标识：产品代号-图号 BQ/1.0

联合作战人机协同博弈挑战赛
智能体开发指南

国防科大系统工程学院

中国电子科技集团公司第五十二研究所

北方自动控制技术研究所

二〇二三年八月

目 次

一、 概述

1.1 整体说明

智能体开发平台与选手自定义规则智能体使用 Python 语言编写，为选手提供开发环境与开发接口。选手需要熟悉自定义规则智能体编程与使用方法，按照规范进行编码开发。开发平台提供自动配置机制，选手只需提供规范化脚本，通过客户端可视化操作即可运行智能体。

1.2 架构介绍

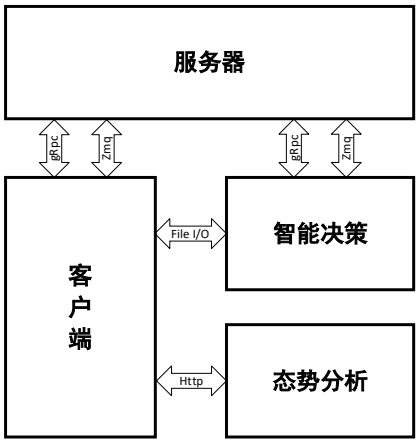


图 1 平台结构图

如图 1 所示，软件整体架构由推演仿真服务器、可视化客户端、智能决策智能体开发平台和态势分析智能体开发平台四个部分组成。推演仿真服务器负责提供仿真引擎指令控制与推演数据功能；可视化客户端负责选手指令操作与态势显示功能；智能决策开发平台负责运行选手自定义决策规则智能体；态势分析开发平台负责运行选手自定义态势分析规则智能体。

二、 功能操作

2.1 功能说明

2.1.1 态势分析与智能决策共有功能

序号	功能项
1	支持加载多个脚本
2	支持同时运行多个脚本
3	支持可视化配置脚本
4	支持可视化控制脚本运行或停止
5	支持编程控制脚本停止

表 1 共有功能表

通过多进程的方式启动多个脚本,加载数量没有限制,态势分析脚本同时运行脚本数量没有限制,智能决策同时运行脚本数量限制为 3 个。由于选手电脑性能不同,启动脚本时间延迟有差异。

通过客户端可视化交互操作即可完成解释器配置,工作路径配置,脚本加载,脚本启动与停止等功能,简化解释器路径配置,工作路径配置,脚本路径文件配置,脚本进程管理等操作。

提供编程控制脚本停止接口,可以实现脚本内主动停止脚本进程,通过此功能可以选择编写长期运行脚本或单次运行脚本。

2.1.2 态势分析特有功能

序号	功能项
1	提供简化 matplot 绘图接口
2	支持控制台输出分析内容
3	支持文件输出分析结果
4	支持文件输出原始态势

表 2 态势分析特有功能表

支持 matplot 绘图,此方式提供了一个直观展现统计态势分析结果的方式.封装 matplot 绘图函数， 通过调用 easy_bar,easy_pie 和 pause,即可实现简单实绘制饼图与直方图,省略窗口管理,画图板管理,图表管理,显示更新管理等复杂操作。

支持控制台输出内容,此方式提供了一个简单直接的态势分析结果输出方式.可以通过 print 将关键态势分析结果输出到控制台进行显示。

支持文件输出自定义内容,此方式提供了一个可扩展和可回溯的态势分析结

果输出方式。通过调用 save_analysis_result 将预期的态势分析结果写入到指定的文件中,可以通过其他软件监控此文件变化并做出响应,也可以保存所有历史态势分析结果方便回溯分析。

支持文件输出原始态势内容,此方式提供了一个可扩展和可回溯的态势输出方式。通过 set_situation_path 设置态势输出到文件的路径,可以通过其他软件监控此文件并作出响应,也可以通过此文件进行对战全过程回溯。注意,由于每局原始态势数据量大,且占用磁盘空间多,因此每次启动态势分析脚本会默认清除默认路径下的原始态势文件。

2.1.3 智能决策特有功能

序号	功能项
1	支持可视化输入脚本数据源
2	支持丰富的编程控制接口

表 3 智能决策特有功能表

支持可视化配置脚本输入源。打开客户端智能体配置界面，可以选择己方单元、情报单元、己方参考点，以及配置用户自定义信息。这些选择的信息在用户配置完成点击确定后会写入配置文件中，文件格式为 json。在智能决策开发平台中提供 api_get_client_select_info 接口来获取完整的脚本输入源信息。也可使用 api_get_client_select_list 接口来获取整理后的脚本输入源列表。api_get_client_select_info 接口返回的内容为 dict 字典，其中包含”unit_info”，”target_info”，”point_info”，”user_define_info”四个模块，分别对应客户端中配置的信息。每个模块中包含仿真系统元素唯一 ID（Guid），经纬度，名字等不同的字段，方便选手进行自定义脚本编程中使用。通过此种可配置脚本输入源的方式，选手可以考虑编写通用化脚本，方便应对战时复杂的态势情况。

支持丰富的编程控制接口。智能决策开发平台提供 40 个接口，覆盖 7 大类功能。接口的设计尽可能简化使用方式，又尽可能多的保留参数控制粒度，方便选手不同程度的编程控制。通过提供的编程接口，选手可以进行分析、任务、单元、武器、条令、平台等多种指令控制，精细化操纵己方推演单元。具体的接口设计与使用详见后续章节说明。

2.2 操作流程

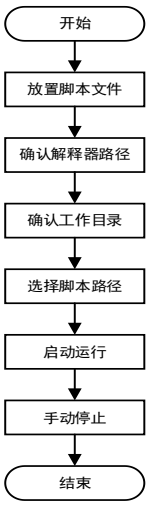


图 2 操作流程图

点击客户端顶部菜单栏“推演”按钮,选择“智能脚本”,打开智能体脚本配置界面。将选手的态势分析脚本、智能决策脚本分别拷贝到“安装目录\intelligent\situation_analysis\agent\script ”、 “安装目录\intelligent_decision-making\agent\sample”目录下。点击智能体脚本配置界面中的“添加”按钮，选择指定路径下的脚本文件。点击“打开”按钮，在脚本配置界面中显示所选脚本，完成加载。

点击脚本对应的“启动”按钮，或“配置脚本”按钮，进入脚本参数配置界面。选择“实体单元”、“情报信息”、“参考点”，然后点击“向右箭头”，添加脚本参数。选择想要删除的参数，点击“向左”按钮，删除对应参数。点击“确定”按钮，完成脚本参数配置。

点击“启动”按钮，配置完成参数后，脚本的状态由“未启动”转换为“启动中”，表示脚本启动成功。点击“停止”按钮，脚本的状态由为“启动中”转换为“未启动”，表示脚本停止成功。

三、 编程说明

3.1 文件说明

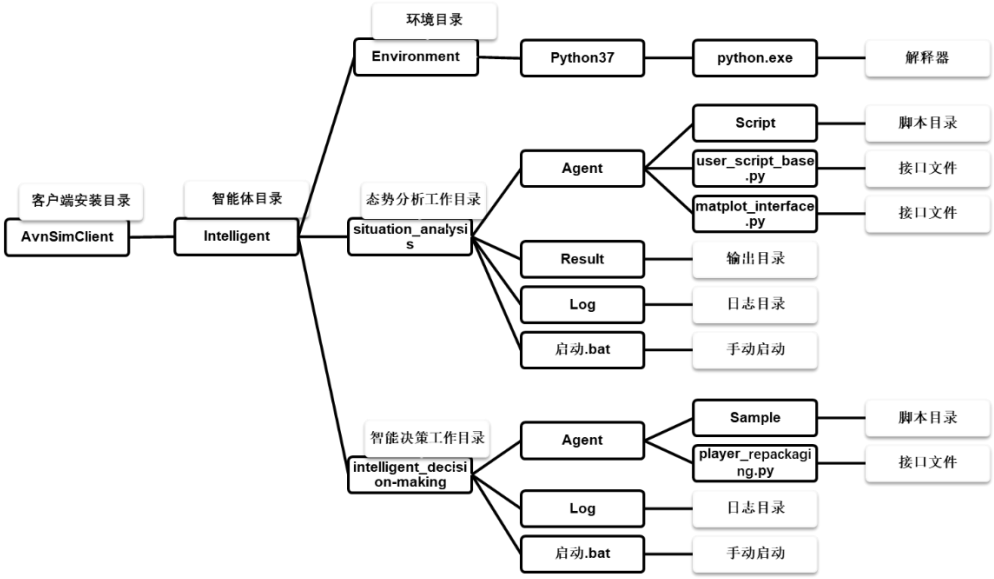


图 3 文件结构图

智能体开发平台的所有相关文件皆放在客户端安装目录（AvnSimClient）下的智能体目录（Intelligent）中。如果选手安装智能体开发平台，需要注意安装的路径是否正确，目录层级是否与文件结构图保持一致。

系统自动为用户配置好启动 python 程序所需的环境目录、工作目录、解释器路径、运行文件、配置文件等内容。只需选手按照推荐的文件目录安装智能体开发平台以及智能体脚本文件，即可使用客户端进行可视化运行控制操作。

在态势分析智能体脚本开发过程中需要关注的内容有：态势分析工作目录（situation_analysis），脚本和接口文件目录（Agent），基础类接口文件（user_script_base.py），绘图类接口文件（matplot_interface.py），示例脚本和选手自定义脚本目录（Script），日志文件目录（Log），分析结果输出目录（Result）。

在智能决策智能体脚本开发过程中需要关注的内容有：智能决策工作目录（Intelligent_decision-making），脚本和接口文件目录（Agent）， 全部接口的文件(player_repackaging.py)， 示例脚本和选手自定义脚本目录(Sample)，日志文件目录（Log）。

3.2 接口设计

3.2.1 态势分析

接口设计-态势分析			
序号	类别	接口说明	接口名
1	基础类	初始运行调用	reset
2		每步运行调用	run
3		结束运行调用	finish
4		获取态势	get_situation
5		查询是否结束	get_finish
6		主动结束脚本	set_finish
7		获取步进粒度	get_run_granularity
8		设置步进粒度	set_run_granularity
9		获取态势保存地址	get_situation_path
10		设置态势保存地址	set_situation_path
11		态势分析 demo	run_analysis_demo
12		获取敌方详细信息	get_targets_info
13		获取敌方统计信息	get_targets_count
14		获取己方详细信息	get_own_units_info
15		获取己方统计信息	get_own_units_count
16		设置图表并显示	set_count_display
17		保存分析结果到文件	save_analysis_result
18		清除文件内容	clear_analysis_result
19	绘图类	新建窗口	new_fig
20		新建画图板	new_subplot
21		新建饼图	new_bar
22		新建直方图	new_pie
23		饼图绘图简化接口	easy_bar

24		直方图绘图简化接口	easy_pie
25		显示	show
26		清除	clf
27		关闭	close
28		重绘	draw
29		暂停	pause

表 4 态势分析接口设计一览表

态势分析智能体开发平台支持两类，共 29 个接口。具体编程接口定义分别在基础类接口文件（user_script_base.py），绘图类接口文件（matplot_interface.py）中。使用说明详见《兵棋大赛态势分析模块接口表-v1.0.0602.docx》。

3.2.2 智能决策

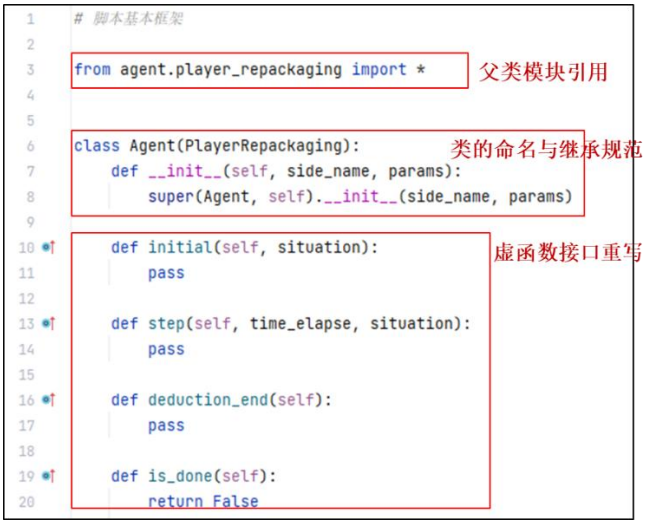
接口设计-智能决策			
序号	**类别**	**接口说明**	**接口名**
1	基础类	初始运行调用	initial
2		每步运行调用	step
3		结束运行调用	deduction_end
4		主动结束脚本	is_done
5		获取前端输入源	api_get_client_select_info
6	分析示例	分析态势	api_situation_analysis_demo
7		分配己方飞机	api_air_army_assign_demo
8		按类型获取己方军队	api_common_army_assign_demo
9		按类型获取敌方目标	api_target_assign_demo
10	任务类	打击任务	api_deploy_strike_mission
11		巡逻任务	api_deploy_patrol_mission
12		支援任务	api_deploy_support_mission
13		转场任务	api_deploy_ferry_mission

14		布雷任务	api_deploy_mine_mission
15		扫雷任务	api_deploy_mine_clear_mission
16		运输任务	api_deploy_cargo_mission
17	条令类	默认	api_doctrine_operation_default
18		电磁管控设置	api_doctrine_electromagnetism
19		武器状态	api_doctrine_weapon
20		接战	api_doctrine_battle
21		通用设置	api_doctrine_common
22		规划与返航	api_doctrine_plan_return
23		反潜作战行动	api_doctrine_navigation_subsurface
24		撤退	api_doctrine_withdraw
25		重新部署	api_doctrine_redeploy
26	武器类	武器使用规则	api_weapon_rule
27	单元类	推演方操作	api_unit_common
28		设置	api_unit_switch
29		航线、速度与高度	api_unit_control
30		攻击	api_unit_attack
31		脱战	api_unit_retreat
32		任务与基地	api_unit_mission_and_base
33		默认	api_unit_operation_default
34	通用类	出动和返航	api_common_out_and_return
35		情报操作	api_common_contact
36		平台操作	api_common_platform
37		任务操作	api_common_mission
38		参考点类操作	api_common_point
39		区域类操作	api_common_zone
40		编组类操作	api_common_group

表 5 智能决策接口设计一览表

智能决策智能体开发平台支持 7 类，共 40 个接口。编程接口定义在接口文件（player_repackaging.py）中。使用说明详见《兵棋大赛智能决策模块接口表-v1.0.0601.docx》。

3.3 编写规范




```
# 脚本基本框架

from agent.player_repackaging import *


class Agent(PlayerRepackaging):
    def __init__(self, side_name, params):
        super(Agent, self).__init__(side_name, params)

    def initial(self, situation):
        pass

    def step(self, time_elapse, situation):
        pass

    def deduction_end(self):
        pass

    def is_done(self):
        return False
```

图 4 脚本基本框架图

3.3.1 文件命名规范

选手自定义脚本文件命名不得包含中文，需要使用英文字母、数字、下划线组合，满足 python 规范与标准。

3.3.2 父类模块引用

选手自定义智能决策脚本必须引用 player_repackaging 接口文件，所有开放编程接口可在文件中具体查看定义方法。其它模块引用，根据选手需求自行添加。态势分析脚本类似。

3.3.3 类的命名与继承规范

智能决策类名必须使用 Agent，且必须继承自 player_repackaging 接口文件中的 PlayerRepacking 接口类。__init__方法与图中定义保持一致。态势分析脚本类似。

3.3.4 虚函数接口重写

选手自定义智能决策脚本中必须重写 initial，step，deduction_end，is_done 函数。函数 initial 方法会在智能体初始化时调用；函数 step 会在仿真推演过程中步进调用；函数 deduction_end 方法会在智能体脚本结束时调用；

函数 is_done 方法会在仿真推演过程中步进调用，根据返回值判断是否结束脚本运行，此方法用于编程控制脚本结束。选手可进行重写，并在预期情况下自行调用。

四、 案例展示

4.1 分析效果

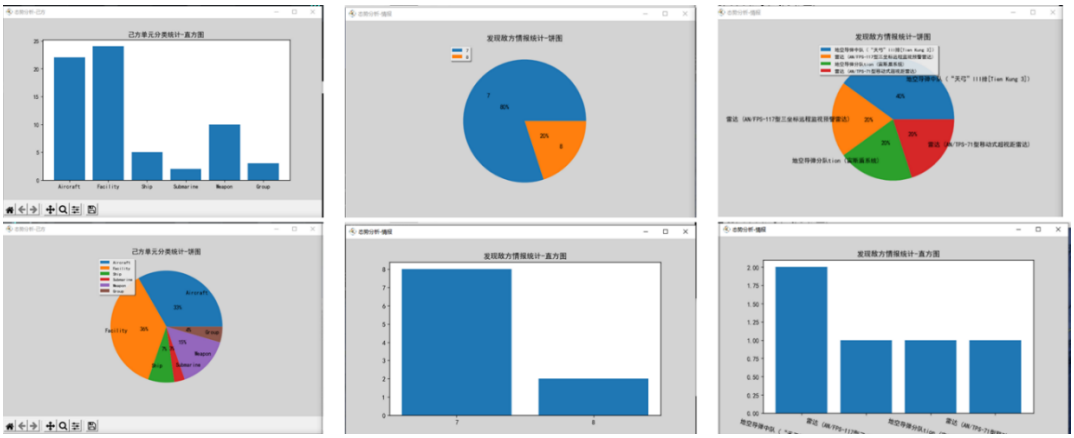


图 5 态势分析结果可视化界面

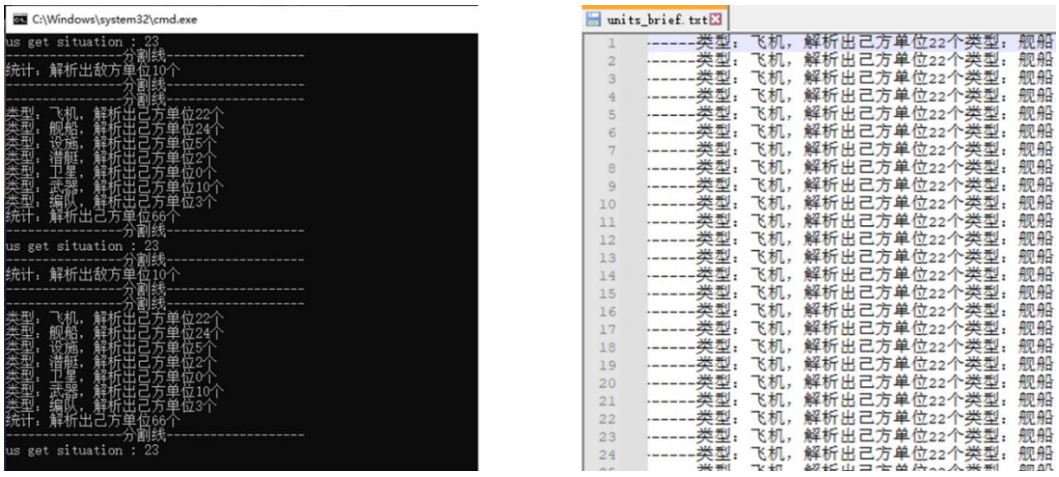


图 6 态势分析结果输出形式

4.2 决策效果

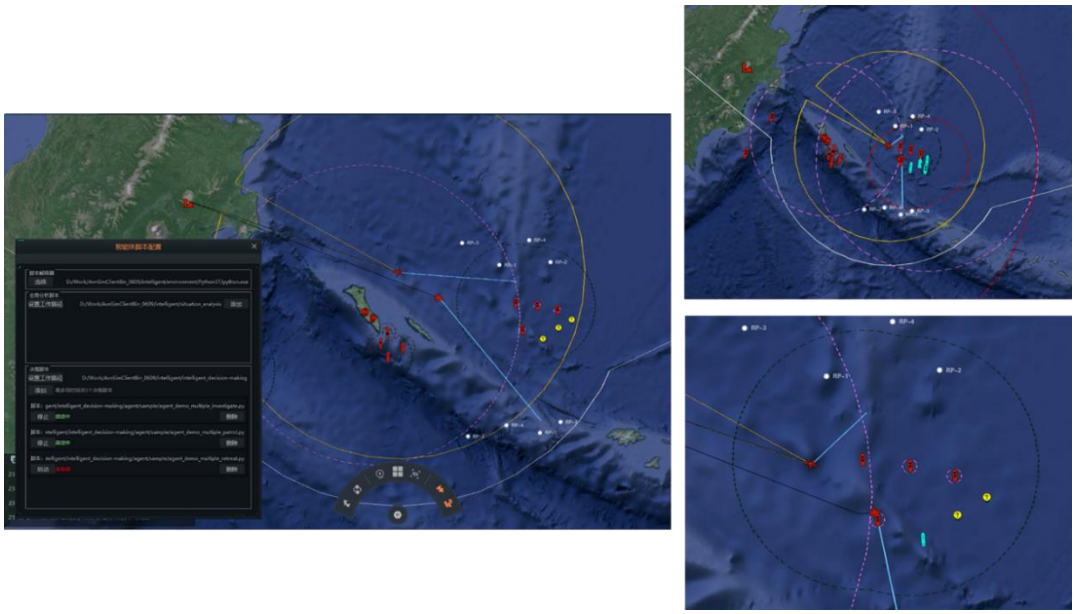


图 7 智能决策脚本运行效果

附件一:

兵棋大赛智能决策模块接口表
V1.0

国防科大系统工程学院

中国电子科技集团公司第五十二研究所

北方自动控制技术研究所

二〇二三年六月

目 次

概述

类别	接口名	接口	参数数量
# 分析示例	*# 分析态势*	api_situation_analysis_demo(self)	0
	# 分配并获取己方飞机 guid 列表	api_air_army_assign_demo(self)	0
	# 根据类型获取己方军队 guid 列表	api_common_army_assign_demo(self	1
	# 根据类型获取敌方目标 guid 列表	api_target_assign_demo(self	1
# 任务类	*# 打击任务*	api_deploy_strike_mission(self	27
	# 巡逻任务	api_deploy_patrol_mission(self	23
	# 支援任务	api_deploy_support_mission(self	18
	# 转场任务	api_deploy_ferry_mission(self	18
	# 布雷任务	api_deploy_mine_mission(self	22
	# 扫雷任务	api_deploy_mine_clear_mission(self	22
	# 运输任务	api_deploy_cargo_mission(self	13
# 条令类	*# 默认*	api_doctrine_operation_default(self)	0
	# 电磁管控设置	api_doctrine_electromagnetism(self	5
	# 武器状态	api_doctrine_weapon(self	4
	# 接战	api_doctrine_battle(self	3
	# 通用设置	api_doctrine_common(self	5
	# 规划与返航	api_doctrine_plan_return(self	7
	# 反潜作战行动	api_doctrine_navigation_subsurface(self	7
	# 撤退	api_doctrine_withdraw(self	4
	# 重新部署	api_doctrine_redeploy(self	4
# 武器类	*# 武器使用规则*	api_weapon_rule(self	7
# 单元	*# 推演方操作*	api_unit_common(self	4
	# 设置	api_unit_switch(self	11

	# 航线、速度与高度	api_unit_control(self	5
	# 攻击	api_unit_attack(self	7
	# 脱战	api_unit_retreat(self	6
	# 任务与基地	api_unit_mission_and_base(self	4
	# 默认	api_unit_operation_default(self	14
# 通用类	*# 出动和返航*	api_common_out_and_return(self	5
	# 情报操作	api_common_contact(self	5
	# 平台操作	api_common_platform(self	9
	# 任务操作	api_common_mission(self	6
	# 参考点类操作	api_common_point(self	4
	# 区域类操作	api_common_zone(self	11
	# 编组类操作	api_common_group(self	4

一、 通用类

1.1 平台操作

- 接口定义

```
api_common_platform(
        self,
        unit_guid: str = None,
        unit_category_type: InfoSequence = None,
        is_get_unit: bool = False,
        is_get_unit_info: bool = False,
        is_get_units: bool = False,
        is_get_situation: bool = False,
        is_get_weather: bool = False,
        is_get_scenarios: bool = False,
        is_get_message_log: bool = False
)
```

- 获取实体

```
self. api_common_platform(is_get_unit=True,unit_guid = “单元 GUID”)
```

- 查询单元信息

```
self. api_common_platform(is_get_unit_info=True,unit_guid =
```

```
“单元 GUID”,unit_category_type = InfoSequence 可选枚举)
```

- 获取本方所有实体信息

```
self. api_common_platform(is_get_units = True)
```

- 获取当前态势

```
self. api_common_platform(is_get_situation = True)
```

- 获取当前天气

```
self. api_common_platform(is_get_weather = True)
```

- 获取想定列表

```
self. api_common_platform(is_get_scenarios = True)
```

- 查询消息日志

```
self. api_common_platform(is_get_message_log = True)
```

1.2 情报操作

- 接口定义

```
def api_common_contact(
        self,
        contact_guid: str,
        new_name: str = str(),
        is_mark_position: bool = False,
        contact_stance: ContactStance = None,
        is_drop_target: bool = False
)
```

- 情报标识重命名

```
self. api_common_contact(contact_guid =“情报 GUID”, new_name =”名字”)
```

- 情报标识标记位置

```
self. api_common_contact(contact_guid =“情报 GUID”, is_mark_position=True)
```

- 标识情报阵营

```
self. api_common_contact(contact_guid =“情报 GUID”, contact_stance=ContactStance 可选列表)
```

- 断开情报标识

```
self. api_common_contact(contact_guid =“情报 GUID”, is_drop_target=True)
```

1.3 出动和返航操作

- 接口定义

```
def api_common_out_and_return(
        self,
        unit_guid: str,
        out_units_guid: List[str] = None,
        is_all_units_hold_position: bool = False,
        new_base_guid: str = str(),
        is_unit_return: bool = False
)
```

- 飞机单机出动

```
self. api_common_out_and_return (unit_guid =“单元 GUID”)
```

- 飞机单机返航

```
self. api_common_out_and_return (unit_guid =“单元 GUID” , is_unit_return=True)
```

- 飞机编组出动

```
self. api_common_out_and_return (out_units_guid =[“单元 GUID”, “单元 GUID”,…])
```

- 单元选择新基地/新港口

```
self. api_common_out_and_return (unit_guid =“单元 GUID”, new_base_guid=”基地 GUID”)
```

- 保持所有单元阵位，所有单元停止机动，留在原地

```
self. api_common_out_and_return (is_all_units_hold_position=True)
```

1.4 任务操作

- 接口定义

```
def api_common_mission(
        self,
        mission_name: str,
        is_del_mission: bool = False,
        is_get_mission: bool = False,
        ass_unit_mission: str = str(),
        del_unit_from_mission: str = str(),
        point_list: list = None
)
```

- 删除任务

```
self. api_common_out_and_return (mission_name =“任务名字”, is_del_mission=True)
```

- 查询任务:通过任务名获取任务对象

```
self. api_common_out_and_return (mission_name =“任务名字”, is_get_mission=True)
```

- 将单元分配到任务

```
self. api_common_out_and_return (mission_name =“任务名字”, ass_unit_mission=”单元GUID”)
```

- 将单元取消分配任务

```
self. api_common_out_and_return (mission_name =“任务名字”, del_unit_from_mission=”单元GUID”)
```

- 增加巡逻任务的警戒区

```
self. api_common_out_and_return (mission_name =“任务名字”, point_list=[ (纬度,经度), (纬度,经度),…])
```

1.5 参考点操作

- 接口定义

```
def api_common_point(
        self,
        add_points: tuple or list = None,
        move_points_by_name: str = None,
        move_points_new_coordinate: tuple = None,
        del_point_by_name: str = None
)
```

- 添加一个或多个参考点

```
self. api_common_point (add_points=” (40.2, 49.6) 或 [(40, 39.0), (41, 39.0)] 或 [(40, 39.0, 'RP1'), (41, 39.0, 'RP2')]，其中纬度值在前，经度值在后”)
```

- 移动参考点的位置

```
self. api_common_point (move_points_by_name=”参考点名”, move_points_new_coordinate=” tuple, 新的经纬度位置 (lat, lon)”)
```

- 删除参考点

```
self. api_common_point (del_point_by_name=”参考点名”)
```

1.6 区域操作

- 接口定义

```
def api_common_zone(
        self,
        add_zone_name: str = None,
        add_zone_points_list: list = None,
        add_zone_type: int = int(),
        set_zone_name: str = None,
```

```
        set_zone_new_name: str = None,
        set_zone_type: int = int(),
        set_zone_is_active: bool = True,
        set_zone_is_locked: bool = True,
        set_zone_mark_as: int = int(),
        remove_zone_type: int = None,
        remove_zone_guid: str = None
)
```

- 删除参考点

```
self. api_common_ zone (add_zone_name =区域名,
        add_zone_points_list = list, 区域点列表,
        add_zone_type = 1: 封锁区   0:禁航区,)
```

- 删除参考点

```
self. api_common_ zone (set_zone_name: str = None,
        set_zone_new_name = “新名字”,
        set_zone_type = int, 1: 封锁区   0:禁航区,
        set_zone_is_active = bool 是否启用,
        set_zone_is_locked = bool 是否区域已锁定(只有禁航区才需要设置),
        set_zone_mark_as= int 2=非友方，3=敌方（封锁区才有的设置）,
)
```

- 删除参考点

```
self. api_common_ zone (remove_zone_type =” int, 1:封锁区，0：禁航区
”, remove_zone_guid=” str, 区域的 guid”)
```

1.7 编组操作

- 接口定义

```
def api_common_group(
        self,
        add_unit_guid_list: list = None,
        detach_unit_guid_list: list = None,
        remove_unit_guid: str = None,
        add_unit_tuple: tuple = None
)
```

- 将多个单元作为一个编队

```
self. api_common_group(add_unit_guid_list=[“guid”,”guid”,…] )
```

- 分离编组单元

```
self. api_common_group(detach_unit_guid_list=[“guid”,”guid”,…])
```

- 将单元移除出编队

```
self. api_common_group(remove_unit_guid=”单元 GUID”)
```

- 编队添加一个单元

```
self. api_common_group(add_unit_tuple  = (”编队 guid”, ”作战单元 guid”))
```

二、 任务类

2.1 打击任务

- 接口定义

```
def api_deploy_strike_mission(
        self,
        mission_name: str = "打击任务-XX",  # 任务名
        mission_type: MissionStrikeType = MissionStrikeType.LAND,  #
MissionStrikeType, 打击任务类型，对空打击，对地打击等
        target_list: list = None,  # 设置打击目标
        start_time: str = "",  # 设置、删除任务开始时间
        end_time: str = "",  # 设置任务：删除任务结束时间
        ass_unit_id_list: list = None,  # 设置任务：将实体分配到任务中来
        un_ass_unit_id_list: list = None,  # 任务取消某单元的任务分配
        remove_target_list: list = None,  # 设置任务：删除打击任务目标
        switch_radar: bool = True,  # 设置任务雷达是否打开
        is_active: bool = True,  # 是否启用任务
        is_only_pre_plan=False,  # 设置任务细节：是否仅考虑计划目标（在目标清单）
        attack_condition: StrikeMinimumTrigger = StrikeMinimumTrigger.NotFriendly,  #
设置打击任务触发条件
        strike_max: StrikeFlyTimeMax = StrikeFlyTimeMax.TWO,  # 设置任务细节：任务允许出动
的最大飞行批次
        flight_size: FlightSize = FlightSize.Two,  # 设置打击任务飞机编队规模
        min_aircraft_req: MinAircraftReq = MinAircraftReq.ONE,  # 设置打击任务所需最少飞
机数
        radar_usage: StrikeRadarUsage = StrikeRadarUsage.ALL_PLAN,  # 设置打击任务雷达运
用规则
        fuel_ammo: StrikeFuleAmmo = StrikeFuleAmmo.MOUNT_SET,  # 设置打击任务燃油弹药规则
        min_dist: int = 100,  # 设置打击任务最小打击半径 # 公里,int
        max_dist: int = 1000,  # 设置打击任务最大打击半径 # 公里,int
        use_flight_size: bool = False,  # 设置打击任务是否飞机数低于编组规模数要求就不能起飞
        use_auto_planner: bool = True,  # 设置打击任务是否多扇面攻击（任务 AI 自动生成）
        one_time_only: bool = True,  # 设置打击任务是否仅限一次
        ship_size: FlightSize = FlightSize.Two,  # 设置打击任务舰艇/潜艇编队规模
        use_ship_size: bool = False,  # 设置打击任务是否强制舰艇/潜艇编队规模
        escort_flight_size: FlightSize = FlightSize.Two,  # 设置打击任务护航飞机编队规模
        min_shooter: FlightSize = FlightSize.One,  # 设置打击护航任务所需最少飞机数
```

```
        max_shooter: FlightSize = FlightSize.Two,  # 设置打击护航任务所需最大飞机数
        response_radius: int = 500  # int,  最大威胁响应半径  海里
)
```

2.2 巡逻任务

- 接口定义

```
def api_deploy_patrol_mission(
        self,
        mission_name='巡逻任务-对空-XX',  # 任务名
        mission_type=MissionPatrolType.AIR,  # MissionPatrolType, 任务类型，对空打击，对地打击等
        point_list: list = None,  # list, 参考点列表
        start_time: str = "",  # 设置、删除任务开始时间
        end_time: str = "",  # 设置任务：删除任务结束时间
        ass_unit_id_list: list = None,  # 设置任务：将实体分配到任务中来
        un_ass_unit_id_list: list = None,  # 任务取消某单元的任务分配
        switch_radar: bool = True,  # 设置任务雷达是否打开
        is_active: bool = True,  # 是否启用任务
        one_third_rule: bool = False,  # 三分之一规则
        station_count: int = 6,  # 为保持阵位的数量
        check_OPA: bool = True,  # 设置任务是否对巡逻区外的探测目标进行分析
        active_EMCON: bool = True,  # 设置任务是否仅在巡逻/警戒区内打开电磁辐射
        check_WWR: bool = True,  # 设置任务是否对武器射程内探测目标进行分析
        flight_size: FlightSize = FlightSize.Two,  # 设置任务编队规模
        use_flight_size: bool = False,  # 是否飞机数低于编队规模不允许起飞
        throttle_transit: Throttle = Throttle.Unspecified,  # 设置任务的出航油门
        throttle_station: Throttle = Throttle.Unspecified,  # 设置任务的阵位油门
        throttle_attack: Throttle = Throttle.Unspecified,  # 设置任务的攻击油门
        transit_altitude: float = 1000,  # float, 出航高度,  单位：米，最多 6 位字符，例：99999.9,  888888
        station_altitude: float = 5000,  # float, 阵位高度,  单位：米，最多 6 位字符
        attack_altitude: float = 5000,  # float, 攻击高度,  单位：米，最多 6 位字符
        attack_distance: int = 500  # int, 攻击距离，单位：公里
)
```

2.3 支援任务

- 接口定义

```
def api_deploy_support_mission(
        self,
        mission_name='支援任务-XX',
```

```
        point_list: list = None,
        start_time: str = "",  # 设置、删除任务开始时间
        end_time: str = "",  # 设置任务：删除任务结束时间
        ass_unit_id_list: list = None,  # 设置任务：将实体分配到任务中来
        un_ass_unit_id_list: list = None,  # 任务取消某单元的任务分配
        switch_radar: bool = True,  # 设置任务雷达是否打开
        is_active: bool = True,  # 是否启用任务
        one_third_rule: bool = False,  # 三分之一规则
        station_count: int = 6,  # int, 保持阵位的数量
        one_time_only: bool = True,  # 设置任务是否仅限一次
        active_EMCON: bool = True,  # 仅在阵位上打开电磁辐射
        single_loop: bool = True,  # 导航类型
        flight_size: FlightSize = FlightSize.Two,  # 编队规模
        use_flight_size: bool = False,  # 是否飞机数低于编队规模不允许起飞
        throttle_transit: Throttle = Throttle.Unspecified,  # 设置任务的出航油门
        throttle_station: Throttle = Throttle.Unspecified,  # 设置任务的阵位油门
        transit_altitude: float = 1000,  # float, 出航高度， 单位：米，最多 6 位字符，例：
99999.9， 888888
        station_altitude: float = 5000  # float, 阵位高度， 单位：米，最多 6 位字符
)
```

2.4 转场任务

- 接口定义

```
def api_deploy_ferry_mission(
        self,
        mission_name='转场任务-XX',  # 任务名
        destination: str = "",  # str, 目的地 guid
        start_time: str = "",  # 设置、删除任务开始时间
        end_time: str = "",  # 设置任务：删除任务结束时间
        ass_unit_id_list: list = None,  # 设置任务：将实体分配到任务中来
        un_ass_unit_id_list: list = None,  # 任务取消某单元的任务分配
        switch_radar: bool = True,  # 设置任务雷达是否打开
        is_active: bool = True,  # 是否启用任务
        ferry_behavior=FerryBehavior.OneWay,  # 设置转场规则
        flight_size: FlightSize = FlightSize.Two,  # 设置任务编队规模
        use_flight_size: bool = False,  # 是否飞机数低于编队规模不允许起飞
        min_aircraft_req: MinAircraftReq = MinAircraftReq.ONE,  # 设置所需最少飞机数
        aircraft_throttle: Throttle = Throttle.Unspecified,  # 飞机转场油门
        ship_throttle: Throttle = Throttle.Unspecified,  # 舰艇转场油门
        submarine_throttle: Throttle = Throttle.Unspecified,  # 潜艇转场油门
        aircraft_altitude: float = 5000,  # 飞机转场高度, float
        submarine_altitude: float = 500,  # 潜艇转场深度,float
```

```
        terrain_follow=False  # 设置转场地形跟随
)
```

2.5 布雷任务

- 接口定义

```
def api_deploy_mine_mission(
        self,
        mission_name="布雷任务-XX",
        point_list: list = None,
        time_delay: str = "",  # str, exp: 以'天数:时数:分数:秒数'表示，如 1 天 4 小时 30 分钟表示为'1:4:30:0'
        start_time: str = "",  # 设置、删除任务开始时间
        end_time: str = "",  # 设置任务：删除任务结束时间
        ass_unit_id_list: list = None,  # 设置任务：将实体分配到任务中来
        un_ass_unit_id_list: list = None,  # 任务取消某单元的任务分配
        switch_radar: bool = True,  # 设置任务雷达是否打开
        is_active: bool = True,  # 是否启用任务
        one_third_rule: bool = False,  # 三分之一规则
        flight_size: FlightSize = FlightSize.Two,  # 设置任务编队规模
        use_flight_size: bool = False,  # 是否飞机数低于编队规模不允许起飞
        group_size: FlightSize = FlightSize.Two,  # 设置任务舰船/潜艇编队规模
        use_group_size: bool = False,  # 是否舰船/潜艇数低于编队规模不允许出发
        min_aircraft_req: MinAircraftReq = MinAircraftReq.TWO,  # 设置所需最少飞机数
        throttle_transit: Throttle = Throttle.Flank,  # 设置任务的出航油门
        throttle_station: Throttle = Throttle.Flank,  # 设置任务的阵位油门
        transit_altitude: float = 10000,  # float, 出航高度， 单位：米，最多 6 位字符，例：99999.9， 888888
        station_altitude: float = 10000,  # float, 阵位高度， 单位：米，最多 6 位字符
        is_submarine: bool = False,  # 是否出潜深度
        transit_terrain_following: bool = True,  # 设置出航地形跟随
        station_terrain_following: bool = True  # 设置阵位地形跟随
)
```

2.6 扫雷任务

- 接口定义

```
def api_deploy_mine_clear_mission(
        self,
        mission_name='扫雷任务-XX',
        point_list: list = None,
        start_time: str = "",  # 设置、删除任务开始时间
```

```
        end_time: str = "",  # 设置任务：删除任务结束时间
        ass_unit_id_list: list = None,  # 设置任务：将实体分配到任务中来
        un_ass_unit_id_list: list = None,  # 任务取消某单元的任务分配
        switch_radar: bool = True,  # 设置任务雷达是否打开
        is_active: bool = True,  # 是否启用任务
        one_third_rule: bool = False,  # 三分之一规则
        flight_size: FlightSize = FlightSize.Two,  # 设置任务编队规模
        use_flight_size: bool = False,  # 是否飞机数低于编队规模不允许起飞
        group_size: FlightSize = FlightSize.Two,  # 设置任务舰船/潜艇编队规模
        use_group_size: bool = False,  # 是否舰船/潜艇数低于编队规模不允许出发
        min_aircraft_req: MinAircraftReq = MinAircraftReq.TWO,  # 设置所需最少飞机数
        throttle_transit: Throttle = Throttle.Flank,  # 设置任务的出航油门
        throttle_station: Throttle = Throttle.Flank,  # 设置任务的阵位油门
        unit_category: ElementType = ElementType.Aircraft,  # 单元类型
        transit_altitude: float = 10000,  # float，出航高度， 单位：米，最多 6 位字符，例：
99999.9， 888888
        station_altitude: float = 10000,  # float，阵位高度， 单位：米，最多 6 位字符
        is_submarine: bool = False,  # bool，是否出潜深度
        transit_terrain_following: bool = False,  # 设置出航地形跟随
        station_terrain_following: bool = False  # 设置阵位地形跟随
)
```

2.7 运输任务

- 接口定义

```
def api_deploy_cargo_mission(
        self,
        mission_name='货运任务-XX',
        point_list: list = None,
        start_time: str = "",  # 设置、删除任务开始时间
        end_time: str = "",  # 设置任务：删除任务结束时间
        ass_unit_id_list: list = None,  # 设置任务：将实体分配到任务中来
        un_ass_unit_id_list: list = None,  # 任务取消某单元的任务分配
        switch_radar: bool = True,  # 设置任务雷达是否打开
        is_active: bool = True,  # 是否启用任务
        throttle_transit: Throttle = Throttle.Flank,  # 设置任务的出航油门
        throttle_station: Throttle = Throttle.Flank,  # 设置任务的阵位油门
        unit_category: ElementType = ElementType.Aircraft, transit_altitude: float =
10000,
        # float，出航高度， 单位：米，最多 6 位字符，例：99999.9， 888888
        station_altitude: float = 10000  # float，阵位高度， 单位：米，最多 6 位字符
)
```

三、 条令类

3.1 默认设置

- 接口定义

```
def api_doctrine_operation_default(self):
        # 条令类操作 : operator base.py

        # 电磁管控设置
        self.doctrine_switch_radar(True)  # 条令中，电磁管控设置，设置雷达开关机
        self.doctrine_switch_oecm(True)  # 条令中，电磁管控设置，设置电子干扰传感器开关机
        self.doctrine_switch_sonar(True)  # 条令中，电磁管控设置，设置声呐开关机
        self.doctrine_SetEMCON_Inherit(True)  # 设置电磁管控是否与上级一致
        self.doctrine_ignore_emcon_under_attack(
            IgnoreEMCONUnderAttack.Ignore_EMCON_While_Under_Attack)  # 受到攻击忽略电磁管
控

        # 武器状态
        self.doctrine_weapon_control_status_air(WeaponControlStatus.Free)  # 武器控制状
态，对空
        self.doctrine_weapon_control_status_surface(WeaponControlStatus.Free)  # 武器
控制状态，对水面
        self.doctrine_weapon_control_status_subsurface(WeaponControlStatus.Free)  # 武
器控制状态，对潜
        self.doctrine_weapon_control_status_land(WeaponControlStatus.Free)  # 武器控制
状态，对地

        # 接战
        self.doctrine_ignore_plotted_course(IgnorePlottedCourseWhenAttacking.Yes)  #
攻击时忽略计划航线设置
        self.doctrine_engage_opportunity_targets
            (EngageWithContactTarget.Yes_AnyTarget)  # 接战临机出现目标
        self.doctrine_engaging_ambiguous_targets
            (BehaviorTowardsAmbigousTarget.Optimistic)  # 接战模糊位置目标

        # 设置
        self.use_nuclear_weapons(EUseNuclear.Yes)  # 条令设置，是否使用核武器
        self.doctrine_automatic_evasion(AutomaticEvasion.Yes)  # 自动规避
        self.doctrine_air_operations_tempo(AirOpsTempo.Surge)  # 空战节奏
        self.doctrine_gun_strafing(GunStrafeGroundTargets.Yes)  # 空对地扫射
        self.torpedoes_kinematic_range(UseTorpedoesKinematicRange.Yes)  # 鱼雷射程
```

```
        # 规划与返航
        self.doctrine_refuel(ERefuel.Yes)  # 加油/途中补给
        self.doctrine_refuel_select(ERefuelSelect.Include_NoBack)  # 加油/途中补给
        self.doctrine_refuel_allied(ERefuelAllied.YesReceiveOnly)  # 给盟军加油/途中补给
        self.doctrine_fuel_state_planned(FuelState.Joker10Percent)  # 燃油状态，预先规划
        self.doctrine_fuel_state_rtb(FuelStateRTB.No)  # 燃油状态，返航
        self.doctrine_weapon_state_planned(WeaponStatePlanned.NoneValue)  # 武器状态，
预先规划
        self.doctrine_weapon_state_rtb(WeaponStateRTB.No)  # 武器状态-返航

        # 反潜作战行动
        self.avoid_contact(AvoidContact.Inherit)  # 反潜作战行动，规避搜索
        self.dive_on_threat(DiveOnThreat.Inherit)  # 反潜作战行动，探测到威胁进行下潜
        self.recharge_on_patrol(RechargePercent.Inherit)  # 反潜作战行动，充电电池 运输/站
点
        self.recharge_on_attack(RechargePercent.Inherit)
        self.use_aip(UseAIP.Inherit)  # 反潜作战行动，充电电池 进攻/防守
        self.dipping_sonar(DippingSonar.Inherit)  # 反潜作战行动， 吊放声呐
        self.navigation_sub(NavigationSub.Inherit)  # 反潜作战行动， 导航

        # 撤退
        self.withdraw_on_damage(DamageThreshold.Ignore)  # 满足如下条件时撤退 - -毁伤程度
大于
        self.withdraw_on_fuel(FuelQuantityThreshold.Ignore)  # 满足如下条件时撤退--燃油少
于
        self.withdraw_on_attack(WeaponQuantityThreshold.Ignore)  # 满足如下条件时撤退--主
要攻击攻击武器至少处于
        self.withdraw_on_defence(WeaponQuantityThreshold.Ignore)  # 满足如下条件时撤退--
主要防御武器至少

        # 重新部署
        self.deploy_on_damage(DamageThreshold.Ignore)  # 满足如下条件时重新部署--毁伤程度小
于
        self.deploy_on_fuel(FuelQuantityThreshold.Ignore)  # 满足如下条件时重新部署--燃油
至少处于
        self.deploy_on_attack(WeaponQuantityThreshold.Ignore)  # 满足如下条件时重新部署--
主要攻击武器处于
        self.deploy_on_defence(WeaponQuantityThreshold.Ignore)  # 满足如下条件时重新部署-
-主要防御武器处于
```

3.2 电磁管控设置

- 接口定义

```
def api_doctrine_electromagnetism(
        self,
        switch_radar: bool = True,  # 条令中，电磁管控设置，设置雷达开关机
        switch_OECM: bool = True,  # 条令中，电磁管控设置，设置电子干扰传感器开关机
        switch_sonar: bool = True,  # 条令中，电磁管控设置，设置声呐开关机
        set_EMCON_inherit: bool = True,  # 设置电磁管控是否与上级一致
        ignore_EMCON_under_attack: IgnoreEMCONUnderAttack
        = IgnoreEMCONUnderAttack.Ignore_EMCON_While_Under_Attack  # 受到攻击忽略电磁管控
)
```

3.3 武器状态设置

- 接口定义

```
def api_doctrine_weapon(
        self,
        to_air: WeaponControlStatus = WeaponControlStatus.Free,  # 武器控制状态，对空
        to_surface: WeaponControlStatus = WeaponControlStatus.Free,  # 武器控制状态，对水面
        to_subsurface: WeaponControlStatus = WeaponControlStatus.Free,  # 武器控制状态，对潜
        to_land: WeaponControlStatus = WeaponControlStatus.Free  # 武器控制状态，对地
)
```

3.4 接战设置

- 接口定义

```
def api_doctrine_battle(
        self,
        ignore_plotted_course: IgnorePlottedCourseWhenAttacking
        = IgnorePlottedCourseWhenAttacking.Yes,  # 攻击时忽略计划航线设置
        engage_opportunity_targets: EngageWithContactTarget
        = EngageWithContactTarget.Yes_AnyTarget,  # 接战临机出现目标
        engaging_ambiguous_targets: BehaviorTowardsAmbigousTarget
        = BehaviorTowardsAmbigousTarget.Optimistic  # 接战模糊位置目标
)
```

3.5 通用设置

- 接口定义

```
def api_doctrine_common(
        self,
        nuclear_weapons: EUseNuclear = EUseNuclear.Yes,  # 条令设置，是否使用核武器
        automatic_evasion: AutomaticEvasion = AutomaticEvasion.Yes,  # 自动规避
        air_operations_tempo: AirOpsTempo = AirOpsTempo.Surge,  # 空战节奏
        gun_strafing: GunStrafeGroundTargets = GunStrafeGroundTargets.Yes,  # 空对地扫
射
        kinematic_range: UseTorpedoesKinematicRange = UseTorpedoesKinematicRange.Yes
# 鱼雷射程
)
```

3.6 规划与返航设置

- 接口定义

```
def api_doctrine_plan_return(
        self,
        refuel: ERefuel = ERefuel.Yes,  # 加油/途中补给
        refuel_select: ERefuelSelect = ERefuelSelect.Include_NoBack,  # 加油/途中补给
        refuel_allied: ERefuelAllied = ERefuelAllied.YesReceiveOnly,  # 给盟军加油/途中
补给
        fuel_state_planned: FuelState = FuelState.Joker10Percent,  # 燃油状态，预先规划
        fuel_state_return: FuelStateRTB = FuelStateRTB.No,  # 燃油状态，返航
        weapon_state_planned: WeaponStatePlanned = WeaponStatePlanned.NoneValue,  # 武
器状态，预先规划
        weapon_state_return: WeaponStateRTB = WeaponStateRTB.No  # 武器状态-返航
)
```

3.7 反潜作战行动设置

- 接口定义

```
def api_doctrine_navigation_subsurface(
        self,
        avoid_contact: AvoidContact = AvoidContact.Inherit,  # 反潜作战行动，规避搜索
        dive_on_threat: DiveOnThreat = DiveOnThreat.Inherit,  # 反潜作战行动，探测到威胁进
行下潜
        recharge_on_patrol: RechargePercent = RechargePercent.Inherit,  # 反潜作战行动，
充电电池 运输/站点
```

```
        recharge_on_attack: RechargePercent = RechargePercent.Inherit,  # 反潜作战行动，充电电池 进攻/防守
        use_aip: UseAIP = UseAIP.Inherit,  # 反潜作战行动，使用 AIP 技术
        dipping_sonar: DippingSonar = DippingSonar.Inherit,  # 反潜作战行动， 吊放声呐
        navigation_sub: NavigationSub = NavigationSub.Inherit  # 反潜作战行动， 导航
)
```

3.8 撤退设置

- 接口定义

```
def api_doctrine_withdraw(
        self,
        damage: DamageThreshold = DamageThreshold.Ignore,  # 满足如下条件时撤退 - -毁伤程度大于
        fuel: FuelQuantityThreshold = FuelQuantityThreshold.Ignore,  # 满足如下条件时撤退--燃油少于
        attack_weapon: WeaponQuantityThreshold = WeaponQuantityThreshold.Ignore,  # 满足如下条件时撤退--主要攻击攻击武器至少处于
        defence_weapon: WeaponQuantityThreshold = WeaponQuantityThreshold.Ignore  # 满足如下条件时撤退--主要防御武器至少
)
```

3.9 重新部署设置

- 接口定义

```
def api_doctrine_redeploy(
        self,
        damage: DamageThreshold = DamageThreshold.Ignore,  # 满足如下条件时重新部署--毁伤程度小于
        fuel: FuelQuantityThreshold = FuelQuantityThreshold.Ignore,  # 满足如下条件时重新部署--燃油至少处于
        attack_weapon: WeaponQuantityThreshold = WeaponQuantityThreshold.Ignore,  # 满足如下条件时重新部署--主要攻击武器处于
        defence_weapon: WeaponQuantityThreshold = WeaponQuantityThreshold.Ignore  # 满足如下条件时重新部署--主要防御武器处于
)
```

四、 武器类

4.1 武器使用规则

- 接口定义

```
def api_weapon_rule(
        self,
        weapon_dbid: str = "",  # {str: 武器的数据库 ID}
        target_type=WRA_WeaponTargetType.NoneValue,  # {str: 目标类型号}
WRA_WeaponTargetType 的值，例如：飞机未指定："2000"
        quantity_salvo=2,  # {str: 'n'-齐射武器数, 'inherit'-继承设置, 'max'-全量齐射,
'none'-禁用}
        shooter_salvo=1,  # {str: 'n'-齐射发射架数, 'inherit'-继承设置, 'max'-全量齐射}
        firing_range='none',  # {str: 'n'-自动开火距离, 'inherit'-继承设置, 'none'-禁用自
动开火}
        self_defense='max',  # {str: 'n'-自动防御距离, 'inherit'-继承设置, 'max'-最大射程
射击, 'none'-禁用自卫}
        escort='false'  # 是否护航任务 {str: 'true'-护航任务, 'false'-非护航任务}
)
```

五、 单元类

5.1 推演方操作

- 接口定义

```
def api_unit_common(
        self,
        unit_guid: str = str(),
        new_name: str = str(),
        contact_guid: str = str(),
        units_contacts: tuple or list = None
)
```

- 修改单元名称

```
self. api_unit_common (unit_guid =”单元 GUID”, new_name =”新命名”)
```

- 开火条件检查

```
self. api_unit_common (units_contacts= list or tuple, example: [(unit1, target1),(unit2, target2)] or (unit1, target1))
```

- 获取情报对象:

```
self. api_unit_common (unit_guid =”单元 GUID”, contact_guid=”情报对象 guid ")
```

- # 获取实体

```
self. api_unit_common (unit_guid =”单元 GUID”)
```

5.2 单元开关操作

- 接口定义

```
def api_unit_switch(
        self,
        unit_guid: str,# ”单元 GUID”
        emcon_abide: bool = True, # 实体传感器面板， 实体是否遵循电磁管控条令
        radar: bool = True, # 实体传感器面板， 实体雷达开关机
        sonar: bool = True, # 实体传感器面板， 实体声呐开关机
        oecm: bool = True, # 实体传感器面板， 实体电子干扰开关机
        follow_terrain: bool = True, # 设置当前单元（飞机）的飞行高度跟随地形
        altitude_manual: bool = True, # 设置飞机或者潜艇手动控制高度
        hold_position: bool = True, # 保持阵位-所选单元
        sonar_type: SonarType = SonarType.Active, # 反潜作战行动,声呐类型

        is_shallow: bool = True, # 反潜作战行动, 是否浅层
        re_fuel_unit_guid: str = str()# 加油/补给
)
```

5.3 航线、速度与高度操作

- 接口定义

```
def api_unit_control(
        self,
        unit_guid: str,
        course_list: list = None,
        throttle: Throttle = None,
        desired_speed: float = 0,
        desired_height: float = 0
)
```

- 实体航线规划

```
self. api_unit_control(unit_guid =”单元 GUID”, course_list = list, [(lat, lon)])
```

- 期望速度

```
self. api_unit_control(unit_guid =”单元 GUID”, desired_speed =” float, 千米/小时”)
```

- 设置实体油门

```
self. api_unit_control(unit_guid =”单元 GUID”, throttle = Throttle 可选列表)
```

- 期望高度值

```
self. api_unit_control(unit_guid =”单元 GUID”, desired_height =海拔高度：m)
```

5.4 攻击操作

- 接口定义

```
def api_unit_attack(
        self,
        unit_guid: str,
        contact_guid: str,
        attack_way: AttackWay = AttackWay.Auto,
        weapon_dbid: int = int(),
        mount_dbid: int = int(), qty_num: int = int(),
        bol_coordinate: (float, float) = (float(), float())
)
```

- 收集攻击信息

```
self. api_unit_attack (unit_guid =”单元 GUID”, contact_guid=”情报单元 GUID”, attack_way
= AttackWay.TargetInfo)
```

- 自动攻击目标

```
self. api_unit_attack (unit_guid =”单元 GUID” contact_guid=”情报单元 GUID”)
```

- 手动攻击(指定 mount)

```
self. api_unit_attack (unit_guid =”单元 GUID” , contact_guid=”情报单元 GUID”,
attack_way = AttackWay. Manual, mount_dbid=”int, 攻击者的装具 DBID”,
weapon_dbid=”int, 攻击者的武器 DBID”, qty_num=”int,分配数量”)
```

- 手动攻击

```
self. api_unit_attack (unit_guid =”单元 GUID” , contact_guid=”情报单元 GUID”,
attack_way = AttackWay. Manual, weapon_dbid=”int, 攻击者的武器 DBID”, qty_num=”int,分
配数量”)
```

- 纯方位攻击

```
self. api_unit_attack (unit_guid =”单元 GUID” , contact_guid=”情报单元 GUID”,
attack_way = AttackWay. Bol, bol_coordinate = “tuple,(纬度,经度)” weapon_dbid=”int,
攻击者的武器 DBID”, qty_num=”int,分配数量”)
```

5.5 脱战操作

- 接口定义

```
def api_unit_retreat(
        self,
        unit_guid: str,
        abandon_attack_guid: str = str(),
        abandon_all_attack: bool = False,
        abandon_all_mission: bool = False,
        clear_all_course: bool = False,
        is_return: bool = False
)
```

- 实体放弃某个之前设定的目标

```
self. api_unit_ retreat (unit_guid =”单元 GUID”, abandon_attack_guid=”情报单元 GUID”)
```

- 将单元取消分配任务

```
self. api_unit_ retreat (unit_guid =”单元 GUID”, abandon_all_mission=True)
```

- 实体放弃所有目标，脱离接战

```
self. api_unit_ retreat (unit_guid =”单元 GUID”, abandon_all_attack=True)
```

- 单元清除航路点

```
self. api_unit_ retreat (unit_guid =”单元 GUID”, clear_all_course=True)
```

- 实体返航

```
self. api_unit_ retreat (unit_guid =”单元 GUID”, is_return=True)
```

5.6 任务与基地操作

- 接口定义

```
def api_unit_mission_and_base(
        self,
        unit_guid,
        new_base_guid: str = str(),
        ass_mission: str = str(),
        ass_escort_mission: str = str()
)
```

- 实体选择新基地/新港口

```
self. api_unit_mission_and_base (unit_guid =”单元 GUID”, new_base_guid=”基地 GUID”)
```

- 分配加入到任务中

```
self. api_unit_mission_and_base (unit_guid =”单元 GUID”, ass_mission=”任务名”)
```

- 将单元分配为某打击任务的护航任务

```
self. api_unit_mission_and_base (unit_guid =”单元 GUID”, ass_escort_mission=”任务名”)
```

5.7 默认操作

- 接口定义

```
def api_unit_operation_default(
        self,
        unit_guid="",
        contact_guid="",
        ass_mission="",
        units_contacts=(),
        course_list: list = None,
        mount_id="",
        weapon_id="",
        qty_num=1,
        BOL_coordinate=(),
        unattack_all=False,
        unattack_contact_guid="",
        unass_mission="",
        new_base_guid="",
        new_name=""
)
```

六、 分析示例类

6.1 分析态势

- 接口定义

```
def api_situation_analysis_demo(self)
```

6.2 分配并获取己方飞机 guid 列表

- 接口定义

```
def api_air_army_assign_demo(self)
```

6.3 根据类型获取己方军队 guid 列表

- 接口定义

```
def api_common_army_assign_demo(
        self,
        army_type: InfoSequence = InfoSequence.Aircraft
)
```

6.4 根据类型获取敌方目标 guid 列表

- 接口定义

```
def api_target_assign_demo(
        self,
        contact_type: ContactType = ContactType.Facility_Mobile
)
```

附件二:

兵棋大赛态势分析模块接口表
V1.0

国防科大系统工程学院

中国电子科技集团公司第五十二研究所

北方自动控制技术研究所

二〇二三年八月

目 次

概述

接口一览表如下:

类别	接口名	接口
基础类	初始运行调用	reset
	每步运行调用	run
	结束运行调用	finish
	获取态势	get_situation
	查询是否结束	get_finish
	主动结束脚本	set_finish
	获取步进粒度	get_run_granularity
	设置步进粒度	set_run_granularity
	获取态势保存地址	get_situation_path
	设置态势保存地址	set_situation_path
	态势分析 demo	run_analysis_demo
	获取敌方详细信息	get_targets_info
	获取敌方统计信息	get_targets_count
	获取己方详细信息	get_own_units_info
	获取己方统计信息	get_own_units_count
	设置图表并显示	set_count_display
	保存分析结果到文件	save_analysis_result
	清除文件内容	clear_analysis_result
绘图类	新建窗口	new_fig
	新建画图板	new_subplot
	新建饼图	new_bar
	新建直方图	new_pie
	饼图绘图简化接口	easy_bar
	直方图绘图简化接口	easy_pie

	显示	show
	清除	clf
	关闭	close
	重绘	draw
	暂停	pause

一、 基础类

1.1 初始运行调用

- 接口定义

```
def reset(self)
```

- 接口说明

  程序初始化时调用，用户可自定义编程内容。

1.2 每步运行调用

- 接口定义

```
def run(self)
```

- 接口说明

  程序运行时按照固定时间间隔调用，用户可自定义编程内容。

1.3 结束运行调用

- 接口定义

```
def finish(self)
```

- 接口说明

  程序运行结束前调用，即步进函数结束后调用，用户可自定义编程内容。

1.4 获取态势

- 接口定义

```
def get_situation(self)
```

- 接口说明

  获取己方完整的原始的态势。

1.5 查询是否结束

- 接口定义

```
def get_finish()
```

- 接口说明

  获取当前是否结束状态，返回布尔值。

1.6 主动结束脚本

- 接口定义

```
def set_finish(flag: bool)
```

- 接口说明

  设置程序是否结束，当程序判断为结束状态，自动结束步进循环。

1.7 获取步进粒度

- 接口定义

```
def get_run_granularity()
```

- 接口说明

  获取步进循环时间间隔，返回 int 类型变量，单位秒。

1.8 设置步进粒度

- 接口定义

```
def set_run_granularity(rg: int)
```

- 接口说明

  设置步进循环时间间隔，传入 int 类型变量，单位秒，范围是不小于 1。

1.9 获取态势保存地址

- 接口定义

```
def get_situation_path(self)
```

- 接口说明

返回原始态势保存地址，基于态势分析主目录的相对路径。默认路径为”/log/save_sit.json”。

1.10 设置态势保存地址

- 接口定义

```
def set_situation_path(self, path: str)
```

- 接口说明

设置原始态势保存地址。建议保存在态势分析目录范围内。

1.11 态势分析 demo

- 接口定义

```
def run_analysis_demo(self)
```

- 接口说明

绘制己方和敌方的统计图表。获取己方和敌方的详细信息和简要信息，并保存在”/result”文件夹下。

1.12 获取敌方详细信息

- 接口定义

```
def get_targets_info(self, situation: dict = None, is_console_show:
bool = True)
```

- 接口说明

获取敌方信息，返回敌方原始态势中情报信息和处理后的简要信息。可以通过 situation 传入需要解析的原始态势，通过 is_console_show 控制是否在控制台打印信息。

1.13 获取敌方统计信息

- 接口定义

```
def get_targets_count(self, situation: dict = None)
```

- 接口说明

获取敌方统计后的信息。返回变量类型为字典 dict，key 为情报单元 Type，Value 为数量 int。

1.14 获取己方详细信息

- 接口定义

```
def get_own_units_info(self, situation: dict = None,
is_console_show: bool = True)
```

- 接口说明

获取己方信息，返回己方原始态势中单元信息和处理后的简要信息。可以通过 situation 传入需要解析的原始态势，通过 is_console_show 控制是否在控制台打印信息。

1.15 获取己方统计信息

- 接口定义

```
def get_own_units_count(self, situation: dict = None)
```

- 接口说明

获取己方统计后的信息。返回变量类型为字典 dict，key 为单元类型，Value 为数量 int。

1.16 设置图表并显示

- 接口定义

```
def set_count_display(self, count, target_count)
```

- 接口说明

传入己方单元统计信息和敌方情报统计信息，绘制出饼图和直方图。

1.17 保存分析结果到文件

- 接口定义

```
def save_analysis_result(self, arg: str, path: str =
"result/analysis_result.txt")
```

- 接口说明

将分析结果存入文件中。传入参数为 string 格式的分析结果和 string 格式的文件路径。

1.18 清除文件内容

- 接口定义

```
def clear_analysis_result(self, path: str =
"result/analysis_result.txt")
```

- 接口说明

清除指定文件的内容。传入参数为 string 格式的文件路径。

二、 绘图类

2.1 新建窗口

- 接口定义

```
def new_fig(self, name: str = None)
```

- 接口说明

新建一个绘图窗口。传入参数为 string 格式的窗口名字。

2.2 新建画图板

- 接口定义

```
def new_subplot(fig, title: str = None)
```

- 接口说明

在指定窗口中新建一个画图板。传入参数为窗口对象 fig 和 string 格式的

画图板标题。

2.3 新建饼图

- 接口定义

```
def new_bar(ax, x: list, labels: list)
```

- 接口说明

新建统计图表-饼图。传入参数为画图板对象 ax，值列表 x（列表元素为 int 或 float），标签列表 labels（列表元素为 string）。

2.4 新建直方图

- 接口定义

```
def new_pie(ax, x, labels)
```

- 接口说明

新建统计图表-直方图。传入参数为画图板对象 ax，值列表 x（列表元素为 int 或 float），标签列表 labels（列表元素为 string）。

2.5 饼图绘图简化接口

- 接口定义

```
def easy_pie(self, x, labels, fig_name: str = None, sub_tittle:
str = None)
```

- 接口说明

简化后的饼图绘制接口，无需对窗口和画图板进行管理，无需传入窗口对象或者画图板对象。传入参数为值列表 x（列表元素为 int 或 float），标签列表 labels（列表元素为 string），窗口名字 fig_name(str 格式)，画图板标题 sub_tittle(str 格式)。

2.6 直方图绘图简化接口

- 接口定义

```
def easy_bar(self, x: list, labels: list, fig_name: str = None,
sub_tittle: str = None)
```

- 接口说明

简化后的直方图绘制接口，无需对窗口和画图板进行管理，无需传入窗口对象或者画图板对象。传入参数为值列表 x（列表元素为 int 或 float），标签列表 labels（列表元素为 string），窗口名字 fig_name(str 格式)，画图板标题 sub_tittle(str 格式)。

2.7 显示

- 接口定义

```
def show(fig)
```

- 接口说明

显示 matplot 窗口。传入参数为窗口对象 fig。

2.8 清除

- 接口定义

```
def clf(fig)
```

- 接口说明

清除 matplot 窗口内容。传入参数为窗口对象 fig。

2.9 关闭

- 接口定义

```
def close(fig)
```

- 接口说明

关闭 matplot 窗口。传入参数为窗口对象 fig。

2.10 重绘

- 接口定义

```
def draw(fig)
```

- 接口说明

绘制 matplot 窗口内容。传入参数为窗口对象 fig。

2.11 暂停

- 接口定义

```
def pause(sec: int = PauseSec)
```

- 接口说明

  暂停 matplot 窗口更新。传入参数为暂停时间 sec，单位秒。