金牛座': 'taurus',  
 '双子座': 'gemini',  
 '巨蟹座': 'cancer',  
 '狮子座': 'leo',  
 '处女座': 'virgo',  
 '天秤座': 'libra',  
 '天蝎座': 'scorpio',  
 '射手座': 'sagittarius',  
 '摩羯座': 'capricorn',  
 '水瓶座': 'aquarius',  
 '双鱼座': 'pisces'  
 }  
  
hot\_trend\_types = {  
 "微博": "wbHot",  
 "虎扑": "huPu",  
 "知乎": "zhihuHot",  
 "哔哩哔哩": "bili",  
 "36氪": "36Ke",  
 "抖音": "douyinHot",  
 "少数派": "ssPai",  
 "IT最新": "itNews",  
 "IT科技": "itInfo"  
  
}  
**import** json  
**import** os  
  
**import** plugins  
**from** bridge.context **import** ContextType  
**from** bridge.reply **import** Reply, ReplyType  
**from** common.log **import** logger  
**from** plugins **import** \*  
  
**from** .lib.WordsSearch **import** WordsSearch  
  
  
@plugins.register(  
 name="Banwords",  
 desire\_priority=100,  
 hidden=True,  
 desc="判断消息中是否有敏感词、决定是否回复。",  
 version="1.0",  
 author="lanvent",  
)  
**class** Banwords(Plugin):  
 **def** \_\_init\_\_(self):  
 super().\_\_init\_\_()  
 **try**:  
 conf = super().load\_config()  
 curdir = os.path.dirname(\_\_file\_\_)  
 **if** **not** conf:  
 config\_path = os.path.join(curdir, "config.json")  
 **if** **not** os.path.exists(config\_path):  
 conf = {"action": "ignore"}  
 with open(config\_path, "w") **as** f:  
 json.dump(conf, f, indent=4)  
  
 self.searchr = WordsSearch()  
 self.action = conf["action"]  
 banwords\_path = os.path.join(curdir, "banwords.txt")  
 with open(banwords\_path, "r", encoding="utf-8") **as** f:  
 words = []  
 **for** line **in** f:  
 word = line.strip()  
 **if** word:  
 words.append(word)  
 self.searchr.SetKeywords(words)  
 self.handlers[Event.ON\_HANDLE\_CONTEXT] = self.on\_handle\_context  
 **if** conf.get("reply\_filter", True):  
 self.handlers[Event.ON\_DECORATE\_REPLY] = self.on\_decorate\_reply  
 self.reply\_action = conf.get("reply\_action", "ignore")  
 **def** on\_handle\_context(self, e\_context: EventContext):  
 **if** e\_context["context"].type **not** **in** [  
 ContextType.TEXT,  
 ContextType.IMAGE\_CREATE,  
 ]:  
 **return**  
  
 content = e\_context["context"].content  
 logger.debug("[Banwords] on\_handle\_context. content: %s" % content)  
 **if** self.action == "ignore":  
 f = self.searchr.FindFirst(content)  
 **if** f:  
 logger.info("[Banwords] %s in message" % f["Keyword"])  
 e\_context.action = EventAction.BREAK\_PASS  
 **return**  
 **elif** self.action == "replace":  
 **if** self.searchr.ContainsAny(content):  
 reply = Reply(ReplyType.INFO, "发言中包含敏感词，请重试: \n" + self.searchr.Replace(content))  
 e\_context["reply"] = reply  
 e\_context.action = EventAction.BREAK\_PASS  
 **return**  
  
 **def** on\_decorate\_reply(self, e\_context: EventContext):  
 **if** e\_context["reply"].type **not** **in** [ReplyType.TEXT]:  
 **return**  
  
 reply = e\_context["reply"]  
 content = reply.content  
 **if** self.reply\_action == "ignore":  
 f = self.searchr.FindFirst(content)  
 **if** f:  
 logger.info("[Banwords] %s in reply" % f["Keyword"])  
 e\_context["reply"] = None  
 e\_context.action = EventAction.BREAK\_PASS  
 **return**  
 **elif** self.reply\_action == "replace":  
 **if** self.searchr.ContainsAny(content):  
 reply = Reply(ReplyType.INFO, "已替换回复中的敏感词: \n" + self.searchr.Replace(content))  
 e\_context["reply"] = reply  
 e\_context.action = EventAction.CONTINUE  
 **return**  
  
 **def** get\_help\_text(self, \*\*kwargs):  
 **return** "过滤消息中的敏感词。"

**class** Hello(Plugin):  
 **def** \_\_init\_\_(self):  
 super().\_\_init\_\_()  
 self.handlers[Event.ON\_HANDLE\_CONTEXT] = self.on\_handle\_context  
 logger.info("[Hello] inited")  
 self.config = super().load\_config()  
  
 **def** on\_handle\_context(self, e\_context: EventContext):  
 **if** e\_context["context"].type **not** **in** [  
 ContextType.TEXT,  
 ContextType.JOIN\_GROUP,  
 ContextType.PATPAT,  
 ContextType.EXIT\_GROUP  
 ]:  
 **return**  
 **if** e\_context["context"].type == ContextType.JOIN\_GROUP:  
 **if** "group\_welcome\_msg" **in** conf():  
 reply = Reply()  
 reply.type = ReplyType.TEXT  
 reply.content = conf().get("group\_welcome\_msg", "")  
 e\_context["reply"] = reply  
 e\_context.action = EventAction.BREAK\_PASS *# 事件结束，并跳过处理context的默认逻辑*  
 **return**  
 e\_context["context"].type = ContextType.TEXT  
 msg: ChatMessage = e\_context["context"]["msg"]  
 e\_context["context"].content = f'请你随机使用一种风格说一句问候语来欢迎新用户"{msg.actual\_user\_nickname}"加入群聊。'  
 e\_context.action = EventAction.BREAK *# 事件结束，进入默认处理逻辑*  
 **if** **not** self.config **or** **not** self.config.get("use\_character\_desc"):  
 e\_context["context"]["generate\_breaked\_by"] = EventAction.BREAK  
 **return**  
   
 **if** e\_context["context"].type == ContextType.EXIT\_GROUP:  
 **if** conf().get("group\_chat\_exit\_group"):  
 e\_context["context"].type = ContextType.TEXT  
 msg: ChatMessage = e\_context["context"]["msg"]  
 e\_context["context"].content = f'请你随机使用一种风格跟其他群用户说他违反规则"{msg.actual\_user\_nickname}"退出群聊。'  
 e\_context.action = EventAction.BREAK *# 事件结束，进入默认处理逻辑*  
 **return**  
 e\_context.action = EventAction.BREAK  
 **return**  
   
 **if** e\_context["context"].type == ContextType.PATPAT:  
 e\_context["context"].type = ContextType.TEXT  
 msg: ChatMessage = e\_context["context"]["msg"]  
 e\_context["context"].content = f"请你随机使用一种风格介绍你自己，并告诉用户输入#help可以查看帮助信息。"  
 e\_context.action = EventAction.BREAK *# 事件结束，进入默认处理逻辑*  
 **if** **not** self.config **or** **not** self.config.get("use\_character\_desc"):  
 e\_context["context"]["generate\_breaked\_by"] = EventAction.BREAK  
 **return**  
  
 content = e\_context["context"].content  
 logger.debug("[Hello] on\_handle\_context. content: %s" % content)  
 **if** content == "Hello":  
 reply = Reply()  
 reply.type = ReplyType.TEXT  
 msg: ChatMessage = e\_context["context"]["msg"]  
 **if** e\_context["context"]["isgroup"]:  
 reply.content = f"Hello, {msg.actual\_user\_nickname} from {msg.from\_user\_nickname}"  
 **else**:  
 reply.content = f"Hello, {msg.from\_user\_nickname}"  
 e\_context["reply"] = reply  
 e\_context.action = EventAction.BREAK\_PASS *# 事件结束，并跳过处理context的默认逻辑*  
  
 **if** content == "Hi":  
 reply = Reply()  
 reply.type = ReplyType.TEXT  
 reply.content = "Hi"  
 e\_context["reply"] = reply  
 e\_context.action = EventAction.BREAK   
  
 **if** content == "End":  
  
 e\_context["context"].type = ContextType.IMAGE\_CREATE  
 content = "The World"  
 e\_context.action = EventAction.CONTINUE   
  
 **def** get\_help\_text(self, \*\*kwargs):  
 help\_text = "输入Hello，我会回复你的名字\n输入End，我会回复你世界的图片\n"  
 **return** help\_text

@plugins.register(  
 name="coze",  
 desire\_priority=99,  
 hidden=True,  
 desc="coze",  
 version="1.1",  
 author="masterke",  
)  
**class** coze(Plugin):  
 **def** \_\_init\_\_(self):  
 super().\_\_init\_\_()  
 self.handlers[Event.ON\_HANDLE\_CONTEXT] = self.on\_handle\_context  
 logger.info("[chajian] inited")  
 self.config = super().load\_config()  
  
 **def** on\_handle\_context(self, e\_context: EventContext):  
 **if** e\_context["context"].type != ContextType.TEXT:  
 **return**  
 reply = None  
 **if** os.path.exists('config.json'):  
 config\_path = os.path.join(os.path.dirname(\_\_file\_\_), "config.json")  
 with open(config\_path, 'r') **as** file:  
 config\_data = json.load(file)  
 apiUrl = config\_data['apiUrl']  
 apiKey = config\_data['apiKey']  
 channelId = config\_data['channelId']  
 createImgPrefix = config\_data['createImgPrefix']  
 **else**:  
 text = "请先配置config.json文件"  
 reply = Reply(ReplyType.ERROR, text)  
 e\_context["reply"] = reply  
 e\_context.action = EventAction.BREAK\_PASS  
  
 query = e\_context["context"].content.strip()  
 **if** query.startswith(createImgPrefix):  
 **try**:  
 header = {  
 "Content-Type": "application/json",  
 "Authorization": apiKey  
 }  
 data = {  
 "channelId": channelId,  
 "messages": [  
 {  
 "role":"user",  
 "content":query  
 }  
 ],  
 "stream": False  
 }  
 json\_data = json.dumps(data)  
 **if** apiKey:  
 results = requests.post(apiUrl, data=json\_data, headers=header)  
 **if** results.status\_code == 200:  
 data\_string = results.json()  
 **print**(data\_string)  
 coze\_reply = data\_string["choices"][0]["message"]["content"]  
 **def** contains\_img(text):  
 *# 正则表达式匹配http或https开头的URL*  
 pattern = re.compile(r'https?://\S+')  
 *# 检查文本是否包含匹配的URL*  
 **return** pattern.search(text) **is** **not** None  
 **if** contains\_img(coze\_reply):  
 url\_pattern = re.compile(r'!\[.\*?\]\((.\*?)\)')  
 *# 使用findall()函数查找所有匹配的URL*  
 urls = url\_pattern.findall(coze\_reply)  
 img\_url = urls[0] **if** urls **else** None  
 logger.info(img\_url)  
 reply = Reply(ReplyType.IMAGE\_URL, img\_url)  
 e\_context["reply"] = reply  
 e\_context.action = EventAction.BREAK\_PASS  
 **else**:  
 logger.info(coze\_reply)  
 reply = Reply(ReplyType.TEXT, coze\_reply)  
 e\_context["reply"] = reply  
 e\_context.action = EventAction.BREAK\_PASS  
 **else**:  
 requests\_error\_reply = (f"创建画图任务失败😭\n状态码：{results.status\_code}!")  
 reply = Reply(ReplyType.ERROR, requests\_error\_reply)  
 e\_context["reply"] = reply  
 e\_context.action = EventAction.BREAK\_PASS  
 **except** Exception **as** e:  
 **print**(e)  
 run\_error\_reply = (f"发生异常,等待修复哦...😭\n先玩游戏其他的功能吧")  
 reply = Reply(ReplyType.ERROR, run\_error\_reply)  
 e\_context["reply"] = reply  
 e\_context.action = EventAction.BREAK\_PASS  
  
 **def** get\_help\_text(self, \*\*kwargs):  
 help\_text = "请先配置config.json文件，删除app.py中config.json中的画图前缀，或者修改不同的画图前缀"  
 **return** help\_text

**class** RolePlay:  
 **def** \_\_init\_\_(self, bot, sessionid, desc, wrapper=None):  
 self.bot = bot  
 self.sessionid = sessionid  
 self.wrapper = wrapper **or** "%s" *# 用于包装用户输入*  
 self.desc = desc  
 self.bot.sessions.build\_session(self.sessionid, system\_prompt=self.desc)  
  
 **def** reset(self):  
 self.bot.sessions.clear\_session(self.sessionid)  
  
 **def** action(self, user\_action):  
 session = self.bot.sessions.build\_session(self.sessionid)  
 **if** session.system\_prompt != self.desc: *# 目前没有触发session过期事件，这里先简单判断，然后重置*  
 session.set\_system\_prompt(self.desc)  
 prompt = self.wrapper % user\_action  
 **return** prompt  
  
  
@plugins.register(  
 name="Role",  
 desire\_priority=0,  
 namecn="角色扮演",  
 desc="为你的Bot设置预设角色",  
 version="1.0",  
 author="lanvent",  
)  
**class** Role(Plugin):  
 **def** \_\_init\_\_(self):  
 super().\_\_init\_\_()  
 curdir = os.path.dirname(\_\_file\_\_)  
 config\_path = os.path.join(curdir, "roles.json")  
 **try**:  
 with open(config\_path, "r", encoding="utf-8") **as** f:  
 config = json.load(f)  
 self.tags = {tag: (desc, []) **for** tag, desc **in** config["tags"].items()}  
 self.roles = {}  
 **for** role **in** config["roles"]:  
 self.roles[role["title"].lower()] = role  
 **for** tag **in** role["tags"]:  
 **if** tag **not** **in** self.tags:  
 logger.warning(f"[Role] unknown tag {tag} ")  
 self.tags[tag] = (tag, [])  
 self.tags[tag][1].append(role)  
 **for** tag **in** list(self.tags.keys()):  
 **if** len(self.tags[tag][1]) == 0:  
 logger.debug(f"[Role] no role found for tag {tag} ")  
 **del** self.tags[tag]  
  
 **if** len(self.roles) == 0:  
 **raise** Exception("no role found")  
 self.handlers[Event.ON\_HANDLE\_CONTEXT] = self.on\_handle\_context  
 self.roleplays = {}  
 logger.info("[Role] inited")  
 **raise** e  
  
 **def** get\_role(self, name, find\_closest=True, min\_sim=0.35):  
 name = name.lower()  
 found\_role = None  
 **if** name **in** self.roles:  
 found\_role = name  
 **elif** find\_closest:  
 **import** difflib  
  
 **def** str\_simularity(a, b):  
 **return** difflib.SequenceMatcher(None, a, b).ratio()  
  
 max\_sim = min\_sim  
 max\_role = None  
 **for** role **in** self.roles:  
 sim = str\_simularity(name, role)  
 **if** sim >= max\_sim:  
 max\_sim = sim  
 max\_role = role  
 found\_role = max\_role  
 **return** found\_role  
  
 **def** on\_handle\_context(self, e\_context: EventContext):  
 **if** e\_context["context"].type != ContextType.TEXT:  
 **return**  
 btype = Bridge().get\_bot\_type("chat")  
 **if** btype **not** **in** [const.OPEN\_AI, const.CHATGPT, const.CHATGPTONAZURE, const.LINKAI]:  
 **return**  
 bot = Bridge().get\_bot("chat")  
 content = e\_context["context"].content[:]  
 clist = e\_context["context"].content.split(maxsplit=1)  
 desckey = None  
 customize = False  
 sessionid = e\_context["context"]["session\_id"]  
 trigger\_prefix = conf().get("plugin\_trigger\_prefix", "$")  
 **if** clist[0] == f"{trigger\_prefix}停止扮演":  
 **if** sessionid **in** self.roleplays:  
 self.roleplays[sessionid].reset()  
 **del** self.roleplays[sessionid]  
 reply = Reply(ReplyType.INFO, "角色扮演结束!")  
 e\_context["reply"] = reply  
 e\_context.action = EventAction.BREAK\_PASS  
 **return**  
 **elif** clist[0] == f"{trigger\_prefix}角色":  
 desckey = "descn"  
 **elif** clist[0].lower() == f"{trigger\_prefix}role":  
 desckey = "description"  
 **elif** clist[0] == f"{trigger\_prefix}设定扮演":  
 customize = True  
 **elif** clist[0] == f"{trigger\_prefix}角色类型":  
 **if** len(clist) > 1:  
 tag = clist[1].strip()  
 help\_text = "角色列表：\n"  
 **for** key, value **in** self.tags.items():  
 **if** value[0] == tag:  
 tag = key  
 **break**  
 **if** tag == "所有":  
 **for** role **in** self.roles.values():  
 help\_text += f"{role['title']}: {role['remark']}\n"  
 **elif** tag **in** self.tags:  
 **for** role **in** self.tags[tag][1]:  
 help\_text += f"{role['title']}: {role['remark']}\n"  
 **else**:  
 help\_text = f"未知角色类型。\n"  
 help\_text += "目前的角色类型有: \n"  
 help\_text += "，".join([self.tags[tag][0] **for** tag **in** self.tags]) + "\n"  
 **else**:  
 help\_text = f"请输入角色类型。\n"  
 help\_text += "目前的角色类型有: \n"  
 help\_text += "，".join([self.tags[tag][0] **for** tag **in** self.tags]) + "\n"  
 reply = Reply(ReplyType.INFO, help\_text)  
 e\_context["reply"] = reply  
 e\_context.action = EventAction.BREAK\_PASS  
 **return**  
 **elif** sessionid **not** **in** self.roleplays:  
 **return**  
 logger.debug("[Role] on\_handle\_context. content: %s" % content)  
 **if** desckey **is** **not** None:  
 **if** len(clist) == 1 **or** (len(clist) > 1 **and** clist[1].lower() **in** ["help", "帮助"]):  
 reply = Reply(ReplyType.INFO, self.get\_help\_text(verbose=True))  
 e\_context["reply"] = reply  
 e\_context.action = EventAction.BREAK\_PASS  
 **return**  
 role = self.get\_role(clist[1])  
 **if** role **is** None:  
 reply = Reply(ReplyType.ERROR, "角色不存在")  
 e\_context["reply"] = reply  
 e\_context.action = EventAction.BREAK\_PASS  
 **return**  
 **else**:  
 self.roleplays[sessionid] = RolePlay(  
 bot,  
 sessionid,  
 self.roles[role][desckey],  
 self.roles[role].get("wrapper", "%s"),  
 )  
 reply = Reply(ReplyType.INFO, f"预设角色为 {role}:\n" + self.roles[role][desckey])  
 e\_context["reply"] = reply  
 e\_context.action = EventAction.BREAK\_PASS  
 **elif** customize == True:  
 self.roleplays[sessionid] = RolePlay(bot, sessionid, clist[1], "%s")  
 reply = Reply(ReplyType.INFO, f"角色设定为:\n{clist[1]}")  
 e\_context["reply"] = reply  
 e\_context.action = EventAction.BREAK\_PASS  
 **else**:  
 prompt = self.roleplays[sessionid].action(content)  
 e\_context["context"].type = ContextType.TEXT  
 e\_context["context"].content = prompt  
 e\_context.action = EventAction.BREAK  
  
 **def** get\_help\_text(self, verbose=False, \*\*kwargs):  
 help\_text = "让机器人扮演不同的角色。\n"  
 **if** **not** verbose:  
 **return** help\_text  
 trigger\_prefix = conf().get("plugin\_trigger\_prefix", "$")  
 help\_text = f"使用方法:\n{trigger\_prefix}角色" + " 预设角色名: 设定角色为{预设角色名}。\n" + f"{trigger\_prefix}role" + " 预设角色名: 同上，但使用英文设定。\n"  
 help\_text += f"{trigger\_prefix}设定扮演" + " 角色设定: 设定自定义角色人设为{角色设定}。\n"  
 help\_text += f"{trigger\_prefix}停止扮演: 清除设定的角色。\n"  
 help\_text += f"{trigger\_prefix}角色类型" + " 角色类型: 查看某类{角色类型}的所有预设角色，为所有时输出所有预设角色。\n"  
 help\_text += "\n目前的角色类型有: \n"  
 help\_text += "，".join([self.tags[tag][0] **for** tag **in** self.tags]) + "。\n"  
 help\_text += f"\n命令例子: \n{trigger\_prefix}角色 写作助理\n"  
 help\_text += f"{trigger\_prefix}角色类型 所有\n"  
 help\_text += f"{trigger\_prefix}停止扮演\n"  
 **return** help\_text

@itchat.msg\_register([TEXT, VOICE, PICTURE, NOTE, ATTACHMENT, SHARING])  
**def** handler\_single\_msg(msg):  
 **try**:  
 cmsg = WechatMessage(msg, False)  
 **except** NotImplementedError **as** e:  
 logger.debug("[WX]single message {} skipped: {}".format(msg["MsgId"], e))  
 **return** None  
 WechatChannel().handle\_single(cmsg)  
 **return** None  
  
  
@itchat.msg\_register([TEXT, VOICE, PICTURE, NOTE, ATTACHMENT, SHARING], isGroupChat=True)  
**def** handler\_group\_msg(msg):  
 **try**:  
 cmsg = WechatMessage(msg, True)  
 **except** NotImplementedError **as** e:  
 logger.debug("[WX]group message {} skipped: {}".format(msg["MsgId"], e))  
 **return** None  
 WechatChannel().handle\_group(cmsg)  
 **return** None  
  
  
**def** \_check(func):  
 **def** wrapper(self, cmsg: ChatMessage):  
 msgId = cmsg.msg\_id  
 **if** msgId **in** self.receivedMsgs:  
 logger.info("Wechat message {} already received, ignore".format(msgId))  
 **return**  
 self.receivedMsgs[msgId] = True  
 create\_time = cmsg.create\_time *# 消息时间戳*  
 **if** conf().get("hot\_reload") == True **and** **int**(create\_time) < **int**(time.time()) - 60: *# 跳过1分钟前的历史消息*  
 logger.debug("[WX]history message {} skipped".format(msgId))  
 **return**  
 **if** cmsg.my\_msg **and** **not** cmsg.is\_group:  
 logger.debug("[WX]my message {} skipped".format(msgId))  
 **return**  
 **return** func(self, cmsg)  
  
 **return** wrapper  
  
  
*# 可用的二维码生成接口*  
*# https://api.qrserver.com/v1/create-qr-code/?size=400×400&data=https://www.abc.com*  
*# https://api.isoyu.com/qr/?m=1&e=L&p=20&url=https://www.abc.com*  
**def** qrCallback(uuid, status, qrcode):  
 *# logger.debug("qrCallback: {} {}".format(uuid,status))*  
 **if** status == "0":  
 **try**:  
 **from** PIL **import** Image  
  
 img = Image.open(io.BytesIO(qrcode))  
 \_thread = threading.Thread(target=img.show, args=("QRCode",))  
 \_thread.setDaemon(True)  
 \_thread.start()  
 **except** Exception **as** e:  
 **pass**  
  
 **import** qrcode  
  
 url = f"https://login.weixin.qq.com/l/{uuid}"  
  
 qr\_api1 = "https://api.isoyu.com/qr/?m=1&e=L&p=20&url={}".format(url)  
 qr\_api2 = "https://api.qrserver.com/v1/create-qr-code/?size=400×400&data={}".format(url)  
 qr\_api3 = "https://api.pwmqr.com/qrcode/create/?url={}".format(url)  
 qr\_api4 = "https://my.tv.sohu.com/user/a/wvideo/getQRCode.do?text={}".format(url)  
 **print**("You can also scan QRCode in any website below:")  
 **print**(qr\_api3)  
 **print**(qr\_api4)  
 **print**(qr\_api2)  
 **print**(qr\_api1)  
 \_send\_qr\_code([qr\_api3, qr\_api4, qr\_api2, qr\_api1])  
 qr = qrcode.QRCode(border=1)  
 qr.add\_data(url)  
 qr.make(fit=True)  
 qr.print\_ascii(invert=True)  
  
  
@singleton  
**class** WechatChannel(ChatChannel):  
 NOT\_SUPPORT\_REPLYTYPE = []  
  
 **def** \_\_init\_\_(self):  
 super().\_\_init\_\_()  
 self.receivedMsgs = ExpiredDict(conf().get("expires\_in\_seconds"))  
 self.auto\_login\_times = 0  
  
 **def** startup(self):  
 **try**:  
 itchat.instance.receivingRetryCount = 600 *# 修改断线超时时间*  
 *# login by scan QRCode*  
 hotReload = conf().get("hot\_reload", False)  
 status\_path = os.path.join(get\_appdata\_dir(), "itchat.pkl")  
 itchat.auto\_login(  
 enableCmdQR=2,  
 hotReload=hotReload,  
 statusStorageDir=status\_path,  
 qrCallback=qrCallback,  
 exitCallback=self.exitCallback,  
 loginCallback=self.loginCallback  
 )  
 self.user\_id = itchat.instance.storageClass.userName  
 self.name = itchat.instance.storageClass.nickName  
 logger.info("Wechat login success, user\_id: {}, nickname: {}".format(self.user\_id, self.name))  
 *# start message listener*  
 itchat.run()  
 **except** Exception **as** e:  
 logger.exception(e)  
  
 **def** exitCallback(self):  
 **try**:  
 **from** common.linkai\_client **import** chat\_client  
 **if** chat\_client.client\_id **and** conf().get("use\_linkai"):  
 \_send\_logout()  
 time.sleep(2)  
 self.auto\_login\_times += 1  
 **if** self.auto\_login\_times < 100:  
 chat\_channel.handler\_pool.\_shutdown = False  
 self.startup()  
 **except** Exception **as** e:  
 **pass**  
  
 **def** loginCallback(self):  
 logger.debug("Login success")  
 \_send\_login\_success()  
  
 *# handle\_\* 系列函数处理收到的消息后构造Context，然后传入produce函数中处理Context和发送回复*  
 *# Context包含了消息的所有信息，包括以下属性*  
 *# type 消息类型, 包括TEXT、VOICE、IMAGE\_CREATE*  
 *# content 消息内容，如果是TEXT类型，content就是文本内容，如果是VOICE类型，content就是语音文件名，如果是IMAGE\_CREATE类型，content就是图片生成命令*  
 *# kwargs 附加参数字典，包含以下的key：*  
 *# session\_id: 会话id*  
 *# isgroup: 是否是群聊*  
 *# receiver: 需要回复的对象*  
 *# msg: ChatMessage消息对象*  
 *# origin\_ctype: 原始消息类型，语音转文字后，私聊时如果匹配前缀失败，会根据初始消息是否是语音来放宽触发规则*  
 *# desire\_rtype: 希望回复类型，默认是文本回复，设置为ReplyType.VOICE是语音回复*  
 @time\_checker  
 @\_check  
 **def** handle\_single(self, cmsg: ChatMessage):  
 *# filter system message*  
 **if** cmsg.other\_user\_id **in** ["weixin"]:  
 **return**  
 **if** cmsg.ctype == ContextType.VOICE:  
 **if** conf().get("speech\_recognition") != True:  
 **return**  
 logger.debug("[WX]receive voice msg: {}".format(cmsg.content))  
 **elif** cmsg.ctype == ContextType.IMAGE:  
 logger.debug("[WX]receive image msg: {}".format(cmsg.content))  
 **elif** cmsg.ctype == ContextType.PATPAT:  
 logger.debug("[WX]receive patpat msg: {}".format(cmsg.content))  
 **elif** cmsg.ctype == ContextType.TEXT:  
 logger.debug("[WX]receive text msg: {}, cmsg={}".format(json.dumps(cmsg.\_rawmsg, ensure\_ascii=False), cmsg))  
 **else**:  
 logger.debug("[WX]receive msg: {}, cmsg={}".format(cmsg.content, cmsg))  
 context = self.\_compose\_context(cmsg.ctype, cmsg.content, isgroup=False, msg=cmsg)  
 **if** context:  
 self.produce(context)  
  
 @time\_checker  
 @\_check  
 **def** handle\_group(self, cmsg: ChatMessage):  
 **if** cmsg.ctype == ContextType.VOICE:  
 **if** conf().get("group\_speech\_recognition") != True:  
 **return**  
 logger.debug("[WX]receive voice for group msg: {}".format(cmsg.content))  
 **elif** cmsg.ctype == ContextType.IMAGE:  
 logger.debug("[WX]receive image for group msg: {}".format(cmsg.content))  
 **elif** cmsg.ctype **in** [ContextType.JOIN\_GROUP, ContextType.PATPAT, ContextType.ACCEPT\_FRIEND, ContextType.EXIT\_GROUP]:  
 logger.debug("[WX]receive note msg: {}".format(cmsg.content))  
 **elif** cmsg.ctype == ContextType.TEXT:  
 *# logger.debug("[WX]receive group msg: {}, cmsg={}".format(json.dumps(cmsg.\_rawmsg, ensure\_ascii=False), cmsg))*  
 **pass**  
 **elif** cmsg.ctype == ContextType.FILE:  
 logger.debug(f"[WX]receive attachment msg, file\_name={cmsg.content}")  
 **else**:  
 logger.debug("[WX]receive group msg: {}".format(cmsg.content))  
 context = self.\_compose\_context(cmsg.ctype, cmsg.content, isgroup=True, msg=cmsg)  
 **if** context:  
 self.produce(context)  
  
 *# 统一的发送函数，每个Channel自行实现，根据reply的type字段发送不同类型的消息*  
 **def** send(self, reply: Reply, context: Context):  
 receiver = context["receiver"]  
 **if** reply.type == ReplyType.TEXT:  
 itchat.send(reply.content, toUserName=receiver)  
 logger.info("[WX] sendMsg={}, receiver={}".format(reply, receiver))  
 **elif** reply.type == ReplyType.ERROR **or** reply.type == ReplyType.INFO:  
 itchat.send(reply.content, toUserName=receiver)  
 logger.info("[WX] sendMsg={}, receiver={}".format(reply, receiver))  
 **elif** reply.type == ReplyType.VOICE:  
 itchat.send\_file(reply.content, toUserName=receiver)  
 logger.info("[WX] sendFile={}, receiver={}".format(reply.content, receiver))  
 **elif** reply.type == ReplyType.IMAGE\_URL: *# 从网络下载图片*  
 img\_url = reply.content  
 logger.debug(f"[WX] start download image, img\_url={img\_url}")  
 pic\_res = requests.get(img\_url, stream=True)  
 image\_storage = io.BytesIO()  
 size = 0  
 **for** block **in** pic\_res.iter\_content(1024):  
 size += len(block)  
 image\_storage.write(block)  
 logger.info(f"[WX] download image success, size={size}, img\_url={img\_url}")  
 image\_storage.seek(0)  
 itchat.send\_image(image\_storage, toUserName=receiver)  
 logger.info("[WX] sendImage url={}, receiver={}".format(img\_url, receiver))  
 **elif** reply.type == ReplyType.IMAGE: *# 从文件读取图片*  
 image\_storage = reply.content  
 image\_storage.seek(0)  
 itchat.send\_image(image\_storage, toUserName=receiver)  
 logger.info("[WX] sendImage, receiver={}".format(receiver))  
 **elif** reply.type == ReplyType.FILE: *# 新增文件回复类型*  
 file\_storage = reply.content  
 itchat.send\_file(file\_storage, toUserName=receiver)  
 logger.info("[WX] sendFile, receiver={}".format(receiver))  
 **elif** reply.type == ReplyType.VIDEO: *# 新增视频回复类型*  
 video\_storage = reply.content  
 itchat.send\_video(video\_storage, toUserName=receiver)  
 logger.info("[WX] sendFile, receiver={}".format(receiver))  
 **elif** reply.type == ReplyType.VIDEO\_URL: *# 新增视频URL回复类型*  
 video\_url = reply.content  
 logger.debug(f"[WX] start download video, video\_url={video\_url}")  
 video\_res = requests.get(video\_url, stream=True)  
 video\_storage = io.BytesIO()  
 size = 0  
 **for** block **in** video\_res.iter\_content(1024):  
 size += len(block)  
 video\_storage.write(block)  
 logger.info(f"[WX] download video success, size={size}, video\_url={video\_url}")  
 video\_storage.seek(0)  
 itchat.send\_video(video\_storage, toUserName=receiver)  
 logger.info("[WX] sendVideo url={}, receiver={}".format(video\_url, receiver))  
  
**def** \_send\_login\_success():  
 **try**:  
 **from** common.linkai\_client **import** chat\_client  
 **if** chat\_client.client\_id:  
 chat\_client.send\_login\_success()  
 **except** Exception **as** e:  
 **pass**  
  
**def** \_send\_logout():  
 **try**:  
 **from** common.linkai\_client **import** chat\_client  
 **if** chat\_client.client\_id:  
 chat\_client.send\_logout()  
 **except** Exception **as** e:  
 **pass**  
  
**def** \_send\_qr\_code(qrcode\_list: list):  
 **try**:  
 **from** common.linkai\_client **import** chat\_client  
 **if** chat\_client.client\_id:  
 chat\_client.send\_qrcode(qrcode\_list)  
 **except** Exception **as** e:  
 **pass**

**class** WechatMessage(ChatMessage):  
 **def** \_\_init\_\_(self, itchat\_msg, is\_group=False):  
 super().\_\_init\_\_(itchat\_msg)  
 self.msg\_id = itchat\_msg["MsgId"]  
 self.create\_time = itchat\_msg["CreateTime"]  
 self.is\_group = is\_group  
  
 **if** itchat\_msg["Type"] == TEXT:  
 self.ctype = ContextType.TEXT  
 self.content = itchat\_msg["Text"]  
 **elif** itchat\_msg["Type"] == VOICE:  
 self.ctype = ContextType.VOICE  
 self.content = TmpDir().path() + itchat\_msg["FileName"] *# content直接存临时目录路径*  
 self.\_prepare\_fn = **lambda**: itchat\_msg.download(self.content)  
 **elif** itchat\_msg["Type"] == PICTURE **and** itchat\_msg["MsgType"] == 3:  
 self.ctype = ContextType.IMAGE  
 self.content = TmpDir().path() + itchat\_msg["FileName"] *# content直接存临时目录路径*  
 self.\_prepare\_fn = **lambda**: itchat\_msg.download(self.content)  
 **elif** itchat\_msg["Type"] == NOTE **and** itchat\_msg["MsgType"] == 10000:  
 **if** is\_group **and** ("加入群聊" **in** itchat\_msg["Content"] **or** "加入了群聊" **in** itchat\_msg["Content"]):  
 *# 这里只能得到nickname， actual\_user\_id还是机器人的id*  
 **if** "加入了群聊" **in** itchat\_msg["Content"]:  
 self.ctype = ContextType.JOIN\_GROUP  
 self.content = itchat\_msg["Content"]  
 self.actual\_user\_nickname = re.findall(r"\"(.\*?)\"", itchat\_msg["Content"])[-1]  
 **elif** "加入群聊" **in** itchat\_msg["Content"]:  
 self.ctype = ContextType.JOIN\_GROUP  
 self.content = itchat\_msg["Content"]  
 self.actual\_user\_nickname = re.findall(r"\"(.\*?)\"", itchat\_msg["Content"])[0]  
  
 **elif** is\_group **and** ("移出了群聊" **in** itchat\_msg["Content"]):  
 self.ctype = ContextType.EXIT\_GROUP  
 self.content = itchat\_msg["Content"]  
 self.actual\_user\_nickname = re.findall(r"\"(.\*?)\"", itchat\_msg["Content"])[0]  
   
 **elif** "你已添加了" **in** itchat\_msg["Content"]: *#通过好友请求*  
 self.ctype = ContextType.ACCEPT\_FRIEND  
 self.content = itchat\_msg["Content"]  
 **elif** "拍了拍我" **in** itchat\_msg["Content"]:  
 self.ctype = ContextType.PATPAT  
 self.content = itchat\_msg["Content"]  
 **if** is\_group:  
 self.actual\_user\_nickname = re.findall(r"\"(.\*?)\"", itchat\_msg["Content"])[0]  
 **else**:  
 **raise** NotImplementedError("Unsupported note message: " + itchat\_msg["Content"])  
 **elif** itchat\_msg["Type"] == ATTACHMENT:  
 self.ctype = ContextType.FILE  
 self.content = TmpDir().path() + itchat\_msg["FileName"] *# content直接存临时目录路径*  
 self.\_prepare\_fn = **lambda**: itchat\_msg.download(self.content)  
 **elif** itchat\_msg["Type"] == SHARING:  
 self.ctype = ContextType.SHARING  
 self.content = itchat\_msg.get("Url")  
  
 **else**:  
 **raise** NotImplementedError("Unsupported message type: Type:{} MsgType:{}".format(itchat\_msg["Type"], itchat\_msg["MsgType"]))  
  
 self.from\_user\_id = itchat\_msg["FromUserName"]  
 self.to\_user\_id = itchat\_msg["ToUserName"]  
  
 user\_id = itchat.instance.storageClass.userName  
 nickname = itchat.instance.storageClass.nickName  
  
 *# 虽然from\_user\_id和to\_user\_id用的少，但是为了保持一致性，还是要填充一下*  
 *# 以下很繁琐，一句话总结：能填的都填了。*  
 **if** self.from\_user\_id == user\_id:  
 self.from\_user\_nickname = nickname  
 **if** self.to\_user\_id == user\_id:  
 self.to\_user\_nickname = nickname  
 **try**: *# 陌生人时候, User字段可能不存在*  
 *# my\_msg 为True是表示是自己发送的消息*  
 self.my\_msg = itchat\_msg["ToUserName"] == itchat\_msg["User"]["UserName"] **and** \  
 itchat\_msg["ToUserName"] != itchat\_msg["FromUserName"]  
 self.other\_user\_id = itchat\_msg["User"]["UserName"]  
 self.other\_user\_nickname = itchat\_msg["User"]["NickName"]  
 **if** self.other\_user\_id == self.from\_user\_id:  
 self.from\_user\_nickname = self.other\_user\_nickname  
 **if** self.other\_user\_id == self.to\_user\_id:  
 self.to\_user\_nickname = self.other\_user\_nickname  
 **if** itchat\_msg["User"].get("Self"):  
 *# 自身的展示名，当设置了群昵称时，该字段表示群昵称*  
 self.self\_display\_name = itchat\_msg["User"].get("Self").get("DisplayName")  
 **except** KeyError **as** e: *# 处理偶尔没有对方信息的情况*  
 logger.warn("[WX]get other\_user\_id failed: " + str(e))  
 **if** self.from\_user\_id == user\_id:  
 self.other\_user\_id = self.to\_user\_id  
 **else**:  
 self.other\_user\_id = self.from\_user\_id  
  
 **if** self.is\_group:  
 self.is\_at = itchat\_msg["IsAt"]  
 self.actual\_user\_id = itchat\_msg["ActualUserName"]  
 **if** self.ctype **not** **in** [ContextType.JOIN\_GROUP, ContextType.PATPAT, ContextType.EXIT\_GROUP]:  
 self.actual\_user\_nickname = itchat\_msg["ActualNickName"]