端到端-中控设计

一、背景

1. 对话系统包含了多个模块，需要一个中控服务来组织和串联对话
2. 作为流量的入口和出口与设备通信

二、目标

💡 **MVP目标：**

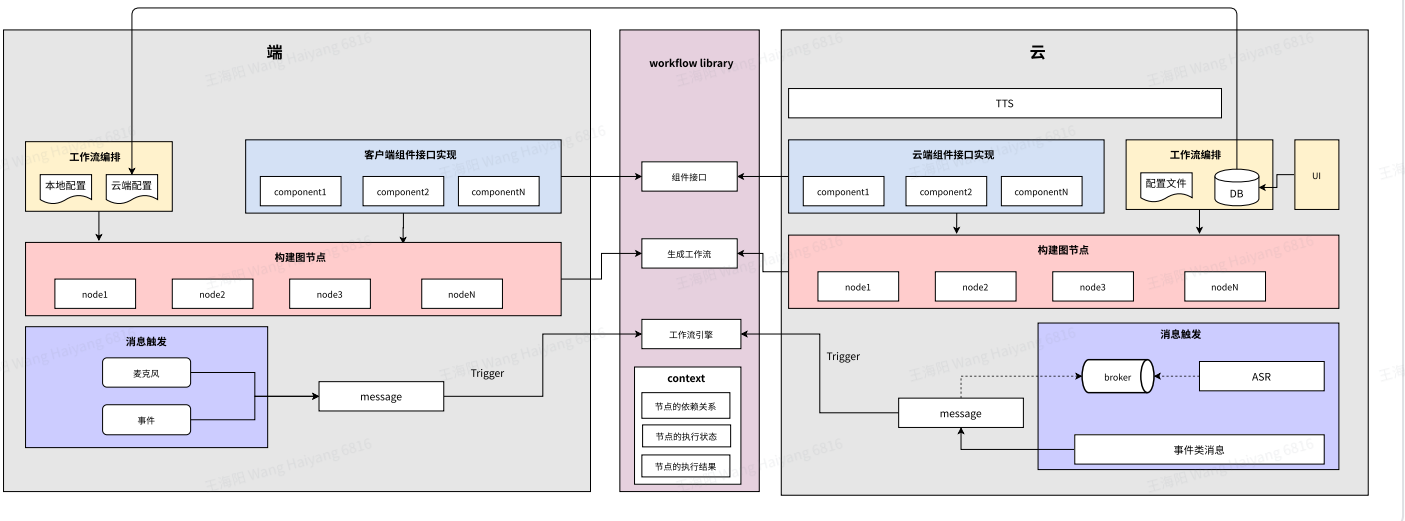
1. 串联起对话链路，包括nlu、tracker、action、nlg
2. 提供接口级别的端到端体验

远期目标：

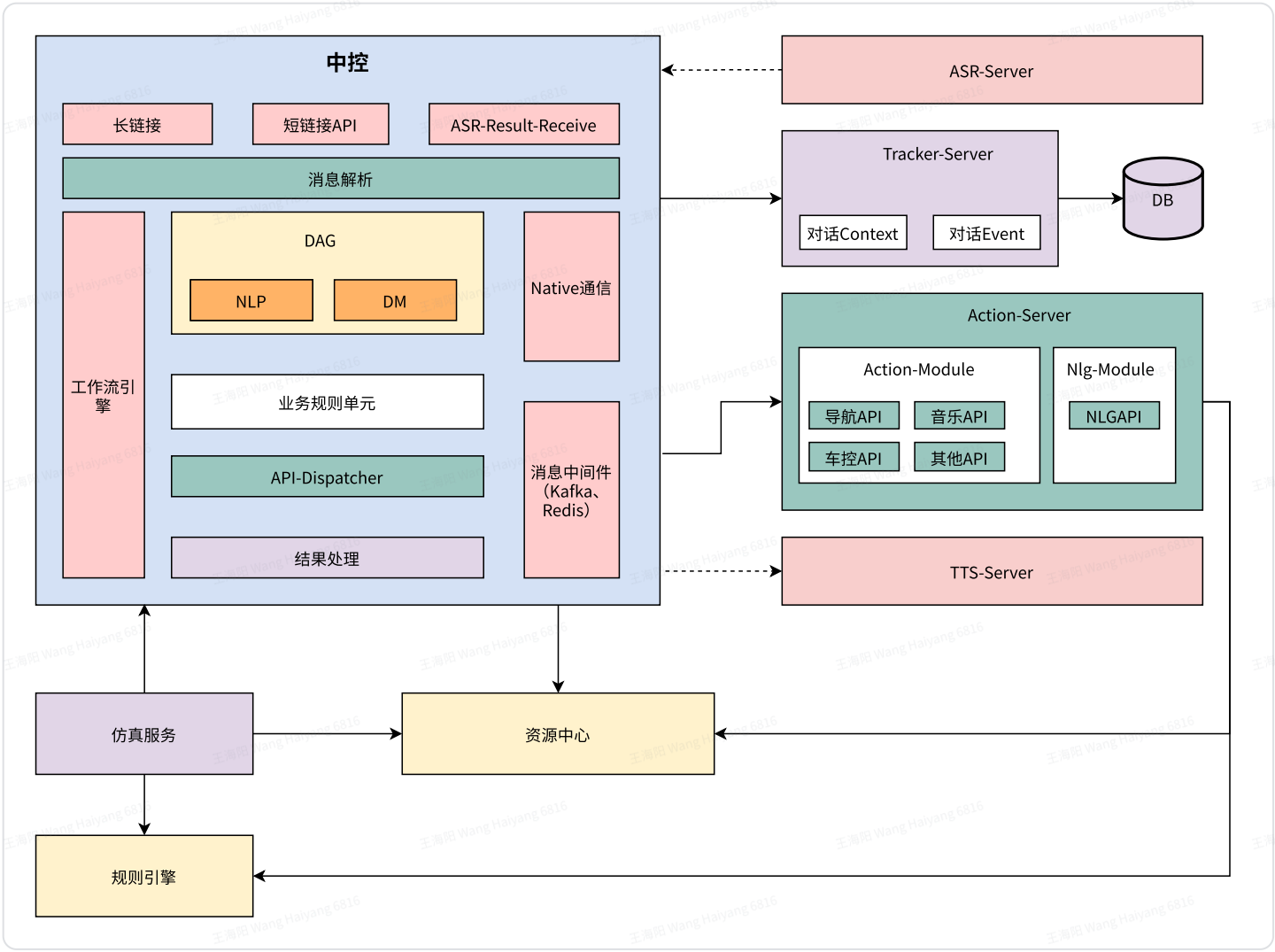
1. 对所有外接的服务支持可插拔能力
2. 支持服务编排，达到平台化标准
3. 打通与大屏的通信，协议向前兼容

三、设计

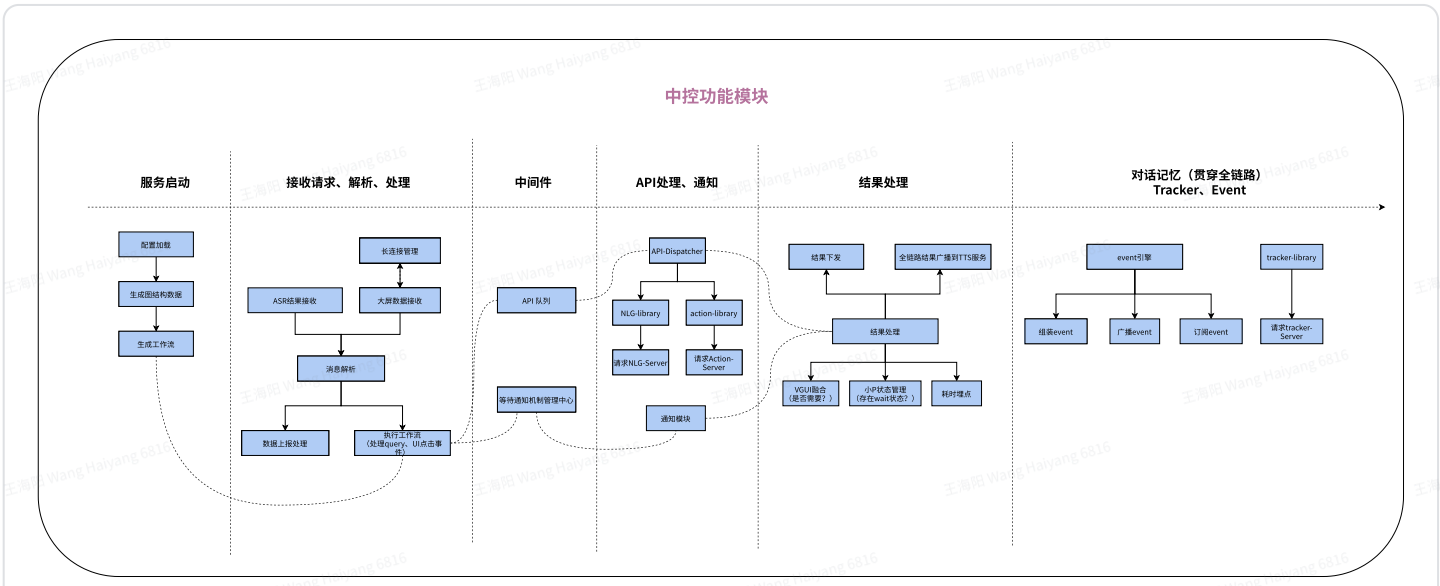
3.1、端云架构



3.2、云架构设计



3.3、功能模块设计



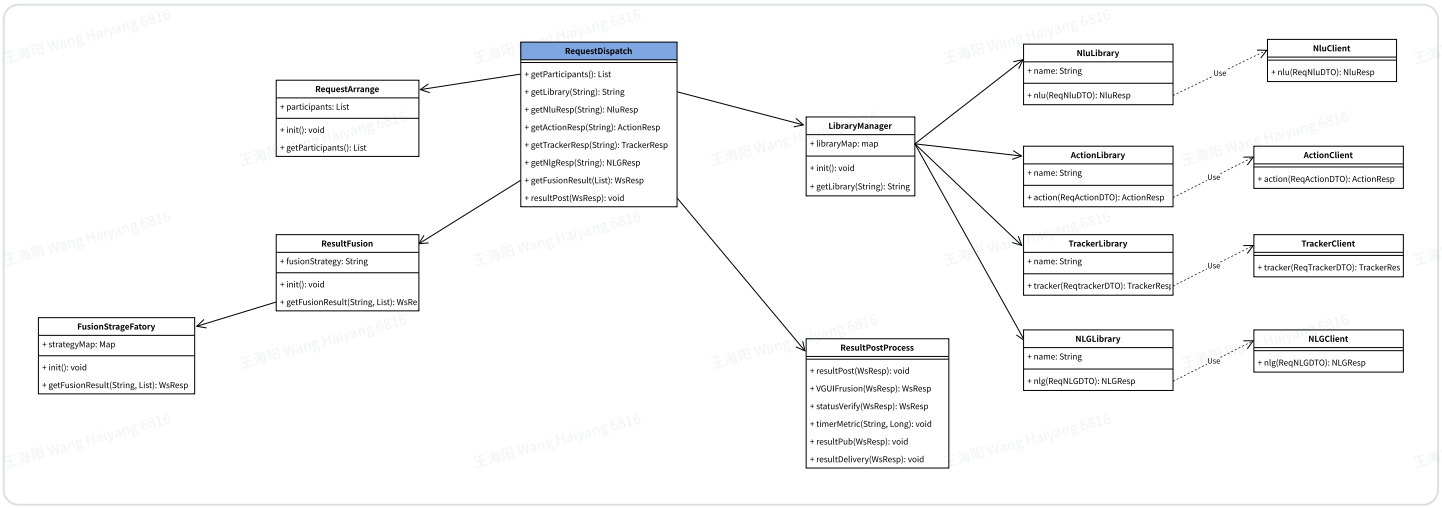
模块

	模块	版本规划	全链路阶段	功能说明
▼	接收请求...			
1	长连接管理	远期	接收请求、解析处理	有效性验证 长连接的本地存储
2	ASR结果接收	远期	接收请求、解析处理	订阅从ASR服务广播的AS 验证ASR是否需要处理
3	大屏数据接收	远期	接收请求、解析处理	接收大屏的数据，包括端 传、事件请求等
4	消息解析	远期	接收请求、解析处理	对订阅的ASR结果 、大屏 屏上报的数据进行数据解
▼	结果处理			
1	全链路结果后下发	远期	结果处理	全链路的结果下发
2	全链路结果广播到TTS服务	远期	结果处理	广播最总的全链路结果
3	VGUI融合	远期	结果处理	统一管理VUI和GUI的状态
4	小P状态管理	远期	结果处理	处理小P在不同场景下的 着业务走
5	耗时埋点	远期	结果处理	全链路结果的耗时以及
▼	对话记忆			
1	event引擎	远期	对话记忆	订阅外部event、组装eve event
2	tracker-library	远期	对话记忆	对接Tracker服务，支持r 本地的记忆存储
3	请求tracker-Server	MVP	对话记忆	对接Tracker服务，持久化
4	组装Event	远期	对话记忆	不同的消息组装成不同的
5	广播Event	远期	对话记忆	对封装的事件进行广播
6	订阅Event	远期	对话记忆	订阅外部服务产生的事件 tracker的属性状态
▼	服务启动			

1	配置加载	远期	服务启动	通过配置中心 或者 从 DB 配置消息
2	生成图结构数据	远期	服务启动	通过组件的依赖关系，构 点和边
3	生成工作流	远期	服务启动	通过节点和边数据构造业 业务工作流可以动态的变
▼ 中间件				
1	API队列	远期	中间件	由AF组件产生的API放入 由API-Dispatcher消费
2	等待、通知管理中心	远期	中间件	由产生API的组件注册wa 模块发出notify信号
▼ API处理...				
1	API-Dispatcher	远期	API处理、通知	API分发器，用于消费API
2	NLG-library	MVP	API处理、通知	对接NLG服务，支持rem 的记忆存储
3	请求NLG-Server	MVP	API处理、通知	NLG功能的具体API调用
4	action-library	远期	API处理、通知	对接Action服务，支持re 地的记忆存储
5	请求Action-Server	MVP	API处理、通知	对接Action功能的具体实

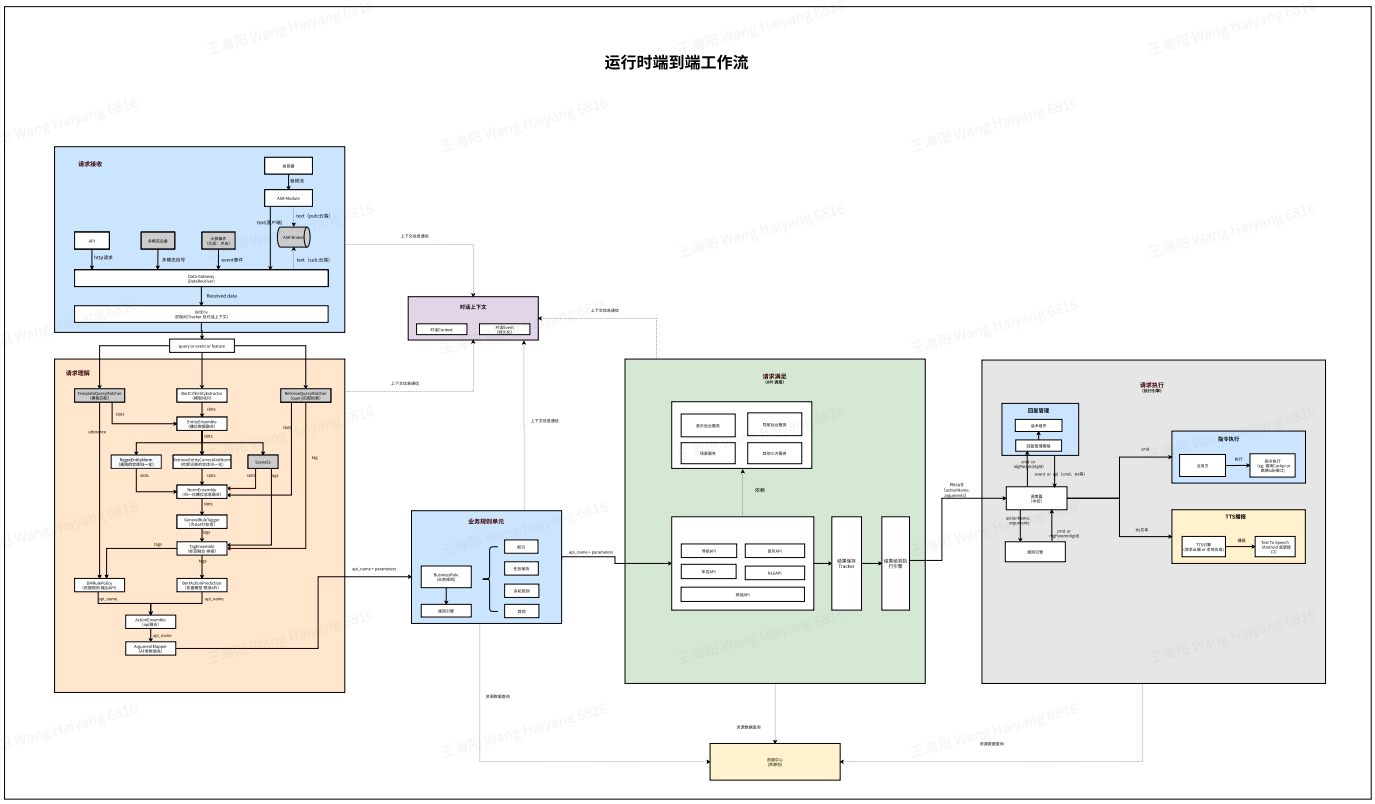
3.4、核心类图

> 以**RequestDispatch为切入点进行业务的处理（MVP阶段实现逻辑，远期由工作流替代）**



3.5、整体数据流图

input (query) --> output (CMD + NLG + VUI + 小Pexpression)



模块概述

模块级输入输出

	模块	input	output	备注
1	请求接收	（audio | event | statusData） + fea...	（文本 | event | statusD...	statusData：车
2	请求理解	（文本 | event | statusData） + feat...	apiName + arguments	NavigationPoi
3	业务执行单元	apiName + arguments + lastApi + f...	apiName + arguments	NlgMultiPoiCo
4	请求满足	apiName + arguments	actionName + arguments	NavigationPoi
5	请求执行	actionName + arguments	cmd | tts	得到cmd or tts

5 条记录

3.5.1、请求接收

1. 用户语音
2. 大屏点击事件
3. 状态数据（比如：页面发生变化时，对应action：给出一些提示词）
4. 特征数据（比如：当前所有车窗是否关闭）

3.5.2、请求理解

职能：根据输入： （文本 | event | statusData） + featureData ，输出对应的api+arguments

示例：

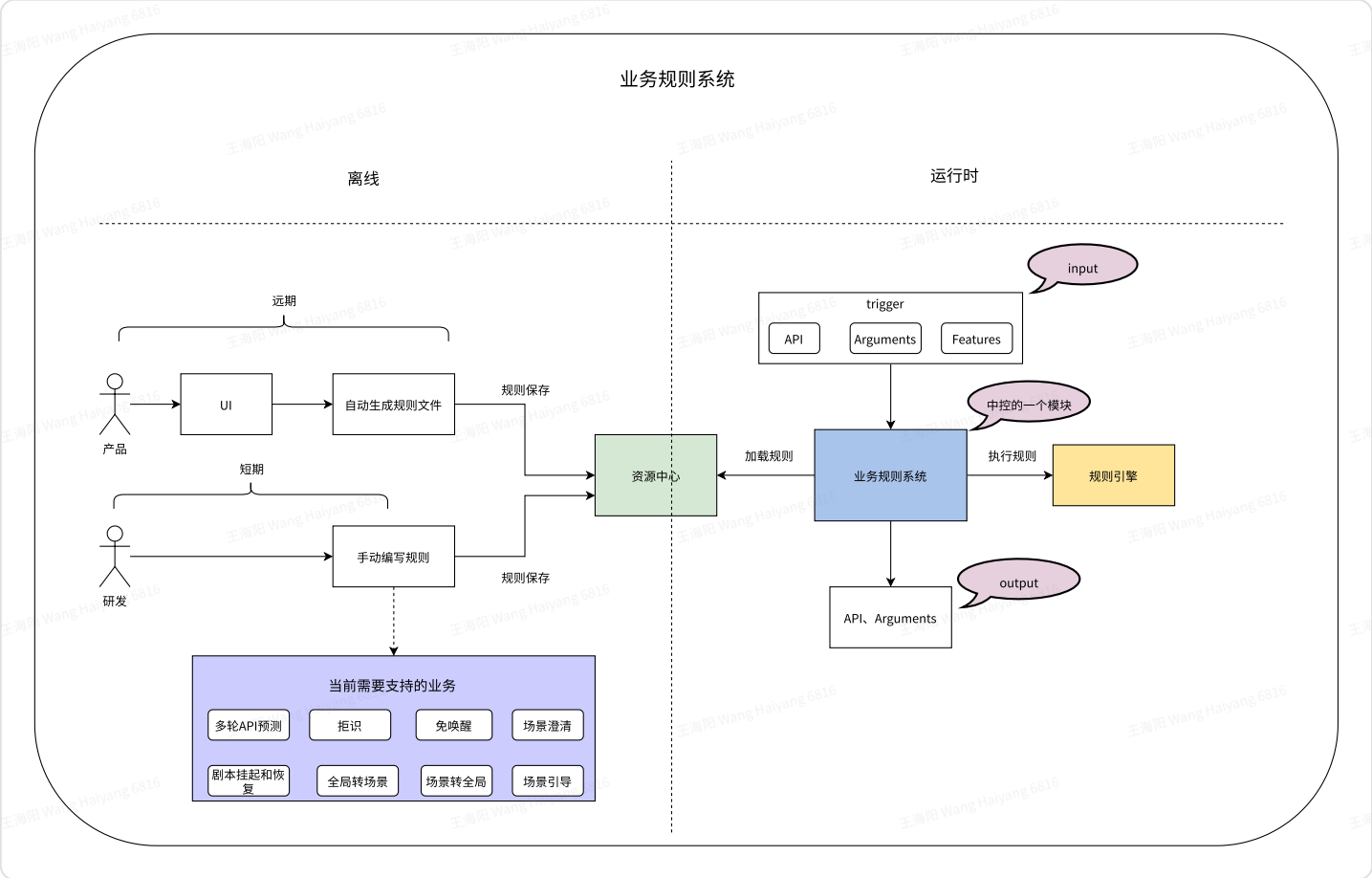
input：导航去北京大学，

output：{"apiName":"SearchNavigationPOI","arguments":[{"name":"poi_name","pos":[3,6],"rawvalue":"北京大学","type":"string","value":"北京大学"}]}

ps：针对此case输出SearchNavigationPOI及arguments之后该模块的使命已经完成

3.5.3、业务规则单元

🎶 根据apiName 、arguments 、lastApi 、 lastApiResp 、features 等数据支撑产品定义的业务。短期会手动平移原二代架构中的规则，远期会提供一个UI界面让产品来定义业务，服务自动实现



示例一：根据 lastApi + lastApiResp 预测下一个api（多轮预测）

场景：导航去北京大学，依赖请求理解模块输出的SearchNavigationPOI所对应的poiSize，输出Nlg相关API

```
if( lastApi == "SearchNavigationPOI" &&  lastApiResp.getPoiList().size() > 1 ) {
    apiName = "NlgMultiPoiConfirm";
}
```

示例二：根据 apiName + lastApi 判定lastApi对应的任务是否需要保持（任务保持）

场景：

U：导航去北京大学

U：打开空调（需要保持导航任务）

U：第一个（能选中第一个poi进行路线选择）

```
if( apiName == "AcOpen" && lastApi == "SearchNavigationPOI") {
```

> 在对话上下文中记录任务状态
>
> 目前承载的能力有：
>
> 1：对AcOpen 产生的结果打上一个标签，用于回溯该设备对应任务的lastTurnInfo
>
> 2：在执行AcOpen对应的cmd or tts时，不能破坏SearchNavigationPOI对应结果的UI形态

```
    workContext.put( "taskStatus", "suspend" );
```

}

示例三：根据 apiName + featureData 改写api（拒识（API改写））

~~U：打开车窗（车外触发的query拦截）（若满足车辆静止、所有车窗都是关闭状态、所有车门关闭三个条件，不执行打开车窗的操作)~~

```
~~if( apiName == "WindowOpen" && featureData.statusIsStatic== true &&~~
~~featureData.allWindowsAreClosed == true && featureData.allDoorsAreClosed== true && ) {~~
~~    apiName = "NoiseAction";~~
~~}~~
```

3.5.4、请求满足

满足所有api的请求, 根据api + arguments 输出对应的actionName + arguments

1. 通过api + arguments找到对应需要执行的函数
2. 运行其函数得到对应的response（获取三方资源《若需要》 or 一些校验异常的结果《比如：听音乐时发现未登录》）
3. 通过api + arguments找到对应的ActionName + arguments，同时若response不为空，则用response 完善arguments
4. 最终输出 actionName + arguments

3.5.5、请求执行

通过解析请求满足模块的输出，得到可以执行的指令或者文本，然后进行trigger执行

1. 根据 actionName + arguments 获取对应的规则
2. 通过规则引擎解析该规则 输出 需要执行的cmd or tts
3. 把cmd or tts 给到对应的业务方 or tts引擎进行执行

3.5.6、对话上下文

用于存储运行时对话状态发生改变时产生的所有的数据，同时提供读接口能力，包含两个维度：

1. 内存维度，提供最近两轮对话的上下文信息。
2. 磁盘维度，提供多轮历史对话上下文信息（此信息会被持久化 及 定期的归档）。（用于回溯对话或者一些特定场景下读取某条指定的上下文信息）

3.5.7、资源中心

包含所有对话中用到的数据

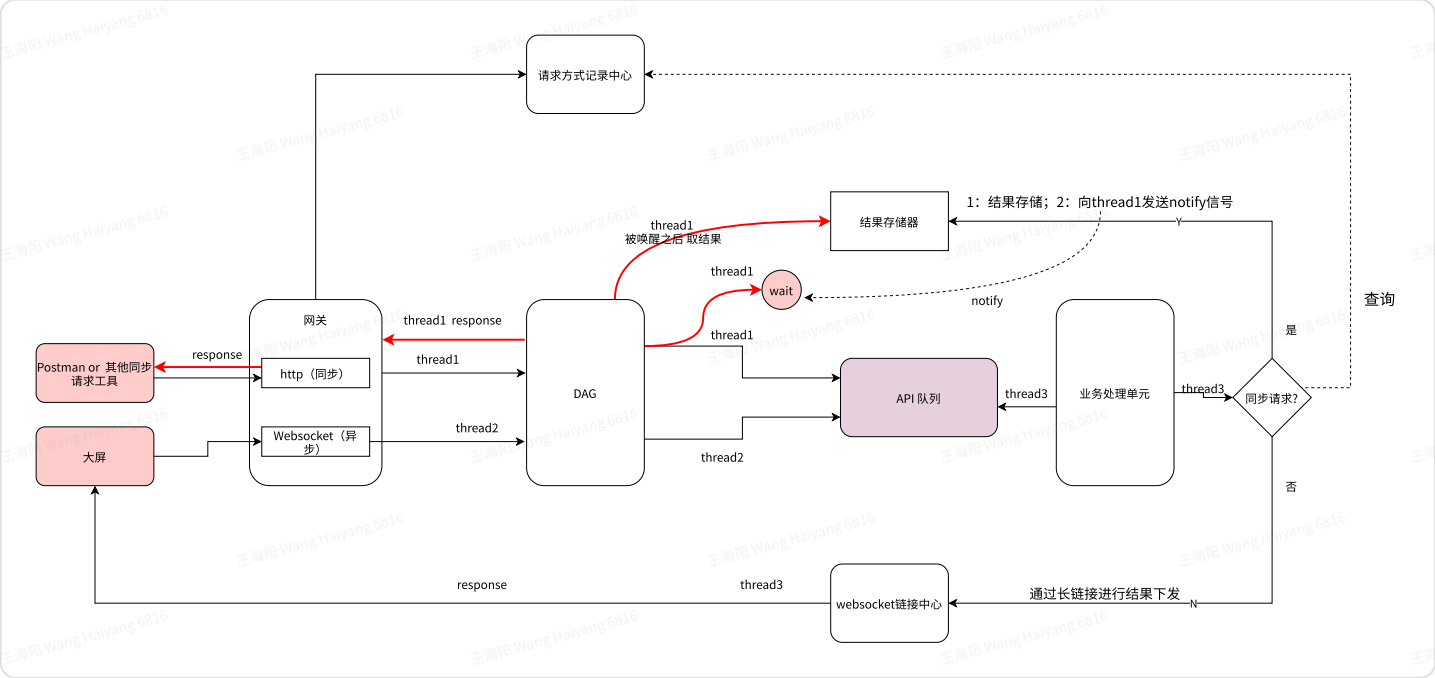
1. 词典
2. 数据间映射关系

3. 元数据
    a. api
    b. 原子指令
    c. 感知点
    d. 回复话术
    e. 信号源
    f. 功能函数
    g. 运算符
4. 规则数据
    a. 预制规则
    b. 条件
    c. 条件组
    d. 动作
    e. 动作组

3.6、全链路能力支持方式

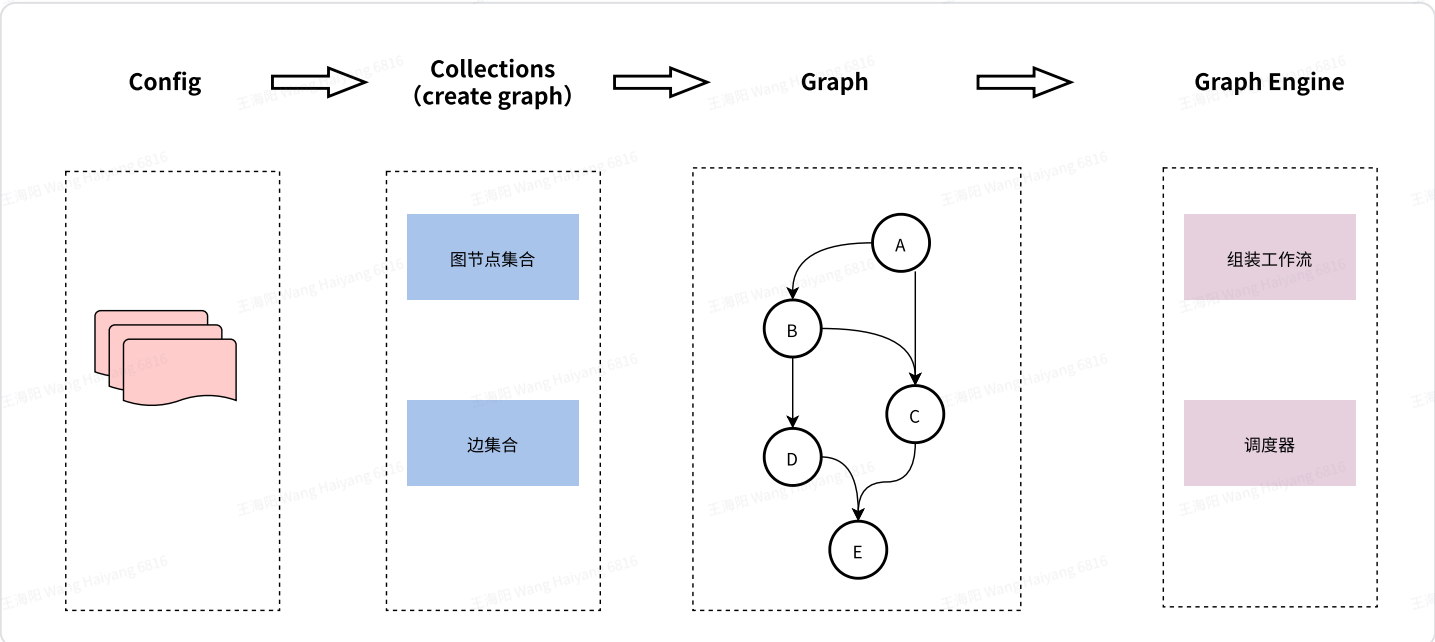
当前整体架构是异步化处理，同时也提供了http同步请求的能力（方便开发or测试进行debug）

1. 同步：http请求
    a. 异步转同步方案 详见红线部分（thread1把DAG结果放入API队列后进入wait态，等待被唤醒，然后取结果进行响应）
2. 异步：websocket途径请求
    a. thread2把DAG结果放入队列后针对该次请求其使命已结束，接下来由业务处理单元的work thread进行处理

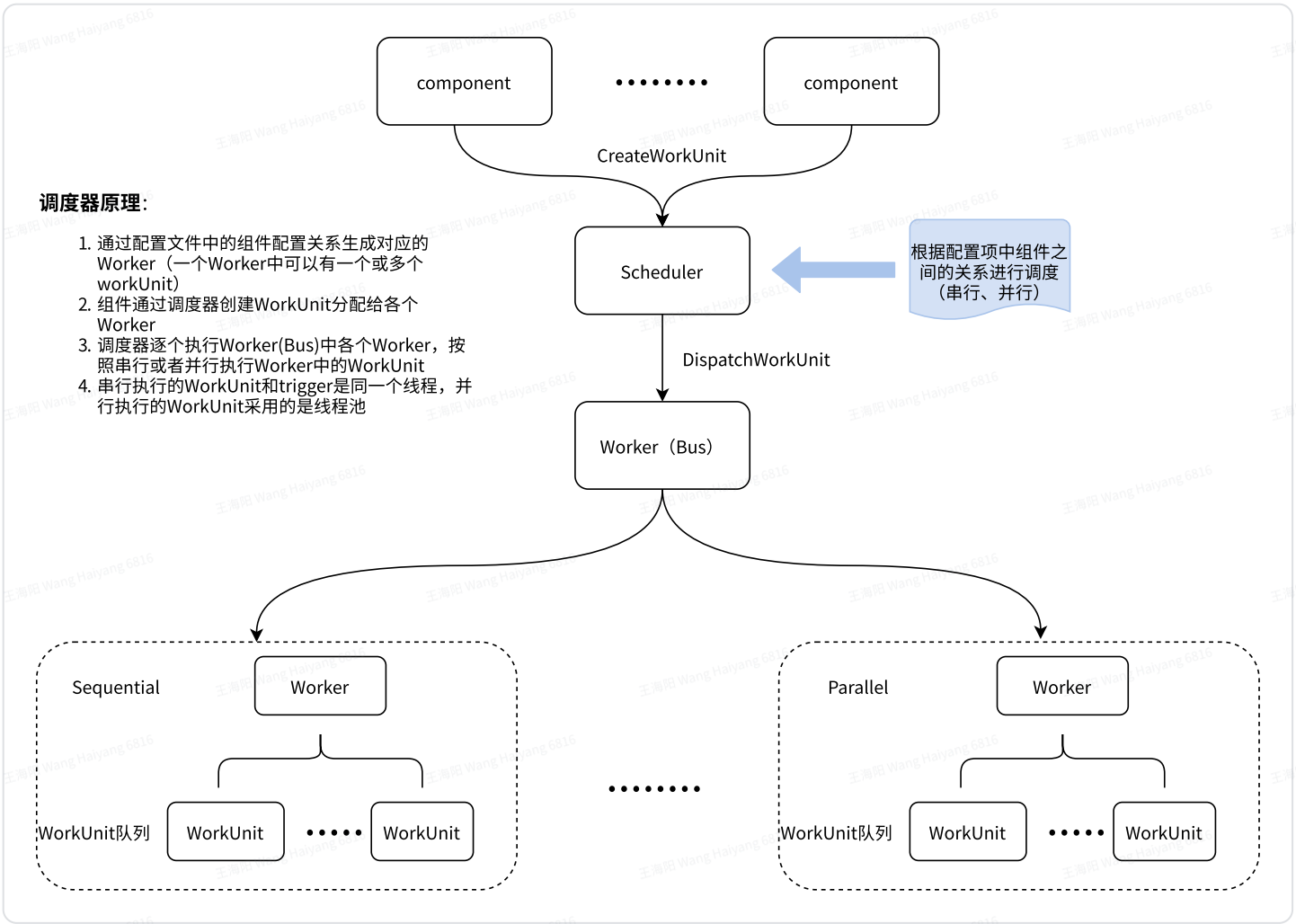


3.7、部分逻辑阐述

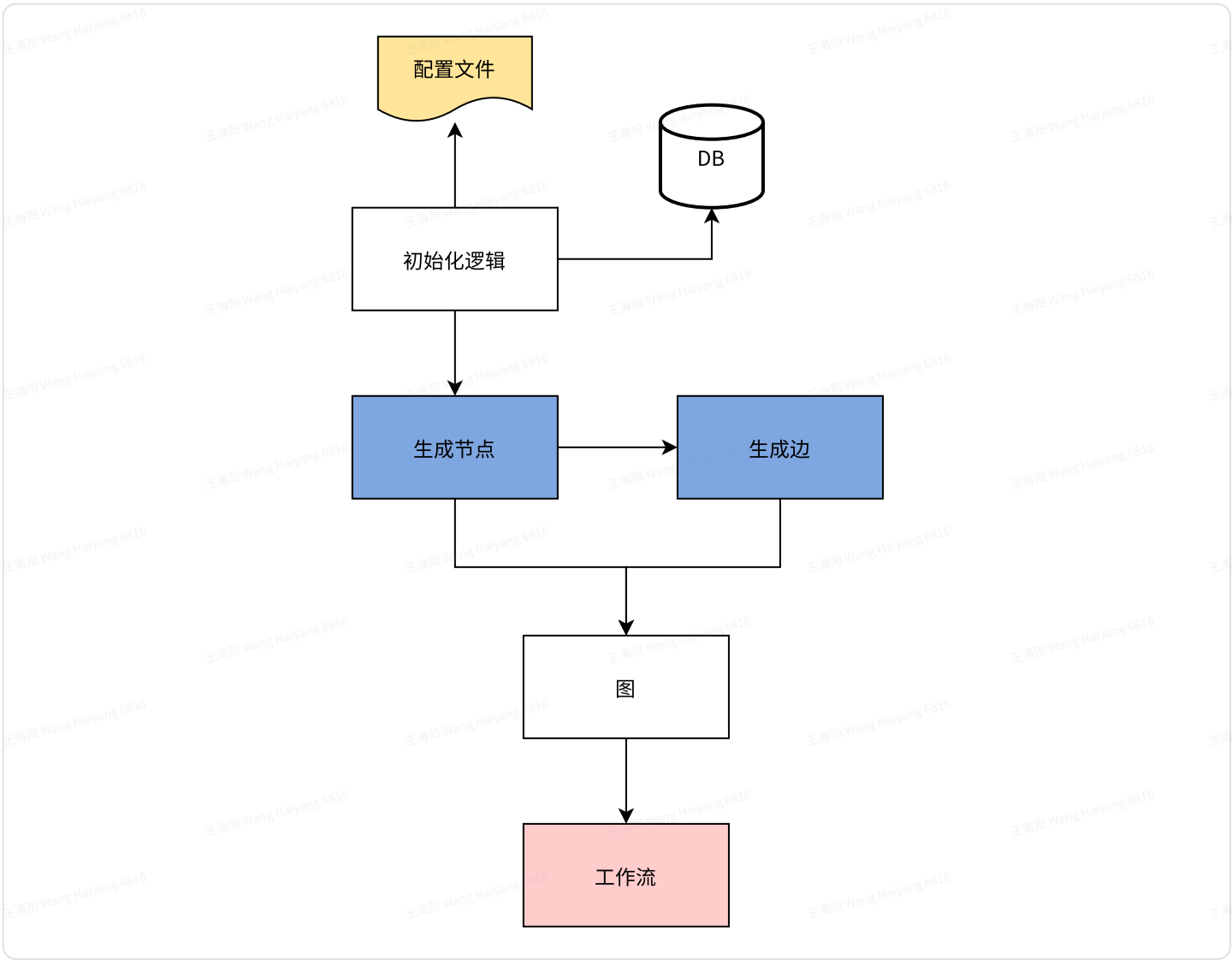
DAG整体流程如下： 详见 NGC-DAG工程化设计



调度器原理如下图：



3.7.1、DAG工作流生成



工作流数据结构：[{"node":"xxx","childNodes":["xxx","xxx"],"parentNodes":["xxx"],"priority":"xxx"}]

重点工作：输出各个节点执行的顺序

方案简述：为每个节点计算优先级，数值越小优先级越高。

优先级计算公式：所有父节点的和 + 入度

3.7.2、DAG工作流执行

运行时支持4种处理方式：串行、并行、条件式、循环

其中：

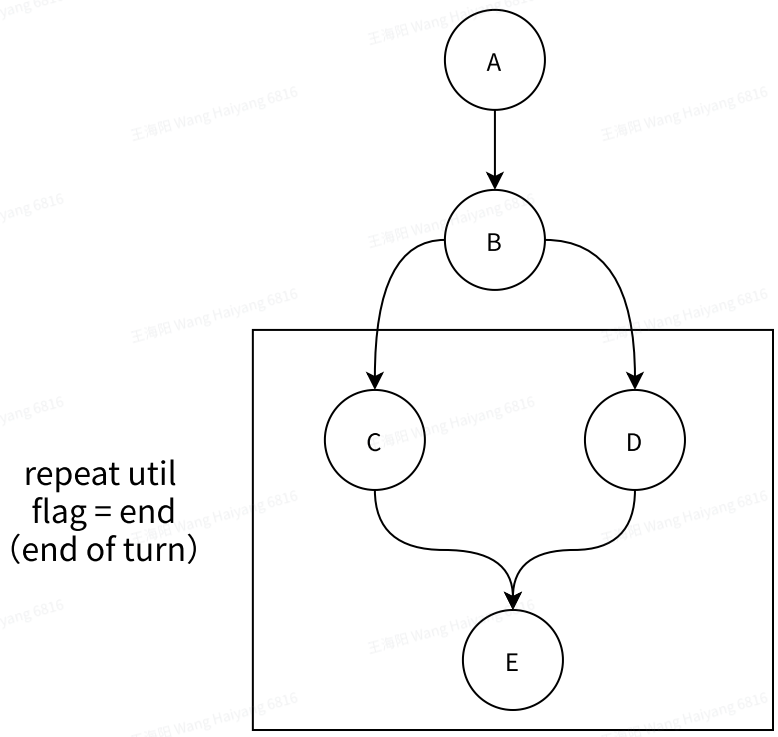
1. 串行、并行根据 childNodes 和 parentNodes的执行状态（是否结束）来决定；
2. 条件式、循环根据配置来决定。
3. 头节点的处理方式：通过获取图中头节点的数量（等于1串行，大于1并行）和配置

示例：

配置

```yaml
graph:
  components:
    - name: A # 节点名称
      dependency:  # 依赖的节点
    - name: B
      dependency: A
    - name: C
      dependency: B
    - name: D
      dependency: B
    - nodeName: E
      dependency: C,D
```

图结构



对应工作流：

```
[
    {
        "node": "A",
        "childNodes": [
            "B"
```

```
        ],
        "parentNodes": [

        ],
        "priority": 0
    },
    {
        "nodeName": "B",
        "childNodes": [
            "C",
            "D"
        ],
        "parentNodes": [
            "A"
        ],
        "priority": 1
    },
    {
        "nodeName": "C",
        "childNodes": [
            "E"
        ],
        "parentNodes": [
            "B"
        ],
        "priority": 2
    },
    {
        "nodeName": "D",
        "childNodes": [
            "E"
        ],
        "parentNodes": [
            "B"
        ],
        "priority": 2
    },
    {
        "nodeName": "E",
        "childNodes": [

        ],
        "parentNodes": [
            "C","D"
        ],
        "priority": 4
    }
```

```
53 ]
```

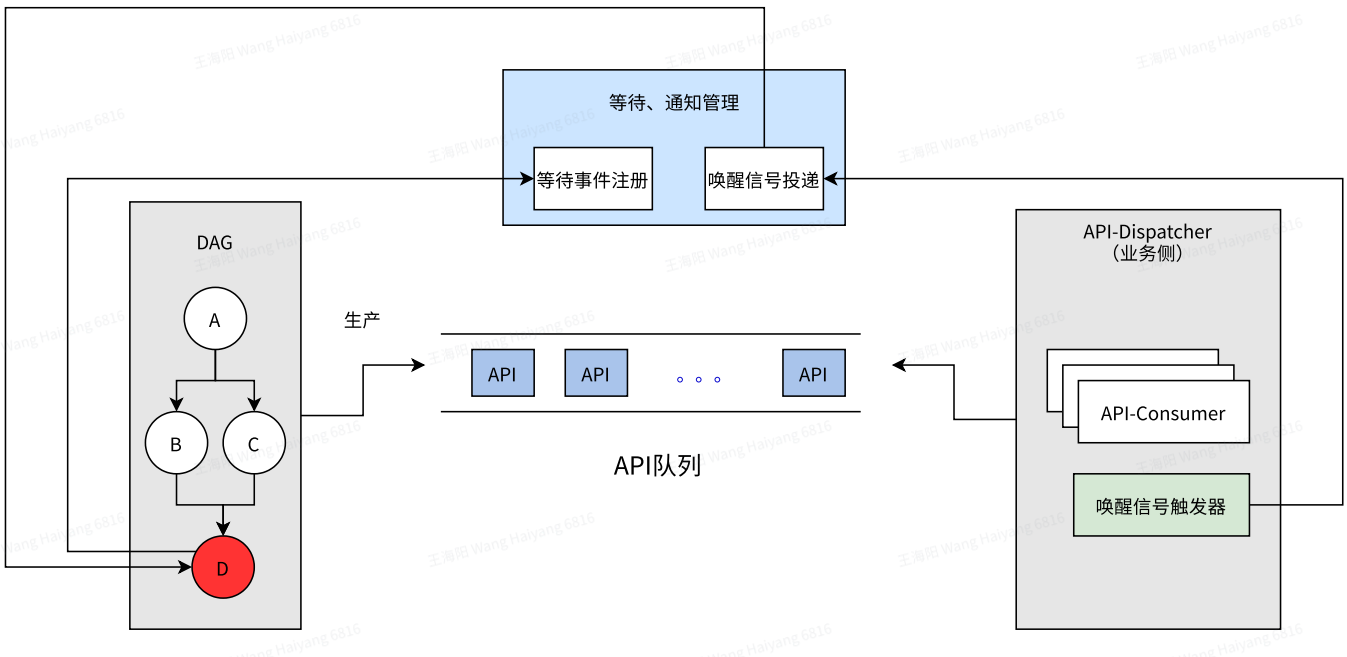
执行伪代码：

```
sequentialOperator()
        .execute(A)
        .then(B)
        .then(repeatOperator()
                .repeat(sequentialOperator()
                        .execute(parallelOperator()
                                .execute(C,D))
                        .then(E)
                        )
                .until("flag == end")
                )
```

3.7.3、DAG与业务模块的集成

核心牵涉到两块能力：（这两块能力端云共用）

1. API的生产、API的消费、API队列
2. 等待、通知管理
   a. 注册等待的key(数据结构)：{msgId}_{apiName}

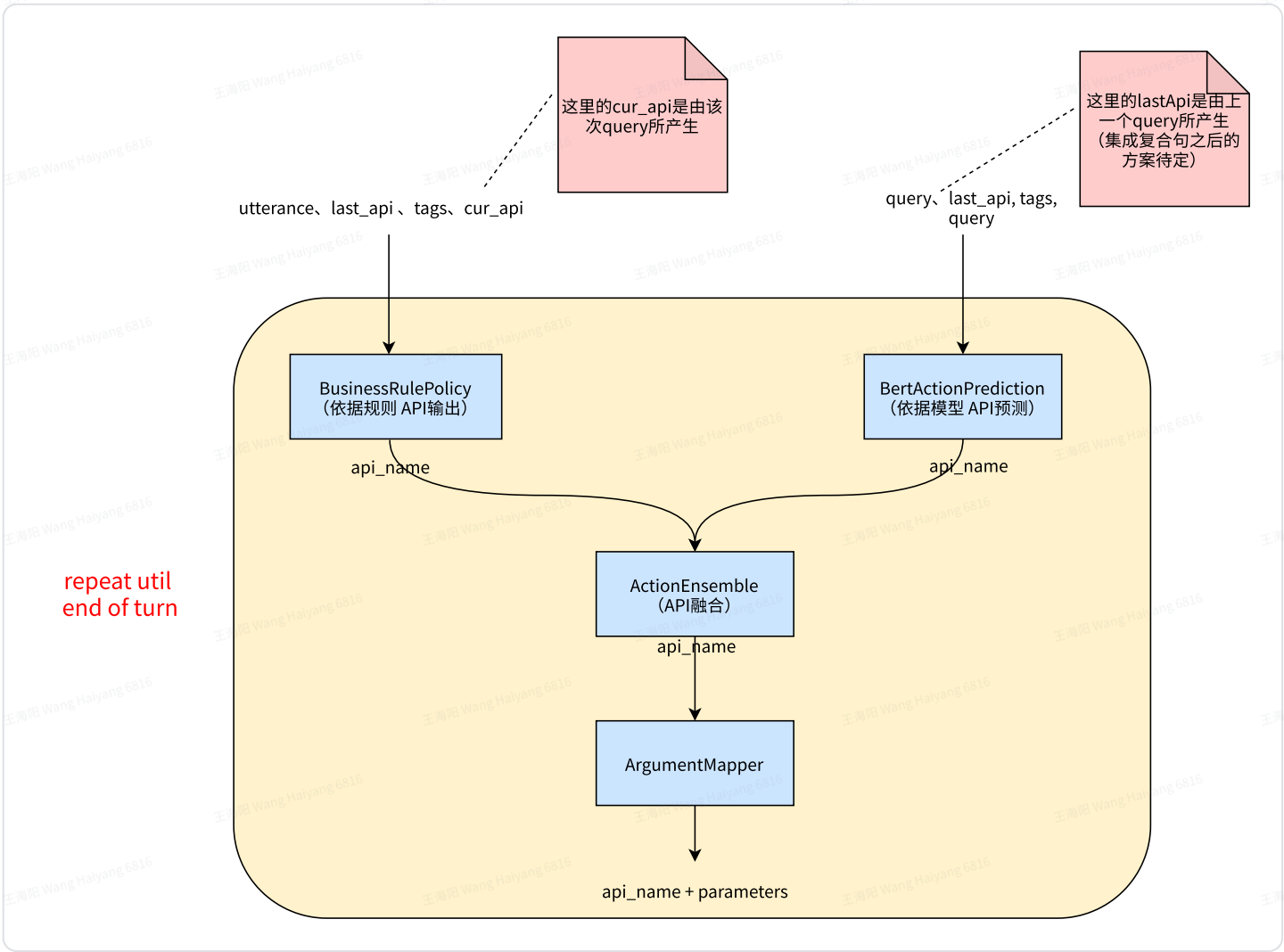


关于API的流程做一下阐述：

1. 计算节点DAG负责产生API
2. 产生的API放入到队列
3. API队列监听器监听到API事件时回调API的handler
4. APIHandler调用业务侧事情处理逻辑

关于等待通知做一下阐述：

前置背景同步：当前（rulepolicy+ap）是支持多轮预测，方案：循环执行BusinessRulePolicy、BertActionPrediction、ActionEnsemble、ArgumentMapper 这四个组件，直至BusinessRulePolicy组件输出 End of turn 标记（ActionEnd）



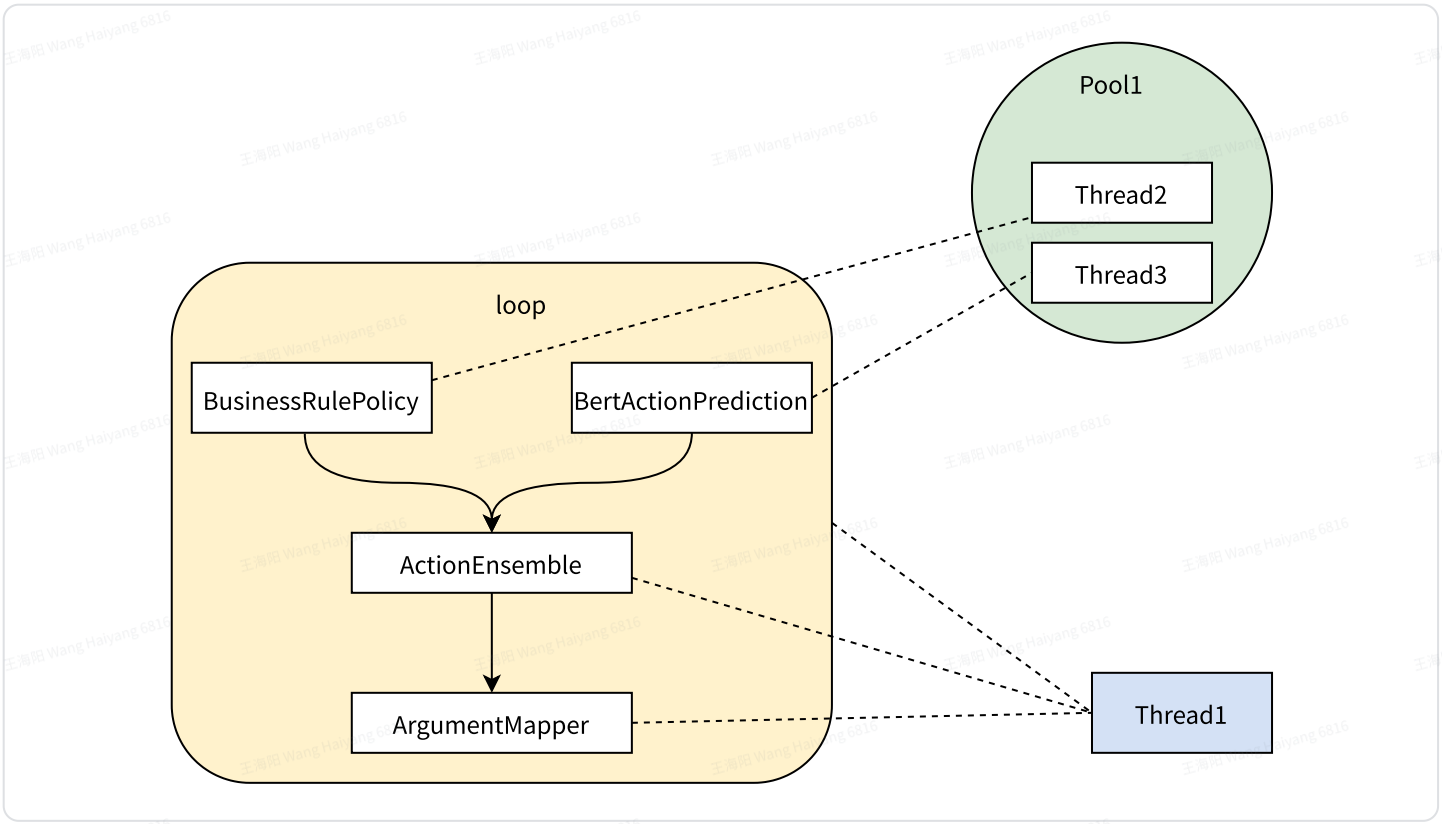
等待通知示例：导航去中关村

1. 当进入到 loop_2，BusinessRulePolicy组件通过判断NavigationPoiSearch需要依赖第三方资源，若依赖的资源不存在此时会输出ActionWait
2. 当ArgumentMapper组件识别到ActionWait标记时会到 等待、通知管理中心进行 wait事件注册（注册ID：{msgId}_NavigationPoiSearch），然后进入wait态，此时loop会被block住

3. 业务侧处理完NavigationPoiSearch这个API后到等待、通知管理中心查询一下是否被注册过（查询ID：{msgId}_NavigationPoiSearch），若被注册过则进行notify
4. ArgumentMapper组件收到notify之后，loop继续，进入到loop_3

	BusinessRulePolicy		BertActionPrediction		actionEnsemble		ArgumentMapper	
	input	output	input	output	input	output	input	output
loop_1	None	None	导航去中关村	NavigationPoiSearch	NavigationPoiSearch	NavigationPoiSearch	NavigationPoiSearch	NavigationPoiSearch(中关村)
loop_2	NavigationPoiSearch	ActionWait	NavigationPoiSearch	None	ActionWait	ActionWait	ActionWait	ActionWait
loop_3	NavigationPoiSearch	NlgMultiSelect	NavigationPoiSearch	None	NlgMultiSelect	NlgMultiSelect	NlgMultiSelect	NlgMultiSelect
loop_4	NlgMultiSelect	ActionEnd	NlgMultiSelect	None	ActionEnd	ActionEnd	ActionEnd	ActionEnd

线程模型：



结合这个线程模型延申一下：整个DAG上的节点只要是使用了Thread1，都支持wait操作

3.7.4、BusinessRulePolicy与BertActionPrediction

1. 这两个组件是并行关系
2. 这两个组件的input 和 output 是一致的
3. BusinessRulePolicy结果优先
4. 都可以输出end of turn 标记
    a. 当前都是由BusinessRulePolicy，具体BertActionPrediction未来什么时间输出end of turn标记需要算法同学对APmodel的迭代升级了

3.8 接口定义

3.8.1 算法DAG

说明：输出api+params

接口地址：/v1/full-link/dag

测试地址：http://logan-gateway.test.logan.xiaopeng.local/xp-ngc-console-boot/v1/full-link/dag

接口类型：POST

input：详见 新架构-端云交互协议 4.1

Output: 各个组件的输出结果，见下方json格式示例

字段	类型	是否可能为空	
msgId	string	no	query的消息Id
query	string	no	query语句
api	string	no	算法DAG输出的最终apiName
params	List<ArgumentBO>	yes	DAG输出的参数

~~ArgumentBO~~

字段	类型	是否可能为空	
name	string	no	参数名称
type	string	no	参数类型
value	obj	no	参数值

示例：

```
{
    "query": "导航去中关村",
    "status": "start",
    "hardwareId": "hardwareIdNGC001",
    "soundArea": "LF",
    "carType": "E38",
    "hid": "REV01",
    "msgType": "req",
    "city": "广州市",
    "sign": "9fcea87666a9ec4bbcd1594298335c77",
    "msgId": "6986e84056d166",
    "lon": "113.37981777777777",
    "vid": "YRD88",
    "uid": "67042",
    "appV": "V1.1.1",
    "model": "1",
    "vin": "L1NSPGHBXLA000552",
    "appId": "xmart:appid:002",
    "lat": "23.059987777777778",
    "timestamp": 1592722880069,
    "speechV": "2.7.0",
    "activeApp": "com.xiaopeng.montecarlo",
```

```
    "bid": "5",
    "activePage": "main"
}
```

返回示例《为了便于定位问题，一并把所有涉及的组件的返回值全部返回》：

```
{
    "entrySet": [
        {
            "actionEnsembleOutput": {
                "componentApiRes": {
                    "from": "BertActionPredictionComponent",
                    "apiName": "AcOpen"
                }
            }
        },
        {
            "argumentMapperOutput": {
                "apiName": "AcOpen",
                "slotArgsList": [
                    {
                        "name": "device",
                        "value": "空调",
                        "pos": [
                            2,
                            3
                        ],
                        "rawvalue": "空调",
                        "type": "string"
                    }
                ]
            }
        },
        {
            "tagEnsembleOutput": {
                "tagSpliceRes": []
            }
        },
        {
            "regexEntityNormOutput": {
                "slots": [
                    {
                        "name": "device",
                        "value": null,
                        "pos": [
```

```
                        2,
                        3
                    ],
                    "rawvalue": "空调",
                    "type": "string"
                }
            ]
        }
    },
    {
        "generalRuleTaggerOutput": {
            "tagListRes": []
        }
    },
    {
        "msgId": "6986e84056d166"
    },
    {
        "bertActionPredictionOutput": {
            "apiName": "AcOpen",
            "apiType": null,
            "apiRankingList": [
                {
                    "confidence": 11.6006365,
                    "name": "AcOpen"
                },
                {
                    "confidence": 3.4787364,
                    "name": "NoiseAction"
                },
                {
                    "confidence": 2.061916,
                    "name": "AcSet"
                }
            ]
        }
    },
    {
        "entityEnsembleOutput": {
            "slots": [
                {
                    "name": "device",
                    "value": null,
                    "pos": [
                        2,
                        3
                    ],
```

```
                    "rawvalue": "空调",
                    "type": "string"
                }
            ]
        }
    },
    {
        "dagOutput": {
            "query": "打开空调",
            "api": "AcOpen",
            "params": [
                {
                    "name": "device",
                    "type": "string",
                    "value": "空调",
                    "props": null
                }
            ],
            "msgId": "6986e84056d166"
        }
    },
    {
        "userQueryReq": {
            "vid": "YRD88",
            "vin": "L1NSPGHBXLA000552",
            "uid": "67042",
            "query": "打开空调",
            "status": "start",
            "hardwareId": "hardwareIdNGC001",
            "soundArea": "LF",
            "carType": "E38",
            "hid": "REV01",
            "msgType": "req",
            "city": "广州市",
            "sign": "9fcea87666a9ec4bbcd1594298335c77",
            "msgId": "6986e84056d166",
            "lon": 113.37981777777777,
            "lat": 23.059987777777778,
            "appV": "V1.1.1",
            "model": "1",
            "appId": "xmart:appid:002",
            "timestamp": 1592722880069,
            "speechV": "2.7.0",
            "activeApp": "com.xiaopeng.montecarlo",
            "bid": "5",
            "activePage": "main",
            "originalText": null,
```

```
                "params": null,
                "eventType": null,
                "eventData": null,
                "sceneIds": null,
                "recordId": null,
                "originalTextFirstTime": null,
                "originalTextLastTime": null,
                "continuousDisplay": null
            }
        },
        {
            "templateQueryMatcherOutput": {
                "matcherResult": null,
                "matcherStatus": null,
                "slots": null,
                "utterances": null
            }
        },
        {
            "senderId": "hardwareIdNGC001@LF"
        },
        {
            "normEnsembleOutput": {
                "slots": [
                    {
                        "name": "device",
                        "value": "空调",
                        "pos": [
                            2,
                            3
                        ],
                        "rawvalue": "空调",
                        "type": "string"
                    }
                ]
            }
        },
        {
            "subTurnCount": 0
        },
        {
            "bertCrfEntityExtractorOutput": {
                "slots": [
                    {
                        "name": "device",
                        "value": null,
                        "pos": [
```

```
                            2,
                            3
                        ],
                        "rawvalue": "空调",
                        "type": "string"
                    }
                ]
            }
        },
        {
            "retrieveEntityCorrectAndNormOutput": {
                "slots": [
                    {
                        "name": "device",
                        "value": "空调",
                        "pos": [
                            2,
                            3
                        ],
                        "rawvalue": "空调",
                        "type": "string"
                    }
                ]
            }
        }
    ]
}
```

3.8.2 云端全链路

说明：收到用户请求到最终结果返回

HTTP接口地址：/v1/full-link/workflow

HTTP测试地址：http://logan-gateway.test.logan.xiaopeng.local/xp-ngc-console-boot/v1/full-link/workflow

长链接测试地址：ws://speech-int.xiaopeng.com/ngc/v1/semantic?speech_v=3.0.0&hardwareId=hardwareIdNGC001&timestamp=1668144987988

接口类型：POST

input：详见 新架构-端云交互协议 4.1

Output 详见 新架构-端云交互协议 4.2

四、难点及挑战点

1. Java wrap模型
    a. 当前能力已实现
2. Java wrap 非模型模块
    a. 主要牵涉到python使用的一些lib库功能 java如何实现，调研阶段
3. 工作流生成
4. 工作流执行
5. 图节点的等待、通知
6. 端云公共lib库的建设
7. 集成ASR 流式相关能力（待设计）
8. 集成复合句断句能力（待设计）

五、待集成能力及设计

1. ASR 流式相关
2. 复合句断句

目前这两块依赖算法团队的最终实现方案

六、实施计划

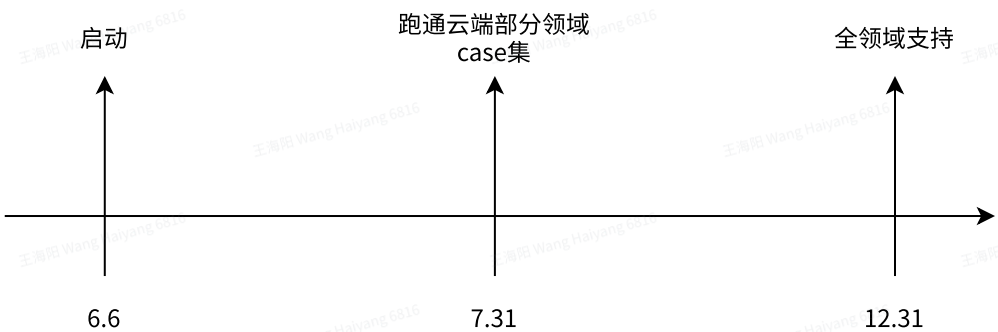
阶段一、 731-MVP-项目

阶段二、 H93 端到端落地规划

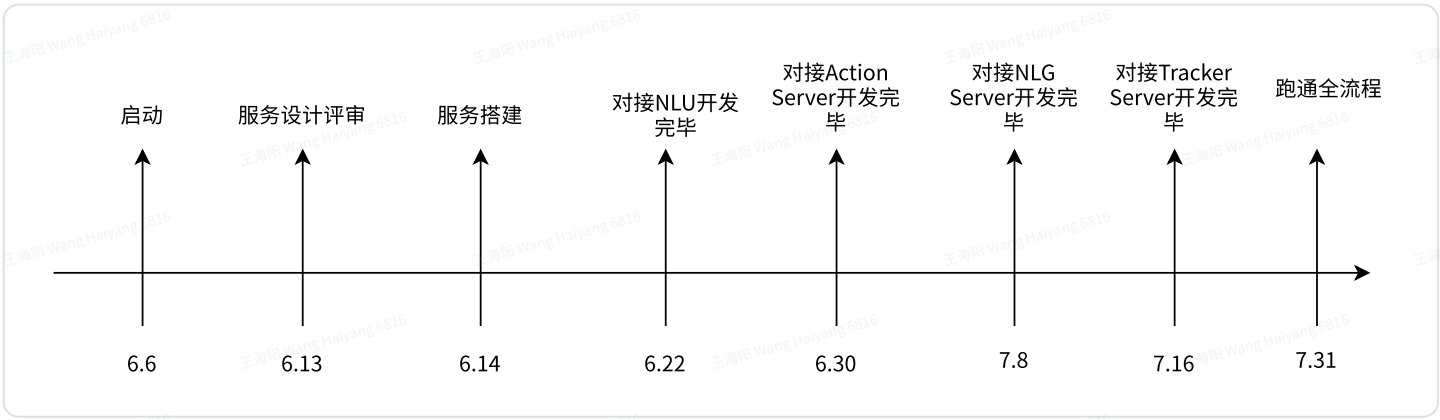
七、相关协议

详见 MVP版本服务及协议定义

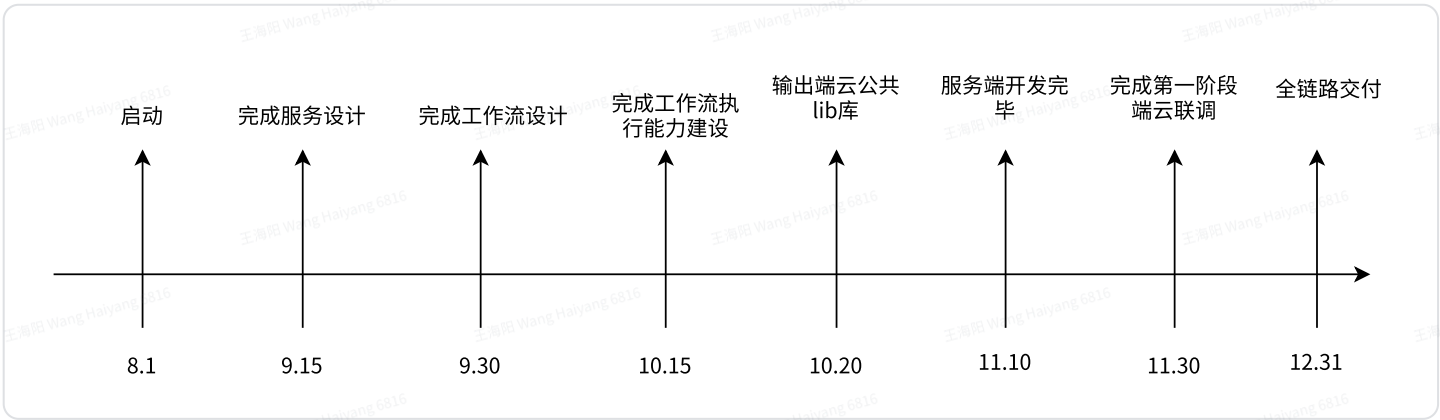
八、里程碑



阶段一



阶段二



九、相关文档

NGC-DAG工程化设计

基于1024一些个人想法 NGC-workflow

1024 工作流梳理 NGC-workflow（V1.0）

H93 0.6版本 NGC-workflow（V2.0）

DAG组件的input & output 新架构中控DAG 组件输入输出