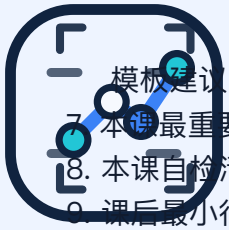
Week03 | 课时 2 | 批量采集主链路：幂等写入、重跑设计与完整性校验

Table of contents



先把 batch 主链路做稳，再谈更复杂的增量与流式

这一讲先解决最容易被轻视的问题：

Manifest 写了，不代表 batch ingest 就天然可重跑；真正的关键是批次边界、写入语义、运行记录和结果对账能不能闭合。

进入课时 3 回看课时 1 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

课时 1 已经把 Week03 的总判断建立起来了：

> ingest 的价值，不是“把数据搬上来”，而是让输入进入系统这件事 **可重复、可追溯、可恢复、可对账**。

这一讲要进一步把这个判断推进成一个更具体的工程动作：

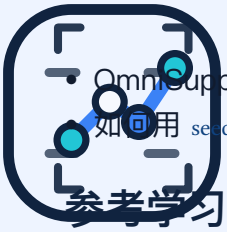
把批量采集链路做成最小但可靠的主链路。

很多团队一听到 batch，就会下意识觉得它“老”“慢”“过渡性强”。但在真实企业数据工程里，批量链路依然是最稳的 baseline，因为它最适合建立：

- 明确的 batch 边界
- manifest 驱动的可重复执行
- 幂等写入
- 完整性校验
- 重跑与补数的最小语义

你在这一讲不会去追求“最酷的实时流”，而是会先把下面几件事做扎实：

- 为什么 batch 仍然是当前课程主线最稳的起点
- 幂等写入、重跑、完整性校验、reconcile 到底是四件什么不同的事



- OmniSupport Copilot 当前 repo 里的 ingest baseline 是怎样被组织起来的
- 如何用 seed_loader、ticket_ingest 与 contract tests 做一次可靠的 batch smoke flow

参考学习时间（50–60 分钟）

如果你只阅读正文，大约需要 30–35 分钟；如果你跟着本课一起对照 seed_loader.py、ticket_ingest.py 和 contract tests 跑一遍最小 batch smoke flow，并顺手整理 batch_ingestion_design_v1.md，建议预留 50–60 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 batch 不是“落后方案”，而是可靠 ingest baseline。
2. 区分 4 个常被混淆的概念：**幂等写入、重跑、完整性校验、reconcile**。
3. 看懂 OmniSupport Copilot 当前 repo 中 seed_loader、ticket_ingest、contract tests 各自负责什么。
4. 跑通一次最小 batch smoke flow，并解释输出里的关键计数。
5. 写出一份最小《batch ingest 设计说明》与《完整性检查报告》草稿。

本课产出

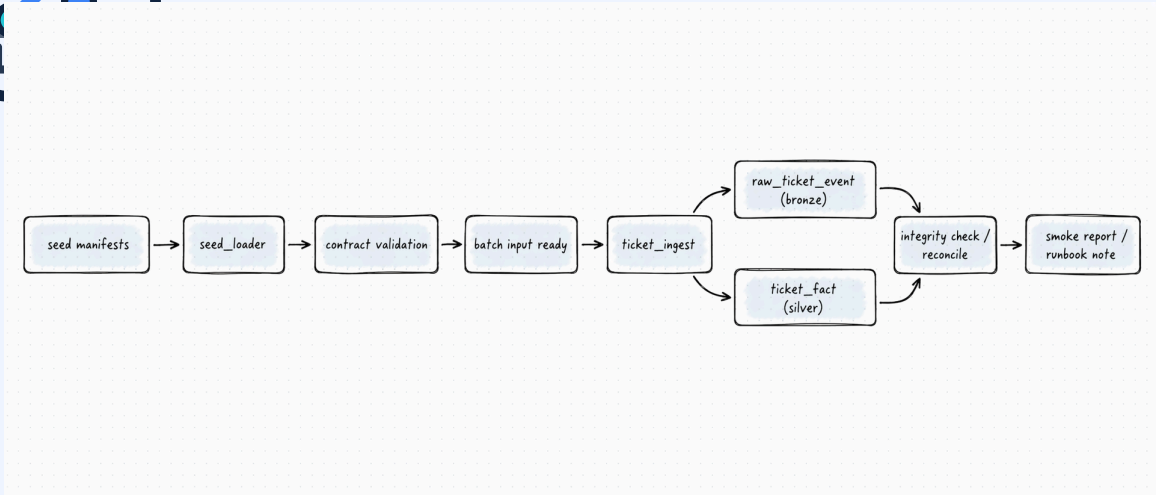
完成这节课后，你至少应该在仓库中补齐或确认这两份文档：

- docs/blueprints/week03/batch_ingestion_design_v1.md
- reports/week03/batch_ingest_smoke_report.md

可选加分项：

- reports/week03/batch_integrity_checklist.md

先看一张总图



这张图要表达的重点是：

本课不是教你“把一批数据写进去”，而是教你怎样让一批数据以后还能被解释、被复核、被安全地重跑。

1. 为什么当前课程主线先从 batch 讲起

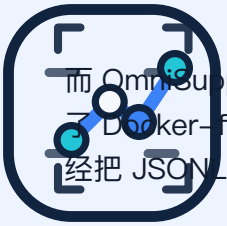
Week03 的周名里写了 Batch / CDC / Stream。 这并不意味着学生本地实践必须从第一分钟就跑完整实时链路。

对当前课程主线来说，batch 是最合理的入口，原因很直接：

为什么先选 batch	工程上的真实好处
边界最清晰	一次 manifest、一批输入、一轮结果，最容易对账
最适合讲幂等	你能清楚看到“同一批次重跑”会发生什么
最适合讲完整性	可以直接比对 count / inserted / invalid / skipped / errors
最适合接回放	replay / backfill 的边界天然以批次或窗口来表达
最适合教学	先建立可靠性，再引出增量、CDC、流式

换句话说：

> **Week03 不是先追求快，而是先追求稳。**



而 OmniSupport Copilot 当前 repo 的实际结构也支持这种教学顺序：README 已经给出 Docker-first 启动、seed_loader、pytest tests/contract/ -v 这些最小可跑入口；ticket_ingest.py 又已经把 JSONL -> contract validation -> Bronze + Silver 写入的基本路径搭出来了。[1][2][3]

2. 先把 3 个最容易混淆的概念拆开

很多同学会把下面这四件事混成一件：

- 幂等写入
- 重跑
- 完整性校验
- reconcile

其实它们解决的问题完全不同。

2.1 幂等写入：同一条记录重复到达，结果不能越写越脏

幂等写入解决的是：

> **同样的数据如果被处理两次，最终结果不能失真。**

在 ticket_ingest.py 里，你能直接看到这种设计已经在发生：

- Bronze 层 raw_ticket_event 用 ON CONFLICT (event_id) DO NOTHING
- Silver 层 ticket_fact 用 ON CONFLICT (ticket_id) DO UPDATE

这就是非常典型的幂等思路： 同样的 ticket 再来一遍，不是简单重复插入，而是按主键控制写法。[4]

2.2 重跑（rerun）：同一批次失败后，系统能否安全地再执行一次

重跑解决的是：

> **如果这次 ingest 中途失败，我重新跑一次，会不会留下半成品、重复数据、或者让上下游状态更乱。**

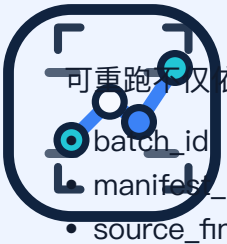
---

[1] OmniSupport Copilot 当前 README 已经把 Docker-first 启动、seed_loader、pytest tests/contract/ -v 作为最小工程基线给出。OmniSupport Copilot README

[2] seed_loader.py 当前已经承担 manifest 目录加载、source_manifest_schema.json 校验、业务规则校验与 dry-run report，并且文件头显式标注“Week03: 接入真实 MinIO 上传 + PostgreSQL 写入”。seed_loader.py

[3] ticket_ingest.py 当前已支持 JSONL 输入、JSON Schema 校验、最小业务规则校验、--batch-id、--dry-run、--limit，并将数据写入 raw_ticket_event Bronze 与 ticket_fact Silver。ticket_ingest.py

[4] ticket_ingest.py 当前已支持 JSONL 输入、JSON Schema 校验、最小业务规则校验、--batch-id、--dry-run、--limit，并将数据写入 raw_ticket_event Bronze 与 ticket_fact Silver。ticket_ingest.py



可重跑不仅依赖幂等写入，还依赖：

- batch_id
- manifest_id
- source_fingerprint
- checkpoint / cursor（如果是增量）
- 报告与 runbook

所以：

- **有幂等** != 一定 **可重跑**
- 但 **要想可重跑**，通常必须先有幂等

2.3 完整性校验：这批数据“写完了”不等于“写对了”

完整性校验解决的是：

> **我怎么知道这次 ingest 没有 quietly 丢掉一部分数据，也没有 silently 把坏数据放进下游。**

你至少要能解释这些数字：

- total
- valid
- invalid
- inserted
- skipped
- errors

ticket_ingest.py 最后的 summary 就是围绕这些数字组织的。[5] 而 seed_loader.py 里的 IngestResult 也明确区分了 success_count / skip_count / fail_count。[6]

2.4 Reconcile：把输入声明、写入结果和异常记录真正对上

reconcile 解决的是：

> **manifest 里宣称要处理的输入，和最终写进去、跳过、拒绝、失败的结果，能不能被一张表解释清楚。**

---

[5] ticket_ingest.py 当前已支持 JSONL 输入、JSON Schema 校验、最小业务规则校验、--batch-id、--dry-run、--limit，并将数据写入 raw_ticket_event Bronze 与 ticket_fact Silver。ticket_ingest.py

[6] seed_loader.py 当前已经承担 manifest 目录加载、source_manifest_schema.json 校验、业务规则校验与 dry-run report，并且文件头显式标注“Week03: 接入真实 MinIO 上传 + PostgreSQL 写入”。seed_loader.py

它通常会对比：

- manifest coverage
- source count
- valid / invalid / rejected
- Bronze / Silver 写入结果
- 是否存在未解释缺口

reconcile 和完整性校验有关，但不等价：

- 完整性校验更像检查“结果是否合理”
- reconcile 更像检查“输入声明和输出结果是否闭合”

一张表把 4 个概念彻底拆开

概念	它解决什么	没有它会怎样
幂等写入	同一记录重复到达时，结果不会被污染	指标翻倍、重复写入、状态错乱
重跑（rerun）	一次失败后能安全再执行同一条 job	每次恢复都像“赌一次”
完整性校验	你能知道这批数据到底有没有写全、写对	看起来成功，实际上 quietly 缺口或脏写
reconcile	你能把 manifest、写入结果和异常记录真正对上	结果看似完整，但没人能证明是否漏处理了一部分输入

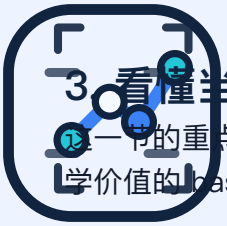
先把 rerun 和 replay 分开

本课先把 rerun 讲清，目的是让你知道：

- rerun 更像重新执行同一条批处理作业
- replay 更像 later-stage 的输入级重放，强调“重新消费同一批或同一来源”

所以课时 2 的重点不是把 replay 讲完，而是先把 rerun 所依赖的批次边界、幂等写入和 reconcile 站稳。

> ! 这一讲最重要的工程判断
>
> **批量链路的可靠性，不是靠“脚本能跑完”来证明，而是靠幂等、可重跑和完整性校验三件事共同成立。**



3. 看懂当前 repo 里的最小 batch baseline

这一节的重点不是让你读完整个 repo，而是认清下面这 3 个对象已经组成了一个非常有教学价值的 baseline。

3.1 seed_loader.py：先把 manifest 语义建立起来

seed_loader.py 当前承担的是：

- 读取 data/seed_manifests/*.json
- 先按 source_manifest_schema.json 做结构校验
- 再做业务规则校验
- 在 dry-run 模式下输出每个 manifest / asset 的处理结果
- 为 Week03 真实 MinIO + PostgreSQL 写入预留接口

而且它自己在文件头已经明确写出阶段定位：

- Week01–02：完整的 manifest 驱动采集框架
- Week03：接入真实 MinIO 上传 + PostgreSQL 写入[7]

所以你应该把 seed_loader 理解成：

Week03 把 manifest 变成 ingest baseline 的第一个台阶。

3.2 ticket_ingest.py：第一条“真正写数据”的基线

相比 seed_loader 更偏“计划 + 验证”，ticket_ingest.py 更像第一条真正的 batch ingest 主链路。

它已经具备这些结构化能力：

- 读取 JSONL 输入
- 用 ticket_contract.json 做 JSON Schema 校验
- 做最小业务规则校验（例如 ticket_id 格式、created_at 必填）
- 写入 raw_ticket_event Bronze
- 写入 ticket_fact Silver
- 支持 --batch-id
- 支持 --dry-run
- 支持 --limit
- 输出 summary 统计

[7] seed_loader.py 当前已经承担 manifest 目录加载、source_manifest_schema.json 校验、业务规则校验与 dry-run report，并且文件头显式标注“Week03: 接入真实 MinIO 上传 + PostgreSQL 写入”。seed_loader.py

这恰好就是本课最该拿来用的例子。[8]

3.3 tests/contract/test_json_schemas.py：让 contract 先站住

这份测试文件的价值不是“我们有测试”，而是它把下面这些最基本的 contract 约束锁住了：

- 4 类数据契约文件必须存在
- 数据契约必须是合法 JSON
- ticket_contract.json 必须包含关键 required 字段
- 所有 seed manifests 至少要有 manifest_id / modality / assets
- 所有 seed manifests 需要能通过 source_manifest_schema.json 校验[9]

你可以把它理解成：

> **在 Week03 真正谈 ingest 之前，contract 和 manifest 至少要先站住。**

4. 这条主链路里最容易 quietly 出错的地方

4.1 输入记录本身合法，但业务意义已经漂了

比如：

- status 枚举合法，但语义变了
- updated_at 存在，但不是可靠的增量时间
- ticket_id 在，但上游换了生成规则

这类问题往往不会在 schema 层立刻爆红，但后面会慢慢把下游拖歪。

4.2 Bronze 写成了，Silver 却没写全

这是批量 ingest 里很典型的一类事故：

- 原始 payload 已经被收到了
- 但 curated 层没成功 upsert
- 或者某些字段被 quietly 解析成空值

所以 batch 完成之后，不能只看“有没有落盘”，还要看：

- 落在哪一层
- 每层行数对不对

---

[8] ticket_ingest.py 当前已支持 JSONL 输入、JSON Schema 校验、最小业务规则校验、--batch-id、--dry-run、--limit，并将数据写入 raw_ticket_event Bronze 与 ticket_fact Silver。ticket_ingest.py

[9] tests/contract/test_json_schemas.py 当前已校验 4 类数据契约文件存在、JSON 结构合法、ticket_contract.json 关键字段存在，以及 seed manifests 对 source_manifest_schema.json 的符合性。test_json_schemas.py

- 是否和 manifest coverage 一致

4.3 `--dry-run` 通过，不代表真实写入就一定没问题

这节课特别要建立一个判断：

> dry-run 很重要，但 dry-run 不是终局。

dry-run 只能先回答：

- manifest 合法不合法
- schema 通过不通过
- 这批输入大致会怎么走

它还不能完全替代：

- 数据库约束
- upsert 冲突
- 事务失败
- 写入层完整性检查

所以 lesson 2 要做的是：

先用 dry-run 建立可解释性，再用真实写入理解完整性。

5. 动手实践：先跑一次最小 batch smoke flow

这一段请直接在 OmniSupport Copilot 当前 repo 中操作。

第 1 步：启动项目环境

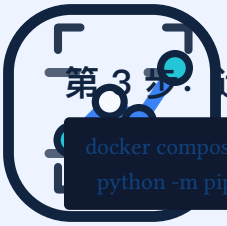
```
docker compose --env-file infra/env/.env.local -f infra/docker-compose.yml up -d --build
```

第 2 步：先跑一遍 contract tests

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

这里的目标不是“庆祝测试全绿”，而是确认：

- 4 类 contract 至少存在且结构合法
- seed manifests 至少满足 schema 约束
- 你接下来要 ingest 的批次，最基本的边界已经站住



第 3 步：运行 manifest 级 dry-run

```
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
```

你在这一步应该重点观察：

- 哪些 manifest 被加载
- 有没有 manifest 被 reject
- dry-run summary 里 success / skip / fail 是怎么统计的

第 4 步：对 ticket 源做一条更具体的 batch 演练

ticket_ingest.py 本身已经给出了 CLI 入口，优先先跑 dry-run，再考虑真实写入：[10]

```
docker compose --profile tools --env-file infra/env/.env.local -f infra/docker-compose.yml run --rm devbox \
  python -m pipelines.ingestion.ticket_ingest \
  --input data/canonization/tickets/tickets-seed-001.jsonl \
  --batch-id batch-20260415-001 \
  --dry-run \
  --limit 20
```

如果你看到 summary 中出现类似下面这些计数，就说明这条 batch 主链路已经开始对你“说人话”了：

- Total
- Valid
- Invalid
- Inserted
- Errors

第 5 步：整理一次 smoke 观察结果

请把你的观察收口到：

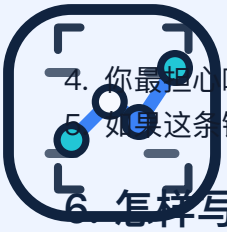
- docs/blueprints/week03/batch_ingestion_design_v1.md
- reports/week03/batch_ingest_smoke_report.md

建议至少写清楚：

1. 这次跑的是哪类源
2. 用的是哪条 manifest / input path
3. dry-run 看到了哪些统计项

---

[10] ticket_ingest.py 当前已支持 JSONL 输入、JSON Schema 校验、最小业务规则校验、--batch-id、--dry-run、--limit，并将数据写入 raw_ticket_event Bronze 与 ticket_fact Silver。ticket_ingest.py



4. 你最担心哪类 quietly fail
5. 如果这条链路要进入 Week04，最先补哪一类检查

6. 怎样写一份最小《batch ingest 设计说明》

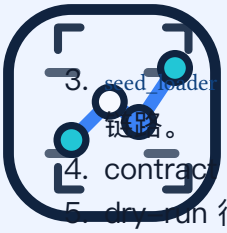
你不需要一上来写设计文档大全。 先把最关键的 5 个问题写清楚就够了。

模板建议

```
# batch_ingestion_design_v1

## 1. 本次 ingest 处理什么
- source:
- modality:
- input path:
- expected target:

## 2. 边界怎么定义
- manifest_id:
- batch_id:
- contract:
- schema_version:

## 3. 幂等如何保证
- primary key:
- on conflict strategy:
- dry-run before write:

## 4. 完整性如何检查
- total:
- valid:
- invalid:
- inserted:
- errors:
- coverage notes:

## 5. 下一步风险
- replay risk:
- backfill risk:
- schema drift risk:
```

7. 本课最重要的 7 个判断

1. Batch 不是落后方案，而是当前课程主线最稳的 ingest baseline。
2. 幂等写入、重跑、完整性校验、reconcile 是四件不同的事。



3. seed_loader 更像 manifest 驱动的入口，ticket_ingest 更像第一条真正的 batch ingest 主链路。
4. contract test 通过，只说明边界站住了，不说明 ingest 一定已经可靠。
5. dry-run 很重要，但不能替代真实写入后的完整性检查。
6. Bronze 写成、Silver 写对、summary 可解释，这三件事必须一起成立。
7. Lesson 2 的真正目标不是“跑一个脚本”，而是为 Lesson 3 的 incremental / cursor / late-arriving data 做铺垫。

8. 本课自检清单

学完这一讲后，你应该能勾掉下面这些项：

- ☐ 我能解释为什么 batch 仍然是 Week03 的最佳入口
- ☐ 我能区分幂等写入、重跑、完整性校验、reconcile
- ☐ 我知道 seed_loader 和 ticket_ingest 分别负责什么
- ☐ 我已经跑过一次 pytest tests/contract/ -v
- ☐ 我已经跑过一次 seed_loader 的 manifest 级 dry-run
- ☐ 我已经跑过一次 ticket_ingest 的 dry-run
- ☐ 我已经写出 batch_ingestion_design_v1.md
- ☐ 我已经写出 batch_ingest_smoke_report.md

9. 课后最小行动

在进入课时 3 前，请至少完成下面这组动作：

1. 再次确认 ticket_contract.json 中 required 字段和你的 batch 理解一致
2. 阅读 seed_loader.py 中 manifest 校验与 IngestResult 的定义
3. 阅读 ticket_ingest.py 中 summary 统计与 Bronze / Silver 写入逻辑
4. 把你今天的 smoke report 补完整

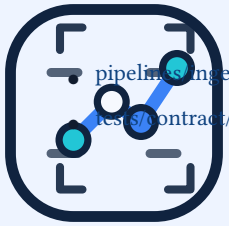
因为下一讲就会开始进入：

- incremental cursor
- watermark
- late-arriving data
- checkpoint / replay

如果这一讲的 batch baseline 没站稳，下一讲会非常痛苦。

延伸阅读

- OmniSupport Copilot README
- pipelines/ingestion/seed_loader.py



- pipelines/ingestion/ticket_ingest.py
- tests/contract/test_json_schemas.py