

AliothStudio 应用构建操作手册

蓝皮书 · 从环境准备到部署发布的完整操作路径

版本 v1.1 · 2026 年 10 月 · 杭州宇器科技有限公司

适用版本: Alioth 模型 v10.0.44 · AliothStudio 0.1.x

目录

- 平台构成：两条链路
- 环境准备
- 模型侧：模型中心与模型发布
- 应用构建：AppAgent workflows
- 模块、Block 与 Service 开发要点
- 扩展：extensions/*.yaml
- 构建与部署
- 质量门禁速查
- 在线实践：SaaS 学习版

1. 平台构成：两条链路

AliothStudio 由两条职责分离的链路构成，理解其边界是正确使用平台的前提：

链路	职责	驱动方	产出
模型发布链	数据基座的结构演进（表/列/继承/维度编码）	Meta 模型中心（人工通道）	Alioth/v* 发布包（DDL · 种子 · 机读契约 · 验证报告）
应用构建链	在冻结基座上组装业务应用	AppAgent 智能体 + 开发者	应用配置、Service 后端、模块前端、Deploy/{ns} 部署包

红线：isahl schema 结构冻结。应用构建链中任何「加表/加列/改列」诉求都必须回到模型发布链（模型中心人工裁决），禁止以 DDL、迁移脚本或元数据直写绕行。

2. 环境准备

2.1 前置依赖

依赖	用途
PostgreSQL 14+	数据基座（模型 v10.0 为表继承结构）
mise	任务与环境统一入口（ <code>mise run <task></code> ）
Rust toolchain / pnpm / bun	后端构建、前端构建、脚本工具链

2.2 数据库初始化与启动

```
# 开发库初始化（从发布快照重放；会 DROP 重建目标库，仅限 aliothstudio_dev）
bash scripts/db/reset-db.sh --dev

# Meta 平台（模型管理 + AppAgent）：后端 :4949，前端 :49494
mise run meta

# 命名空间开发环境（Gateway + SSO），{ns} ∈ alioth | wz | avic | cosmic | se | mat
mise run dev:{ns}
```

数据库为四层隔离：生产 `aliothstudio` ← 预发 `aliothstudio_pre` → 开发 `aliothstudio_dev` → 测试 `aliothstudio_test`。环境与 DATABASE_URL 强制绑定；schema 查询一律走 `cd Meta/backend && mise run schema-info -- <command>`（DB 是唯一真相源，禁止读 DDL 文件推断）。

3. 模型侧：模型中心与模型发布

3.1 模型中心（Meta UI）

全部结构演进经 Meta 前端模型中心完成：新建集合（collection，挂接父表形成继承）、注册字段（fields）、维护维度编码（场景/要素/职能）。集合与字段的注册会由平台程序化同步为物理表与继承挂接——开发者不编写任何建表 DDL。

3.2 模型发布

模型中心发起发布后，管线导出 `isahl schema (pg_dump --schema-only + 268 张种子表数据 + 维度种子 seed-dimensions.sql)`，后处理为纯 `CREATE/ALTER` 形态，写入 `Alioth/v{version}/` 并在专用验证库全量重建验证（结构往返 + 种子往返 + uid 唯一性），报告落盘 `verify-report.json`。当前最新发布 `v10.0.44`：1022 张 isahl 表、1204 条继承边，268 张种子表逐表往返一致，验证状态 `verified`。发布流水线产出 7 个进度事件（`prepare / schema / isahl_dump / seed_dimensions / manifest / verify / done`）；集合行层级由模型侧 `row_layer`（`abstract / carry`）声明承载，未裁决父表持行即阻断发布。

模型落地与 DB 派生面同批耦合：`refresh-derived-artifacts.ts` 是 `fk-index`、`relation-index`、`value-domain-columns`、`dk-projection-tables` 四件派生产物的唯一刷新入口，已接入 `reset-db.sh`；产物 diff 即模型拓扑评审面，防摘除由门禁 W9 保证。

3.3 维度与坐标

业务语义经三维坐标定位：`dk_scene`（场景）、`dk_factor`（要素）、`dk_function`（职能）。维度码由类目码 + 序号符号自动复合（如 `JC` = 系统管理）。坐标按命名空间局部解释；Service 的每个 API 静态绑定固定坐标三元组。

4. 应用构建：AppAgent workflow

应用构建链由 AppAgent 驱动，标准工具序列：

```
analyze_ontology    # 需求/原型 → OntologyModel IR (含五领域完备性硬校验)
build_module        # flow-plan → module.json (模块声明)
build_block         # flow-plan → block.json (能力块声明)
transfer_ontology   # 本体对齐 → alignment-graph.json + mapped-entities.json
generate_service_api # 本体实体 → Service 后端 (Rust, 坐标静态绑定)
compose_app         # 模块组装 → app.json + extensions/*.yaml + 原型产物
```

产物落位：`Pre-Proc/{ns}/Apps/{app}/`（`app.json`、`extensions`、流程产物）与 `Pre-Proc/{ns}/Sources/Apps/{Modules,Blocks,Services}/`（模块/块/服务声明与代码）。本体映射产物（`ontology-output.json`）的条目形状受契约门禁校验，坐标必须为 DB 真实码。

5. 模块、Block 与 Service 开发要点

层	关键约束
Module	Scene/Factor 组合容器； <code>module.json</code> 必填字段完整；禁止 DashboardPage（根路径直跳实体列表页）
Block	能力单元； <code>factors / services</code> 引用的 ID 必须存在（孤儿引用有专项门禁）
Service 后端	<code>handlers/models/repositories/services</code> 标准骨架；坐标三元组静态绑定（同表不同坐标拆 API）；禁止 auth 中间件（Gateway 统一 NGAC）；禁止模块内 <code>tracing crate</code>
前端	状态管理一律 Jotai v2；计量字段按「数值 + 单位单选」制式；真实后端联调，禁止 mock；数据访问统一走 <code>@alioth/api</code> 资源级 transport（list/get/mutate，信封经 <code>items() / unwrap()</code> 解包）；引用选择器一律使用 <code>ReferenceSelectDescriptor</code> 描述符，禁止按端点类型外推契约

5.1 数据写入铁律

- **INSERT 落叶表**：写入目标必须是继承偏序的叶表（门禁用 `pg_inherits` 实测，不接受静态猜名）
- **状态走关系表**：实体状态/类目/标签写入 `zc_id_lifecycle_r_*` 关系表，禁止实体自建状态列（`flag` 仅状态字典行使用）
- **类写入契约**：lifecycle 子表写入必须提供六前缀之一的 `dk_function`（`_f_ / _t_` 自动派生，禁止手写字面量）
- **标量引用**：`qk_*` 列是值空间指针（bigint），禁止对其算术运算，读取经 `_refs` 解析
- **时间语义**：未来切片走 task 族、过去切片走 event 族，完成动作落事件切片（切片翻转）

6. 扩展：extensions/*.yaml

应用级扩展（`constraints / rules / statemachines / workflows`）放置于 `Pre-Proc/{ns}/Apps/{app}/extensions/`，Gateway 启动时加载，是扩展生效的唯一驱动面——无需重新生成代码即可调整校验、规则与流程。

7. 构建与部署

```
# 唯一构建入口 (feature 门控 + target 隔离 + 签名自动处理)
bash scripts/build-ns.sh <NAMESPACE> release    # 或 dev (debug 模式)

# 发布 (Apps / seed / .env / apps.json 同步; 不自行构建)
bash scripts/deploy/release-to-namespace.sh <NAMESPACE>
```

产物：`Deploy/{ns}/bin/{ns}-server`（启用 OpenActivity 的命名空间另有 `Deploy/{ns}/Open/bin/openactivity-server`）。发布到 production 阶段自动重启 Gateway。非 namespace 应用 (Meta / SSO) 走 `scripts/meta|sso/deploy.sh`，落 `Release/`。

8. 质量门禁速查

仓库以「规约执行基板」强制质量：每条可检查规约配检查脚本 + 基线 + 门禁登记（当前登记门禁 229 条：commit 阶段 137 条 / push 阶段 92 条）；规约库现有 403 个 capability 活规约，每条 MUST/NEVER 条文必须以结构化字段「执行面」声明其执行位（门禁 id / 脚本 / 测试路径），散文声明不再放行。高频接触项：

门禁	拦截对象	阶段
<code>check-isahl-ddl-boundary</code>	任何绕道的 isahl 结构写	commit
<code>check-leaf-insert</code>	INSERT 未落叶表 / 手写 <code>_f/_t</code> 字面量	commit
<code>check-code-semantics</code>	类型/枚举语义写入 code 列	commit
<code>check-ontology-contract</code>	本体映射产物形状/坐标伪造	commit
<code>check-bridge-host-domain</code>	关系表左端宿主域错位	commit
<code>formal-proofs</code>	Lean 定理构建失败或含证明占位	push
<code>check-all</code> (T1/T2)	版本对齐 / OpenSpec 合规 / i18n / Service DAG 等	push

阻断时不要绕过：按门禁输出的修复面处理；存量违规走基线归线（只减不增），新增违规必须修复。

9. 在线实践：SaaS 学习版

无需本地环境即可体验完整平台：<https://cosmic-tools.shop/> 提供 AliothStudio 的在线 SaaS 学习实践版——面向个人用户的精简部署（核心建模、应用组装与运行线完整可用，企业级多命名空间编排与发布管线从略）。建议路径：注册进入 → 跟随本手册第 2–7 章顺序实操 → 用第 8 章门禁清单自检产物。

配套文献：《Alioth 模型理论白皮书》——理解本手册全部约束背后的数学结构。

© 2025–2026 杭州宇器科技有限公司 (CosmicTools.ltd)。