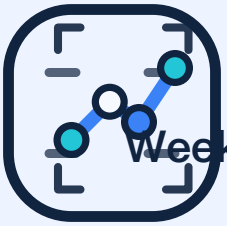


Week02｜课时 5｜契约驱动的采集策略：Manifest、增量窗口、拦截与 Week03 起跑线

Table of contents

Week02 的真正收官，不是写完 contract，而是让 contract 开始驱动 ingest admission

这一讲要把前三课真正翻译成运行闭环：

manifest -> select contract -> schema check -> seed_loader dry-run -> accept / quarantine / reject -> emit run evidence

如果课时 4 解决的是“contract 长什么样”，那么课时 5 要解决的就是：contract 怎么开始真正驱动采集策略，并成为 Week03 的起跑线。

回看课时 4 进入实验 返回 Week02 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

到这里，你已经完成了 Week02 的前三件大事：

- 你盘清了有哪些输入资产值得进入系统
- 你为文档 / 音频 / 视频 / 工单定义了最小元数据与 PII 边界
- 你把这些规则写进了机器可读的 Data Contract

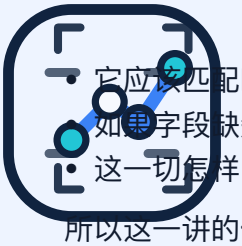
但系统此时还没有真正“跑起来”。

为什么?

因为 **contract 只定义“什么样的数据是合格的”**，
还没有定义 **“这一次到底准备接哪一批数据、用什么窗口接、失败后怎么处理、结果记到哪里”**。

也就是说，你现在还缺 **运行时输入控制面**：

- 这次 ingest 的批次是谁
- 它属于全量、增量、CDC，还是回放 / 补数



- 它应该匹配哪份 contract
- 如果字段缺失、枚举漂移、PII 标记不完整，应该放行、隔离，还是拒收
- 这一切怎样自然接到 Week03 的 ingest / replay / backfill

所以这一讲的任务，是把 Week02 真正收口成：

contract + manifest + loader dry-run + run evidence

参考学习时间（55–70 分钟）

如果你只阅读正文，大约需要 30–40 分钟；如果你跟着本课一起读 manifest schema、补练习版 manifest，并跑一次 Docker devbox 的 `seed_loader` dry-run，建议预留 55–70 分钟。

本课你将完成什么

学完这一讲，你应该能做到：

1. 解释为什么 **Data Contract 不是 ingest plan**。
2. 用一份 `manifest_week02_practice_v1.json` 把一次 ingest 批次声明清楚。
3. 区分 `full_snapshot` / `incremental_cursor` / `cdc` / `replay` / `backfill` 五种采集策略。
4. 设计 **accept / quarantine / reject / warn** 四类门禁动作。
5. 在仓库里跑通一个最小输入起跑线：
   - 先阅读 `source_manifest_schema.json`
   - 再对照现有 manifest
   - 最后用 Docker devbox 跑一次 `seed_loader` dry-run

本课产出

完成这一讲后，你至少应该产出这些工件：

- `data/seed_manifests/source_manifest_schema.json`
- `data/seed_manifests/manifest_tickets_synthetic_v1.json`
- `data/seed_manifests/manifest_edge_gateway_pdf_v1.json`
- `data/seed_manifests/manifest_workspace_helpcenter_v1.json`
- `data/seed_manifests/manifest_week02_practice_v1.json`
- `docs/blueprints/week02/ingest_strategy_v1.md`
- 一次 `seed_loader` dry-run 观察记录

> 到这一讲结束，你手里就不再只是“规则文件”，而是一套真正能指导 Week03 采集与入湖的最小基线。

先看一张总图

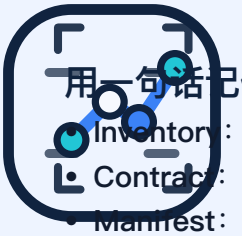
Figure 1: Week02 课时 5 总图

Week02 的真正收官不是“把 contract 写完”，
而是把 contract 变成 **运行时准入机制**。

1. 你现在真正需要区分的 4 个对象

很多同学到这里会把 contract、manifest、ingest plan 混在一起。
最稳的记法是下面这张表。

对象	它回答什么问题	当前 repo 里通常落在哪里	是否会频繁变化
asset inventory	我们到底有哪些数据资产	docs/blueprints/week02/	中
data contract	什么样的数据才算合格	contracts/data/*.json	中低
source manifest	本次到底要接哪一批数据	data/seed_manifests/*.json	高
run evidence	这次 dry-run / seed loader 到底发生了什么	控制台输出、报告、release 记录	高



用一句话记住它们

- **Inventory**：世界地图
- **Contract**：合格标准
- **Manifest**：本次装车清单
- **Run evidence**：通关结果

为什么 manifest 不是文件清单，而是运行时意图

Manifest 最容易被误用成“把路径写出来就算完成”。真实工程里，这样只会留下一个静态目录，不会留下任何可复现的 ingest 语义。

你把 manifest 当成什么	会出什么问题
文件清单	只能列路径，无法复现 ingest 语义
运行时意图声明	才能绑定 contract、窗口、load mode、owner、release

2. Manifest anatomy：每一组字段各守什么门

Phase 2 的统一原则是：

不再把 YAML manifest 当作 Week02 的主实践格式。

这周应该直接围绕当前 repo 已存在的 JSON manifest 体系工作：

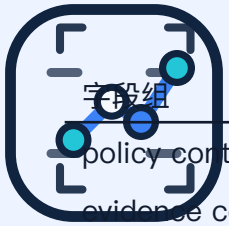
- data/seed_manifests/manifest_tickets_synthetic_v1.json
- data/seed_manifests/manifest_edge_gateway_pdf_v1.json
- data/seed_manifests/manifest_workspace_helpcenter_v1.json
- data/seed_manifests/source_manifest_schema.json

如果需要课程练习版，再新增：

- data/seed_manifests/manifest_week02_practice_v1.json

一份 manifest 里常见的字段组

字段组	它守的门	典型字段
source identity	这批数据到底是谁	source_id, asset_type, owner
location	系统去哪里读它	path, uri, bucket, prefix
contract binding	它应该服从哪份 gate	contract_ref
load semantics	这次是怎么接的	load_mode, cursor_field, window_start, window_end, snapshot_date



字段组	它守的门	典型字段
policy context	运行时边界和版本	pii_level, access_scope, release_id
evidence context	以后怎么追	generated_at, manifest_version, notes

一份练习版 manifest 应该至少解决 6 件事

字段组	作用	没有它会怎样
source_id / asset_type	识别这是什么输入	无法分流到正确 contract
path / uri	知道这一批数据在哪	只能靠脚本硬编码
contract_ref	指向本批次适用的 contract	无法执行统一 gate
load_mode	声明全量 / 增量 / CDC / replay	无法正确规划窗口
window_* / cursor_* / snapshot_*	解释“本次范围”	不能复现这次 ingest
owner / pii_level / release_id	明确责任、风险和版本	后面难审计、难追责

一份最小练习版 JSON manifest 示例

```
{
 "manifest_version": "v1",
 "release_id": "week02-core-r1",
 "generated_at": "2026-04-14T09:00:00Z",
 "sources": [
  {
   "source_id": "ticket_core_daily",
   "asset_type": "ticket",
   "path": "data/seed_manifests/manifest_tickets_synthetic_v1.json",
   "contract_ref": "contracts/data/ticket_contract.json",
   "load_mode": "incremental_cursor",
   "cursor_field": "updated_at",
   "window_start": "2026-04-13T00:00:00Z",
   "window_end": "2026-04-14T00:00:00Z",
   "owner": "support-ops",
   "pii_level": "restricted"
  },
  {
   "source_id": "kb_manual_docs",
   "asset_type": "document",
```

```
    "path": "data/seed_manifests/manifest_edge_gateway_pdf_v1.json",
    "contract_ref": "contracts/data/doc_asset_contract.json",
    "load_mode": "full_snapshot",
    "snapshot_date": "2026-04-14",
    "owner": "knowledge-platform",
    "pii_level": "public"
  }
 ]
}
```

> ℹ 记住这个关系
>
> - **Contract** 约束记录长什么样
> - **Manifest** 约束批次怎么进来
> - **Loader / gate** 决定这批数据现在能不能进

manifest 字段组如何被 loader / gate 消费

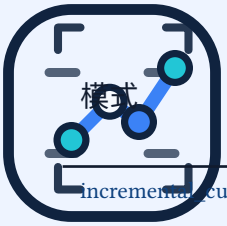
字段组	loader / gate 会拿它干什么
source identity	路由到正确输入类型
location	找到数据
contract binding	加载对应 gate
load semantics	规划窗口与状态逻辑
policy context	约束边界
evidence context	记录 run 证据

3. 五种采集模式不要死记，背后是五种业务语义

站在 AI 数据工程视角里，这些模式其实是在回答：

这一批数据与上一批数据到底是什么关系。

模式	你在 manifest 里至少要声明什么	它对应的业务语义	最常见风险
full_snapshot	snapshot_date	这次提供的是某个时点的完整世界	成本高、重复索引



模式	你在 manifest 里至少要声明什么	它对应的业务语义	最常见风险
incremental_cursor	cursor_field + window_start + window_end	只要最近变化过的对象	漏数、重数、时区错误
cdc	checkpoint_field / cdc_cursor	我们关心事件流而不是静态表	配置复杂、补数策略难
replay	replay_from_manifest / replay_reason	不是新数据，是重跑旧批次	重复写入、版本混淆
backfill	backfill_range / reason	不是在线变化，是补历史空洞	影响范围过大、资源争抢

别背模式，先问 3 个问题

1. 这一批和上一批到底是什么关系？
2. 这一次需要覆盖完整世界，还是只接变化部分？
3. 如果失败，后面应该重放、补数，还是继续追增量？

4. 运行时门禁不是二元判断，而是 4 类动作

到这里，你已经有两样东西：

- contract
- manifest

剩下的就是把它们真正接到一起。

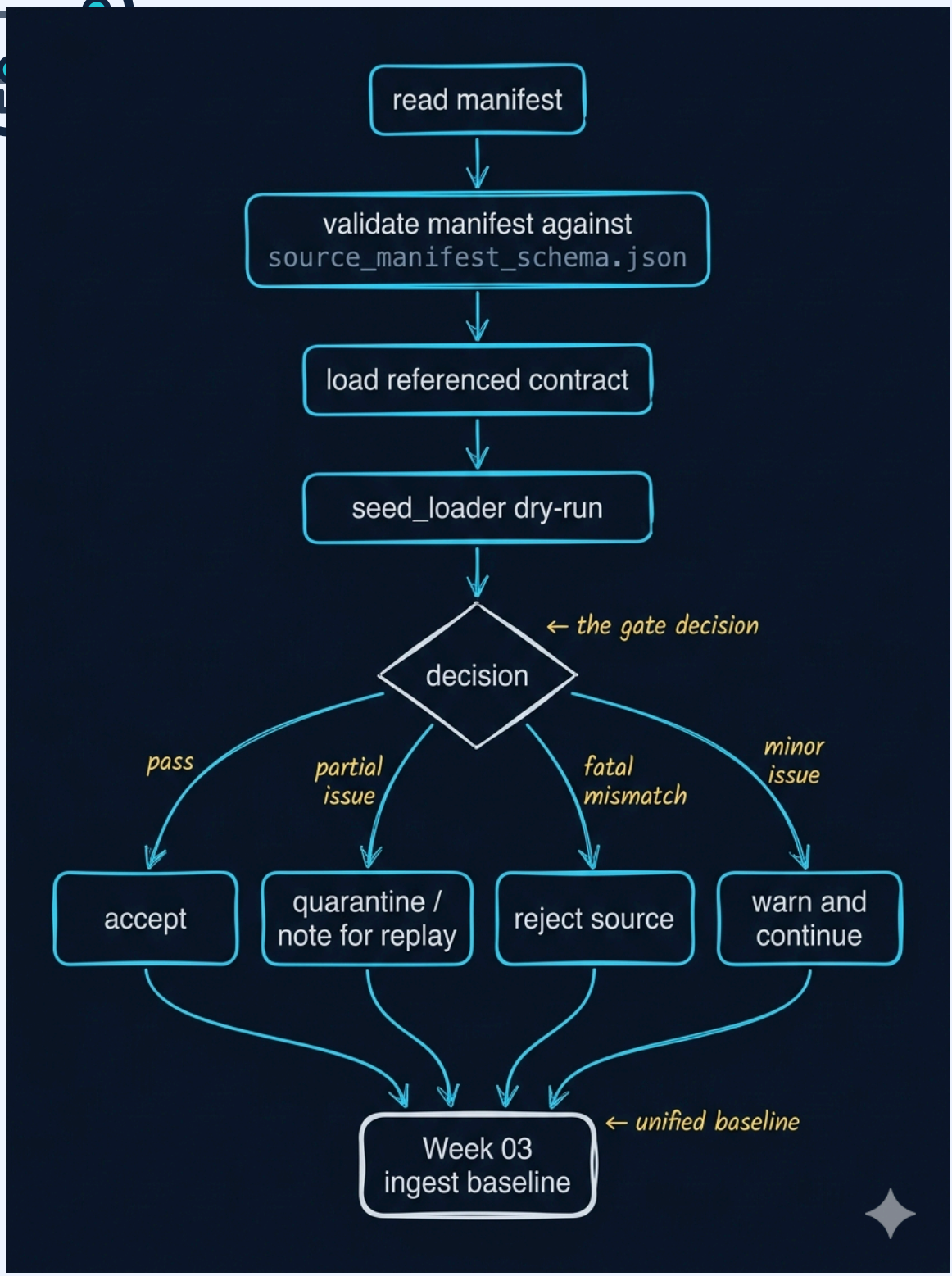
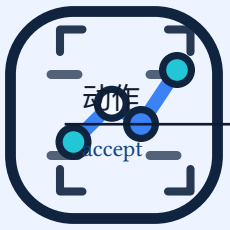
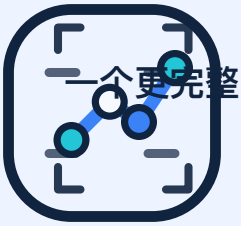


Figure 2: Week02 运行时门禁四类动作图



动作	含义	什么时候用
accept	记录合格，可以进入下一层	字段、元数据、PII、枚举全部通过
quarantine	先隔离，暂不进主链路	个别记录坏了，但整批还有可用部分
reject	整个 source 不接收	manifest 严重错误、contract 不匹配、关键字段缺失
warn	暂时放行，但必须记日志	可容忍问题，例如非关键描述字段缺失



一个更完整的决策树

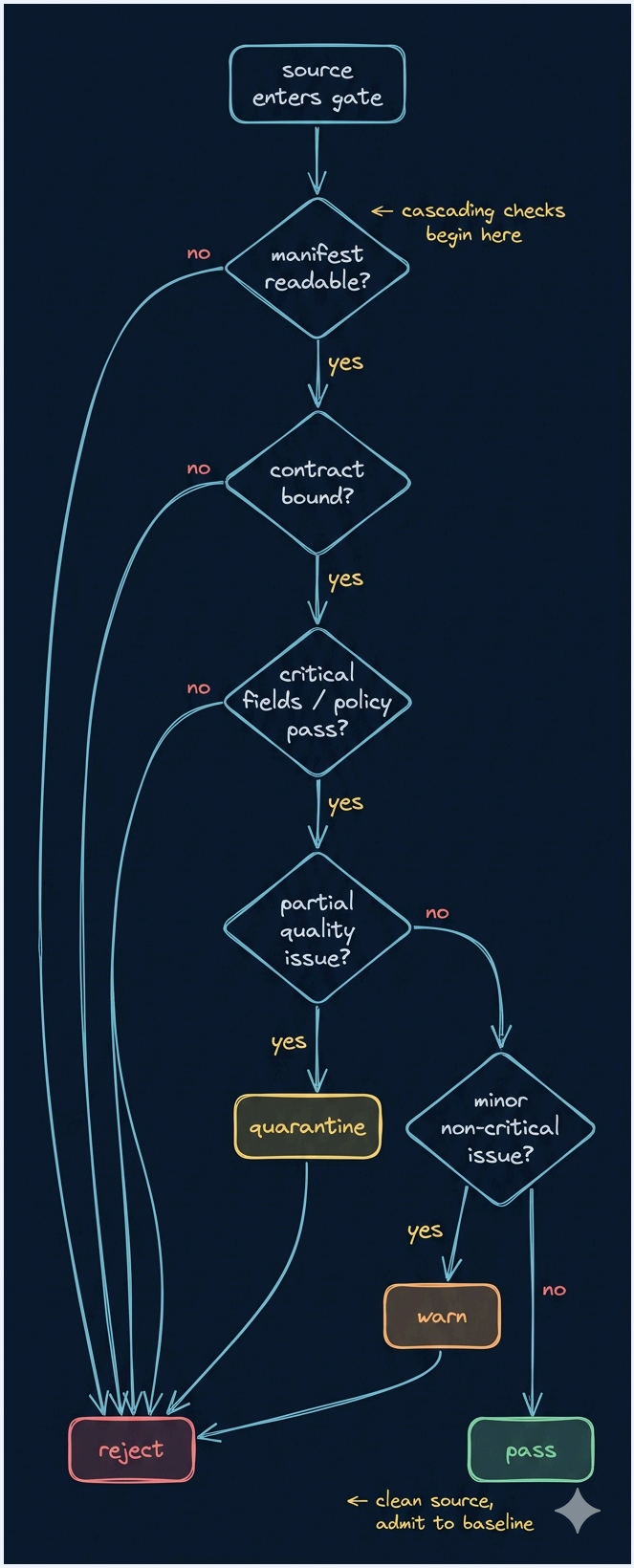


Figure 3: Week02 gate 决策树

Week02 的目标不是把所有坏数据拒之门外，而是学会把问题分层处理。

5. 一次 dry-run 报告到底应该怎么读

很多同学第一次跑 dry-run 时，只看“有没有报错”。这不够。

你至少应该从输出里读出下面 4 层信息：

你应该看什么	它说明什么
哪个 manifest 被读取	本次批次边界是否明确
它引用了哪份 contract	source 和准入标准是否闭环
哪些 source 被 accept / quarantine / reject	运行时门禁是否开始发挥作用
有没有 release_id / source_id / owner 这类证据字段	后续追踪和 Week03 衔接是否可能

典型阅读顺序

1. 先看 manifest schema 是否通过
2. 再看 source 是否绑定到了正确 contract
3. 再看 gate judgment 是 accept / quarantine / reject / warn 哪一种
4. 最后看这一轮输出能否沉淀成 run evidence

dry-run 报告怎么读，才能接上 Week03

你在报告里看到什么	Week03 会怎么继续消费它
manifest 被读到	批次存在性成立，说明 state 有起点
contract 绑定成功	admission 前提成立，说明 gate 可复用
某 source 被 quarantine	需要 patch、replay 或补 metadata
某 source 被 reject	不能进入 ingest 主链，必须先修准入问题
window 声明不清	state / watermark 无法建立

6. 直接动手：把 JSON manifest 和 Docker-first loader 跑起来

第 1 步：先看 schema，再看 manifest

先按这个顺序阅读：

```
data/seed_manifests/source_manifest_schema.json
data/seed_manifests/manifest_tickets_synthetic_v1.json
data/seed_manifests/manifest_edge_gateway_pdf_v1.json
data/seed_manifests/manifest_workspace_helpcenter_v1.json
```

带着下面几个问题去看：

- source_id / asset_type / contract_ref 有没有闭环
- load_mode 需不需要 cursor_field / window_* / snapshot_date
- 这次 ingest 的责任人和 release 边界有没有写清

第 2 步：补一份课程练习版 manifest

```
touch data/seed_manifests/manifest_week02_practice_v1.json
touch docs/blueprints/week02/ingest_strategy_v1.md
```

建议你在 docs/blueprints/week02/ingest_strategy_v1.md 里记录：

- 为什么某个 source 选 full_snapshot
- 为什么某个 source 选 incremental_cursor
- 哪种 source 如果 contract 不匹配必须 reject
- 哪种 source 可以先 quarantine

第 3 步：统一用 Docker devbox 跑 seed loader 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
```

这条命令跑完，你至少应该能解释什么

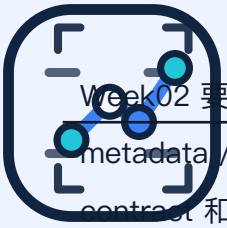
- loader 这次读取了哪些 manifest
- 哪些 source 属于“批次语义不清”
- 哪些 source 就算 schema 没坏，也会因为 contract / metadata / PII 问题被拦

7. Week02 为什么只做到这里，不直接把 Week03 也做完

这是一个非常容易误解的点。

Week02 到这里停下，不是因为课程没讲完，而是因为工程边界本来就该这样切：

Week02 要解决什么	Week03 再解决什么
输入资产和值得接入的 source	真正的 batch / incremental ingestion 实现



Week02 要解决什么	Week03 再解决什么
metadata / PII 最低标准	幂等、补数、回放、失败恢复
contract 和 manifest 的准入门槛	真正的数据搬运与入湖
dry-run 与 run evidence 的起点	运行时调度与长期运维

也就是说，Week02 结束时你拿到的不是“完整 ingest 系统”，而是：

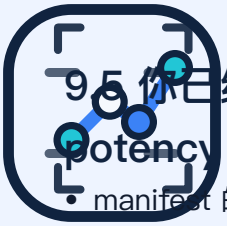
一个可信的 ingest baseline。

8. 本课最容易误解的 5 件事

误解	正确理解
manifest 只是列几个文件路径	manifest 是一次 ingest 批次的运行时声明
只要 contract 正确，manifest 随便写	manifest 决定的是“这一批怎么进来”
dry-run 通过就说明 Week03 全部完成	dry-run 只是起跑线，不是终局
quarantine 和 reject 差不多	quarantine 代表可隔离观察，reject 代表整批不能接
Week02 应该把 ingest 做完	Week02 的职责是把 ingest baseline 和 run evidence 建起来

9. Week02 工件 -> Week03 会继续消费什么

Week02 工件	Week03 会继续消费什么
asset_inventory_v1.csv	决定哪些 source 真正进入 ingest 计划
metadata_minimums_v1.md	决定 parse / normalize 后哪些字段必须保留
pii_policy_matrix_v1.csv	决定 redact、filter、tool boundary
contracts/data/*.json	决定记录级 gate 和 schema 演进判断
manifest_week02_practice_v1.json	决定本次 batch / incremental / replay 的入口
seed_loader dry-run 记录	成为 Week03 run evidence 和排障参照



9.5 你已经走到 Week03 的门口了：state、watermark、idempotency 从哪里长出来

- manifest 的 load_mode 会直接决定 Week03 使用哪种状态模型。
- contract 的 shape 和 semantics 会决定幂等写入与去重到底能不能成立。
- gate 的 accept / quarantine / reject 会决定 replay、backfill 和补丁修复的边界。

也就是说，Week03 不是从零开始长出 ingest runtime，而是继续消费 Week02 已经定义好的准入规则。

10. 小结

这一讲最重要的，不是记住 5 种采集模式的名字，而是建立这条判断：

contract 解决“什么数据算合格”，manifest 解决“这次到底接哪一批”，gate 解决“现在能不能放行”，run evidence 解决“以后怎么追”。

只有这四件事连起来，Week03 才有一个稳定起跑线。

11. 课后最小行动

做完这一讲，请至少完成下面三件事：

- ☐ 阅读 source_manifest_schema.json 和 3 个现有 manifest
- ☐ 创建或补齐 manifest_week02_practice_v1.json
- ☐ 跑一次 Docker devbox 的 seed_loader dry-run，并写下你看到的门禁判断

12. 下一步衔接

下一步不是再讨论 manifest 理论，而是把这些 Week02 工件真正用到实验页和 Week03 的 ingest 起跑线里。