# NGC-DAG工程化设计

	### ### ##############################	7 2
名词	释义  When the Halyang 6816  When the Halyang 6816  When the Halyang 6816	
DAG	Directed Acyclic Graph 有向无环图,任意一个节点出发,根据方向无法回到原节点的图	
入度	有向图中某点作为图中边的终点的次数之和	3
出度	对于有向图来说出边条数即为该顶点的出度	

### 一、Why DAG



- 1. 优化模型的执行
  - a. 支持节点的并行计算,提升性能
  - 2. 可以灵活地支持不同语种、车型的对话架构
    - a. 根据节点配置和组件池能够快速建立起对话全链路服务
  - 3. 工作流任务存在执行顺序,运行较为复杂。
    - a. 每一个工作流任务,是由一系列的子任务和任务与任务间的数据依赖关系构成,任务与任务之间存在数据上的交互,某些任务需要上一个任务的完成才能继续执行,由于这种依赖关系的复杂性,因此使用DAG来描述一个工作流任务

### 二、目标



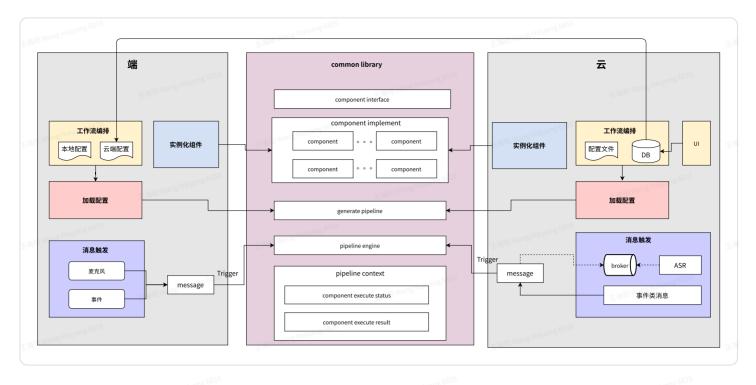
🖍 短期:使用工作流引擎及默认的配置跑通云端全链路服务

长期:

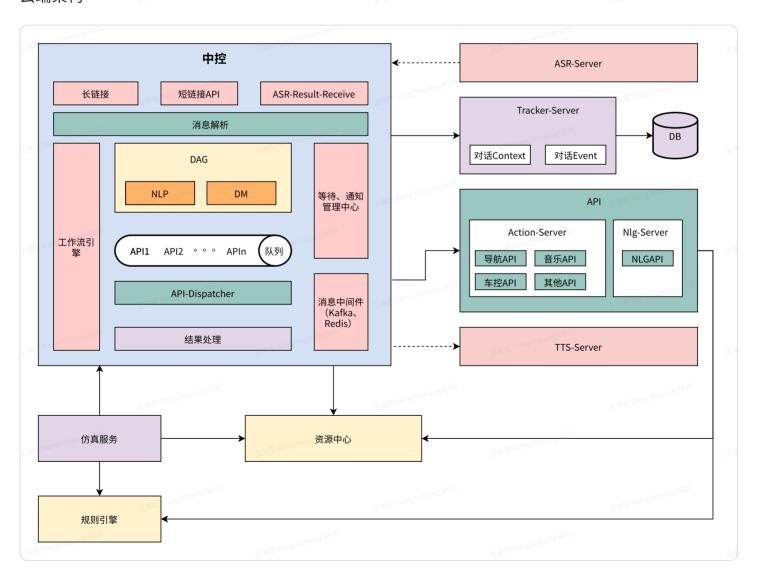
- 1. 支持平台化部署: 通过编排组件池中的组件,全链路服务可以支持不同的语种和车型
- 2. 端云一体化部署

# 三、架构设计

### 3.1、整体架构

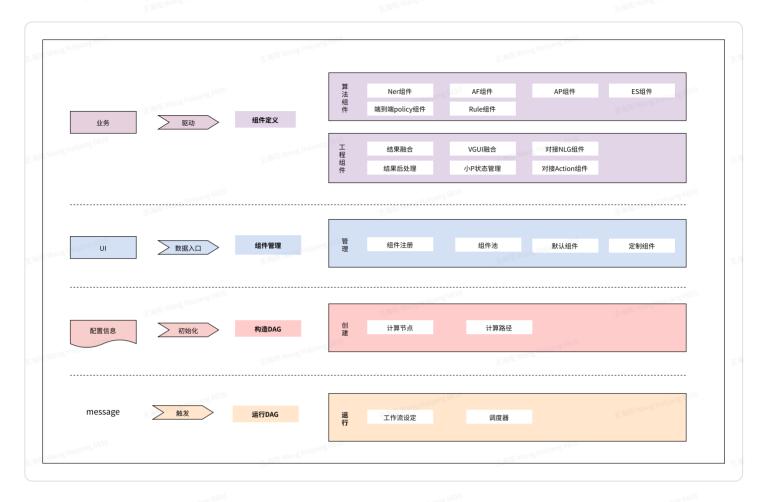


#### 云端架构



待讨论点: ASR、TTS要不要作为组件进行编排? (当前ASR、TTS独立部署,客户端直连)

### 3.2、详细设计

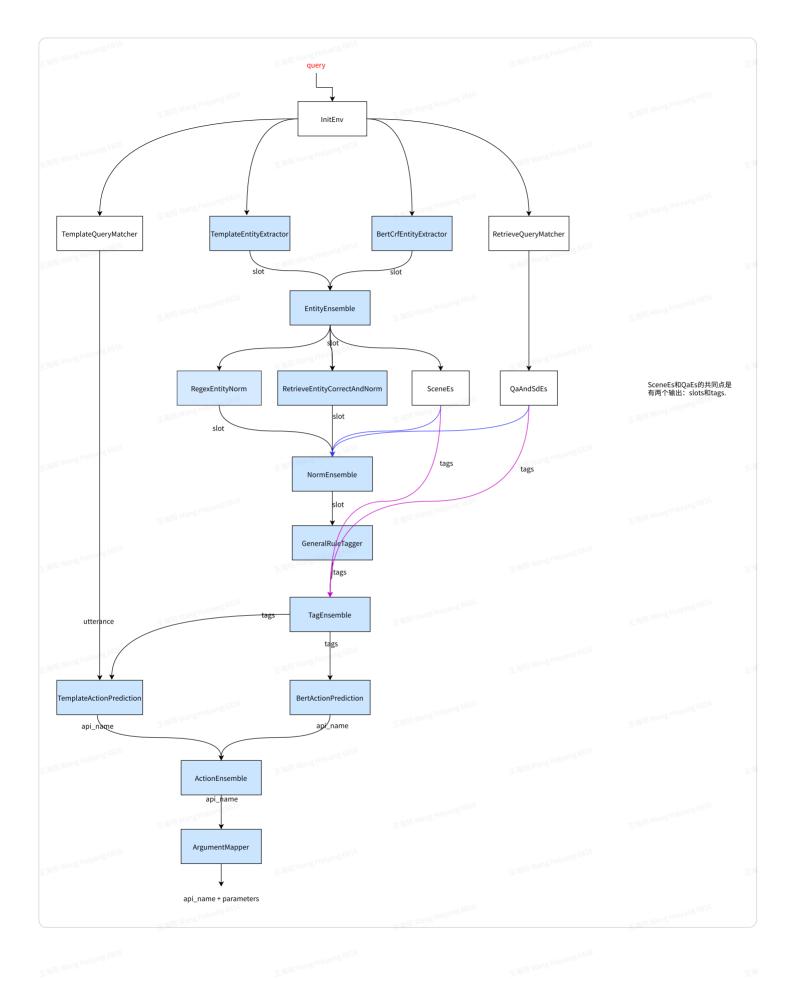


### 3.2.1、组件定义

### 3.2.1.1、组件梳理

初步划分出18个组件

(Tracker服务作为存储服务,对应交互没在图中体现,放到业务逻辑侧)



#### 3.2.1.2、Schema定义

### 3.2.1.2.1、用户请求消息

UserQueryReq

字段	类型 <sub>王海阳 Wang P</sub>	必要数据	备注  Ellife Wang halyang 58.16
query	string	yes	请求的文本
msgld	string	yes	er Halyang 6816
uid	long	no	小鹏账号的id 未登录: -1
city	string	no EMP	Nang Haiyang GB to 王海阳 Wang Haiyang GB to
lon ag Halyang 6816	double	no	经度小数点后6位 从导航客户端获取
lat	double	no	经度小数点后6位 从导航客户端获取
ts	long	yes	时间戳,如:1492397512117
activeApp	string	yes	当前app
activePage	string	yes ± MR	前台app的页面
status	string	yes	端上对话状态: start/continue/end/wait
	og Holyanii 6816		og Halyang 6816
preCmd	string	no	上一个指令码
preTs	long	no	上一个指令时间戳
pre_intent	string	no	上一个意图
vin	string	no sue	vin码, 大屏需要申请才有VIN码,真实车辆存在
switchData	string	no	服务器下发的保留字段,大屏回传
params	string	no EMP	json格式,预留字段
originalText	string EMPRINANCE	no	前端asr识别的原始结果
speaking	boolean	no	true: 大屏正在tts播报; false
scenelds	List <string></string>	no	场景ID
eventType	string	no	大屏事件处理定义
eventData	string	no ENE	Nang Haiyang <sup>(981)</sup> 王海阳 Wang Haiyang <sup>(981)</sup>
hardwareId	string ************************************	no	设备唯一编号。高级的
continuousDisplay	String	yes	连续上屏模式,loose/tight

### 3.2.1.2.2、实体词提取组件

### Input

字段	类型	必要数据	备注	
query	string	yes	请求的文本	
msgld	string	yes	消息Id	

### output

### NerModelBO array结构

字段 <sub># 38</sub> 58 Wester Hallyang 68.16	类型 <sub>全规图 Wang</sub> Y	必要数据	备注  ### Wang Halyang 6816
name	string	yes	实体词名称
pos	array	no	槽所在文本中的位置
rawvalue	string	no	原始槽值

# 3.2.1.2.3、特征提取归一化组件

# Input

字段	类型	必要数据	备注	
query	string EMB Wand H	yes	请求的文本	
msgld	string	yes	消息Id	
slots MRR Wang Halyang 6816	list <nermodelbo></nermodelbo>	no ayang sala	由ner model 输出作为输入	

# output

字段	类型	必要数据	备注	
query	string	yes	实体词名称	
msg	string	no	异常提示消息	
slots	List <slot></slot>	no No	归一化之后的槽值	

字段	类型	必要数据 📨	*************************************
name Managasas	string	yes	实体词名称
pos	array	no	槽所在文本中的位置
value	string	no no	归一化之后的槽值
rawvalue	string	no	原始槽值
type	string	no EN	槽值的类别

### 3.2.1.2.4、AP组件

### Input

字段	类型 <sub>至海阳 Wang P</sub>	必要数据	备注 <sub>王海阳Wang Hayang Gal G</sub>
query	string	yes	请求的文本
msgld	string	yes	消息Id
lastApi	string	no	上一轮的api名称
tag	string	no EAR	上一轮的api标签

### output

字段	类型 <sub>unang</sub>	必要数据	备注
actionName	string	yes	需要请求的具体api名称

### 3.2.1.2.5、Rule组件

该组件暂未启动,待与 🚷 樊骏锋 Fan Junfeng 对齐

### Input

字段	类型 宝 <sup>MPE Wang Y</sup>	必要数据	备注  EMER Warts Halyans	
query	string	yes <sub>EMB</sub>	请求的文本	
msgld	string	yes	消息Id	

lastApi	string EMPE WARE	aly <b>no</b> <sup>936</sup>	上一轮的api名称	
tag	string	no	上一轮的api标签	

### output

字段	类型	必要数据	备注	王海阳 Wang Haiyang 6816
actionName	string	yes	需要请求的具体api名称	

### 3.2.1.2.6、policy组件

目前还未启动,待与算法同学对齐 🎈 杨如栋 Yang Rudong 🛮 🗞 樊骏锋 Fan Junfeng

🧌 魏子兵 Wei Zibing 🏻 🌜 宁洪珂 Ning Hongke

字段	类型	必要数据	备注	王海阳 Wans ·	
query	string EMPH Wang	yes	请求的文本		
msgld	string	yes	消息Id	王海阳 Wang Halyang 6816	
lastApi	string	no	上一轮的api名称		
tag	string	no	上一轮的api标签	ng 6816	

### Output

字段	类型	必要数据	aaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaa
actionName	string	yes	需要请求的具体api名称
endOfTurn	boolean	yes	该轮是否结束,需要和算法同学对齐,牵涉到多轮

### 3.2.1.2.7、AF组件

### Input

	5110	王河明	Water
字段	类型	必要数据	备注  What is a second of the s
msgld	string	yes	消息Id
actionName	string	yes <sub>±MR</sub>	需要请求的具体api名称
msgld	string	yes	消息Id <sub>主题码 Wang Halyang 68.16</sub>

			- 1 6816
slots	List <slot></slot>	no	归一化之后的槽值

# output

字段	类型	必要数据	备注	
actionName	string	yes	需要请求的具体api名称	
params	list <actionparambo< td=""><td>no no</td><td>请求参数</td><td></td></actionparambo<>	no no	请求参数	
	29 6816			

### ActionParamBO

字段	类型	必要数据	<b>备注</b> 。68.18	<sub>Jang Haiyang 6816</sub>
name	string	yes	名称	
value	string	yes	参数值	6836
type	string	yes	类型	gang Halyans

### 3.2.1.2.8、Action组件

### Input

字段	类型	必要数据	备注	
msgld	string	yes	消息Id	
name	string	yes	需要请求的具体api名称	
arguments	list <actionparam BO&gt;string</actionparam 	yes	参数 <sub>mans</sub> transport of the	

### output

字段	类型	必要数据	备注
responses	list <responsebo></responsebo>	yes	actionName对应的查询数据,支持批量查询
events	list <event></event>	no EMPEWA	回调事件

### ResponseBO

字段	类型	必要数据	No Warrier Gests  A 注
code	int 王海阳Wang Haiya	yes	code码,根据此值判定响应结果的正确性
msg	string of	no gg Wa	提示信息
text	string	no	tts文本
actionName	string	no	Action的名称
command	CommandBO	no <sub>主海阳W</sub>	需要执行的指令
rule g Hayang 6816	RuleBO	no	待执行的规则
contentType	string	no	列表数据类型,默认值为none,即无列表
content	list	no	查询内容

### CommandBO

字段	类型 <sub>宝海路 Wang Halya</sub>	必要数据	备注 <sub>至海阳 Wang Halyang 0816</sub>	
code	string	yes	执行的指令	
param	obj	no 6816	执行参数	
runSequence	int 王海姆Wang Haby	no	执行顺序	
delay	a int ang 6816	no	Action的名称	

# RuleBO

字段	类型	必要数据	<b>备注</b>	
ruleId	string	no	规则Id	
param	obj	no <sub>主海阳 W</sub> 。	执行参数	

# 3.2.1.2.9、Nlg组件

# Input

		wang 6810			
字段	类型 <b>医</b>	必要数据	备注		

msgld	string 25678 Wants	yes	消息Id <b>E</b> 類El Wang Halyang Gold	Ŧ
name	string	yes	需要请求的具体api名称	
arguments	list <actionparam BO&gt;string</actionparam 	yes	参数 <sub>主網路 Wang Halyang 6816</sub>	H

### Output

确认输出 🏮 杨如栋 Yang Rudong 🏻 🚵 易晖 Yi Hui

字段	类型	必要数据	Alline Halverton American English Manus Halvanis Charles American English Manus Ch	
nlg ng Halyang 6816	string EMPR Wang P	yes	播报文本	7.

### 3.2.1.2.10、结果融合组件

### Input

字段	类型	必要数据	w 备注
msgld and 68.16	string	yes	消息Id <sub>主馬阿 Wang Hayang Sol</sub> 5
actionOutput	ActionOutput	no	action结果
nlgOutput	NlgOutput	no	nlg结果

### Output

### WsResp

字段	类型	必要数据	<b>备注</b>	
code	int	yes	code码	王海阳 Wang Halyang 6816
msg	string EMPR Wang	no	异常时赋值	
msgld	string	yes	消息Id	
msgType	string	yes	消息类别	
hardwareId	string	yes	设备ID	
data	UserQueryResp	yes	数据体	王海阳 Wang Halyane 6816

### UserQueryResp

字段	类型	必要数据	Mariante Basis <b>备注 E #FEL Wante Halyane</b> Basis
nlg ng Halyang 5816	string	no	播报文本
command	CommandBO	no	需要执行的指令
rule	RuleBO	no	待执行的规则
contentType	string	no	列表数据类型,默认值为none,即无列表
answer EMPH	list	no	查询内容
status pang 6816	string	yes	对话状态
priority	int ng Halyang 6816	yes	分值 mans Halyang 6816
query	string	no	用户说的话

### 3.2.1.2.11、结果后处理组件

input

WsResp

Output

WsResp

### 3.2.2、组件管理

### 3.2.2.1、注册

由于需要平台化部署,所以组件需要持久化。

Type: post

Url: {host}/v1/component/register

请求体:

字段	类型	允许为空	释义	备注
name	string	N aiyang 6816	组件名称	比较重要
desc	string	Υ	组件描述	
className and Hang Hangar	string	N E	组件的类对象	王海阳 Wang Halyang Gaste
outputClassName	string EMPELWARE	Y 6816	王海阳W	angHalyang 6816

			组件的输出类对 象	ang Halyang 6816
inputClassName (September 1997)	string	Y ENNE	组件的输入类对象	工海阳 Wang Halyang 6816
parents	list 8 08315	N EME	父节点 ************************************	依赖的节点,为空时表示头节 点
childs	list Emps wang	™ N 11 687e	子节点	作为子节点的输入,为空时表 示叶子节点
supportedLanguages	list	A Eller	支持的语种	若为空表示支持所有的语种
supportedCarTypes	list <sub>王海阳 Wang</sub>	Y Awang 6816	支持的车型	若为空表示支持所有的车型
modelPath	string	Y	模型存储路径	工程组件时为空
				29 6816

### 响应:

字段	类型	允许为空	释义	示例	
code	E Mars Wang Haiyang Cint	N	code码	200	
data MER Wang Halyang 6816 MER Wang Halyang 6816	list 王爾冠 Wang Hayang 6816		数据体	{	
實語 Wang Haiyang 6816		王海阳 Wang Halyang 6816		王洞阳Winn}Halyang 6816	
msg	string	Υ	消息提示	王海田 Wang Haiyang	

# 3.2.2.2、修改

Type: post

Url: {host}/v1/component/update

请求体:

字段 	类型	允许为空	释义	备注

id Emilia wang hayang 6816	long EMER Wang	<sup>131</sup> <b>N</b> 6 6876	数据id	ang Halyang 6816
name	String	N	组件名称	比较重要
desc	String	Y = 6816	组件描述	- c816 王湖起 Ann. 18
parents	list EMPERWANE	N Table	父节点 Name Haryang Sale	依赖的节点,为空时表示头节点
childs EMPR Wang Halyang 6816	list	N Jaiyang 6816	子节点	作为子节点的输入,为空时表 示叶子节点
supportedLanguages	list	Y ENER	支持的语种	若为空表示支持所有的语种
supportedCarTypes	list	<sub>1811</sub> Yng 6816	支持的车型	若为空表示支持所有的车型
path <sub>王剛阳 Wang Haya</sub>	string	Y	模型存储路径	王海阳 Wang Halyang 6816

### 响应:

字段		类型	允许为空	释义	示例
code		int	N	code码	200
data	王海阳 Wang Haiyan	list	N	数据体	{
			Nang Haiyang 6816		"id": 1,
					"name": "ner",
			壬	實用 Wang Haiyang 6816	"desc": "实体词识别"
Wang Haiyang 6816			Nang Naiyang 6816		} Halyang 6816
msg		string	Υ	消息提示	3.79
			至	海阳 Wang Haiyang 6816	王海阳 Wang Haiyang 6816

# 3.2.2.3、注销

Type: get

Url: {host}/v1/component/delete?id=xxx

请求体:

字段	类型	允许为空	释义	备注
				aug Haise

id	long	Wang Hair N & 6816	数据id	— mpH Wang Haiyang 6816	
----	------	--------------------	------	-------------------------	--

### 响应:

字段		类型	允许为空	释义	示例
code		int	N EN	code码	200 Ears wans Halyang 58 16
data Hayang 6816				数据体 <sub>English</sub>	"id": 1  "isDelete": 1 SE WATE HAIYATE SEAS  }
msg	王海阳 Wang Halyan	string	Υ	消息提示	王洞阳 Wang Halyang 6816

### 3.2.2.4、查询

Type: post

Url: {host}/v1/component/list

请求体:

字段	类型	允许为空	释义	备注
id	long	Y	数据id	ang Halyang 651°
pageIndex	int	Y	Nang Halyang 6816	王海阿 Wang Halyang 6816
pageSize	int = NAME WANE	13HY 86816	主海阳平	ingHalyang 6816

### 响应:

字段	类型	允许为空	释义	示例	177
code State Manus Har	int	N ENE	code码	200 EMB Wang Halyang 6816	
data	list ENDEWARDS	N 6816	数据体	[{ "id": 1	- 5
王海阳 Wang Hal	yang 6816		Wang Haiyang 6816	"name": "ner", " "ayang sala	
王海兒Wang Halyang 6816	王海阳Wang	Maiyang 6816	王海阳》	"desc": "实体词识别"	111

msg	王海阳 Wang Halyar	string	Y	消息提示	王海阳 Wang Halyang 6816
				王海阳 Nang Halyang 6816	93 "path":""
王海阳Wang Halyang 6816 王海阳 Wang Halyang 6816			yang haiyang 6816 Yang haiyang 6816	王海阳 Wang Haiyang 6816	"parents": []  "childs": []  "supportedLanguages":  cn  "supportedCarTypes":H

#### 3.2.3、构造DAG

#### 3.2.3.1、概述

- 1. 对模块化功能进行组件化,一个组件可以作为一个图节点进行编排
- 2. 通过指定的配置项进行DAG的构造
  - a. 短期: 仅通过配置 车型和语种 两个维度的数据即可,走默认的配置节点及工作流
  - b. 长期:由于后期可能会引入更多的组件,将提供一个UI界面把所有的组件进行展示供编排
- 3. 由于对话场景的特殊性,构造的DAG,其入度、出度为0的节点各一个
- 4. 在编排时初始化默认的节点:

		Wang Haiye	Wang Hary	
节点	名称	备注		
ner	命名实体识别		王海阳 Wang Halyans CR15	王海阳Wang
featurizer	特征提取、归一化	E搁在 Mang Halyang 6816	至獨阳 Wang Halyang	816
ар	action预测		主海阳 Wang Halyang 6816	
apRule	ap配套的规则	Wang Haiyang 6816	ang wang Halyang <sup>(</sup>	
policy	对话策略	决策turn及se		
af	参数填充		王海阳 Wang Halyang Obste	王海阳 Wang
action	action服务	三海阳 Wang Haiyang 6816	王獨阳 Wang Halyang	816
nlg	nlg服务	了	若后期被action直接引用了,那	就可以摘除
resultFusion	结果融合	E海阳 Wang Haiyang 6816	王海阳 Wang Halyang	2810

resultPost	结果后处理	当前先按一个组件处理,后期按需拆分。里面包含了 小P
		状态管理、VGUI融合、耗时埋点、结果下发、结果广播等

### 3.2.3.2、图

### 图schema:

字段名词	类型	允许为空	释义	备注
nodes	list <sub>E</sub> 6816	N Empl	所有参与的节点 名称集合	
edges	list <edge></edge>	<sup>917</sup> N <sup>8</sup> 6876	边集合	根据组件名称和childNodes

### 边定义

字段名词	类型 基準短Wang	允许为空	释义	备注	
source East Wang Haryar	GraphNode	N ENR	边起点		
destination	GraphNode	N g 6816	边终点		

### 3.2.3.3、节点

# GraphNode

字段名词	类型	允许为空	释义	备注 ** #55 Wang Halyang 6816
name	string	N Haiyang 6816	节点名词	一般和组件名词保持一致
MR Manus USES 王剛居 Manus Halv	string	N	对应的组件类路径	用于构造执行实例
parentNodes	List <graph Node&gt;</graph 	Yyang 6816	依赖的父节点	
childNodes 5/8/8 1888 1888 1888	List <graph Node&gt;</graph 	Y E	子节点	王海阳 Wang Haiyang balis
eager EMENAREHAN	boolean	Y	是否需要加载(默 认为true)	特指模型是否立即加载初始 化
ast Wang Haiyang 6816	ro War	g Haiyang 6816		

isTarget	boolean	N 126816	是否是目标节点	如果target节点执行完毕,相 当于整个计算结束。面向编 排时使用
工海阳 Wang Haily				排列使用
input	InputDescri ptor	Y gHaiyang 6816	节点的输入	
Output  ENER Wang Halyang 6816	OutputDesc riptor	Y <sub>E Halyang 6816</sub>	节点的输出	扩展用,可以灵活支持数据 的存储和应用,本地缓存 or 分布式缓存

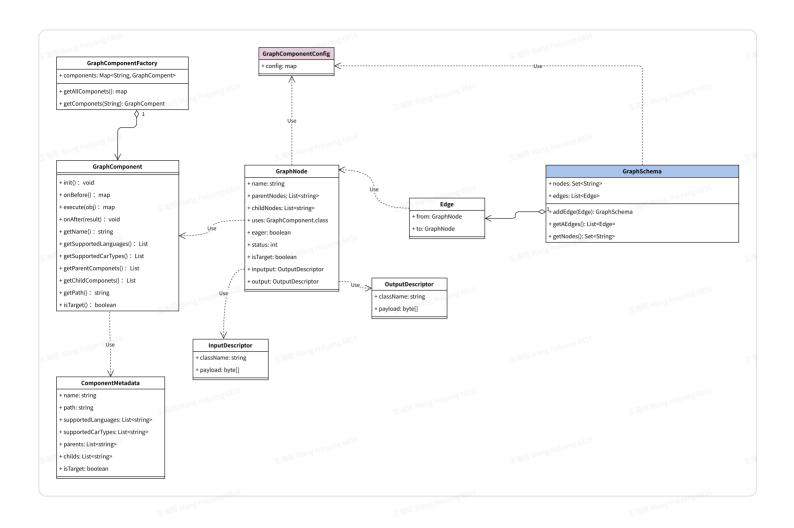
# InputDescriptor

		Haiyalla		La Haiyalla
字段名词	类型	允许为空	释义	备注
className	string	N	类名	
payload	byte[]	gHNang6816	数据体	
	og 6816		296816	

# OutputDescriptor

字段名词	类型	允许为空	释义	备注 <sub>生期</sub> 和 Wang Halyang 5816
className	string	N Halyang 6816	类名	g Halyang 6816
payload	byte[]	N	数据体 <sub>Stationarie</sub> co.15	王海阳 Wang Halyang 6816

# 3.2.3.4、核心类图

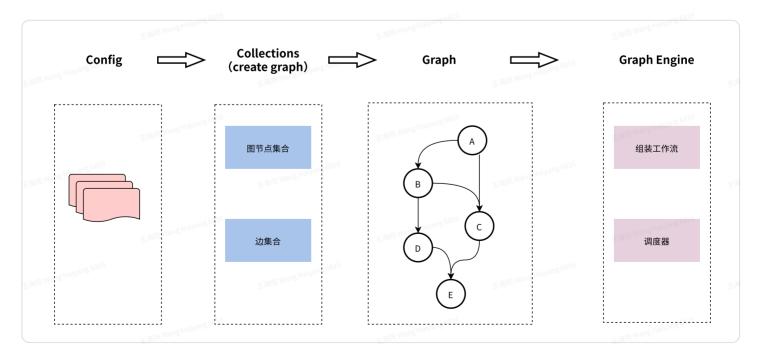


### 3.2.4、DAG Workflow Engine

基于开源easy-flow框架进行二次开发

调研: 冒工作流引擎方案对比

整体流程图如下:



#### 3.2.4.1、配置文件

当前V2.0阶段使用配置文件进行支持,主要关于两部分信息:组件的依赖关系和分组信息,如下示例:

```
components:
 2
        - name: A
          dependency:
 3
 4
        - name: B
 5
          dependency: A
        - name: C
 6
 7
          dependency: A,B
        - name: D
 8
          dependency: B
 9
10
        - name: E
          dependency: C,D
11
```

#### 3.2.4.2、图数据

根据配置文件中的组件信息在服务启动时进行构图,主要生成节点和边核心代码如下:

```
1 public static Graph composeGraph(List<ComponentInfo> components) {
 2
       Set<GraphNode> nodes = new HashSet<>(components.size());
 3
       Map<String, GraphNode> nodeMap = new HashMap<>(components.size());
       for (ComponentInfo component: components) {
 4
           GraphNode node = GraphNode.builder()
 5
 6
                    .name(component.getName())
                    .className(component.getClassName())
 7
 8
                    .parentNodes(component.getDependency())
                    .groupName(component.getGroupName())
9
                    .build();
10
           nodes.add(node);
11
           nodeMap.put(component.getName(), node);
12
13
       Set<Edge> edges = new HashSet<>();
14
       for (GraphNode node : nodes) {
15
           final List<String> parentNodes = node.getParentNodes();
16
           for (String resource : parentNodes) {
17
18
               if (StringUtils.isBlank(resource)) {
                    continue;
19
20
               Edge edge = Edge.builder().sourceNodeName(resource).destinationNodeN
21
               edges.add(edge);
22
23
```

```
24 }
25
26 return Graph.builder().nodes(nodes).edges(edges).build();
27 }
```

#### 示例如下:

```
1 {
 2
        "edges":[
            {
                 "destinationNodeName": "C",
 4
                 "sourceNodeName": "B"
 5
 6
            },
 7
            {
 8
                 "destinationNodeName": "D",
                 "sourceNodeName": "B"
 9
10
            },
11
            {
                 "destinationNodeName": "B",
12
                 "sourceNodeName": "A"
13
14
            },
15
            {
                 "destinationNodeName": "C",
16
                 "sourceNodeName": "A"
17
            },
18
19
            {
                 "destinationNodeName": "E",
20
                 "sourceNodeName": "D"
21
            },
22
23
                 "destinationNodeName": "E",
24
                 "sourceNodeName": "C"
25
26
            }
27
        ],
        "nodes":[
28
            {
29
                 "name": "E",
30
                 "parentNodes":[
31
                     "C",
32
                     "D"
33
34
                 ]
            },
35
36
37
                 "name": "A",
                 "parentNodes":[
38
```

```
39
40
                 1
41
             },
             {
42
                  "name": "B",
43
                  "parentNodes":
44
                      "A"
45
46
47
             },
             {
48
                  "groupName": "norm",
49
                  "name": "C",
50
                  "parentNodes": [
51
                       "A",
52
                       "B"
53
54
             },
55
56
             {
                  "groupName": "norm",
57
                  "name": "D",
58
                  "parentNodes": [
59
                      "B"
60
61
                  ]
62
             }
63
        ]
64 }
```

#### 3.2.4.3、生成执行工作流

根据图数据生成对应的执行工作流,目的是生成工作流引擎需要的执行代码。主要思想包含两部分:

- 1. 根据图数据为每个节点生成对应的执行优先级(用权重进行标识,权重越小执行优先级越高) 权重计算公式: 所有父节点的权重 + 当前节点的入度
- 2. 根据分组信息为相关节点打上组信息,根据组的执行属性进行相关代码逻辑的生成核心代码如下:

```
1 /**
2 * 按图顺序从header节点到tail节点 组装工作流
3 * 执行步骤:
4 * 1.根据边获取到各个节点的依赖和孩子节点
5 * 2.根据边上的各个节点及对应的孩子节点计算各个节点的权重,权重越小计算的优先级越高
6 * 2.1 设置被指向节点优先级=所有父节点的和+入度
7 * 2.2.遍历边表,先生成一个父节点和子节点的映射map
8 * 2.3.再次遍历边表,如果是头节点,默认值设0,如果是尾节点默认设0,已存在=节点原有值+1
9 * 2.4.递归给尾节点的所有下游节点+1
```

```
10
    * 3.根据权重进行节点的排序
    * 4.根据有序的节点集合构造对应的工作流节点集合
11
12
    * @param graph 图
13
    * @param groupInfos 分组信息
14
    * @return 带执行舒徐的工作流节点集合
15
16
17 public static AIWorkFlow composeWorkFlow(Graph graph, Map<String, GroupInfo> gro
       Set<Edge> edges = graph.getEdges();
18
       //存放孩子节点
19
20
       HashMultimap<String, String> childMap = HashMultimap.create();
21
       HashMultimap<String, String> parentMap = HashMultimap.create();
22
       //获取所有的节点的孩子和依赖节点
23
       for (Edge s : edges) {
24
25
           String sourceNodeName = s.getSourceNodeName();
          String destinationNodeName = s.getDestinationNodeName();
26
27
          childMap.put(sourceNodeName, destinationNodeName);
          parentMap.put(destinationNodeName, sourceNodeName);
28
29
       }
       //根据边上的各个节点及对应的孩子节点计算各个节点的权重,权重越小执行的优先级越高
30
       TreeMap<String, Integer> priorityMap = new TreeMap<>();
31
       for (Edge edge : edges) {
32
33
          String sourceNodeName = edge.getSourceNodeName();
          String destinationNodeName = edge.getDestinationNodeName();
34
          priorityMap.putIfAbsent(sourceNodeName, 0);
35
          priorityMap.putIfAbsent(destinationNodeName, 0);
36
          priorityMap.computeIfPresent(destinationNodeName, (k, v) \rightarrow v = v + 1);
37
           //更新目标节点+递归更新自己的孩子节点
38
           updateChild(priorityMap, childMap, destinationNodeName);
39
40
       //取工作流中所有的节点集合
41
       List<WorkFlowNode> workFlowNodes = getWorkFlowNodes(priorityMap, childMap, p
42
       return AIWorkFlow.builder().workFlowNodes(workFlowNodes).groupInfos(groupInf
43
44 }
```

#### 生成工作流示例如下:

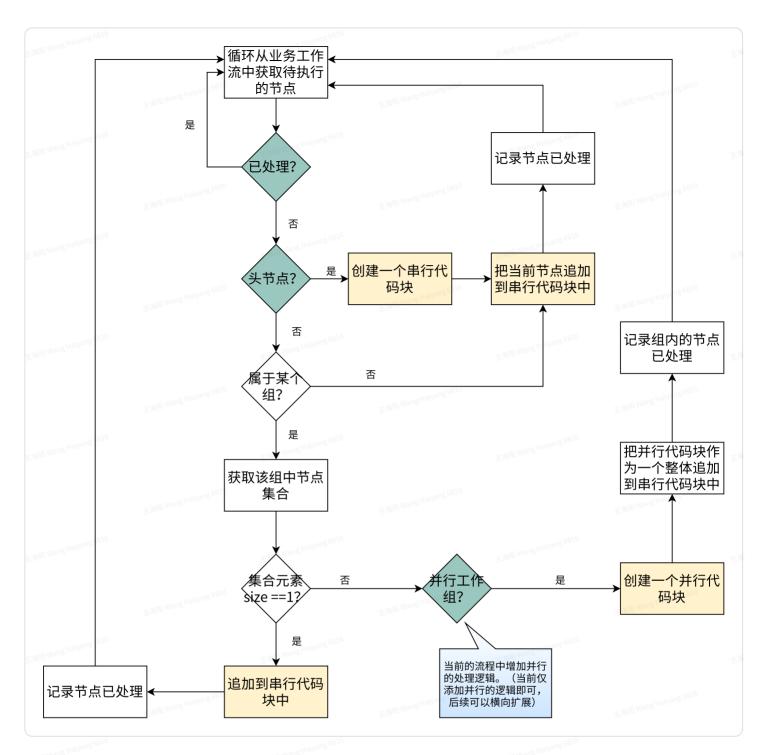
```
"executeType": "CONCURRENT",
 8
                 "name": "norm"
 9
10
            }
        },
11
        "workFlowNodes": [
12
13
            {
14
                 "childNodes":[
                     "В",
15
                     "C"
16
17
                 ],
                 "nodeName": "A",
18
                 "parentNodes":[
19
20
21
                 ],
                 "priority":0
22
23
            },
            {
24
                 "childNodes":[
25
                     "D",
26
                     "C"
27
28
                 ],
                 "nodeName": "B",
29
                 "parentNodes":[
30
                     "A"
31
32
                 ],
                 "priority":1
33
34
            },
            {
35
                 "childNodes":[
36
                     "E"
37
38
                 "groupName": "norm",
39
                 "nodeName":"D",
40
                 "parentNodes":[
41
                     "B"
42
43
                ],
                 "priority":2
44
            },
45
46
            {
                 "childNodes":[
47
                     "E"
48
49
                 "groupName":"norm",
50
51
                 "nodeName": "C",
                 "parentNodes":[
52
                     "A",
53
                     "B"
54
```

```
55
                "priority":3
56
            },
57
            {
58
59
                "childNodes":[
60
                ],
61
                "nodeName": "E",
62
                "parentNodes":[
63
                    "D",
64
                     "C"
65
66
                "priority":7
            }
68
69
       ]
70 }
```

### 3.2.4.4、动态生成引擎执行的代码

流程如下:

版本: V1.0



#### 核心代码如下:

```
1 private WorkFlow getDynamicWorkflow(AIWorkFlow workFlow, Map<String, GraphCompon
       final List<WorkFlowNode> workFlowNodes = workFlow.getWorkFlowNodes();
 3
       final Map<String, GroupInfo> groupInfos = workFlow.getGroupInfos();
       //记录节点的处理记录,是否已处理; 1-是, (0 or null) -否
 4
       Map<String, Integer> nodeProcessRecordMap = new HashMap<>();
 5
       SequentialFlow.Builder.NameStep nameStep = SequentialFlow.Builder.aNewSequen
 6
       SequentialFlow.Builder.ThenStep thenStep = null;
 7
       final int nodeSize = workFlowNodes.size();
 8
       for (int i = 0; i < nodeSize; i++) {</pre>
 9
10
           final WorkFlowNode node = workFlowNodes.get(i);
           final String nodeName = node.getNodeName();
11
```

```
12
           //如果节点已处理则pass
          if (hadProcessed(nodeName, nodeProcessRecordMap)) {
13
14
              continue;
          }
15
           final GraphComponent component = componentMap.get(nodeName);
16
           //头节点,由于业务的特殊性,需要初始化运行上下文,所以用初始化环境组件作为了头节点。
17
           // 这里头节点的判定未使用GraphUtil.findHeadNodes()函数进行更多复杂的操作,直接
18
          if (thenStep == null) {
19
              thenStep = nameStep.named("start dynamic workflow").execute(componen
20
              markNodeHadProcessed(nodeProcessRecordMap, nodeName);
21
22
           } else {
              final String groupName = node.getGroupName();
23
              //不属于任何一个组,独立运行即可
24
              if (null == groupName) {
25
                  thenStep.then(component);
26
27
                  markNodeHadProcessed(nodeProcessRecordMap, nodeName);
              } else {
28
29
                  final GroupInfo groupInfo = groupInfos.get(groupName);
                  //组中只包含自己一个组件(理论上不推荐这么配置),增加groupInfo == nul
30
                  if (groupInfo == null || onlyContainsSelf(groupInfo.getComponent
31
32
                      thenStep.then(component);
                      markNodeHadProcessed(nodeProcessRecordMap, nodeName);
33
                  } else {
34
35
                      final List<String> components = groupInfo.getComponents();
                      Work[] works = new Work[components.size()];
36
                      String[] workNames = new String[components.size()];
37
                      int k = 0;
38
39
                      for (int j = 0; j < components.size(); <math>j++) {
                          final String name = components.get(j);
40
                          //防止组件多次执行的情况-处理配置出现错误的场景
41
42
                          if (hadProcessed(name, nodeProcessRecordMap)) {
                              continue;
43
44
                          }
                          works[k] = componentMap.get(name);
45
46
                          workNames[k] = name;
47
                          k++;
                      }
48
                      //若组内的组件需并行操作(目前业务上仅有并行这一种情况,后续其他的逻
49
                      if (ExecuteTypeEnum.CONCURRENT.name().equals(groupInfo.getEx
50
                          final ParallelFlow parallelFlow = ParallelFlow.Builder.a
51
                          thenStep.then(parallelFlow);
52
                          markNodeHadProcessed(nodeProcessRecordMap, workNames);
53
                      }
54
55
                  }
              }
56
57
          }
58
```

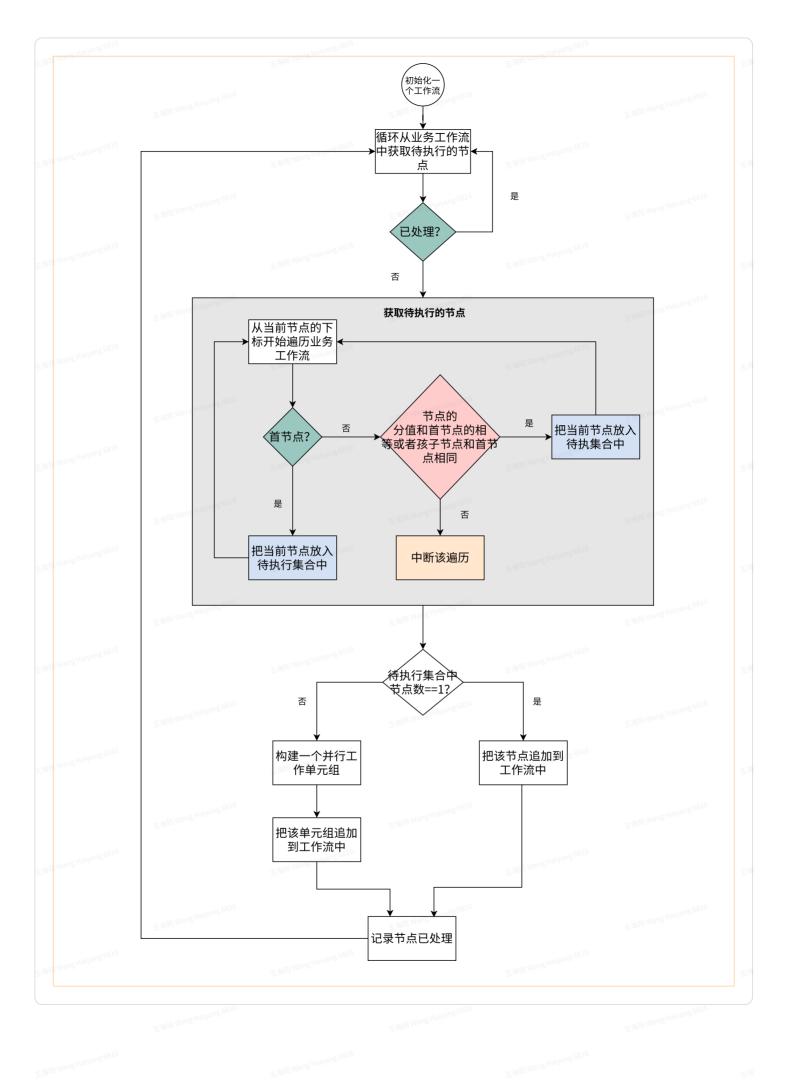
```
if (thenStep == null) {
    throw new RuntimeException("an exception occurred during get dynamic wor
}
return thenStep.build();
}
```

#### 感兴趣的同学可以看下对应的mr:

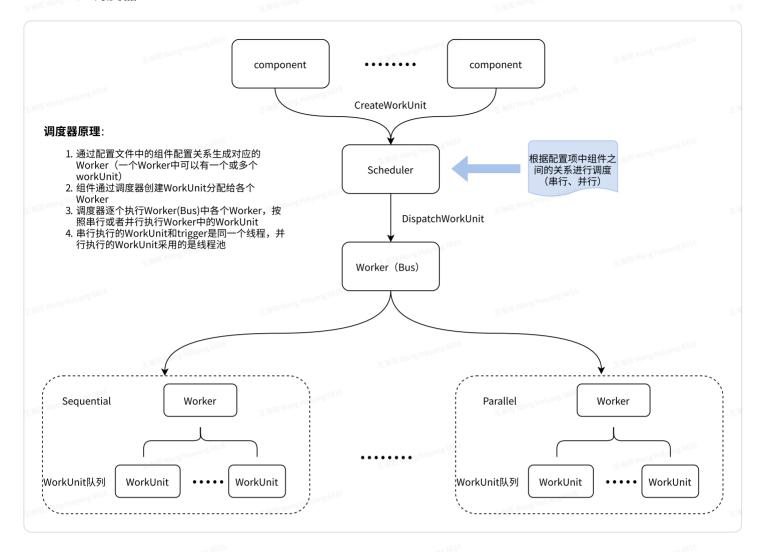
http://gitlab.xiaopeng.local:18080/xai/ngc/xp-ngc-library/merge\_requests/94/diffs (端云公共 lib)

http://gitlab.xiaopeng.local:18080/xai/ngc/xp-ngc-console/merge\_requests/55/diffs (中控)

版本: V2.0 (当前使用的是该版本)



#### 3.2.4.5、调度器



#### 串行处理逻辑

```
public WorkReport execute(WorkContext workContext) {

WorkReport workReport = null;

for (Work work : workUnits) {

workReport = work.execute(workContext);

if (workReport != null && FAILED.equals(workReport.getStatus())) {

break;

}

return workReport;

}
```

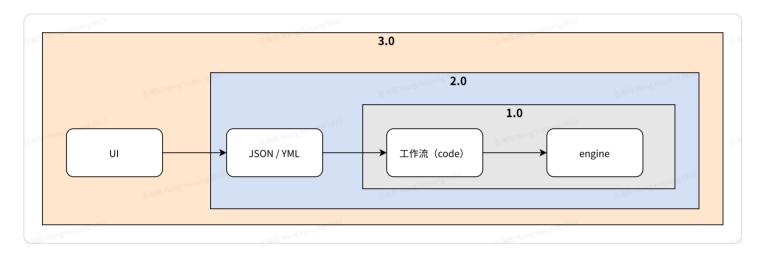
#### 并行处理逻辑

```
public ParallelFlowReport executeInParallel(List<Work> workUnits, WorkContext wo
   List<Callable<WorkReport>> tasks = new ArrayList<>(workUnits.size());
   workUnits.forEach(work -> tasks.add(() -> work.execute(workContext)));
```

```
List<Future<WorkReport>> futures = this.workExecutor.invokeAll(tasks);
       Map<Work, Future<WorkReport>> workToReportFuturesMap = new HashMap<>();
 5
       for (int index = 0; index < workUnits.size(); index++) {</pre>
 6
           workToReportFuturesMap.put(workUnits.get(index), futures.get(index));
 7
 8
       List<WorkReport> workReports = new ArrayList<>();
9
       for (Map.Entry<Work, Future<WorkReport>> entry : workToReportFuturesMap.entr
10
           workReports.add(entry.getValue().get());
11
12
       }
       ParallelFlowReport workFlowReport = new ParallelFlowReport();
13
       workFlowReport.addAll(workReports);
14
       return workFlowReport;
15
16 }
```

#### 3.2.4.6、工作流演进方案

分三个阶段进行推进,当前处于V2.0阶段



- V1.0、硬编码引擎所需的技术工作流,使用引擎跑通全链路
- V2.0、通过配置项生成业务工作流,通过业务工作流生成引擎需要的技术工作流,再把技术工作流灌 入引擎执行
- V3.0、通过UI提供编排组件的能力,通过编排生成业务工作流,通过业务工作流生成引擎需要的技术工作流,再把技术工作流灌入引擎执行

### 四、难点

- 1. 各个组件schema定义,组件包括:
  - a. 模型组件
  - b. 工程组件
- 2. java工程加载模型及部署
- 3. 工作流编排

- 4. 工作流引擎的执行
- 5. 端云一体化

# 五、工作项-List

### ■ 工作项

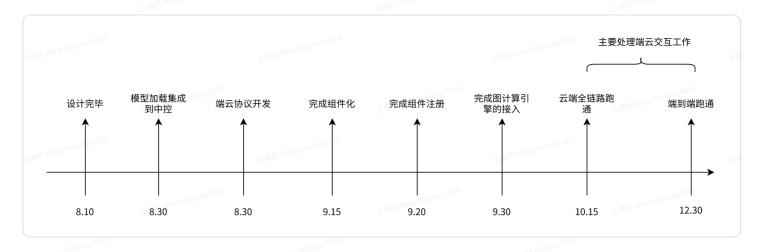
	♣ AE 工作项	🖽 日期	A= 人员	⊙ 版本规划	
		3			
<b>v</b> 1	0 10条记录	wang 6816			
1	整体设计	2022/08/10	杨如栋	1.0	
2 ang Haiyar	模型加载集成到中控	2022/08/30	杨如栋	1.0	
3	端云协议开发	2022/08/30	杨如栋	1.0	
4	完成组件化	2022/09/15	杨如栋	1.0	
5 Haiyar	java加载模型部署到GPU服务节点	2022/08/30	杨如栋	1.0	
6	组件注册	2022/09/20	杨如栋	1.0	
7	工作流编写	2022/09/21	杨如栋	1.0	
3	完成工作流引擎的接入	2022/09/30	杨如栋	1.0	
) J	云端全链路集成	2022/10/15	杨如栋	1.0	
.0	集成客户端	2022/12/30	杨如栋	1.0	
▼ 2.0 5 条记录		m Wang Halyang 6816			
1	输出工作流library	2023/01/05	杨如栋	2.0	
2	客户端组件化开发	王海阳 Wang Haiyang 6816		2.0	
3 Na Haiyar	客户端基于工作流引擎跑通本地对话			2.0	
4	本地加载模型实验	3	海阳 Watty	2.0	
5	基于配置自动转化为工作流	2022/11/30	杨如栋	2.0	
▼ 3.0 3条记录		<sub>主海阳 Wang Halyang 6816</sub>			
1	工作流编排提供UI界面			3.0	
2	基于UI自动完成工作流的编写	2023/03/30	杨如栋	3.0	
3 云端代码进行裁剪客户端运行实验		2023/04/28	杨如栋	3.0	

18 条记录

ps: 人员分配TBD

### 六、里程碑

短期:~12.30



### 七、远期规划

1. 端云一体化,工作流共享: 【library形式输出】

2. 提供UI支持组件编排

3. 日志国际化

a. 不同语种输出的日志进行国际化配置

### 八、组件化收益

#### 田 组件化

		1		
王海阳 Vang Hali	â Æ 组件名称	A≅ 语种	A= 车型	
1	实体词提取	n	m Emma Wang	
2	ES(特征提取、归一化)	n	m	
王海阳 Wang Hair	AP组件	n	Haiyang <del>0810</del> <b>M</b>	
4	Rule组件	n Haiyang 6816	m	
5	policy	1 毛海阳 Wang Haiyana	<b>1</b> 王海阳 Wans	
EMER W 68 Hair	AF组件	1	$H\dot{f 1}$ ang 68 $16$	
7	Action组件	n	1~m	
8	Nlg组件	1 王海阳 Wang Hayang 6816	1 王海阳Wang	
± 1 <b>5</b> FB W 9 g Hair	结果融合组件	1	$^{\rm H}$ $^{ m 1}$ ang $^{ m 6816}$	
10	结果后处理组件	1	1~m	
		-016		

# 九、相关调研

- 1. Rasa-DAG
- 2. Apache tez https://tez.apache.org/index.html
  - a. 基于DAG的数据处理框架
  - b. 优化了MapReduce计算
  - c. https://github.com/apache/tez
- 3. 分布式工作流引擎 https://github.com/okayrunner/piper
- 4. 工作流引擎 https://github.com/j-easy/easy-flows