Zigent 多智能体系统源码解读

核心架构

1. 基础组件

- **BaseAgent**: 基础智能体类，提供了智能体的基本功能
- **ManagerAgent**: 管理者智能体，负责协调和管理其他智能体
- **TaskPackage**: 任务包装器，用于封装和传递任务信息
- **AgentAct**: 智能体动作类，定义智能体可执行的动作

2. 关键类解析

ManagerAgent 类

```
class ManagerAgent(BaseAgent):
    def __init__(self, llm, name, role, TeamAgents=[], ...):
        # 初始化管理者智能体
        # 可以管理多个劳工智能体(TeamAgents)
```

主要功能：

- 团队管理：维护和协调多个智能体
- 任务分发：将任务分配给适当的智能体
- 结果整合：收集和整合各个智能体的执行结果

Philosopher 类

```
class Philosopher(BaseAgent):
    def __init__(self, philosopher, llm, actions=[], manager=None, ...):
        # 初始化哲学家智能体
        # 每个哲学家都有自己的角色定义和思考方式
```

特点：

- 角色特化：每个哲学家都有独特的思考视角
- 动作处理：实现了 forward 方法来处理动作执行
- 错误防护：添加了参数检查机制

工作流程

1. **任务初始化**
    - 创建任务包(TaskPackage)
    - 分配给管理者智能体
2. **任务执行**
    - 管理者分析任务
    - 选择合适的智能体
    - 转发任务并收集结果
3. **结果处理**

- 整合各个智能体的输出
- 生成最终答案

常见 Bug 及解决方案

1. FinishAct 缺少 response 参数

```
TypeError: FinishAction.__call__() missing 1 required positional argument:
'response'
```

问题原因：

- FinishAct 动作需要必需的 response 参数
- 某些智能体在执行 Finish 动作时未提供该参数

解决方案：

```
def forward(self, task: TaskPackage, agent_act: AgentAct) -> str:
    if agent_act.name == FinishAct.action_name and 'response' not in
agent_act.params:
        agent_act.params['response'] = "默认回答"
    return super().forward(task, agent_act)
```

2. 智能体匹配问题

当前实现：

```
def agent_match(self, agent_name: str, agent: ABCAgent) -> bool:
    if agent_name == agent.name:  # exact match
        return True
    return False
```

潜在问题：

- 仅支持精确匹配
- 可能错过相似名称的智能体

改进建议：

- 添加模糊匹配支持
- 实现智能体别名系统
- 添加智能体角色匹配

最佳实践

1. **智能体初始化**
    - 为每个智能体提供清晰的角色定义
    - 确保必要的参数都已正确设置
2. **任务设计**
    - 任务指令要清晰明确
    - 提供足够的上下文信息
3. **错误处理**

- 实现参数验证
- 添加适当的默认值
- 记录详细的错误信息

Bug 深入分析

1. 模型能力与Prompt设计分析

示例任务设计

```
# 当前的示例任务过于简单
exp_task = "What do you think the meaning of life?"
exp_task_pack = TaskPackage(instruction=exp_task)

# 示例动作链过于理想化
act_1 = AgentAct(
    name=ThinkAct.action_name,
    params={INNER_ACT_KEY: """Based on my thought..."""}
)
obs_1 = "OK. I have finished my thought..."
```

问题分析

1. **示例局限性**
   - 示例任务过于简单，没有体现复杂场景
   - 示例回答过于标准化，缺乏多样性
   - 动作链没有展示错误处理场景
2. **Prompt不足**
   - 缺少对必需参数的明确说明
   - 没有展示错误情况下的处理方式
   - 未包含多轮对话的示例
3. **改进建议**

```
# 更好的示例设计
exp_task = "分析苹果公司最近五年的财务状况并给出投资建议"
exp_task_pack = TaskPackage(instruction=exp_task)

# 展示参数验证的动作示例
act_1 = AgentAct(
    name=ThinkAct.action_name,
    params={
        INNER_ACT_KEY: "首先需要收集财务数据...",
        'validation': True,
        'required_params': ['response', 'data_source']
    }
)

# 展示错误处理的动作示例
act_2 = AgentAct(
    name=FinishAct.action_name,
    params={
        'response': "根据分析结果...",
```

```
        'error_handling': {
            'missing_data': "数据不完整，建议补充...",
            'invalid_params': "参数验证失败，请检查..."
        }
    }
)
```

4. **完整的Prompt示例**

```
role_prompt = f"""你是一个专业的{philosopher}智能体，需要：
1. 每个动作必须包含 response 参数
2. 思考(Think)动作后必须有明确的结论
3. 完成(Finish)动作必须总结前面的思考
4. 遇到错误时，需要提供详细的错误信息

示例格式：
- Think动作：{{"name": "Think", "params": {{"response": "具体思考内容..."}}}}
- Finish动作：{{"name": "Finish", "params": {{"response": "最终结论..."}}}}
"""
```

2. 模型能力限制

1. **上下文理解**
   - 模型可能无法完全理解复杂的上下文关系
   - 在多轮对话中容易丢失前文信息
2. **参数完整性**
   - 模型生成的动作可能忽略必需参数
   - 参数格式可能不符合预期
3. **解决方案**
   - 增加更多的错误示例和处理方法
   - 在prompt中明确强调参数要求
   - 实现参数自动补全机制
   - 添加动作验证层

3. 改进后的工作流程

1. **智能体初始化**

```
class Philosopher(BaseAgent):
    def __init__(self, philosopher, llm, actions=[], manager=None):
        self.validation_layer = self._init_validation()
        self.error_handler = self._init_error_handler()
        super().__init__(...)

    def _init_validation(self):
        return {
            'Think': ['response'],
            'Finish': ['response', 'summary']
        }

    def forward(self, task, agent_act):
        # 添加参数验证
```

```
        if not self._validate_params(agent_act):
            return self._handle_missing_params(agent_act)
        return super().forward(task, agent_act)
```

2. **动作验证机制**

```
def _validate_params(self, agent_act):
    required_params = self.validation_layer.get(agent_act.name, [])
    return all(param in agent_act.params for param in required_params)

def _handle_missing_params(self, agent_act):
    # 自动补充缺失参数
    if agent_act.name == FinishAct.action_name:
        agent_act.params['response'] = self._generate_default_response()
    return f"已自动补充缺失参数: {agent_act.params}"
```

这些改进建议主要针对：

1. 提供更完善的示例
2. 增加参数验证机制
3. 添加错误处理层
4. 实现自动补全功能

通过这些改进，可以大大减少因模型能力限制导致的错误。

Prompt 完整示例分析

1. Prompt 结构

根据 `prompt_utils.py` 中的定义，完整的 prompt 包含以下部分：

```
PROMPT_TOKENS = {
    "instruction": {"begin": "[Instruction]", "end": "[End of Instruction]"},
    "role": {"begin": "[Role]", "end": "[End of Role]"},
    "constraint": {"begin": "[Constraint]", "end": "[End of Constraint]"},
    "action": {"begin": "[Action_Doc]", "end": "[End of Action_Doc]"},
    "example": {"begin": "[Example]", "end": "[End of Example]"},
    "action_format": {"begin": "[ActionFormatExample]", "end": "[End of
ActionFormatExample]"},
    "execution": {"begin": "[Execution]", "end": "[End of Execution]"},
    "team": {"begin": "[Team_Doc]", "end": "[End of Team_Doc]"},
}
```

2. 生成 Finish Action 的完整 Prompt 示例

2.1 基础指令部分

```
[Instruction]
You are an intelligent agent. You should follow your role, action documentation
to take actions. Your generation should follow the example format. Finish the
task as best as you can.
[End of Instruction]

[Role]
You are Socrates, the famous educator in history. You are very familiar with
Socrates's Book and Thought. Tell your opinion on behalf of Socrates.
[End of Role]

[Constraint]
You generation should be simple and clear.
[End of Constraint]
```

2.2 动作文档部分

```
[Action_Doc]
[
    {
        "name": "Think",
        "description": "Think about the problem deeply",
        "parameters": {"response": "Your thinking process and analysis"}
    },
    {
        "name": "Finish",
        "description": "Complete the task with a final answer",
        "parameters": {"response": "Your final conclusion"}
    }
]
[End of Action_Doc]
```

2.3 示例部分

```
[Example]
Task: What do you think the meaning of life?
Action:Think[{"response": "Let me contemplate this profound question. As a
philosopher, I believe we must examine life's purpose through rational
inquiry..."}]
Observation: OK. I have finished my thought.
Action:Finish[{"response": "After careful consideration, I conclude that the
meaning of life lies in the pursuit of wisdom and virtue..."}]
Observation: Task Completed. The meaning of life is to pursue wisdom and virtue.
[End of Example]
```

2.4 当前执行部分

```
[Execution]
Task: 先有鸡还是先有蛋?
Action:Think[{"response": "让我从哲学的角度思考这个问题。这是一个关于因果关系和起源的深刻问
题..."}]
Observation: OK
Action:
```

3. Prompt 生成过程

1. **初始化阶段**

```
# ManagerPromptGen 初始化
manager_agent = ManagerAgent(
    name="manager_agent",
    role="you are managing Confucius, Socrates and Aristotle to discuss on
questions...",
    llm=llm
)

# 添加示例
manager_agent.prompt_gen.add_example(
    task=exp_task_pack,
    action_chain=exp_act_obs
)
```

2. **动作生成阶段**

```
# BasePromptGen.action_prompt 方法构建完整 prompt
prompt = f"{self.instruction}\n{self.__role_prompt__(self.agent_role)}\n"
prompt += f"{self.__constraint_prompt__()}\n"
prompt += f"{self.__act_doc_prompt__(actions)}\n"
prompt += f"{self.__prompt_example__(prompt_example)}\n"
prompt += f"{PROMPT_TOKENS['execution']['begin']}\n{cur_session}\n"
prompt += "Action:"
```

3. **实际调用示例**

```
# 实际生成的完整 prompt 示例
"""
[Instruction]
You are a manager agent. You can assign a task to those agents in your team...
[End of Instruction]

[Role]
you are managing Confucius, Socrates and Aristotle to discuss on questions...
[End of Role]

[Team_Doc]
{
    "Confucius": "The famous Chinese philosopher...",
    "Socrates": "The ancient Greek philosopher...",
    "Aristotle": "The great philosopher who..."
}
[End of Team_Doc]

[Action_Doc]
[
    {
        "name": "Think",
        "description": "Think about the problem",
        "parameters": {"response": "thinking content"}
```

```
    },
    {
        "name": "Finish",
        "description": "Finish the task",
        "parameters": {"response": "final answer"}
    }
]
[End of Action_Doc]

[Example]
Task: What do you think the meaning of life?
Action:Think[{"response": "Let me ask each philosopher..."}]
Observation: OK
Action:Confucius[{"Task": "What is your opinion on the meaning of life?"}]
Observation: Life is about moral cultivation and social harmony...
Action:Finish[{"response": "Based on the philosophers' insights..."}]
[End of Example]

[Execution]
Task: 先有鸡还是先有蛋?
Action:Think[{"response": "这是一个深刻的哲学问题..."}]
Observation: OK
Action:
"""
```