航班场景v0.4

1. 建模范围与目标

本章节说明本模型的应用场景、整体建模目标，以及文档中各类表示方法（O/E/LT/AT）的使用约定。

1.1 业务范围

本模型聚焦于 **航班运行（Flight Operation）与机场协同（Airport Collaborative Decision Making, A-CDM）** 的整体知识体系建设，覆盖如下典型业务：

- 航班计划制定（计划层 / Flight Plan）
- 运行航班执行（执行层 / Operational Flight）
- 航班状态变更（延误、取消、返航、备降等）
- 机场资源管理（登机口、跑道、机位、滑行道等）
- 塔台 / 管制单元在内的行为主体协同
- 事件驱动的动作触发（Action）
- 多系统之间的数据交换（AODB、FOC、MRO、ADS、ATFM 等）

本体建模的最终目标是构建一个可解释、可复用、可推理的“航班运行知识层”，为智能体调度、事件治理、预测模型提供统一的语义基础。

1.2 建模目标

本模型的设计目标包括：

（1）建立统一的对象体系（FO / O / E）

- FO：形式对象，表达“结构可能是什么”
- O：本体对象，描述类型、模板、结构定义
- E：实体对象，描述真实世界中的具体对象

确保业务模型具备跨系统的一致性和可扩展性。

（2）明确领域内核心本体对象及其关系

包括：

- 航班（计划层 / 执行层）
- 飞机、机场资源、跑道、机位等实体
- 行为主体（塔台、管制、外部系统）
- 事件对象（延误、计划变更、资源冲突等）

（3）构建事件驱动模型（Event → Action → Update）

为调度系统提供统一的“事件触发链路”，包括：

- 哪种事件触发哪些动作
- 动作如何影响实体状态
- 状态变化如何驱动后续事件

此机制构成机场协同与航班运行管理的核心。

（4）提供可推理、可验证的关系与约束

例如：

- Gate 不可被两个航班同时占用
- 后续航班时间必须晚于当前航班落地 + 最小过站时间
- 跑道占用必须遵守安全最小时间间隔

通过本体化表达，实现自动化冲突检查与规则治理。

（5）形成可复用的建模基线

为未来扩展场景（如 ATFM、机场客流管理、地面保障协同）提供统一基础，保证新业务能够平滑添加。

1.3 文档约定（Notation）

为确保整个模型结构清晰，本文档采用统一命名规范：

类型	记法	示例
本体对象（Ontology）	**O_XXX**	O_Flight, O_Gate
实体对象（Entity）	**E_XXX**	E_Flight_CA123_20240208
静态关系（LINK TYPE）	**O1 (LT) O2**	O_Flight (uses) O_Gate
动态关系（ACTION TYPE）	**O1 (AT) O2**	O_FlightDelayedEvent (AT) O_NotifyTowerAction
继承关系	**extends**	O_Flight extends O_FlightBase

其他约定：

- 所有时间均使用 ISO8601 格式：`YYYY-MM-DDThh:mm:ss`
- 所有事件均通过 eventType 区分业务语义
- 所有实体均要求 ID 全局唯一

---

2. FO / O / E 建模基础

本章节系统定义本体建模中最基础的三类对象：
FO（形式对象）、O（本体对象）、E（实体对象），并结合航班场景给出可复用的结构性说明。

2.1 FO（Form Object）定义

形式对象（FO） 是最抽象的对象概念，用于表达“对象的结构与内容”，由若干键值对 (key: value) 组成。
FO 不要求内容完整，也不要求与真实世界一一对应。

FO 的核心特征

- FO = {}（空集）本身就是一个形式对象
- 向 FO 中加入任何属性 `(key: value)` 后仍然是 FO
- FO 不要求所有属性具备确定值
- FO 是 O（本体对象）与 E（实体对象）的共同超集

数学表达

代码块

```
FO = {O, E}
```

belongsto（结构最小差异）关系

若：

- 从 FO1 删除 **最少数量 K 个属性** 后可得到 FO2
- 则称：
  `FO1 (belongsto) FO2`

含义：

- FO2 是 FO1 的“更抽象”结构
- belongsto 是一种结构层级关系，可用于自动推理对象之间的结构继承

2.2 O（Ontology Object）定义

本体对象（O） 是 FO 的子集，用于描述“类型、模板、结构定义”（即 schema）。
O 描述的是一类对象，而非现实中的具体实例。

本体对象的判定条件

一个 FO 若满足：

- 存在至少一个未确定的属性值（string、enum、datetime 类型定义）
- 尚不能与物理世界一一映射
- 用于表达某一类对象的结构模板

则该 FO 是 **本体对象 O**。

示例：航班计划本体（O_FlightPlan）

代码块

```
O_FlightPlan = {
  planId: string,
  flightNumber: string,
  STD: datetime,
  STA: datetime,
}
```

本体对象的主要作用

- 描述对象的属性结构（schema）
- 定义允许出现的 **LINK TYPE / ACTION TYPE**
- 为实体对象（E）提供约束与类型规范

- 为团队提供统一的建模基线

2.3 E（Entity Object）定义

实体对象（E） 是 FO 的子集，用于描述真实世界中的 **具体、可追踪对象**。

实体对象的判定条件

一个 FO 若满足：

- 所有属性具有唯一、明确的具体值
- 与物理世界存在一一映射
- 具备可追踪的唯一 ID（如航班号+日期、机场资源编号）
- 表达的是真实业务对象，而非结构定义

则 FO 是 **实体对象 E**。

示例：执行航班实体

代码块

```
E_Flight_CA123_20240208 = {
  flightId: "CA123-20240208",
  flightNumber: "CA123",
  status: "Active",
  actualDepTime: "2024-02-08T09:42",
}
```

2.4 O 层继承体系（Ontology Inheritance）

为支持跨场景复用，各类对象在 O 层必须形成清晰的继承结构。

以下为航班运行场景的推荐继承树：

代码块

```
O_Object
  ├─ O_Asset              // 机场资源
  │    ├─ O_Airport
  │    ├─ O_Runway
  │    ├─ O_Gate
  │    └─ O_ParkingStand
  │
```

```
├─ O_Actor              // 行为主体
│    ├─ O_Tower
│    ├─ O_ATCUnit
│    └─ O_ExternalSystem
│
├─ O_FlightBase         // 航班类
│    ├─ O_FlightPlan    // 计划层
│    └─ O_Flight        // 执行层
│
└─ O_Event              // 事件类（父对象）
     ├─ O_FlightDelayedEvent
     ├─ O_FlightPlanChangedEvent
     └─ O_ResourceConflictEvent
```

该结构确保：

- 属性可以从父类继承
- LINK / ACTION 类型可在父类上统一定义
- 扩展新类型时保持兼容性（如新增 O_WeatherEvent）

---

3. 核心本体对象（O 层）

本章定义航班场景中所有关键对象的 **本体层（O）结构**，包括属性、类型、继承关系，并明确哪些对象用于计划层、执行层、资源层与事件层。

所有本体对象均为 FO 的子集，用于描述一类对象的结构模板，而不是具体实例。

3.1 航班类本体（Flight Objects）

航班类对象是本场景的核心，分为两大类：

- **航班计划（O_FlightPlan）**：业务计划层
- **运行航班（O_Flight）**：执行运行层

必要时，可通过 `O_FlightBase` 抽象共性属性。

3.1.1 航班基础本体（可选）

代码块

```
O_FlightBase = {
  flightNumber: string,        // 航班号
  origin: string,              // 计划起飞机场
  destination: string,         // 计划到达机场
}
```

用于容纳计划层与执行层的共性信息。

3.1.2 航班计划本体（O_FlightPlan）

描述机场/航司发布的计划航班信息，是 O 层（本体）对象。

代码块

```
O_FlightPlan extends O_FlightBase = {
  planId: string,              // 计划唯一 ID
  STD: datetime,               // 计划起飞时间
  STA: datetime,               // 计划到达时间
  planStatus: enum(Planned, Updated, Canceled),
}
```

语义说明：

- 所有字段均为类型定义，未对应具体值 → 属于 O 层
- 计划层可被事件变更（如 O_FlightPlanChangedEvent）

3.1.3 运行航班本体（O_Flight）

描述真实执行过程中的航班运行对象（执行层）。

该本体用于创建实体 E_Flight。

代码块

```
O_Flight extends O_FlightBase = {
  flightId: string,            // 航班运行实例 ID
  aircraftType: string,        // 飞机型号
  status: enum(
    Created, Planned, Active, Landed, Canceled
  ),
  scheduleSource: string,      // 来源系统
}
```

执行层航班的关键语义：

- 必须具备 flightId 类型定义
- 状态机受事件驱动（详见第 5 章）
- 运行航班可引用资源（Gate、Runway、Stand）

3.2 机场资源本体（Airport Assets）

机场资源属于 **O_Asset** 子类，是本场景中最关键的资源类对象，用于支撑机位分配、跑道冲突检测、机场容量管理等推理任务。

3.2.1 机场（O_Airport）

代码块

```
O_Airport extends O_Asset = {
  airportCode: string,        // 机场三字码，如 ZBAA
  name: string,
  timezone: string,
}
```

3.2.2 跑道（O_Runway）

代码块

```
O_Runway extends O_Asset = {
  runwayId: string,
  category: enum(CAT_I, CAT_II, CAT_III),
  length: int,
}
```

3.2.3 登机口（O_Gate）

代码块

```
O_Gate extends O_Asset = {
  gateId: string,
  gateType: enum(Contact, Remote),
}
```

3.2.4 机位（O_ParkingStand）

代码块

```
O_ParkingStand extends O_Asset = {
  standId: string,
  standCategory: string,      // 适用机型范围
}
```

3.3 行为主体本体（Actor）

行为主体是场景中触发动作（Action）的关键角色，例如塔台、管制单元、系统。

3.3.1 塔台（O_Tower）

代码块

```
O_Tower extends O_Actor = {
  towerId: string,
  region: string,
}
```

3.3.2 管制单元（O_ATCUnit）

代码块

```
O_ATCUnit extends O_Actor = {
  unitId: string,
  controlArea: string,
}
```

3.3.3 外部系统（O_ExternalSystem）

代码块

```
O_ExternalSystem extends O_Actor = {
  systemId: string,
  systemType: string,
}
```

3.4 事件本体（Event Base）

事件本体描述“变化”，是动态模型（Action）的触发源。

3.4.1 通用事件本体（O_Event）

代码块

```
O_Event = {
  eventId: string,
  eventType: string,
  eventTime: datetime,
  severity: enum(Info, Warning, Critical),
  sourceSystem: string,
}
```

说明：

- 所有其他事件均继承该结构
- 为事件生命周期建立统一约定（见第 5 章）

3.4.2 航班延误事件（O_FlightDelayedEvent）

代码块

```
O_FlightDelayedEvent extends O_Event = {
  delayMinutes: int,
  delayPhase: enum(DEP, ARR, TURNAROUND),
  reasonCode: string,
}
```

3.4.3 航班计划变更事件（O_FlightPlanChangedEvent）

代码块

```
O_FlightPlanChangedEvent extends O_Event = {
  oldSTD: datetime,
  newSTD: datetime,
  changeReason: string,
}
```

3.4.4 资源冲突事件（O_ResourceConflictEvent）

代码块

```
O_ResourceConflictEvent extends O_Event = {
  resourceType: string,
```

```
  conflictDescription: string,
}
```

本章总结

本章为航班场景提供了系统的 O 层对象定义：

包括航班类、资源类、行为主体类、事件类，形成完整且可扩展的本体体系。

下一章（第 4 章）将定义实体对象（E 层），并展示 O → E 的实例化方式。

4. 实体对象（E 层）

实体对象（E）用于描述 **真实业务世界中的具体对象实例**，是 O 层（本体）的实例化结果。

与 O 层相比，E 层具有关键特征：

- 所有属性均具有 **明确、唯一、确定值**
- 能与物理世界 **一一映射**
- 必须存在 **唯一 ID**
- 能够随运行过程发生变化（由事件驱动）

本章将定义航班场景中的主要实体对象及示例。

4.1 航班实体（E_Flight）

航班实体对应某一天、某一班次的实际运行航班，例如 “CA123 / 2024-02-08”。

其结构来自 O_Flight，本体字段全部填入具体值：

代码块

```
E_Flight_CA123_20240208 = {
  flightId: "CA123-20240208",
  flightNumber: "CA123",
  aircraftType: "A320",
  status: "Active",
  origin: "ZBAA",
  destination: "ZSPD",
  actualDepTime: "2024-02-08T09:42",
  actualArrTime: null
}
```

说明：

- flightId 为全局唯一
- status 会由事件触发更新（见第 7 章）
- 所有值均已确定 → 属于 E 层

4.2 航班计划实体（E_FlightPlan）

运行前，每个计划航班在计划层也是一个实体（通常来自 AODB 或航司排班系统）。

代码块

```
E_FlightPlan_CA123_20240208 = {
  planId: "FP-CA123-20240208",
  flightNumber: "CA123",
  STD: "2024-02-08T09:00",
  STA: "2024-02-08T11:10",
  planStatus: "Published"
}
```

说明：

- planStatus 可随计划变更事件更新
- 用于执行层航班的调度与校验

4.3 机场资源实体（E_Gate / E_Runway / E_Stand）

各类资源均来自 O_Asset 派生对象，实体样例如：

4.3.1 登机口（Gate）

代码块

```
E_Gate_T3_52 = {
  gateId: "T3-52",
  gateType: "Contact"
}
```

4.3.2 跑道（Runway）

代码块

```
E_Runway_01L = {
  runwayId: "01L",
  category: "CAT_III",
  length: 3800
}
```

4.3.3 机位（Stand）

代码块

```
E_ParkingStand_245 = {
  standId: "245",
  standCategory: "A320/A321/B738"
}
```

说明：

资源实体一般不会频繁变更，但会参与“资源占用”、“冲突检测”等推理任务。

4.4 行为主体实体（E_Tower / E_ATCUnit / E_ExternalSystem）

塔台实体

代码块

```
E_Tower_ZBAA_T1 = {
  towerId: "ZBAA-T1",
  region: "Terminal 1"
}
```

管制单元实体

代码块

```
E_ATCUnit_DEP = {
  unitId: "ATC-DEP",
  controlArea: "Departure"
}
```

外部系统实体

```
E_ExternalSystem_MRO = {
  systemId: "MRO",
  systemType: "Maintenance"
}
```

用途：

用于定义 Action 的触发接收方（如通知塔台、同步到 MRO）。

4.5 事件实体（E_Event）

事件实体对应真实发生的一次航班变化，如一次延误事件。

示例：

代码块

```
E_FlightDelayed_20240208_001 = {
  eventId: "EVT-20240208-001",
  eventType: "FlightDelayed",
  eventTime: "2024-02-08T09:30",
  severity: "Warning",
  sourceSystem: "AODB",

  delayMinutes: 50,
  delayPhase: "DEP",
  reasonCode: "WX"
}
```

说明：

- eventId 全局唯一
- 事件实体不可修改过去，只能关闭（进入 Closed 状态）
- 触发后续 ACTION（见第 7 章）

本章总结

实体层（E）是对本体层（O）的真实投射，通过填入具体值形成“真实世界的镜像”。
E 层内容是运行系统中最核心的数据对象，并将被 **事件驱动模型**（Event → Action → Update）持续更新。

下一章（第 5 章）将定义实体与事件的 **生命周期模型（State Machine）**，用于规范状态迁移逻辑。

5. 生命周期模型（State Machine）

生命周期（State Machine）用于描述实体与事件在业务过程中的 **状态变化路径**。
生命周期模型不仅帮助系统进行状态管理，也为后续的 **事件触发（Event）、动作执行（Action）、约束检查（Constraint）** 提供语义基础。

本章分别定义：

- 航班计划生命周期
- 运行航班生命周期
- 事件生命周期

5.1 航班计划生命周期（Flight Plan Lifecycle）

航班计划（O_FlightPlan）在运行前由航司或 AODB 系统生成，其生命周期通常如下：

```
代码块
1   Draft → Published → Updated → Archived
```

状态语义说明

状态	含义
Draft	初版计划，未对外发布
Published	已正式发布，可被调度系统使用
Updated	被事件（如计划变更）修改过的版本
Archived	历史归档，不再参与调度

典型事件触发

- `O_FlightPlanChangedEvent` 会使计划进入 **Updated** 状态
- 计划一旦用于生成运行航班，可进入 **Archived**

5.2 执行航班生命周期（Operational Flight Lifecycle）

运行航班（O_Flight）在真实业务中具有明确的时间线与状态流。

推荐状态机如下：

```
代码块
```

```
Created → Planned → Active → Landed → Closed
```

状态语义说明

状态	含义
Created	航班从计划层实例化，但未进行调度
Planned	已确定资源（飞机/时刻/机位）并准备执行
Active	航班处于执行过程，例如离港滑行、起飞、在途等
Landed	航班已落地，等待地面程序处理
Closed	全部流程结束，进入历史区

状态变更由事件驱动

事件	状态变化
FlightDelayedEvent	Planned → Planned（状态不变但属性更新）
FlightDepartureEvent	Planned → Active
FlightArrivalEvent	Active → Landed
FlightCloseEvent	Landed → Closed

说明

- “延误”本质上不是状态变化，而是属性变化
- 状态变化通常由执行类事件触发（如起飞、落地）

5.3 事件生命周期（Event Lifecycle）

事件实体（E_Event）本身也具备独立生命周期，确保其在系统中可追踪。

推荐状态机如下：

代码块

```
New → Processing → Closed → Archived
```

状态意义

状态	含义
New	刚产生的事件，还未处理
Processing	已开始执行相关动作（Action）
Closed	动作链路已全部执行完毕
Archived	历史事件，仅用于审计与推理

事件驱动说明

- 状态从 **New → Processing** 代表系统开始执行动作链路
- 事件执行完 Action 后进入 **Closed**
- 定期将旧事件进入 **Archived**

5.4 生命周期统一规则（跨对象）

所有生命周期遵循以下统一原则：

（1）不能逆向迁移

例如航班不能从 Landed 回到 Active。

（2）状态迁移必须由事件触发

如：

```
代码块
1  E_Flight.status = "Active"
2  ⇐ 由 E_FlightDepartureEvent 触发
```

（3）状态机之间可能存在联动

例如：

- 航班进入 Landed → 触发资源释放动作
- 航班进入 Active → 触发跑道占用检查
- 计划变更事件 → 触发执行航班属性调整

（4）状态变化必须写入历史（不可篡改）

用于后续：

- 审计（Audit）

- AI 预测（ML）
- 知识推理（Reasoning）

本章总结

生命周期模型为整个航班运行知识体系提供了 **可验证的时序逻辑**。
通过事件驱动状态流转，系统可实现：

- 自动调度
- 时序一致性校验
- 行为监督与预测
- 资源冲突检测

下一章（第 6 章）将进入更关键的部分：
静态关系（LINK TYPE）建模，用于表达航班、机场资源、事件之间的结构性关联。

6. 静态关系（LINK TYPE, LT）

静态关系（LINK TYPE, LT）用于描述 **对象之间在结构上、业务上、语义上的长期性关系**。
与动态关系（Action Type）不同，LT 不涉及事件触发，也不表达操作，只描述“是什么关系”。

在知识图谱与本体建模中，LINK TYPE 是构建可推理图谱的核心。

6.1 LT 定义说明（方向、多重性、约束）

每个 LT 必须明确以下语义：

（1）方向（Direction）

关系是否具备显式方向，例如：

```
代码块
1   O_Flight → O_Gate
```

用于表达谁“使用/引用/绑定”谁。

（2）多重性（Cardinality）

指定关系的数量约束：

- 1:1 —— 一个航班只对应一个 Gate

- 1:N —— 一架飞机可以执行多个航班
- N:N —— 多个系统可管理多个对象

（3）可选性（Optional / Required）

关系是否为必需条件，例如：

- 航班必须有飞机（required）
- 航班可选登机口（某些机场无廊桥）

（4）生命周期绑定（Owned / Referenced）

表示对象是否被另一个对象“所有（Owned）”或只是“引用（Referenced）”：

- 航班引用 Gate（Referenced）
- 文件夹包含文件（Owned，非本案例）

（5）业务约束（Constraints）

例如：

- “同一时间段一个 Gate 不能服务多个航班”
- “跑道分配必须符合机型等级”

这些约束将在第 8 章进一步展开。

6.2 航班与资源（Flight–Resource Links）

航班在执行过程中需要占用多类资源，包括机位、登机口、跑道等。

6.2.1 航班使用登机口

代码块

```
O_Flight (uses) O_Gate
 direction: Flight → Gate
 cardinality: 1:1
 required: false
 lifecycle: referenced
 constraints:
   - 同一时间段 Gate 不可被多个航班占用
```

6.2.2 航班使用跑道

```
O_Flight (uses) O_Runway
 direction: Flight → Runway
 cardinality: 1:1
 required: true
 lifecycle: referenced
 constraints:
   - 跑道使用必须满足最小时间间隔
   - 跑道与航班方向必须一致（01L/19R）
```

6.2.3 航班使用机位（Stand）

代码块

```
O_Flight (parkedAt) O_ParkingStand
 direction: Flight → Stand
 cardinality: 1:1
 required: true
 constraints:
   - 机位类型必须支持对应飞机型号
```

6.3 航班与飞机（Flight–Aircraft Links）

航班在执行阶段必须绑定具体飞机。

代码块

```
O_Aircraft (assignedTo) O_Flight
 direction: Aircraft → Flight
 cardinality: 1:N
 required: true
 lifecycle: referenced
 constraints:
   - 同一时刻一架飞机只能执行一个航班
   - 后续航班起飞时间 >= 前序航班到达时间 + 最小过站时间
```

此 LT 将与 **约束（Constraint）** 紧密相关。

6.4 事件作用对象（Event–Entity Links）

事件必须明确其“影响对象”（Affected Entity），否则无法形成事件驱动链路。

6.4.1 事件影响航班

```
O_Event (impacts) O_Flight
 cardinality: N:1
 constraints:
   - 每个事件必须绑定唯一作用航班
```

示例：

- 延误事件 → 作用于某一航班
- 计划变更事件 → 作用于航班计划对象

6.4.2 事件影响资源

```
O_Event (impacts) O_Gate
 cardinality: N:1
 constraints:
   - 资源冲突事件必须绑定冲突资源
```

示例：

- O_ResourceConflictEvent → 作用于 Gate 或 Stand
- O_RunwayCloseEvent → 作用于 Runway

6.5 资源之间的结构性关系（Asset–Asset Links）

某些资源之间也有父子结构或引用关系，例如机场下管理多个跑道：

```
O_Airport (manages) O_Runway
 cardinality: 1:N
```

```
O_Airport (manages) O_Gate
 cardinality: 1:N
```

这些关系有助于推理：

- 机场容量
- 资源可用性
- 机场内部结构导航（例如根据站坪位置选择可用跑道）

6.6 行为主体与对象的关系（Actor–Object Links）

塔台管理跑道

代码块

```
O_Tower (controls) O_Runway
 cardinality: 1:N
```

管制单元负责航班

代码块

```
O_ATCUnit (manages) O_Flight
 cardinality: 1:N
```

外部系统负责事件来源

代码块

```
O_ExternalSystem (produces) O_Event
 cardinality: 1:N
```

本章总结

- LINK TYPE 描述对象之间的 **结构性、持久性关系**
- 包含方向、多重性、可选性、生命周期绑定与业务约束语义
- 为 **动态关系（Action Type）** 和 **事件链路（Event Flow）** 提供语义基础
- 在知识图谱中可直接转为图谱边类型（Edge Type）

接下来（第 7 章）将介绍最关键的 **动态关系（ACTION TYPE, AT）**，用于描述事件触发动作链路。

7. 动态关系（ACTION TYPE, AT）

动态关系（ACTION TYPE, AT）用于描述 **事件（Event）如何触发动作（Action），从而影响实体对象（E）**。
与静态关系（LINK TYPE, LT）描述长期结构关系不同，ACTION TYPE 描述的是 **过程、行为、因果链路**。

ACTION TYPE 是整个航班运行知识图谱中实现“事件驱动”与“自动调度”的核心语义。

7.1 ACTION TYPE（AT）定义与结构字段

一个 AT 必须包含如下核心语义：

（1）触发事件（trigger event）

动作发生的前提，例如：

```
代码块
1   eventType = "FlightDelayed"
```

（2）目标对象（target entity）

动作影响的实体对象，例如：

- 影响航班 → E_Flight
- 影响资源 → E_Gate
- 影响外部系统 → E_Tower

（3）动作效果（effect）

描述动作想要“做什么”，例如：

- 通知塔台
- 更新航班状态
- 重新分配资源

（4）更新规则（update rules）

用于描述动作如何修改实体属性，例如：

```
代码块
1   E_Flight.status = "Delayed"
```

（5）幂等性约束（idempotency）

确保动作不会重复执行，例如：

- 通知只能发一次
- 状态只能在正确条件下迁移

ACTION TYPE 模板结构

代码块

```
O_Action = {
  actionType: string,
  triggerEvent: O_Event,
  targetEntity: O_Entity,
  effect: string,
  updateRules: {...},
  idempotent: boolean
}
```

7.2 典型动作定义（AT 示例）

以下给出航班运行场景中最核心的 ACTION TYPE。

7.2.1 航班延误 → 通知塔台（NotifyTower）

此动作由航班延误事件触发。

代码块

```
O_FlightDelayedEvent (AT) O_NotifyTowerAction
```

触发条件

代码块

```
triggerEvent.eventType = "FlightDelayed"
```

动作效果

- 向塔台系统发送航班延误通知
- 推动管制部门提前规划时刻与跑道排队顺序

更新规则

```
代码块
1   E_Flight.status = "Delayed"          // 状态更新
2   E_Flight.delayMinutes = event.delayMinutes
```

幂等性

```
代码块
1   idempotent = true
```

此动作可重复触发，但不应多次写入相同通知。

7.2.2 航班计划变更 → 更新计划对象（UpdatePlan）

```
代码块
1   O_FlightPlanChangedEvent (AT) O_UpdateFlightPlanAction
```

动作效果

- 将新的起飞/到达时间写回 E_FlightPlan
- 若执行层航班已创建，则同步更新执行层属性

更新规则示例

```
代码块
1   E_FlightPlan.STD = event.newSTD
2   E_FlightPlan.STA = event.newSTA
```

7.2.3 计划变更 → 再分配机场资源（ReassignGate）

计划调整（如时间提前/延后）可能导致资源冲突，需要重新分配。

```
1  O_FlightPlanChangedEvent (AT) O_ReassignGateAction
```

动作效果

- 检查 Gate 时间占用冲突
- 若冲突 → 自动重新分配可用 Gate
- 更新 E_Flight 所使用的 Gate

更新规则示例

```
1  E_Flight.gateId = NewGate
```

7.2.4 航班落地 → 释放资源（ReleaseResources）

```
1  O_FlightArrivalEvent (AT) O_ReleaseResourcesAction
```

动作效果

- 释放跑道
- 释放机位
- 释放登机口

更新规则

```
1  E_Gate.status = "Available"
2  E_ParkingStand.status = "Available"
```

7.2.5 资源冲突事件 → 资源冲突告警

```
1  O_ResourceConflictEvent (AT) O_AlertConflictAction
```

动作效果

- 发送冲突告警
- 启动冲突解决流程（如 Gate 调整、时刻协调）

7.3 ACTION TYPE 的执行原则

（1）事件必须绑定目标实体

没有目标实体的 Action 不应执行。

（2）Action 只能修改实体（E），不能直接修改本体（O）

例如：

- 不能修改 O_Flight.status（这是模板）
- 只能修改 E_Flight.status（实体实例）

（3）Action 必须与状态机配合（第 5 章）

例如：

- 状态只能从 Planned → Active → Landed → Closed
- Action 不能越级修改状态

（4）Action 可以生成新的事件

如：

- Gate 冲突 → 触发 O_ResourceConflictEvent
- 延误事件 → 触发额外通知事件

（5）Action 应具备幂等性

避免重复执行导致状态异常，例如重复通知、重复分配资源。

本章总结

本章定义了：

- ACTION TYPE 的结构模型
- 动作触发链路（Event → Action → Entity Update）

- 多个典型航班场景下的动作样例
- 动作执行的业务约束原则

ACTION TYPE 是构建“智能调度”“自动响应”的关键语义模块，为下章（第 8 章）业务约束模型（Constraint）提供动态触发基础。

8. 业务约束模型（Constraint）

业务约束（Constraint）用于表达航班运行与机场协同中必须满足的硬性规则与逻辑条件，是维持系统一致性、避免冲突和保证安全运行的关键机制。

Constraint 通常与 **静态关系（LT）**、**动态关系（AT）**、**生命周期（State Machine）** 共同作用，形成一个可推理、可验证的知识体系。

8.1 Gate 使用冲突（Gate Conflict Constraint）

登机口是稀缺资源，需要严格的时间占用约束。

约束描述

代码块

```
Constraint_GateConflict = {
  scope: O_Gate,
  rule: "同一时间段内，Gate 不得被两个航班同时占用",
  severity: error
}
```

检查逻辑

当一个航班执行 Gate 分配（或计划调整）时，必须检查：

代码块

```
E_Flight_A.timeRange ∩ E_Flight_B.timeRange = ∅
```

违反即视为冲突，触发：

- 资源冲突事件（O_ResourceConflictEvent）
- Gate 重分配动作（O_ReassignGateAction）

8.2 Turnaround（过站）规则

过站时间是飞机连续执行航班的硬性安全要求。

约束描述

```
代码块
1  Constraint_Turnaround = {
2    scope: O_Flight,
3    rule: "后续航班起飞时间 >= 前序航班落地时间 + 最小过站时间",
4    severity: warning
5  }
```

检查逻辑

```
代码块
1  STD_next >= ATA_prev + minTurnaroundTime
```

若不满足：

- 触发提醒
- 建议重新排班
- 可触发计划变更事件

8.3 Runway 使用限制（Runway Occupancy Constraint）

跑道是最关键的容量资源之一，需严格遵守占用间隔与适航配置。

约束描述

```
代码块
1  Constraint_RunwayOccupancy = {
2    scope: O_Runway,
3    rule: "连续两架飞机使用跑道之间必须满足最小安全间隔",
4    severity: error
5  }
```

典型间隔规则

- Wake Turbulence（尾流）
- Arrival → Arrival 需 ⩾ X 秒
- Arrival → Departure 需 ⩾ Y 秒

执行时机

在航班状态进入“Active”或“预计落地”时强制检查。

8.4 飞机连续航班的时间约束（Aircraft Rotation Constraint）

飞机执行多个航班时必须满足可行性：

约束描述

代码块

```
Constraint_AircraftRotation = {
  scope: O_Aircraft,
  rule: "同一架飞机执行多个航班时，不得出现时间冲突",
  severity: error
}
```

示例逻辑

代码块

```
flightA.ATA + taxiIn + serviceTime <= flightB.ETD
```

否则：

- 触发航空器排班冲突事件（O_RotationConflictEvent）
- 建议换机（Aircraft Swap Action）

8.5 航班计划与执行一致性（Plan–Ops Consistency Constraint）

执行层（E_Flight）与计划层（E_FlightPlan）之间必须保持部分一致性。

约束描述

代码块

```
Constraint_PlanOpsConsistency = {
  scope: O_FlightPlan,
  rule: "执行航班的关键属性不可偏离计划超出阈值",
  severity: warning
}
```

核查字段

- 航班号（flightNumber）
- 航线（origin/destination）
- STD/STA 偏差阈值（如超过 30 分钟需触发事件）

8.6 航班状态机合法性（State Machine Validity Constraint）

约束航班状态时间线合法性（对应第 5 章）。

约束描述

代码块

```
Constraint_FlightStateTransitions = {
  scope: O_Flight,
  rule: "航班状态必须按定义的生命周期路径迁移",
  severity: error
}
```

不合法例子

- Landed → Active（非法）
- Planned → Closed（非法）
- Active → Planned（非法）

非法迁移应触发：

- 状态异常事件（O_InvalidStateTransitionEvent）
- 自动纠正动作（O_AutoFixFlightStatusAction）

8.7 数据一致性与引用完整性（Data Integrity Constraints）

确保实体之间的引用关系有效。

示例约束：

- 航班分配的 Gate 必须存在
- Flight.origin / Flight.destination 必须是有效 Airport
- Gate 必须属于该 Airport（跨机场引用禁止）

```
代码块
1  Constraint_ReferenceIntegrity = {
2    scope: O_Flight,
3    rule: "所有引用对象必须存在于数据库或知识图谱中",
4    severity: error
5  }
```

本章总结

本章定义了航班运行系统中最核心的约束机制，包括：

- Gate 冲突
- Turnaround 过站限制
- Runway 占用间隔
- Aircraft 排班可行性
- 计划层与执行层一致性
- 状态机合法性
- 数据引用一致性

这些约束共同构成航班运行的 **安全边界与语义规则**，并为事件触发、预测模型、冲突检测引擎提供基础逻辑。

接下来将在 **第 9 章** 展示如何将以上所有模型结合起来，形成“端到端建模示例”。

9. 典型业务链路示例（端到端）

本章通过一个完整的端到端案例展示：
如何将 **本体对象（O） → 实体对象（E） → 静态关系（LT） → 动态关系（AT） → 约束（Constraint） → 状态更新（State Update）** 串联成为一条完整的业务链路。

示例场景：
“航班延误 → 通知塔台 → 再分配 Gate → 更新航班状态”

这是机场协同（A-CDM）与航班运行管理中最常见的链路之一。

9.1 输入实体（E 层）

在链路开始时，系统中已有如下实体：

（1）航班实体 E_Flight

```
代码块
1  E_Flight_CA123_20240208 = {
2    flightId: "CA123-20240208",
3    flightNumber: "CA123",
4    status: "Planned",
5    origin: "ZBAA",
6    destination: "ZSPD",
7    gateId: "T3-52"
8  }
```

（2）登机口实体 E_Gate

```
代码块
1  E_Gate_T3_52 = {
2    gateId: "T3-52",
3    gateType: "Contact"
4  }
```

（3）塔台实体 E_Tower

```
代码块
1  E_Tower_ZBAA_T3 = {
2    towerId: "ZBAA-T3",
3    region: "Terminal 3"
4  }
```

这些实体之间在 LT（静态关系）中已有结构关联：

- `O_Flight (uses) O_Gate`
- `O_Tower (controls) O_Runway`（用于推理通知链路）

9.2 输入事件（Event）

航班延误事件从外部系统（如 AODB）传入：

```
代码块
1   E_FlightDelayed_20240208_001 = {
2     eventId: "EVT-CA123-001",
3     eventType: "FlightDelayed",
4     eventTime: "2024-02-08T09:30",
5     delayMinutes: 50,
6     delayPhase: "DEP",
7     severity: "Warning",
8     sourceSystem: "AODB"
9   }
```

静态关系使事件与目标对象关联：

```
代码块
1   O_Event (impacts) O_Flight
```

因此自动可推断：

```
代码块
1   E_FlightDelayed_20240208_001 → E_Flight_CA123_20240208
```

9.3 动作链路（Action Flow）

事件触发动态关系（ACTION TYPE）。

本场景将触发三个主要 Action：

9.3.1 Action 1：通知塔台（NotifyTower）

```
代码块
1   O_FlightDelayedEvent (AT) O_NotifyTowerAction
```

执行逻辑

- 系统生成通知任务（如 MQTT、Webhook、系统消息）
- 发送至塔台实体 E_Tower_ZBAA_T3
- 塔台系统可以提前安排跑道时刻

更新规则

```
代码块
1   E_Flight.status = "Delayed"
```

9.3.2 Action 2：重新检查 Gate 占用（CheckGateConflict）

延误可能导致占用时间推迟，与其他航班形成冲突。

```
代码块
1   O_FlightDelayedEvent (AT) O_CheckGateConflictAction
```

执行逻辑

检查新时间范围是否与其他航班冲突：

```
代码块
1   NewTimeRange ∩ OtherFlightRange ≠ ∅ ?
```

如有冲突 → 执行 Action 3。

9.3.3 Action 3：重新分配 Gate（ReassignGate）

如果 Gate 冲突，系统执行：

```
代码块
1   O_FlightDelayedEvent (AT) O_ReassignGateAction
```

执行逻辑

- 查询可用 Gate
- 按机型/区域/时段筛选
- 分配新 Gate，例如：

```
代码块
1   E_Flight.gateId = "T3-66"
```

并生成新的事件：

```
代码块
1   O_GateReassignedEvent → 触发后续相关处理
```

9.4 实体状态更新结果（State Update）

在上述 Action 执行后，实体层被更新：

航班实体更新

```
代码块
1   E_Flight_CA123_20240208.status = "Delayed"
2   E_Flight_CA123_20240208.delayMinutes = 50
3   E_Flight_CA123_20240208.gateId = "T3-66"    // 如需重分配
```

事件状态更新（事件生命周期）

```
代码块
1   Event.status: New → Processing → Closed
```

Gate 更新

如有变动：

```
代码块
1   E_Gate_T3_52.status = "Available"
```

9.5 约束验证流程（Constraint Check）

在上述整个链路中，以下约束参与了推理：

约束	影响
Gate 冲突约束	决定是否触发 Gate 重分配
航班状态机合法性约束	确保状态从 Planned → Delayed 合法
计划–执行一致性约束	判断延误是否需要同步更新计划
数据一致性规则	确保新的 Gate 存在且合法

所有动作执行结束后，系统进行最终一致性校验。

本章总结

通过本章的端到端链路示例，我们展示了：

- 如何从 **实体层（E）** 构建业务对象
- 如何通过 **事件（Event）** 驱动动作（Action）
- 如何通过 **动态关系（AT）** 执行自动化处理
- 如何通过 **约束（Constraint）** 确保业务逻辑正确
- 如何自动完成 **实体状态更新（State Update）**

这一链路清楚展示了本体化建模如何支持智能调度、冲突检测与事件处理。

下一章（第 10 章）将介绍模型的复用与扩展策略。

10. 扩展与复用说明

本章说明如何将航班场景 v0.4 建模体系扩展到更广泛的业务场景，并提供可复用的建模方法，确保本体结构具备长期可演进性。

10.1 如何新增事件类型（Event Type）

新增事件时，应遵循以下原则：

（1）事件必须继承自 `O_Event`

例如新增一个“天气影响事件”：

```
代码块
1  O_WeatherImpactEvent extends O_Event = {
2    weatherType: enum(WX_Storm, WX_Fog, WX_Snow),
3    impactLevel: enum(Minor, Major, Severe)
4  }
```

（2）事件必须具备：

- 固有字段：eventId、eventType、eventTime、severity
- 业务字段：与事件语义直接相关

（3）事件必须定义其作用对象（LINK TYPE）

如：

```
代码块
1  O_WeatherImpactEvent (impacts) O_Flight
```

（4）事件必须指定触发的动作（ACTION TYPE）

如：

```
代码块
1  O_WeatherImpactEvent (AT) O_AlertWeatherAction
```

10.2 如何新增资源类型（Asset Type）

新资源必须继承自 `O_Asset`，并附加自身特有属性。

示例：新增滑行道对象

```
代码块
1  O_Taxiway extends O_Asset = {
2    taxiwayId: string,
3    length: int,
4    direction: string
5  }
```

原则：

1. 所有资源均通过静态关系绑定机场：

```
代码块
1   O_Airport (manages) O_Taxiway
```

2. 所有资源均可成为事件影响对象：

```
代码块
1   O_TaxiwayClosedEvent (impacts) O_Taxiway
```

3. 如资源参与调度，则应定义 Resource Constraint，例如最小占用间隔。

10.3 如何新增动作类型（Action Type）

所有新 Action 必须：

1. **有明确的触发事件**
2. **指向可执行的目标实体 E**
3. **具备 effect 与 updateRules**
4. **符合整体状态机逻辑**

例如新增“流控延误 → 更新航班流控字段”：

```
代码块
1   O_FlowControlEvent (AT) O_UpdateFlowControlInfoAction
```

updateRules：

```
代码块
1   E_Flight.flowControlStatus = event.flowControlStatus
```

10.4 如何扩展到更多机场/航司场景

由于本体设计具有层次化、可抽象特性，可以轻松扩展至：

（1）A-CDM（机场协同决策）

新增：

- O_MilestoneEvent
- O_TargetTimeEvent
- O_CDMPhase

用于描述 TOBT、TSAT、TTOT 等要素。

（2）ATFM（空管流量管理）

新增：

- O_CapacityRestrictionEvent
- O_SlotAllocationAction

用于描述航路/区域流控。

（3）地面保障（GSE & Handling）

新增资源：

- O_GroundVehicle
- O_BaggageSystem

新增事件：

- O_VehicleDelayEvent
- O_GroundServiceReadyEvent

（4）飞机维修（MRO）

新增：

- O_MaintenanceOrder
- O_MaintenanceEvent

可用本体继承体系进行扩展。

10.5 如何复用本体结构到其它行业

本体系不仅能用于航班场景，也能映射到其他需要“事件驱动 + 资源调度”的行业，例如：

- **物流运输**：车辆、任务、仓库、路由规划
- **制造生产**：设备、工单、产线资源、异常事件
- **能源调度**：电网资源、调度事件、动作链路

- **医疗应急**：病人状态、医疗资源、急救事件

关键在于：

复用 FO/O/E 的三层结构

```
FO → 抽象结构
O  → 类型定义
E  → 实体实例
```

复用事件驱动链路（Event → Action → Update）

所有业务场景都可采用该模式。

10.6 模型演化与版本升级建议

为保证模型的长期演进能力：

（1）保持 O 层的稳定性

O 层是系统的 “Schema”，变更要谨慎。

（2）事件类型可以快速迭代

事件是反映业务变化的关键点，适合敏捷扩展。

（3）Action 可以随业务变化灵活调整

Action 是动态逻辑，允许根据策略迭代。

（4）Constraint 需要严格版本管理

Constraint 的一处改动可能影响整个调度过程。

（5）实体层（E）永远是“实时镜像”

E 随业务实时变化，不进行版本管理。

本章总结

本章提供了 v0.4 模型的扩展与复用方法，包括：

- 新增事件 / 资源 / Action 的方式

- 如何适应 A-CDM、ATFM、MRO 等复杂业务
- 如何将模型复用到其他行业
- 如何管理模型版本演化

至此，航班场景 v0.4 建模体系已经形成完整、可复用、可扩展、可推理的知识图谱基础。