AI 智能体技能规范

OpenSAI 使用 Skill 规范

借助当前 MCP 连接暴露的 `opensai_*` 工具，将用户的自然语言计算需求转化为可控的分布式执行，并清晰反馈业务输出、实际耗时与关键调度决策（MCP 连接需由客户端预先配置完成）。

产品概览 | 快速上手 | 使用手册 | 智能体接入 | FAQ 与维护

> ◈ **人类读者导引（关于本文档的用途）**
>
> 本文档对应代码包内的 `skills/zh-CN/opensai/SKILL.md`，是专门编写给支持 MCP 的 AI 智能体（如 Claude、Cursor、OpenClaw）加载的**操作行为规范**。阅读本文档可帮助你直观了解智能体如何将自然语言请求映射为工具参数、幂等轮询与指标解释；如需从零完成服务安装与客户端连接，请先参阅 《智能体接入指南》。

1. 选择正确的执行入口

1. **优先复用已注册应用**：先调用 `opensai_list_apps` 检查可用应用及其真实输入（`inputs`）、可调旋钮（`knobs`）与默认目标（`qos`），匹配时优先使用 `opensai_run_app`。
2. **按需组合算法或单步调用**：若无现成应用，调用 `opensai_catalog` 查询底层算法参数并组织多步骤 DAG（`opensai_submit_dag`）；单一算法直接调用 `opensai_submit_task`。按需调用 `opensai_list_devices` 与 `opensai_list_data`，在同一会话内复用仍有效的发现信息。
3. **区分执行授权与只读查询**：当用户明确提出“运行 / 计算 / 测试某任务”时，视为已授权该次计算，直接调用执行工具无需反复追问；当用户仅要求查看状态或解释历史作业时，只调用只读工具。

2. 将用户意图映射为调用参数

用户表达的需求	对应的工具参数设置
本次处理的具体业务输入（如点数、帧号、场景名）	填入 `inputs` 字典
业务上必须固定不变的参数档位	填入 `knobs` 字典锁定；未锁定的旋钮留给平台自动适配
“在 N 秒内完成”（时限优先，兼顾最高可行质量）	`mode: "deadline"`，同时设置 `deadline_s: N`
“尽快跑完”（速度优先，可附最低质量底线）	`mode: "fastest"`；如有要求则附加 `min_quality`
“按时完成且尽量节省算力资源”	`mode: "efficient"`，同时设置 `deadline_s`（可选 `min_quality`）

- **接口位置差异**：MCP 工具中 `mode`、`deadline_s`、`min_quality` 直接作为顶层参数传入；Python SDK 则放入 `qos={...}` 字典。未覆盖字段沿用应用清单默认值；质量分数始终按该应用自身声明的尺度解释。

调用 `opensai_run_app` 运行分布式聚类的标准示例：

```json
{
  "app": "cluster_analytics",
  "inputs": {"points": 30000, "seed": 1},
  "deadline_s": 3,
  "wait": true,
  "wait_timeout": 60
}
```

3. 单次提交与幂等跟踪

- 提交后立即保存返回的 `job_id`，通过 `opensai_wait_result` 等待完成或用 `opensai_get_result` 查询快照。
- 若返回 `done=false`（单次等待窗口超时），代表后台作业仍在继续运行，**必须复用原 `job_id` 继续查询，切勿重新提交新作业**。
- 因网络抖动需要重试同一次提交请求时，传入相同的 `idempotency_key` 防止重复计算；遇到 `RATE_LIMITED` 时严格按返回的 `retry_after_s` 退避重试。

4. 编排自定义 DAG 或注册新应用包

- **编排 DAG（`dag_spec`）**：顶层包含 `name` 与 `nodes` 列表；每个节点具有唯一正整数 `id`、目录中存在的 `algorithm`、业务参数 `params` 以及前驱 ID 列表 `before_nodes`。提交前必须先调用 `opensai_validate_dag` 校验无误。节点资源置于 `resources`（`cores` 为整数核数，`mem_mb` / `storage_mb` 单位为 MiB，注意 `bw_mbps` 在当前接口中单位为 **MB/s**）；如需绑定特定设备，将设备列表中的 `allocation_index` 填入 `direct_allocation_id`。
- **封装应用包**：由 `app.yaml` 清单与自包含的 `steps/Algorithm_*.py` 步骤文件组成。每个步骤文件须导出 `__all__`，定义同名类并实现 `__init__(self, param_dict, data_dict)` 与 `run(self)`。`_artifact_in`、`_artifact_out`、`_alloc`、`_data_dir` 由平台在运行期自动注入，调用方不得手动传入以下划线开头的保留字段；`run()` 返回业务结果字符串，并可通过 `self.quality`（0～1）和 `self.metrics` 上报实测质量与数值指标。
- **安全注册边界**：先调用 `opensai_validate_app` 通过本地检查，再执行用户已明确要求的应用注册；仅当操作会覆盖其他来源的同名应用或超出用户授权范围时先向用户确认。记住：注册仅同步步骤源码，第三方 Python 库、模型权重与大型数据集需在执行设备上预先就绪。

5. 核对数据本地性与传输路径

- 使用 `opensai_list_data` 核实数据集所在设备；应用步骤的 `data` 字段或 DAG 的 `requires_data` 用于声明强制数据亲和。调用 `opensai_upload_data` 时用 `devices` 指定目标设备序号（省略将默认分发至全部在线设备）。
- 远程连接时，注册与上传工具中的 `path` 均指 **MCP 服务所在机器** 的路径；若代码不在该机器上，改用 `manifest` 与 `files` 字段直接传递文本内容。
- 跨设备中间产物默认经 QUIC 直连拉取；中继多链路需由管理员在启动中枢前配置 `OPENSAI_MULTIPATH=1`（MCP 工具不远程篡改中枢启动环境变量，解释实际传输行为时以作业路径记录为准）。

6. 向用户汇报清晰可核验的结果

1. **结论先行**：先说明作业是否成功（`SUCCEEDED` / `FAILED`）及核心业务输出，再附上端到端耗时 `duration_s`、实测业务质量以及关键设备/核数/旋钮选择（调用 `opensai_explain_job` 获取依据，注意区分“预测质量”与“实

测质量”）。

2. **准确解读统计量**：`duration_s` 是整图端到端墙钟时间；`compute_s`、`fetch_s`、`queue_s` 为各节点累计耗时，存在并行分支时不可直接相加充当总时长。`core_seconds` 表示分配的 CPU 核秒；`bytes_moved` 表示设备拉取的有效载荷字节数（分析多跳中继流量时查看路径级传输记录）。
3. **故障定位与长输出处理**：作业失败时优先读取失败节点的具体报错文本——报缺应用/算法时刷新目录，报缺数据时核对 `opensai_list_data`；已有 `job_id` 时先查询状态再决定下一步。若长结果文本被截断，可将 `opensai_get_result` 的 `max_result_chars` 调高至上限 `4000`，或提示用户使用 Python SDK 导出完整 JSON。