Week03｜课时 1｜从“能采上来”到“可重复采集”：为什么 ingest 可靠性决定下游一切

Table of contents

先把 ingest baseline 讲清，再谈更复杂的增量与流式

这一讲先立一个判断：

系统不是“采不到数据”才危险，而是今天能采、明天漂、后天想重跑却找不到当时到底发生了什么。

Week02 让你知道哪些输入允许进入系统；Week03 从这一讲开始，让你真正面对 ingestion baseline 的运行时现实。

进入课时 2 返回 Week03 总览

下载讲义

提供适合离线阅读的 PDF 版和适合批注整理的 Word 版。

PDF 版 · 打印 / 离线阅读 Word 版 · 批注 / 二次整理

这节课解决什么问题

到了 Week03，课程开始从“输入准入”正式进入“输入运行”。

Week02 已经把下面这些问题定义清楚了：

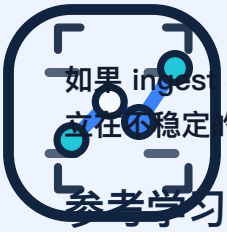
- 哪些数据值得进入系统
- 四类数据分别应该长什么样
- 最小 metadata 和 PII 边界是什么
- Data Contract 怎么把输入约束写成机器可读规则
- Manifest 和 Gate 为什么是 Week03 的起跑线

但这还不够。

因为在真实系统里，真正难的从来不只是“能采上来”，而是：

- 这次采集能不能重复执行
- 失败之后能不能可靠重跑
- 同一批数据有没有重复、缺口或静默丢失
- 入湖后的结果能不能和 manifest / contract / batch 对得上
- 以后出问题时，能不能快速知道“断在了哪里”

所以，这节课要建立的核心判断是：



如果 ingest 不可重复、不可追溯、不可恢复，下游所有索引、RAG、Agent 和评测都会建立在不稳定的基础上。

参考学习时间（45–55 分钟）

如果你只阅读正文，大约需要 30–35 分钟；如果你跟着当前 repo 一起跑一遍 seed_loader 最小入口，并顺手整理 ingestion_baseline_v1.md 与 smoke report，建议预留 45–55 分钟。

学完这一讲，你应该能做到什么

完成这一讲后，你应该能：

1. 解释为什么 Week03 不是“再讲一次 ETL”，而是把 Week02 的 contract 推进成可运行的数据链路。
2. 区分“能采上来”和“可重复采集”的工程差异。
3. 识别 ingest 可靠性里最关键的 4 类问题：重复、缺口、错位、无追踪。
4. 看懂 OmniSupport Copilot 当前 repo 中和 Week03 最相关的 ingest 基线对象。
5. 跑通一次最小 smoke flow，并输出一份《ingestion baseline》说明。

本课产出

完成这节课后，你至少应该在仓库中补齐或确认这 3 份文档：

- docs/blueprints/week03/ingestion_baseline_v1.md
- docs/blueprints/week03/reliability_checklist.md
- reports/week03/ingest_smoke_report.md

什么叫 Week03 的 ingest baseline

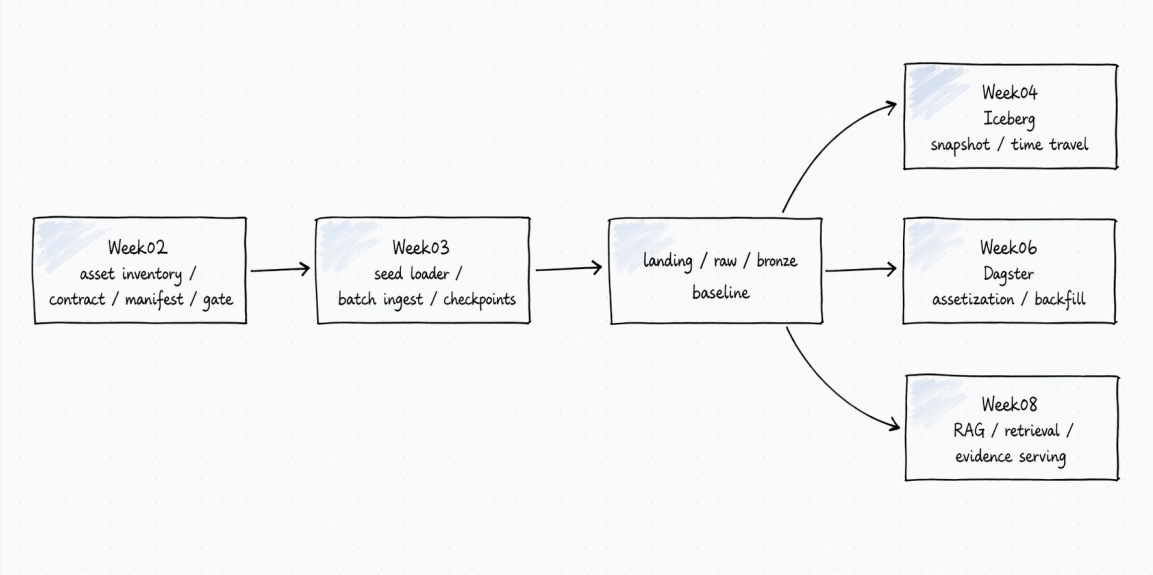
这一讲里的 ingest baseline，不是“现在仓库里终于有了几个采集脚本”。它至少要同时满足下面 5 个条件：

条件	它要求你真正具备什么
输入边界明确	你能说清本次 ingest 由哪个 manifest、哪批 source、哪个 batch window 声明
执行可以重复	同一批次重跑不会把结果越跑越脏
状态可以持久化	你知道 run_id、checkpoint、cursor、watermark 应该落在哪里
结果可以解释	你能用 run evidence 解释 success / skip / reject / fail 到底发生了什么

条件	它要求你真正具备什么
恢复有路径	你知道问题出现时该 retry、replay 还是 backfill，而不是全凭记忆重跑

如果这 5 个条件缺任何一个，Week03 最多只能算“采到数据”，还不能算“站住了 ingestion baseline”。

先看一张总图



这张图要表达的重点是：

- Week02 解决的是 **输入是否有资格进入系统**
- Week03 解决的是 **这些输入如何稳定、重复、可恢复地进入系统**
- Week03 的结果不是最终服务，而是后面所有层的 **ingest baseline**

1. 为什么 Week03 不是“再讲一次 ETL”

很多人会把 Week03 误解成“我们现在开始写采集脚本了”。

这只是表面现象。

真正更准确的理解是：

> Week03 是把 Week02 的规则体系推进成可运行的基础设施。

你现在要分清的两句话

说法	其实还不够	为什么
“我们已经能把文件读进来了”	还不够	可能无法重复、无法校验、无法对账
“我们已经能把数据写进去了”	还不够	可能有重复、缺口、错序、没有 run trace
“manifest 也有了”	还不够	还需要让 manifest 和 contract、run log、落盘结果真正对齐
“测试也过了”	还不够	contract test 通过不等于 ingest 基线已经可回放、可补数

所以 Week03 的视角必须升级成：

从“能跑一次”升级成“能可靠跑很多次”。

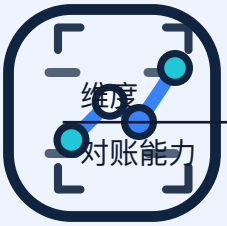
这一讲最关键的判断

> ! 核心判断
>
> **一条 ingest 链路只有在“可重复、可恢复、可校验、可对账”时，才配成为 Week04 以后所有 AI 数据层的上游。**

2. 从“能采上来”到“可重复采集”到底差在哪里

下面这张表，是本课最核心的对比表。

维度	只是“能采上来”	真正“可重复采集”
执行方式	手动跑一次脚本	有 manifest / batch / run 语义
数据边界	靠文件名和记忆	有清晰的 manifest_id / source_fingerprint / batch window
失败恢复	重跑碰碰运气	能定位缺口，知道该 replay、retry 还是 backfill



维度	只是“能采上来”	真正“可重复采集”
对账能力	看起来像成功了	能对 count / checksum / manifest coverage / rejects
下游影响	写进去就算完成	能作为 Iceberg / RAG / Graph / Evals 的稳定上游
复盘能力	出事后靠日志猜	有 runbook、run_id、ingest report、checkpoint

你应该从这张表里直接读出一个结论：

> **Week03 的价值不在于“把数据搬进来”，而在于把“这次搬运”变成一条以后还能解释、还能恢复、还能复现的工程链路。**

3. 先建立 5 个必须被带起来的追踪锚点

Week03 最容易被低估的一件事是： 一旦你没有从第一天开始把这些追踪锚点带起来，后面再补会非常痛。

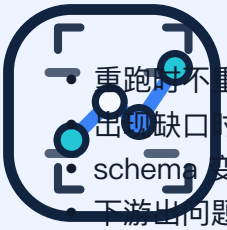
这一讲建议你先把下面这 5 个运行时锚点记住：

锚点	它用来回答什么
manifest_id	这次 ingest 是按哪次声明执行的
batch_id	这次 ingest 对应哪一个批次或哪一个输入窗口
run_id	这次运行到底是哪一次
source_fingerprint	这份输入内容来自哪个版本 / 指纹
trace_id	这次 ingest 和下游验证、告警、run report 如何串起来

状态相关对象最好单独看，不要和运行锚点混成一团：

状态对象	它真正回答什么
checkpoint	成功处理并持久化到了哪里
cursor	下一次增量应该从哪里继续看
watermark	当前批次承认的时间或序列边界是什么

这 5 个锚点不是为了“字段看起来更专业”，而是为了后面这些能力：



- 重跑时不重复
- 出现缺口时能定位
- schema 变化时能回看影响范围
- 下游出问题时能反推上游输入状态

一个很现实的问题

如果今天晚上 ingest 失败了，第二天你最需要知道的不是“服务挂过没有”，而是：

- 哪个 manifest 在跑
- 跑到哪一个 source
- 哪些数据已经写进去
- 哪些没有写进去
- 下游看到的是旧数据还是新数据
- 是 retry，还是 replay，还是 backfill

如果你现在就能感受到这些问题的重要性，说明这节课的定位已经对了。

4. 输入链路里最常见的 4 类事故

这一节是 Week03 的现实感来源。

别把 ingest 想成理想流水线。生产里最常见的不是“完全不能跑”，而是“跑了，但 quietly 出错”。

4.1 重复（Duplicate）

常见来源：

- 同一批次被重复执行
- 增量窗口边界算错
- retry 没有幂等键
- 上游重新投递，消费端没去重

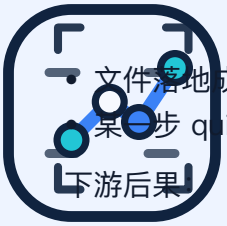
下游后果：

- 指标翻倍
- 同一记录多次进入索引
- Graph / embedding / eval 集出现伪样本膨胀

4.2 缺口（Gap）

常见来源：

- manifest 漏了一部分 source
- 增量 cursor 跳过了晚到数据



- 文件落地成功，但 metadata write 失败
- 某一步 quietly fail，没有被 run log 记录

下游后果：

- RAG 召回缺页、缺段
- KPI 少算一段时间
- 回归评测结果不稳定
- 坏例子无法完整复盘

4.3 错位（Mismatch / Drift）

常见来源：

- 上游 schema 变了
- contract 理解不一致
- 文档版本没对上
- 音频片段时间和文本对不上

下游后果：

- 记录结构合法，但业务语义错了
- evidence anchor 指向错误版本
- ingestion 看似成功，serving 却开始漂

4.4 无追踪（No Traceability）

常见来源：

- 没有 run id
- 没有 manifest 对应关系
- 没有 checkpoint
- 没有 source_fingerprint

下游后果：

- 失败无法精确定位
- 不能说清本次入湖范围
- 回放时不知道从哪一段开始补

Week03 的排障顺序

当你发现 ingest 结果不对时，优先按这个顺序排查：

1. 这次 ingest 是哪个 `manifest_id`
2. 这次 ingest 的 `run_id` 是什么
3. 当前 source 的 checkpoint / cursor 在哪里
4. 写进去的数据和 manifest coverage 是否一致
5. 是否出现 duplicate / gap / drift / missing trace 中的某一类

5. 把 Week02 的 contract / manifest 接到 Week03 的 repo 现实里

这一步很关键。

Week03 不是从空白目录开始，而是要贴着 **OmniSupport Copilot 当前仓库** 往下走。

当前 repo README 已经把最重要的 Week03 基线对象摆出来了：

- `contracts/data/*.json`
- `data/seed_manifests/*.json`
- `pipelines/ingestion/seed_loader.py`
- `tests/contract/test_json_schemas.py`
- Docker-first 命令基线[1]

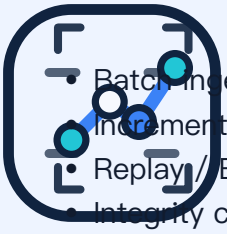
这意味着你在本课实践里，不需要再自己发明第二套世界。 你真正要做的是：**先读懂并跑通当前 repo 已有的 ingest baseline。**

建议你先认识这 4 类对象

对象	当前 repo 中的作用
`contracts/data/*.json`	约束 source 输入边界与 schema
`data/seed_manifests/*.json`	声明 Week03 的最小 ingest 范围
`pipelines/ingestion/seed_loader.py`	作为 manifest-driven ingest baseline 的入口
`tests/contract/test_json_schemas.py`	校验 contract 与 schema 的最小正确性

这也是为什么 Week03 的学生实践主线建议是：

[1] OmniSupport Copilot 当前 README 已明确给出 Week01–Week03 的 Docker-first 基线、`seed_loader.py` 入口以及 `pytest tests/contract/ -v` 作为最小 contract tests 验证方式。OmniSupport Copilot README



- Batch ingest
- Incremental cursor
- Replay / Backfill
- Integrity checks
- Runbook

而不是一上来强制每个学生在本地完整跑 Kafka / Debezium 栈。

6. 动手实践：跑通一次最小 ingest baseline smoke flow

现在开始动手。

第 1 步：确认环境已经起来

如果你前面还没有启动环境，先在项目根目录运行：

```
cp infra/env/.env.example infra/env/.env.local

docker compose --env-file infra/env/.env.local -f infra/docker-compose.yml up -d --build
```

如果你已经在 Week01 / Week02 期间启动过环境，可以直接跳到下一步。

第 2 步：跑一次 seed loader dry-run

执行：

```
docker compose --profile tools --env-file infra/env/.env.local -f infra/docker-compose.yml run --rm devbox \
  python -m pipelines.ingestion.seed_loader --manifest-dir data/seed_manifests
```

这是 Week03 最值得先跑通的一条命令。 因为它会让你第一次看到：

- 当前 repo 里 manifest-driven ingest 的入口在哪里
- 这套 ingest baseline 现在已经能感知哪些 source
- 你后面在课时 2、课时 3 要扩展的地方具体在哪

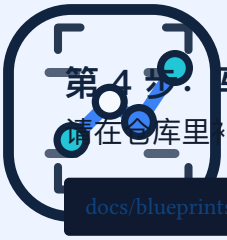
第 3 步：跑一次 contract tests

执行：

```
docker compose --profile tools --env-file infra/env/.env.local -f infra/docker-compose.yml run --rm devbox \
  pytest tests/contract/ -v
```

这一步的意义不是“证明代码能跑”，而是让你确认：

- Week03 的 ingest 不是脱离 contract 的
- source baseline 的第一道门禁仍然是 schema / contract correctness



第 4 步：写一页 baseline 说明

建在仓库里补一份：

```
docs/blueprints/week03/ingestion_baseline_v1.md
```

建议至少写 5 段：

1. 当前 repo 里的 ingest 入口是什么
2. 当前 manifest 体系负责声明什么
3. 当前 contract tests 负责拦什么
4. 现在还缺哪些能力（如 checkpoint、replay、backfill）
5. Week03 后续会优先补什么

第 5 步：写一份 smoke report

再补一份：

```
reports/week03/ingest_smoke_report.md
```

你至少要写清楚：

- 你跑了哪些命令
- 哪些命令通过了
- 哪些对象你确认存在
- 你现在认为 Week03 最大的 ingest 风险是什么

7. 本课最容易忽略的一点：Week03 不是“实时化冲动”，而是“可靠性基线”

很多人一提到 ingest，就会立刻想到：

- 流式
- CDC
- Kafka
- 实时同步
- Exactly-once

这些都重要，但如果你把 Week03 的主线直接写成“我们现在开始做实时平台”，就会偏掉。

更稳的顺序是：

1. 先把 batch ingest 做可靠
2. 再把 incremental cursor 做清楚

3. 再讲 CDC / WAL / logical decoding 的边界
4. 最后再讨论实时一致性和成本权衡

Airbyte 的官方文档对这个顺序其实给了很朴素的说明：incremental sync 之所以有意义，是因为你已经有了可被 cursor 跟踪的变化边界；第一次 incremental 本质上仍然会把源数据先完整读一遍。[2]

也就是说：

> **“实时”不是 Week03 的第一性原理，可靠性和可恢复性才是。**

8. 本课你真正要带进课时 2 的是什么

这节课结束时，你不只是拿到了几个命令。

你真正带进课时 2 的，至少有这 5 样东西：

1. 一个判断：Week03 的重点不是“把数据搬进来”，而是“让数据可重复进入系统”。
2. 一份 baseline 文档：`ingestion_baseline_v1.md`
3. 一份 smoke 报告：`ingest_smoke_report.md`
4. 一个更清晰的 repo 视图：manifest / contract / seed_loader / tests 各自负责什么
5. 一组待补能力清单：checkpoint、state、replay、backfill、integrity checks

课时 2 就会开始把这些东西具体推进成：

- batch ingest baseline
- 幂等写入
- 完整性校验
- raw / landing / bronze 的最小落盘语义

9. 本课最重要的 8 个判断

1. Week03 不是“再讲一次 ETL”，而是把 Week02 的规则推进成可运行链路。
2. “能采上来”不等于“可重复采集”。
3. ingest 可靠性的核心不是快，而是可重复、可恢复、可对账、可追踪。
4. `manifest_id` / `batch_id` / `run_id` / `source_fingerprint` / `trace_id` 是最值得先带起来的运行锚点，而 `checkpoint` / `cursor` / `watermark` 应该单独作为状态对象被记录。
5. 重复、缺口、错位、无追踪，是 Week03 最常见的四类事故。
6. Week03 的学生实践主线应该优先贴着当前 repo 的真实 ingest baseline，而不是另起第二套工程世界。[3]

---

[2] Airbyte 官方文档说明，incremental sync 基于 cursor 识别自上次同步以来新增或更新的数据；如果目标为空，第一次 incremental 本质上会先搬完整数据集。Airbyte Incremental Sync

7. CDC / Stream 要讲清，但不应在当前课程主线里冒进为“本地必须全量搭起 Kafka / Debezium”。
8. 课时 1 的任务不是把所有 ingest 细节讲完，而是先把“采集可靠性决定下游一切”的判断立住。

10. 本课自检清单

学完这一讲后，请确认你至少能勾掉这些项：

- ☐ 我能解释为什么 Week03 不是“再讲一次 ETL”
- ☐ 我能区分“能采上来”和“可重复采集”
- ☐ 我知道 ingest 中最危险的 4 类事故是什么
- ☐ 我知道 manifest、contract、seed_loader、tests 在当前 repo 中分别负责什么
- ☐ 我已经跑过一次 `seed_loader` baseline 命令
- ☐ 我已经跑过一次 `pytest tests/contract/ -v`
- ☐ 我已经补了一份 `ingestion_baseline_v1.md`
- ☐ 我已经补了一份 `ingest_smoke_report.md`

11. 课后最小行动

在进入课时 2 前，请把下面这 3 件事做完：

1. 把 `docs/blueprints/week03/ingestion_baseline_v1.md` 补齐
2. 把 `reports/week03/ingest_smoke_report.md` 补齐
3. 用自己的话写下：
   - 当前 ingest baseline 最缺的 2 项能力
   - 你认为 Week03 里最先该补的是 checkpoint、integrity check、还是 replay，为什么

延伸阅读

- OmniSupport Copilot README / Quick Start / seed loader
- Airbyte / Incremental Sync
- PostgreSQL / Logical Decoding Concepts
- Debezium / Snapshot + Stream
- Dagster / Partitions and Backfills

[3] OmniSupport Copilot 当前 README 已明确给出 Week01–Week03 的 Docker-first 基线、`seed_loader.py` 入口以及 `pytest tests/contract/ -v` 作为最小 contract tests 验证方式。OmniSupport Copilot README