ICS

CCS

XXX

T/SAIAS

上海市人工智能行业协会团体标准

T/SAIAS XXX—2025

智能科学实验协议

Intelligence Scientific Experiment Protocol

（草案稿）

XXXX - XX - XX 发布　　XXXX - XX - XX 实施

上海市人工智能行业协会　　发 布

设置格式[j]: 字体: （默认）黑体, （中文）黑体

设置格式[j]: 字体: （默认）黑体, （中文）黑体

设置格式[j]: 字体: （默认）黑体, （中文）黑体

删除[S]: 2024

设置格式[j]: 字体: （默认）黑体, （中文）黑体

设置格式[j]: 字体: （默认）黑体, （中文）黑体

设置格式[j]: 字体: （默认）黑体, （中文）黑体

设置格式[j]: 字体: （默认）黑体, （中文）黑体

设置格式[j]: 字体: （默认）黑体, （中文）黑体

设置格式[j]: 字体: （默认）黑体, （中文）黑体, 四号

目　　次

设置格式[j]: 字体: （默认）黑体, （中文）黑体
设置格式[j]: 字体: （默认）宋体, （中文）宋体
设置格式[j]: 字体: （默认）宋体, （中文）宋体, 五号
设置格式[j]: 字体: （默认）宋体, （中文）宋体
删除[j]: 前言　II
设置格式[j]: 字体: （中文）宋体, 五号
设置格式[j]: 字体: （默认）宋体, （中文）宋体
设置格式[j]: 字体: （中文）宋体, 五号
设置格式[j]: 字体: （默认）宋体, （中文）宋体, 五号
设置格式[j]: 字体: （中文）宋体, 五号
设置格式[j]: 字体: （默认）宋体, （中文）宋体
设置格式[j]: 字体: （中文）宋体, 五号
设置格式[j]: 字体: （默认）宋体, （中文）宋体, 五号
设置格式[j]: 字体: （中文）宋体, 五号
设置格式[j]: 字体: （默认）宋体, （中文）宋体
设置格式[j]: 字体: （中文）宋体, 五号
设置格式[j]: 字体: （默认）宋体, （中文）宋体, 五号
设置格式[j]: 字体: （中文）宋体, 五号
设置格式[j]: 字体: （默认）宋体, （中文）宋体
设置格式[j]: 字体: （中文）宋体, 五号
设置格式[j]: 字体: （默认）宋体, （中文）宋体, 五号
设置格式[j]: 字体: （中文）宋体, 五号
设置格式[j]: 字体: （默认）宋体, （中文）宋体
设置格式[j]: 字体: （中文）宋体, 五号
设置格式[j]: 字体: （默认）宋体, （中文）宋体, 五号
设置格式[j]: 字体: （中文）宋体, 五号
设置格式[j]: 字体: （默认）宋体, （中文）宋体, 五号
设置格式[j]: 字体: （中文）宋体, 五号
设置格式[j]: 字体: （默认）宋体, （中文）宋体
设置格式[j]: 字体: （中文）宋体, 五号
设置格式[j]: 字体: （默认）宋体, （中文）宋体, 五号
设置格式[j]: 字体: （中文）宋体, 五号

前　　言

本文件按照GB/T 1.1—2020《标准化工作导则　第1部分：标准化文件的结构和起草规则》的规定起草。

请注意本文件的某些内容可能涉及专利。本文件的发布机构不承担识别这些专利的责任。

本文件由上海市人工智能行业协会提出并归口。

本文件起草单位：

本文件主要起草人：

本标准首次制定。

首期执行单位：

设置格式[j]: 字体: （默认）黑体, （中文）黑体

设置格式[j]: 字体: （默认）宋体, （中文）宋体

引　言

设置格式[j]: 字体: （默认）黑体, （中文）黑体

设置格式[j]: 字体: （默认）宋体, （中文）宋体

ISP是为科学实验定制的开源标准，旨在加速科学发现，是科学发现平台的重要基础支撑之一。目标是促进“干湿结合、多体协作”的研究范式，推动研究者、研究工具与研究对象在人工智能驱动的科学探索中合作与协同进化。通过建立对智能科学实验的标准化描述，ISP能对齐不同研究者对同一实验流程的理解和设置。同时，ISP定义了面向科学发现的应用程序与外部研究对象和工具（如实验室仪器、数据库、知识库、大语言模型（LLMs）、专业计算模型、工具及API）高效交互的接口，受模型上下文协议（MCP）模型工具调用的通用性和易用性的启发，本标准额外整合了湿实验设备、数据存储、知识库、LLMs、领域专用模拟模型等研究组件，为实验流程及其中所涉组件提供了一致且灵活的接口和交互协议。

智能科学实验协议

1 范围

本文件定义科学实验中的实验者、实验对象和实验工具协同交互的协议，包含科学实验的统一表达，科学智能体接口定义规范，科学智能体功能和运行要求描述规则，科学智能体注册和发现机制，智能科学实验网关（记忆、审核、管理等），科学智能体通信协议（会话、状态、提示词、同步异步要求、消息体定义、异常处理等）。

本文件适用于科学智能体（包括但不限于人工智能智能体和智能湿实验设备等）功能调用及协同。

2 规范性引用文件

下列文件中的内容通过文中的规范性引用而构成本文件必不可少的条款。其中，注日期的引用文件，仅该日期对应的版本适用于本文件；不注日期的引用文件，其最新版本（包括所有的修改单）适用于本文件。

GB/T 41867-2022 信息技术 人工智能 术语

3 术语和定义

下列术语和定义适用于本文件。

3.1

科学智能体 scientific agent

在科学实验中提供实验步骤所需能力的功能模块，包括但不限于人工智能智能体和智能湿实验设备等。

3.2

实验者 experimenter

实验者是发起、设计、执行、监控、分析并最终解释实验过程与结果的实体。在智能科学实验中，实验者本身可以是一个智能体（AI系统或机器人），也可以是人类研究者，或者人-智能体协作系统。

3.3

实验对象 experimental subject

实验对象在实验中被研究、测量或改造的核心要素，通常为数据、知识库、信息模型或资料集合。其状态、结构或内容是实验分析的焦点。

3.4

实验工具 experimental tool

实验工具是辅助实验者完成实验任务的软硬件设施、平台、环境或方法。它们是实验者用以操控实验条件、观察实验对象、收集和分析数据的手段。

4 缩略语

下列缩略语适用于本文件。

AI 人工智能（Artificial Intelligence）

ISP 智能科学实验协议（Intelligent Scientific experiment Protocol）

JSON JavaScript对象表示法（JavaScript Object Notation）

API 应用程序编程接口（Application Programming Interface）

OPC UA OPC统一架构（OPC Unified Architecture）

设置格式[j]: 字体: 四号
设置格式[j]: 字体: 四号
设置格式[j]: 字体: （默认）黑体, （中文）黑体
设置格式[j]: 字体: （默认）黑体, （中文）黑体
设置格式[j]: 字体: （默认）宋体, （中文）宋体
设置格式[j]: 字体: （默认）黑体, （中文）黑体, 非加粗
设置格式[j]: 字体: （默认）宋体, （中文）宋体
设置格式[j]: 字体: （默认）宋体, （中文）宋体
设置格式[j]: 字体: （默认）黑体, （中文）黑体
设置格式[j]: 字体: （默认）宋体, （中文）宋体
设置格式[j]: 字体: （默认）宋体, （中文）宋体
设置格式[j]: 字体: （默认）黑体, （中文）黑体
设置格式[j]: 字体: （默认）宋体, （中文）宋体
删除[S]:
设置格式[j]: 字体: （默认）黑体, （中文）黑体
设置格式[j]: 字体: （默认）宋体, （中文）宋体
删除[S]:
设置格式[j]: 字体: （默认）黑体, （中文）黑体
设置格式[j]: 缩进: 左侧: 0 毫米, 首行缩进: 7.4 毫米
设置格式[j]: 字体: （默认）宋体, （中文）宋体, 英
删除[S]:
设置格式[S]: 标准文件_术语条一
设置格式[j]: 字体: （默认）黑体, （中文）黑体
设置格式[j]: 字体: （默认）宋体, （中文）宋体
删除[S]:
设置格式[j]: 字体: （默认）黑体, （中文）黑体
设置格式[j]: 字体: （默认）宋体, （中文）宋体
设置格式[j]: 缩进: 左侧: 0 毫米, 首行缩进: 7.4 毫米

PCR 平台配置寄存器（Platform Configuration Register）
UUID 通用唯一识别码（Universally Unique Identifier）
LLM 大语言模型（Large Language Model）
JWT 令牌 （JSON Token）
RBAC 基于角色的访问控制（Role-Based Access Control）
ROS2 机器人操作系统2（Robot Operating System 2）
URI 统一资源标识符（Uniform Resource Identifier）
REST 表述性状态传递（Representational State Transfer）
HTTP 超文本传输协议（Hyper Text Transfer Protocol）

5 智能科学实验协议

5.1 概述

智能科学实验协议（ISP）是一套标准化的实验描述与配套管理框架，其核心由协议本身的规范化 protocol 的结构定义与枢纽系统的协同交互框架构成。该框架通过客户端接入，由枢纽进行实验注册、工具管理、任务编排与合规验证，生成实验 protocol 的统一表达，并指令分布式服务器灵活调用底层的数据、工具、算力及实体设备等资源，从而实现对分布式科学实验全生命周期的自动化执行、状态监控与统一管理，最终构建成一个资源可调度、流程可复现的智能实验框架。实验 protocol 统一表达整体框架如图 1 所示。

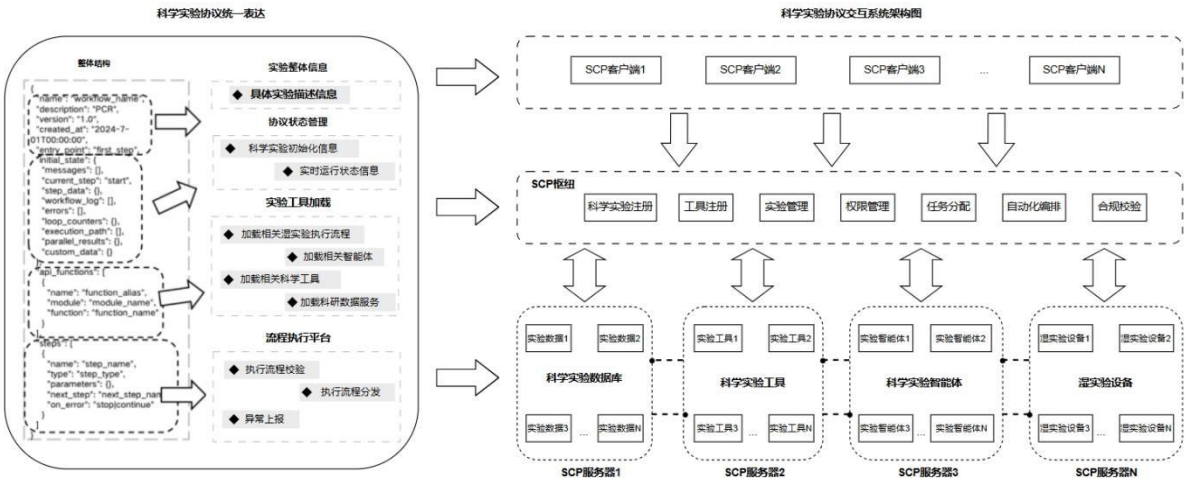

图 1 实验 protocol 统一表达整体框架

5.2 实验 protocol 统一表达

目标：建立科学实验的统一描述语言。
内容：以结构化 protocol 为核心，定义了实验的整体信息、执行步骤和初始状态等内容。通过协议状态管理模块来维护实验的初始化信息与实时运行状态。在执行前，系统会加载所需的湿实验流程、智能体、科学工具与数据服务等资源。最终，由流程执行平台负责对协议进行校验、分发执行，并在此过程中进行异常上报，从而确保 实验流程能够正确、可靠地闭环运行，该框架具备以下性质：

a） 通用性：通过JSON配置创建任意类型的工作流，无需编写代码，适用于各类实验场景。
b） 多样性：支持湿实验、干实验、仿真、数据分析等多种类型。

设置格式[j]: 字体: （默认）宋体, （中文）宋体
删除[S]:
设置格式[j]: 字体: （默认）宋体, （中文）宋体
设置格式[S]: 缩进: 首行缩进: 0 字符
设置格式[j]: 字体: （默认）黑体, （中文）黑体
设置格式[j]: 字体: （默认）黑体, （中文）黑体
设置格式[j]: 字体: （默认）黑体, （中文）黑体
设置格式[j]: 字体: （默认）黑体, （中文）黑体
设置格式[j]: 字体: （默认）宋体, （中文）宋体
设置格式[S]: 缩进: 首行缩进: 7.4 毫米
设置格式[j]: 默认段落字体, 字体: （默认）宋体, （中
删除[S]:
设置格式[S]: 字体: （默认）宋体, （中文）宋体, 五号
设置格式[S]: 标准文件_一级条标题, 缩进: 左侧: 7.4
设置格式[S]: 标准文件_正文表标题, 居中, 缩进: 左侧
设置格式[S]: 字体: （默认）黑体, （中文）黑体
删除[S]:
设置格式[S]: 字体: （默认）微软雅黑, （中文）微软
删除[S]: 图 1
设置格式[S]: 正文, 两端对齐, 制表位: 16.54 字符, 左
设置格式[j]: 字体: （默认）黑体, （中文）黑体
设置格式[j]: 字体: （默认）宋体, （中文）宋体, 英
设置格式[j]: 字体: （默认）宋体, （中文）宋体
设置格式[j]: 字体: （默认）宋体, （中文）宋体, 英
设置格式[j]: 字体: （默认）宋体, （中文）宋体
设置格式[j]: 字体: （默认）宋体, （中文）宋体, 五号
设置格式[j]: 字体: （默认）宋体, （中文）宋体, 英

c) 灵活性：支持循环、条件分支、并行执行等高级控制流，满足复杂实验流程的动态需求。
d) 可扩展性：动态加载外部工具函数，支持API调用，便于集成新工具和功能模块。
e) 安全性：多层次的数据验证和安全检查，保障实验执行安全。

具体实现：通过基于JSON的标准化实验协议描述，完整涵盖实验元数据、资源定义、工具调用和执行流程，确保流程闭环。支持协议模板导入、低代码平台、AI生成和手工编写等多种来源，从而实现智慧科学实验方案的复现和适配，提升实验效率和一致性。Protocol实例详见附录A。

5.3 设备抽象层接口

a) 干/湿实验设备接口定义

目标：实现"设备即插即用"，规范干湿设备调用方式。

内容：制定统一的抽象层规范，通过标准化驱动接口（如OPC UA、ROS2）兼容多品牌设备（如PCR仪/机械臂），屏蔽底层通信差异。

b) 干/湿实验设备统一寻址

目标：实现设备动态注册与状态监控，统一管理多设备调用逻辑，提升跨平台兼容性。

内容：制定分层URI寻址架构（例：scplab://zone3/pcr_machine01/acticon1），结合注册发现协议实现设备动态发现。

5.4 信息交互

a) 跨实验类型通用数据封装

目标：统一数据格式消除设备通信壁垒，压缩传输提升效率，双层兼容机制兼顾性能与开发灵活性。

内容：定跨实验类型的通用数据格式，具备可扩展的Schema（含实验类型标记字段），支持二进制压缩传输，兼容JSON序列化。

b) 数据传输

目标：构建端到端可验证的无损传输管道，保障数据完整性及防篡改能力。

内容：制定统一的数据交换协议标准，确保数据传输工程中无损无篡改流转。

5.5 实验操作

a) 干湿实验操作

目标：构建跨设备的抽象执行层，实现指令标准化与调度统一化。

内容：定义原子操作API，实现实验流程的标准化动作执行体系，确保参数化指令与设备无关性调用。

b) 全流程数据操作

目标：构建跨数据的抽象执行层，实现数据指令标准化与调度统一化。

内容：定义数据操作API，实现全流程数据处理规范执行体系，确保参数化指令与设备无关性调用。

5.6 智能体操作

目标：构建智能体的抽象执行层，实现智能体指令标准化与调度统一化。

内容：定义智能体操作API，实现全流程智能体处理规范执行体系，确保参数化指令与模型无关性调用。

5.7 实验流程

a) 执行流程

目标：实验流程标准化执行，监控与任意断点回溯。构建跨设备的抽象执行层，实现指令标准化与调度统一化。

内容：定义状态机（Created→Running→Paused→Completed/Failed）以及对应的（if/else/while等常见分支流程），支持中间状态推送。

b) 异常语义

目标：系统故障码标准化。

设置格式[j]: 字体: 四号
设置格式[j]: 字体: 四号
删除[S]:
设置格式[S]: 左, 缩进: 左 2 字符
设置格式[j]: 字体: （默认）黑体, （中文）黑体
设置格式[j]: 字体: （默认）宋体, （中文）宋体
设置格式[j]: 编号 + 级别: 1 + 编号样式: a, b, c, ... +
删除[S]:
设置格式[j]: 字体: （默认）宋体, （中文）宋体
设置格式[j]: 编号 + 级别: 1 + 编号样式: a, b, c, ... +
删除[S]:
设置格式[j]: 字体: （默认）黑体, （中文）黑体
设置格式[j]: 字体: （默认）宋体, （中文）宋体
删除[S]:
设置格式[j]: 字体: （默认）宋体, （中文）宋体
删除[S]:
设置格式[S]: 缩进: 左 3 字符
设置格式[j]: 字体: （默认）黑体, （中文）黑体
设置格式[j]: 字体: （默认）宋体, （中文）宋体
设置格式[j]: 字体: （默认）宋体, （中文）宋体
设置格式[j]: 字体: （默认）宋体, （中文）宋体
删除[S]:
设置格式[j]: 字体: （默认）黑体, （中文）黑体
设置格式[j]: 字体: （默认）宋体, （中文）宋体
删除[S]:
设置格式[j]: 字体: （默认）黑体, （中文）黑体
设置格式[j]: 字体: （默认）宋体, （中文）宋体

内容：构建三级错误编码（设备级0x1FFF/数据级0x2FFF/逻辑级0x3FFF）标准，附加可扩展描述字段。

c） 数据统一管理：

目标： 建立面向长周期实验的数据全生命周期管理标准，确保实验数据的完整性、可追溯性与一致性。

内容：构建标准化管理框架，确保长周期实验数据在整个生命周期内的完整性、可用性和安全性，为科学研究的可重复性和数据驱动发现提供坚实基础。

5.8 实验认证

a） 执行流程

目标：构建安全执行流程标准，防止越权操作。

内容：基于OAuth 2.0设备流，采用JWT携带RBAC策略（如"role": "centrifuge_operator"）等方案进行认证。

5.9 智能科学实验服务系统的基本组件

5.9.1 智能科学实验服务器

a） 访问控制：ISP 服务器管理自身及其内部工具的权限，支持动态更新访问权限。

b） 自动注册：ISP 边缘服务器自动向中央服务器注册，简化外部用户访问。

c） 设备管理：ISP 边缘服务器对现有设备和工具进行管理，使其可被外部调用。

d） 任务调用：ISP 边缘服务器接收到 ISP 中心服务器的指令后，对连接的设备、工具和模型执行操作，中间结果和最终结果实时返回。

e） 健康监测：定期评估边缘设备、工具和模型的健康状态，并反馈给 ISP 边缘服务器，以指导内部调度和决策。

5.9.2 智能科学实验枢纽

a） 任务分配：处理外部指令；ISP 中央服务器确定执行逻辑，调用适当的边缘服务器，并实时回传中间和最终结果。

b） 边缘服务器健康管理：定期监控已部署边缘服务的状态，并向 ISP 边缘服务器反馈，以辅助内部调度决策。

c） 意图识别：解析用户输入，识别并分解意图，执行实验任务的后续步骤，或为不支持的请求返回用户友好的指导。

d） 任务编排：根据用户权限和可用资源评估已识别的任务，用户可以自定义编排实验流程也可以通过自动化编排等手段返回前三个执行计划，并在用户选择后执行所选的工作流。

e） 科学实验注册：为新实验方案创建标准化的数字身份，通过协议解析、资源预检和元数据索引，将其唯一注册并纳入平台资源库。

f） 实验管理：对实验全生命周期进行集中管控，包括实例化、状态跟踪、执行控制与数据归档，确保实验过程可监控、结果可复现。

g） 权限管理： 通过角色定义与资源授权，实施细粒度访问控制，确保用户与设备仅能操作其权限范围内的实验与资源。

h） 自动化编排：智能解析步骤依赖与资源需求，进行最优调度与并行执行，驱动复杂实验流程高效、可靠地自动运行。

i） 合规校验：在实验执行前后，自动核查协议规范、环境安全与数据完整性，确保实验过程符合既定标准与法规要求。

5.9.3 智能科学实验客户端

a） 工具清单：根据用户权限，编译并向用户展示所有可用的ISP边缘侧工具列表。

b） 用户调用：允许用户执行端到端实验任务编排，以及调用单个工具。

5.9.4 消息队列服务器

设置格式[j]: 字体: （默认）宋体, （中文）宋体
设置格式[j]: 字体: （默认）宋体, （中文）宋体
删除[S]:
设置格式[S]: 正文, 缩进: 左 2.61 字符
设置格式[j]: 字体: （默认）黑体, （中文）黑体
设置格式[j]: 字体: （默认）宋体, （中文）宋体
删除[S]:
删除[j]:
设置格式[S]: 缩进: 左 3 字符, 首行缩进: 0 毫米
设置格式[j]: 字体: （默认）黑体, （中文）黑体
删除[j]: 协议
设置格式[j]: 字体: （默认）微软雅黑, （中文）微软雅黑
设置格式[j]: 字体: （默认）黑体, （中文）黑体
设置格式[S]: 字体: （默认）宋体, （中文）宋体, 中
设置格式[S]: 标准文件_字母编号列项（一级）, 制表
设置格式[S]: 项目符号和编号
删除[S]:
删除[S]:
删除[S]:
设置格式[j]: 字体: （默认）黑体, （中文）黑体
设置格式[S]: 字体: （默认）宋体, （中文）宋体, 中
设置格式[S]: 标准文件_字母编号列项（一级）, 制表
设置格式[S]: 项目符号和编号
设置格式[S]: 字体: （默认）宋体, （中文）宋体
设置格式[S]: 字体: （默认）宋体, （中文）宋体, 中
设置格式[S]: 字体: （默认）宋体, （中文）宋体
删除[S]:
删除[S]:
删除[S]:
删除[S]:
删除[S]:
设置格式[S]: 标准文件_字母编号列项（一级）, 制表
设置格式[S]: 标准文件_字母编号列项（一级）, 制表
删除[S]:
设置格式[S]: 字体: （默认）宋体, （中文）宋体, 中

a) 设备-集线器解耦：实现边缘设备与中央集线器的完全解耦，允许根据需要动态扩展设备端硬件资源。
b) 消息传递：在边缘设备和集线器之间促进可靠的消息传递和状态监控。

5.9.5 RADIS 记忆服务器

a) 结果缓存：按 UUID 缓存每个中间结果和最终结果，以减少系统整体调用开销。
b) 临时结果缓存：存储临时结果，以最大限度地减少内部通信延迟。
c) 状态日志：对实现过程，中间状态等实验过程数据进行记录。

5.9.6 对象存储服务器

a) 参数共享：对于无法通过函数参数方便传递的大型数据对象，支持中央服务器和边缘服务器之间的数据交换。
b) 其他数据共享：根据需要支持其他数据共享机制。

设置格式[j]: 字体: 四号
设置格式[j]: 字体: 四号
设置格式[j]: 字体: （默认）宋体, （中文）宋体
设置格式[j]: 字体: （默认）黑体, （中文）黑体
设置格式[j]: 字体: （默认）宋体, （中文）宋体
设置格式[j]: 字体: （默认）黑体, （中文）黑体
设置格式[j]: 字体: （默认）宋体, （中文）宋体

附 录 A

（资料性）

智能科学实验交互标准示例

A.1 实验的统一表达【protocol.json】

A.1.1 整体结构

```
{
  "name": "workflow_name",
  "description": "工作流描述",
  "version": "1.0",
  "created_at": "2024-01-01T00:00:00",
  "entry_point": "first_step",
  "initial_state": {
    "messages": [],
    "current_step": "start",
    "step_data": {},
    "workflow_log": [],
    "errors": [],
    "loop_counters": {},
    "execution_path": [],
    "parallel_results": {},
    "custom_data": {}
  },
  "api_functions": [
    {
      "name": "function_alias",
      "module": "module_name",
      "function": "function_name"
    }
  ],
  "steps": [
    {
      "name": "step_name",
      "type": "step_type",
      "parameters": {},
      "next_step": "next_step_name",
      "on_error": "stop|continue"
    }
  ]
}
```

A.1.2 基本编排单元【统一格式】

设置格式[j]: 项目符号和编号
设置格式[j]: 字体: （默认）黑体, （中文）黑体
设置格式[j]: 字体: （默认）黑体, （中文）黑体
设置格式[j]: 字体: （默认）黑体, （中文）黑体
设置格式[j]: 项目符号和编号
设置格式[j]: 标准文件_附录二级条标题, 无项目符号或编号
设置格式[j]: 中文(简体)
设置格式[j]: 字体: （默认）宋体, （中文）宋体
设置格式[j]: 字体: （默认）宋体, （中文）宋体
设置格式[j]: 标准文件_附录二级条标题, 无项目符号或编号
设置格式[j]: 项目符号和编号

A.1.2.1 协议状态管理模块

```
{
    "messages": [],
    "current_step": "start",
    "step_data": {},
    "workflow_log": [],
    "errors": [],
    "loop_counters": {},
    "execution_path": [],
    "parallel_results": {},
    "custom_data": {
        "source_vessel": {"id": "reaction_flask_1", "volume": 200.0},
        "collection_vessels": [{"id": "collection_flask_1", "volume": 0.0}],
        "pressure": 1.0,
        "heating_rate": 5.0,
        "collected_fractions": [],
        "target_temperature": 100.0,
        "current_temperature": 25.0,
        "monitoring_cycles": 0,
        "max_monitoring_cycles": 5
    }
```

设置格式[j]: 字体: 四号

设置格式[j]: 字体: 四号

设置格式[j]: 标准文件_附录三级条标题, 缩进: 首行缩进: 0 毫米, 制表位: 不在 4 字符, 无项目符号或编号

设置格式[j]: 项目符号和编号

设置格式[j]: 字体: （默认）宋体, （中文）宋体

A.1.2.2 工具导入模块

```
 [
{
     "name": "setup_distillation_apparatus",
     "module": "distillation_graph_with_tools",
     "function": "setup_distillation_apparatus"
 },
 {
     "name": "start_heating",
     "module": "distillation_graph_with_tools",
     "function": "start_heating"
  }
]
```

设置格式[j]: 标准文件_附录三级条标题, 缩进: 首行缩进: 0 毫米, 制表位: 不在 4 字符, 无项目符号或编号

设置格式[j]: 项目符号和编号

设置格式[j]: 字体: （默认）宋体, （中文）宋体

A.1.2.3 API 调用步骤

```
{
  "name": "setup_apparatus",
  "type": "api_call",
  "api_function": "setup_distillation_apparatus",
  "parameters": {
    "source_vessel_id": "${custom_data.source_vessel.id}",
    "apparatus_type": "simple_distillation",
    "pressure": 1.0
  },
  "on_error": "stop",
  "next_step": "start_heating"
```

设置格式[j]: 标准文件_附录三级条标题, 缩进: 首行缩进: 0 毫米, 制表位: 不在 4 字符, 无项目符号或编号

设置格式[j]: 项目符号和编号

设置格式[j]: 字体: （默认）宋体, （中文）宋体

```
    }
```

A.1.2.4 循环控制模块【计数循环】

```
{
  "name": "monitoring_loop",
  "type": "loop",
  "max_iterations": 10,
  "condition": {
    "type": "less_than",
    "left": "custom_data.monitoring_cycles",
    "right": "${custom_data.max_monitoring_cycles}"
  },
  "loop_body": [
    {
      "name": "monitor_step",
      "type": "api_call",
      "api_function": "monitor_process"
    }
  ],
  "next_step": "analysis"
}
```

设置格式[j]: 标准文件_附录三级条标题, 缩进: 首行缩进: 0 毫米, 制表位: 不在 4 字符, 无项目符号或编号

设置格式[j]: 项目符号和编号

设置格式[j]: 字体: （默认）宋体, （中文）宋体

A.1.2.5 循环控制模块【条件循环】

```
{
  "name": "temperature_check",
  "type": "conditional",
  "condition": {
    "type": "greater_equal",
    "left": "custom_data.current_temperature",
    "right": 78.0
  },
  "true_step": "collect_fraction",
  "false_step": "continue_heating"
}
```

设置格式[j]: 标准文件_附录三级条标题, 缩进: 首行缩进: 0 毫米, 制表位: 不在 4 字符, 无项目符号或编号

设置格式[j]: 项目符号和编号

设置格式[j]: 字体: （默认）宋体, （中文）宋体

A.1.2.6 并行执行模块

```
{
  "name": "parallel_analysis",
  "type": "parallel",
  "parallel_steps": [
    {
      "name": "stop_heating",
      "type": "api_call",
      "api_function": "stop_process"
    },
    {
```

设置格式[j]: 标准文件_附录三级条标题, 缩进: 首行缩进: 0 毫米, 制表位: 不在 4 字符, 无项目符号或编号

设置格式[j]: 项目符号和编号

设置格式[j]: 字体: （默认）宋体, （中文）宋体

```
            "name": "analyze_results",
            "type": "api_call",
            "api_function": "analyze_data"
        }
    ],
    "next_step": "completion"
}
```

蒸馏工作流实例如下【基础模块】：

```
{
        "name": "enhanced_distillation_workflow",
        "description": "蒸馏工作流程，支持循环监控和条件分支",
        "version": "2.0",
        "created_at": datetime.now().isoformat(),
        "entry_point": "setup_apparatus",
        "initial_state": {
            "messages": [],
            "current_step": "start",
            "step_data": {},
            "workflow_log": [],
            "errors": [],
            "loop_counters": {},
            "execution_path": [],
            "parallel_results": {},
            "custom_data": {
                "source_vessel": {"id": "reaction_flask_1", "volume": 200.0},
                "collection_vessels": [{"id": "collection_flask_1", "volume": 0.0}],
                "pressure": 1.0,
                "heating_rate": 5.0,
                "collected_fractions": [],
                "target_temperature": 100.0,
                "current_temperature": 25.0,
                "monitoring_cycles": 0,
                "max_monitoring_cycles": 5
            }
        },
        "api_functions": [
            {
                "name": "setup_distillation_apparatus",
                "module": "distillation_graph_with_tools",
                "function": "setup_distillation_apparatus"
            },
            {
                "name": "start_heating",
                "module": "distillation_graph_with_tools",
                "function": "start_heating"
            },
            {
```

设置格式[j]: 字体: 四号

设置格式[j]: 字体: 四号

```
            "name": "monitor_distillation",
            "module": "distillation_graph_with_tools",
            "function": "monitor_distillation"
        },
        {
            "name": "collect_fraction",
            "module": "distillation_graph_with_tools",
            "function": "collect_fraction"
        },
        {
            "name": "stop_distillation",
            "module": "distillation_graph_with_tools",
            "function": "stop_distillation"
        },
        {
            "name": "analyze_results",
            "module": "distillation_graph_with_tools",
            "function": "analyze_results"
        }
    ],
    "steps": [
        {
            "name": "setup_apparatus",
            "type": "api_call",
            "api_function": "setup_distillation_apparatus",
            "parameters": {
                "source_vessel_id": "${custom_data.source_vessel.id}",
                "apparatus_type": "simple_distillation",
                "pressure": "${custom_data.pressure}",
                "heating_rate": "${custom_data.heating_rate}"
            },
            "on_error": "stop",
            "next_step": "start_heating_process"
        },
        {
            "name": "start_heating_process",
            "type": "api_call",
            "api_function": "start_heating",
            "parameters": {
                "initial_temp": 25.0,
                "target_temp": "${custom_data.target_temperature}",
                "heating_rate": "${custom_data.heating_rate}"
            },
            "on_error": "stop",
            "next_step": "monitoring_loop"
        },
        {
            "name": "monitoring_loop",
            "type": "loop",
            "max_iterations": 10,
```

```
            "condition": {
                "type": "less_than",
                "left": "custom_data.monitoring_cycles",
                "right": "${custom_data.max_monitoring_cycles}"
            },
            "loop_body": [
                {
                    "name": "monitor_temperature",
                    "type": "api_call",
                    "api_function": "monitor_distillation",
                    "parameters": {
                        "current_temp": 78.0,
                        "vapor_temp": 78.5,
                        "collection_rate": 2.5
                    }
                },
                {
                    "name": "increment_counter",
                    "type": "data_transform",
                    "transform_type": "increment",
                    "source": "custom_data.monitoring_cycles",
                    "target": "custom_data.monitoring_cycles",
                    "increment": 1
                }
            ],
            "next_step": "temperature_check"
        },
        {
            "name": "temperature_check",
            "type": "conditional",
            "condition": {
                "type": "greater_equal",
                "left": "custom_data.current_temperature",
                "right": 78.0
            },
            "true_step": "collect_main_fraction",
            "false_step": "monitoring_loop",
            "next_step": "collect_main_fraction"
        },
        {
            "name": "collect_main_fraction",
            "type": "api_call",
            "api_function": "collect_fraction",
            "parameters": {
                "fraction_name": "主馏分",
                "collection_vessel": "${custom_data.collection_vessels.0.id}",
                "start_temp": 78.0,
                "end_temp": 82.0,
                "volume": 150.0
```

设置格式[j]: 字体: 四号

设置格式[j]: 字体: 四号

```
            },
            "on_error": "continue",
            "next_step": "parallel_analysis"
        },
        {
            "name": "parallel_analysis",
            "type": "parallel",
            "parallel_steps": [
                {
                    "name": "stop_heating",
                    "type": "api_call",
                    "api_function": "stop_distillation",
                    "parameters": {
                        "reason": "蒸馏完成"
                    }
                },
                {
                    "name": "analyze_results",
                    "type": "api_call",
                    "api_function": "analyze_results",
                    "parameters": {
                        "total_fractions": 1,
                        "total_volume": 150.0,
                        "temperature_ranges": "78-82° C"
                    }
                }
            ],
            "next_step": "completed"
        },
        {
            "name": "completed",
            "type": "data_transform",
            "transform_type": "format
            "template": "蒸馏工作流完成，共执行 {cycles} 次监控循环",
            "values": {
                "cycles": "custom_data.monitoring_cycles"
            },
            "target": "step_data.completion_message"
        }
    ]
}
```

A.2 科学实验信息【LabBaseParams】

```
class BaseParams(TypedDict, total=False):
    """Base class for all parameter types."""
    UUID: str  # 调用实例
    user_id: str  # 用户 ID【鉴权】
    organization_id: str  # 机构 ID【鉴权】
```

设置格式[j]: 字体: （默认）黑体, （中文）黑体

设置格式[j]: 项目符号和编号

设置格式[j]: 字体: （默认）宋体, （中文）宋体

```
    creat_time: date  # 创建时间
    experiment_type: Literal["dry-experiment", "wet-experiment", "dry-wet-experiment"]
# 实验类型【枚举】
    experiment_name: str  # 实验名称
    experiment_des: str  # 实验描述
     priority: Literal["high", "med", "low"]  # 优先级【枚举】
```

A.3 湿实验室设备信息【DeviceParams】

```
class DeviceParams(BaseParams, total=False):
    """Base class for all device parameter types."""
    device_name: str  # 实验室名称+内部分类+具体功能
    device_id: str  # 设备编码
    device_des: str  # 设备描述
     customized_params: List[Any]  # 设备定制参数
```

A.4 科学实验智能体【AgentParams】

```
class AgentParams(BaseParams, total=False):
    """Base class for all lab agent parameter types."""
    api_key: str  # 设备调用秘钥【针对外部模型，可选】
    agent_name: str  # 实验室名称+内部分类+具体功能
    agent_id: str  # 智能体编码
    agent_des: str  # 智能体描述
    customized_params: List[Any]  # 设备定制参数
```

A.5 干/湿实验设备接口标准定义

设备驱动抽象层

```
    from abc import ABC, abstractmethod
    class IDeviceDriver(ABC):
        """设备驱动统一接口（适配器模式）"""
        @abstractmethod
        def connect(self, config: dict) -> bool:
            """连接设备（如 IP/串口配置）"""

        @abstractmethod
        def execute(self, command: 'DeviceCommand') -> 'Response':
            """执行原子操作命令"""

        @abstractmethod
        def read_status(self) -> 'DeviceStatus':
            """读取实时状态"""

        @abstractmethod
```

设置格式[j]: 字体: 四号

设置格式[j]: 字体: 四号

设置格式[j]: 字体: （默认）黑体, （中文）黑体

设置格式[j]: 项目符号和编号

设置格式[j]: 字体: （默认）宋体, （中文）宋体

设置格式[j]: 字体: （默认）黑体, （中文）黑体

设置格式[j]: 项目符号和编号

设置格式[j]: 字体: （默认）宋体, （中文）宋体

删除[j]: A.5

设置格式[j]: 标准文件_附录一级条标题

设置格式[j]: 项目符号和编号

设置格式[j]: 字体: （默认）宋体, （中文）宋体

```
        def subscribe(self, event: str, callback: callable):
            """订阅设备事件（如温度超限）"""

    class DeviceCommand:
        """设备指令封装（命令模式）"""
        def __init__(self, operation: str, params: dict):
            self.operation = operation  # e.g. "centrifuge_start"
            self.params = params        # e.g. {"rpm": 3000, "duration": 600}

    class DeviceStatus:
        """设备状态快照"""
        def __init__(self, state: str, metrics: dict):
            self.state = state    # "RUNNING", "PAUSED"
            self.metrics = metrics # {"current_rpm": 2987, "vibration": 0.12}
```

接口调用示例

```
# OPC UA 协议设备（如赛默飞 PCR 仪）
class ThermoFisherDriver(IDeviceDriver):
    def __init__(self):
        import opcua  # 厂商 SDK 封装
        self.client = opcua.Client()

    def connect(self, config):
        self.client.connect(f"opc.tcp://{config['ip']}:4840")

    def execute(self, command):
        if command.operation == "set_temperature":
            node = self.client.get_node("ns=2;s=Heater/Temp")
            node.set_value(command.params["target_temp"])
            return Response(success=True)

# ROS2 协议设备（如 Hamilton 机械臂）
class HamiltonDriver(IDeviceDriver):
    def __init__(self):
        import rclpy  # ROS2 Python 接口
        self.node = rclpy.create_node('hamilton_adapter')

    def execute(self, command):
        from hamilton_msgs.srv import PipetteCommand
        req = PipetteCommand.Request(
            volume=command.params["volume"],
            source=command.params["source_well"]
        )
        future = self.pipette_client.call_async(req)
        return future  # 返回异步句柄
```

A.6 干/湿实验设备统一寻址

删除[j]: A.5

设置格式[j]: 标准文件_附录一级条标题

设置格式[j]: 项目符号和编号

实验设备统一寻址具体实现

```
class DeviceURI:
    """分层 URI 解析与构建"""
    def __init__(self, zone="", device_id="", action=""):
        self.scheme = "scplab"
        self.zone = zone.lower().replace(' ', '-')  # 标准化区域标识
        self.device_id = device_id.lower()          # 设备 ID 强制小写
        self.action = action

    @classmethod
    def parse(cls, uri: str) -> "DeviceURI":
        """从字符串解析 URI（例：scplab://zone3/pcr-01/temperature）"""
        if not uri.startswith("scplab://"):
            raise ValueError("Invalid scheme")

        parts = uri[9:].split('/')  # 去除协议头
        if len(parts) < 2:
            raise ValueError("Missing required components")

        return cls(
            zone=parts[0],
            device_id=parts[1],
            action='/'.join(parts[2:]) if len(parts) > 2 else ""
        )

    def to_string(self) -> str:
        """构建完整 URI 字符串"""
        path = f"{self.zone}/{self.device_id}"
        if self.action:
            path += f"/{self.action}"
        return f"scplab://{path}"

    def validate(self) -> bool:
        """校验 URI 有效性"""
        return all([
            re.match(r"^[a-z0-9\-]+$", self.zone),
            re.match(r"^[a-z0-9_\-]+$", self.device_id),
            not self.action or re.match(r"^[\w/\-]+$", self.action)
        ])

# 使用示例
uri = DeviceURI.parse("scplab://zone3/pcr-01/temperature/set")
```

注册中心客户端

```
import consul
from datetime import datetime, timedelta

class DeviceRegistry:
    """基于 Consul 的设备注册发现服务"""
```

设置格式[j]: 字体: 四号

设置格式[j]: 字体: 四号

设置格式[j]: 字体: （默认）宋体, （中文）宋体

```
    def __init__(self, consul_hosts: list):
        self.consul = consul.Consul(host=consul_hosts[0])  # 多节点负载均衡

    def register(self, descriptor: "DeviceDescriptor") -> bool:
        """设备注册"""
        key = f"devices/{descriptor.uri.zone}/{descriptor.uri.device_id}"
        payload = {
            'type': descriptor.type,
            'status': descriptor.status.value,
            'endpoint': descriptor.endpoint,
            'last_seen': datetime.utcnow().isoformat()
        }
        return self.consul.kv.put(key, json.dumps(payload))

    def discover(self, filter: "DiscoveryFilter") -> list:
        """设备发现"""
        _, data = self.consul.kv.get("devices/", recurse=True)
        results = []

        for item in (data or []):
            # 提取设备信息：devices/zone1/pcr-01 → pcr-01
            device_id = item['Key'].split('/')[-1]
            payload = json.loads(item['Value'])

            # 构建URI对象
            uri = DeviceURI(
                zone=item['Key'].split('/')[1],
                device_id=device_id
            )

            # 构建描述符
            desc = DeviceDescriptor(
                uri=uri,
                type=payload['type'],
                status=DeviceStatus(payload['status']),
                endpoint=payload['endpoint']
            )

            if filter.match(desc):
                results.append(desc)

        return results

    def heartbeat(self, device_id: str):
        """心跳维持"""
        key = f"devices/_heartbeat/{device_id}"
        self.consul.kv.put(key, datetime.utcnow().isoformat(), ttl=30)
```

A.7 跨实验类型通用数据封装

数据传输基础类

```
from google.protobuf import descriptor_pool, message_factory

class SchemaRegistry:
    _instance = None
    SCHEMAS = {
        ExperimentType.PCR: {
            "name": "PCRExperiment",
            "fields": {
                "sample_id": "string",
                "cycles": "int32",
                "denature_temp": "float",
                "annealing_temp": "float",
                "results": "bytes"
            }
        },
        ExperimentType.NGS: {
            "name": "NGSExperiment",
            "fields": {
                "flowcell_id": "string",
                "read_length": "int32",
                "quality_scores": "bytes"
            }
        }
    }

    def __new__(cls):
        if not cls._instance:
            cls._instance = super().__new__(cls)
            cls.pool = descriptor_pool.DescriptorPool()
            # 预加载所有模式
            for exp_type, schema in cls.SCHEMAS.items():
                cls._build_descriptor(exp_type, schema)
        return cls._instance

    @classmethod
    def _build_descriptor(cls, exp_type, schema):
        """动态创建 Protobuf 描述符"""
        file_desc = FileDescriptorProto()
        file_desc.name = f"{schema['name']}.proto"

        # 创建消息描述符
        msg_desc = DescriptorProto(name=schema['name'])
        for idx, (field, ftype) in enumerate(schema['fields'].items(), 1):
            field_desc = msg_desc.field.add()
            field_desc.name = field
            field_desc.number = idx
```

设置格式[j]: 字体: 四号

设置格式[j]: 字体: 四号

删除[j]: A.

设置格式[j]: 标准文件_附录一级条标题

设置格式[j]: 项目符号和编号

删除[j]: 6

设置格式[j]: 字体: （默认）宋体, （中文）宋体

```
            field_desc.label = FieldDescriptorProto.LABEL_OPTIONAL

            # 映射类型
            type_map = {
                "string": FieldDescriptorProto.TYPE_STRING,
                "int32": FieldDescriptorProto.TYPE_INT32,
                "float": FieldDescriptorProto.TYPE_FLOAT,
                "bytes": FieldDescriptorProto.TYPE_BYTES
            }
            field_desc.type = type_map.get(ftype, FieldDescriptorProto.TYPE_BYTES)

        file_desc.message_type.add().CopyFrom(msg_desc)
        cls.pool.Add(file_desc)

    @classmethod
    def get_schema(cls, exp_type):
        """获取对应实验类型的消息类"""
        schema_name = cls.SCHEMAS[exp_type]["name"]
        descriptor = cls.pool.FindMessageTypeByName(schema_name)
        return message_factory.GetMessageClass(descriptor)

class ProtoBufHelper:
    @staticmethod
    def encode(data: dict, schema_class) -> bytes:
        """将字典编码为 Protobuf 二进制"""
        message = schema_class()
        json_format.ParseDict(data, message)
        return message.SerializeToString()

    @staticmethod
    def decode(data: bytes, schema_class) -> dict:
        """将 Protobuf 二进制解码为字典"""
        message = schema_class()
        message.ParseFromString(data)
        return json_format.MessageToDict(message)
```

A.8 干湿实验操作

实验操作基类定义

```
from abc import ABC, abstractmethod

class IAtomicOperation(ABC):
    """原子操作抽象接口"""
    @abstractmethod
    def execute(self, params: dict) -> dict:
        """执行操作并返回结果"""
        pass
```

删除[j]: A.

设置格式[j]: 标准文件_附录一级条标题

设置格式[j]: 项目符号和编号

删除[j]: 7

设置格式[j]: 字体: （默认）宋体, （中文）宋体

```
        @abstractmethod
        def validate(self, params: dict) -> bool:
            """验证参数有效性"""
            pass

    class CentrifugeOperation(IAtomicOperation):
        """离心操作标准化实现"""
        def validate(self, params: dict) -> bool:
            # 参数校验：RPM必须在500-15000之间
            return 500 <= params.get('rpm', 0) <= 15000

        def execute(self, params: dict) -> dict:
            # 构造设备无关命令
            cmd = DeviceCommand(
                operation="centrifuge_start",
                params={
                    "rpm": params["rpm"],
                    "duration": params.get("duration", 300)  # 默认5分钟
                }
            )

            # 通过设备适配器执行
            return OperationEngine.get_instance().device_adapter.execute_command(cmd)

    class PipetteOperation(IAtomicOperation):
        """移液操作标准化实现"""
        def validate(self, params: dict) -> bool:
            # 校验移液量（单位：μL）
            return 0 < params.get("volume", 0) <= 1000

        def execute(self, params: dict) -> dict:
            # 构造设备无关命令
            cmd = DeviceCommand(
                operation="pipette_transfer",
                params={
                    "volume": params["volume"],
                    "source": params["source"],
                    "dest": params["dest"]
                }
            )

            # 通过设备适配器执行
            return OperationEngine.get_instance().device_adapter.execute_command(cmd)
```

A.9 全流程数据操作

数据操作基类定义

```
from .device_proxy import DeviceProxy  # 引用统一寻址模块的DeviceProxy
```

设置格式[j]: 字体: 四号

设置格式[j]: 字体: 四号

设置格式[j]: 标准文件_附录一级条标题

设置格式[j]: 项目符号和编号

删除[j]: A.8

设置格式[j]: 字体: （默认）宋体, （中文）宋体

```
class SCPDataAdapter:
    """SCP协议数据读写适配器"""
    def __init__(self, registry_endpoints: list):
        self.proxy = DeviceProxy(registry_endpoints)

    def read(self, scp_uri: str) -> DataPayload:
        """通过SCP URI读取设备数据"""
        # 解析SCP URI (格式: scplab://zone3/pcr01/temperature)
        device_uri = DeviceURI.parse(scp_uri)

        # 调用设备数据接口
        response = self.proxy.invoke(device_uri, {"action": "read"})

        # 封装为数据负载
        return DataPayload(
            raw=response["data"].encode('utf-8'),
            metadata={
                "source": scp_uri,
                "timestamp": response["timestamp"]
            },
            schema="scp/device"
        )

    def write(self, scp_uri: str, payload: DataPayload) -> bool:
        """通过SCP URI写入设备数据"""
        device_uri = DeviceURI.parse(scp_uri)

        # 转换为设备可接受格式
        if payload.schema == "scp/device":
            data = payload.raw.decode('utf-8')
        else:
            data = json.dumps(payload.to_json())

        # 调用设备写入接口
        response = self.proxy.invoke(
            device_uri,
            {
                "action": "write",
                "value": data
            }
        )
        return response.get("success", False)
```

A.10 __智能体操作

智能体操作基类定义

删除[j]: A.9

设置格式[j]: 标准文件_附录一级条标题

设置格式[j]: 项目符号和编号

设置格式[j]: 字体: （默认）宋体, （中文）宋体

设置格式[j]: 字体: 四号

设置格式[j]: 字体: 四号

```
from abc import ABC, abstractmethod
import time
import random

class IAgentOperation(ABC):
    """智能体原子操作接口"""
    @abstractmethod
    def execute(self, context: 'AgentContext', params: dict) -> 'AgentResult':
        """执行智能体操作"""
        pass

    @abstractmethod
    def validate(self, params: dict) -> bool:
        """验证参数有效性"""
        pass

class ReasoningOperation(IAgentOperation):
    """推理型操作（如LLM思考）"""
    def validate(self, params):
        return "prompt" in params

    def execute(self, context, params):
        start_time = time.time()

        # 从上下文中获取记忆
history = context.recall("conversation_history") or []

# 构造完整提示
full_prompt = f"{params['prompt']}\n\nContext:\n{context.environment}"

# 调用智能体（模型无关）
agent = AgentRegistry.get_agent(params.get("agent_id", "default_llm"))
response = agent.invoke({
    "prompt": full_prompt,
    "temperature": params.get("temperature", 0.7),
    "max_tokens": params.get("max_tokens", 1024)
})

# 更新上下文记忆
context.remember("last_reasoning", response)
history.append({"role": "user", "content": params["prompt"]})
history.append({"role": "assistant", "content": response})
context.remember("conversation_history", history[-10:])  # 保留最近10条

return AgentResult(
    success=True,
    output=response,
    new_context=context,
    cost=calculate_cost(response),
```

```
                duration=time.time() - start_time
            )

    class DataAnalysisOperation(IAgentOperation):
        """数据分析操作"""
        def validate(self, params):
            return "dataset" in params and "question" in params

        def execute(self, context, params):
            # 获取工具管理器
            tool_mgr = context.tools

            # 分步执行分析
            steps = [
                {"tool": "data_loader", "params": {"path": params["dataset"]}},
                {"tool": "clean_data", "params": {"strategy": "auto"}},
                {"tool":  "stat_analysis",  "params":  {"method":  params.get("method",
"describe")}},
                {"tool": "visualize", "params": {"type": "histogram"}}
            ]

            results = []
            for step in steps:
                result = tool_mgr.invoke_tool(step["tool"], step["params"])
                results.append(result)

            # 最终解释
            explanation = AgentRegistry.get_agent("explainer_llm").invoke({
                "prompt": f"Explain this analysis: {results[-1]}",
                "temperature": 0.3
            })

            return AgentResult(
                success=True,
                output={"results": results, "explanation": explanation},
                new_context=context
            )

    class ActionPlanningOperation(IAgentOperation):
        """行动规划操作"""
        def execute(self, context, params):
            # 获取当前环境状态
            env_state = context.environment

            # 调用规划智能体
            planner = AgentRegistry.get_agent("action_planner")
            plan = planner.invoke({
                "goal": params["goal"],
                "constraints": params.get("constraints", ""),
                "current_state": env_state
```

```
})

# 验证计划可行性
validator = AgentRegistry.get_agent("plan_validator")
validation = validator.invoke({
    "plan": plan,
    "safety_rules": context.recall("safety_protocols")
})

return AgentResult(
    success=validation["valid"],
    output=plan if validation["valid"] else validation["issues"],
    new_context=context
)
```

A.11 实验执行流程

实验执行流程基类

```
from enum import Enum
import time
import json
from abc import ABC, abstractmethod

class State(Enum):
    CREATED = 0
    RUNNING = 1
    PAUSED = 2
    COMPLETED = 3
    FAILED = 4

class Condition(ABC):
    """状态转换条件接口"""
    @abstractmethod
    def check(self, context) -> bool:
        pass

class Transition:
    """状态转换规则"""
    def __init__(self, from_state: State, to_state: State, event: str, condition:
Condition = None):
        self.from_state = from_state
        self.to_state = to_state
        self.event = event
        self.condition = condition

    def is_valid(self, context) -> bool:
        """检查转换是否有效"""
        return self.condition is None or self.condition.check(context)
```

设置格式[j]: 字体: 四号

设置格式[j]: 字体: 四号

删除[j]:

设置格式[j]: 字体: （默认）黑体, （中文）黑体

设置格式[j]: 缩进: 首行缩进: 7.4 毫米

删除[j]: A.10

设置格式[j]: 标准文件_附录一级条标题

设置格式[j]: 项目符号和编号

设置格式[j]: 字体: （默认）宋体, （中文）宋体

```
    class ExperimentStateMachine:
        """实验流程状态机引擎"""
        def __init__(self):
            self.current_state = State.CREATED
            self.history = []
            self.transitions = {}
            self.context = ExecutionContext()
            self.listeners = []

            # 初始化默认转换规则
            self._init_default_transitions()

        def _init_default_transitions(self):
            """初始化基础状态转换"""
            self.add_transition(State.CREATED, "start", State.RUNNING)
            self.add_transition(State.RUNNING, "pause", State.PAUSED)
            self.add_transition(State.PAUSED, "resume", State.RUNNING)
            self.add_transition(State.RUNNING, "complete", State.COMPLETED)
            self.add_transition(State.RUNNING, "fail", State.FAILED)
            self.add_transition(State.PAUSED, "abort", State.FAILED)

        def add_transition(self, from_state: State, event: str, to_state: State, condition:
Condition = None):
            """添加状态转换规则"""
            key = (from_state, event)
            self.transitions[key] = Transition(from_state, to_state, event, condition)

        def trigger_event(self, event: str):
            """触发状态转换事件"""
            key = (self.current_state, event)
            transition = self.transitions.get(key)

            if not transition:
                raise InvalidTransitionError(f"无定义转换: {self.current_state} ->
{event}")

            if not transition.is_valid(self.context):
                raise ConditionNotMetError(f"转换条件不满足: {self.current_state} ->
{event}")

            # 记录历史状态
            self._record_history()

            # 更新状态
            old_state = self.current_state
            self.current_state = transition.to_state

            # 通知监听器
            for listener in self.listeners:
```

```
            listener.on_state_changed(old_state, self.current_state, self.context)

    def _record_history(self):
        """记录状态历史"""
        self.history.append(StateHistory(
            state=self.current_state,
            timestamp=time.time(),
            context_snapshot=self.context.capture_snapshot(),
            trigger_event=""
        ))

    def add_listener(self, listener: 'StateListener'):
        """添加状态监听器"""
        self.listeners.append(listener)

    def save_snapshot(self) -> str:
        """保存当前状态快照"""
        snapshot_id = f"snapshot_{int(time.time()*1000)}"
        snapshot = {
            "state": self.current_state.name,
            "context": self.context.capture_snapshot(),
            "history": [h.to_dict() for h in self.history]
        }
        # 实际存储到数据库或文件系统
        SnapshotStorage.save(snapshot_id, snapshot)
        return snapshot_id

    def restore_snapshot(self, snapshot_id: str):
        """恢复历史快照"""
        snapshot = SnapshotStorage.load(snapshot_id)
        if not snapshot:
            raise SnapshotNotFoundError(snapshot_id)

        # 恢复状态
        self.current_state = State[snapshot["state"]]

        # 恢复上下文
        self.context.restore_snapshot(snapshot["context"])

        # 恢复历史记录
        self.history = [
            StateHistory.from_dict(h) for h in snapshot["history"]
        ]

        # 通知监听器
        for listener in self.listeners:
            listener.on_snapshot_restored(snapshot_id)

class StateHistory:
```

设置格式[j]: 字体: 四号

设置格式[j]: 字体: 四号

```
        """状态历史记录"""
        def __init__(self, state: State, timestamp: float, context_snapshot: dict,
trigger_event: str):
            self.state = state
            self.timestamp = timestamp
            self.context_snapshot = context_snapshot
            self.trigger_event = trigger_event

        def to_dict(self):
            return {
                "state": self.state.name,
                "timestamp": self.timestamp,
                "context": self.context_snapshot,
                "event": self.trigger_event
            }

        @classmethod
        def from_dict(cls, data):
            return cls(
                state=State[data["state"]],
                timestamp=data["timestamp"],
                context_snapshot=data["context"],
                trigger_event=data["event"]
            )

    class ExecutionContext:
        """执行上下文管理器"""
        def __init__(self):
            self.variables = {}
            self.device_status = {}
            self.step_counter = 0

        def set_variable(self, name: str, value):
            """设置上下文变量"""
            self.variables[name] = value

        def get_variable(self, name: str, default=None):
            """获取上下文变量"""
            return self.variables.get(name, default)

        def capture_snapshot(self) -> dict:
            """捕获上下文快照"""
            return {
                "variables": self.variables.copy(),
                "device_status": self.device_status.copy(),
                "step_counter": self.step_counter
            }

        def restore_snapshot(self, snapshot: dict):
            """恢复上下文快照"""
```

```
        self.variables = snapshot["variables"].copy()
        self.device_status = snapshot["device_status"].copy()
        self.step_counter = snapshot["step_counter"]
```

设置格式[j]: 字体: 四号

设置格式[j]: 字体: 四号

A.12 异常语义

删除[j]: A.11

设置格式[j]: 字体: （默认）黑体, （中文）黑体

设置格式[j]: 标准文件_附录一级条标题, 缩进: 首行缩进: 0 毫米

设置格式[j]: 项目符号和编号

设置格式[j]: 字体: （默认）宋体, （中文）宋体

异常基础类

```
from enum import Enum
import time
import json

class ErrorLevel(Enum):
    DEVICE = 0x1000
    DATA = 0x2000
    LOGIC = 0x3000

class ErrorSeverity(Enum):
    INFO = 1
    WARNING = 2
    ERROR = 3
    CRITICAL = 4

class LabException(Exception):
    """实验室异常基类"""
    BASE_CODE = 0x0000

    def __init__(self, message: str, context: dict = None):
        self.code = self.BASE_CODE
        self.level = ErrorLevel.DEVICE
        self.message = message
        self.timestamp = int(time.time() * 1000)  # 毫秒时间戳
        self.context = context or {}
        super().__init__(message)

    def get_code(self) -> int:
        """获取完整错误码"""
        return self.level.value | self.code

    def get_level(self) -> ErrorLevel:
        return self.level

    def to_dict(self) -> dict:
        """转换为字典格式"""
        return {
            "code": self.get_code(),
            "level": self.level.name,
```

```
            "message": self.message,
            "timestamp": self.timestamp,
            "context": self.context
        }

    def to_json(self) -> str:
        """转换为 JSON 格式"""
        return json.dumps(self.to_dict(), ensure_ascii=False)

    @classmethod
    def from_code(cls, code: int, message: str = "", context: dict = None):
        """根据错误码创建异常实例"""
        level_code = code & 0xF000
        sub_code = code & 0x0FFF

        if level_code == ErrorLevel.DEVICE.value:
            return DeviceException(sub_code, message, context)
        elif level_code == ErrorLevel.DATA.value:
            return DataException(sub_code, message, context)
        elif level_code == ErrorLevel.LOGIC.value:
            return LogicException(sub_code, message, context)
        else:
            return LabException(f"未知错误码: {hex(code)}", context)

class DeviceException(LabException):
    """设备级异常"""
    BASE_CODE = 0x1000

    def __init__(self, code: int, message: str = "", context: dict = None):
        super().__init__(message, context)
        self.code = code
        self.level = ErrorLevel.DEVICE
        self.device_id = context.get("device_id", "") if context else ""
        self.sensor_readings = context.get("sensor_readings", {}) if context else
{}

class DataException(LabException):
    """数据级异常"""
    BASE_CODE = 0x2000

    def __init__(self, code: int, message: str = "", context: dict = None):
        super().__init__(message, context)
        self.code = code
        self.level = ErrorLevel.DATA
        self.data_source = context.get("data_source", "") if context else ""
        self.data_snapshot = context.get("data_snapshot", {}) if context else {}

class LogicException(LabException):
    """逻辑级异常"""
    BASE_CODE = 0x3000
```

```
def __init__(self, code: int, message: str = "", context: dict = None):
    super().__init__(message, context)
    self.code = code
    self.level = ErrorLevel.LOGIC
    self.function_name = context.get("function", "") if context else ""
    self.input_params = context.get("params", {}) if context else {}
```

设置格式[j]: 字体: 四号

设置格式[j]: 字体: 四号

删除[S]:

参考文献

[1]

设置格式[j]: 字体: （默认）宋体, （中文）宋体, 五号, 英语(美国)

设置格式[j]: 字体: （默认）黑体, （中文）黑体, 五号