DISTRIBUTED SHARED MEMORY

Head First DSM

从共享状态到可验证的一致性边界

26 章 · 6 条学习弧 · 1 条贯穿案例

中文版 · DSM 0.1.2

LeanowTech

源码基线：DSM 505e755 · examples 0eca40d

目录

前言：DSM 的价值与本书学习路径

分布式系统中，一种常见风险是把某种局部保证误当成另一种全局保证。例如，本地写入成功并不表示远端已经可见，副本最终收敛也不表示已发生的业务响应可以撤销。

DSM 是一套面向 Java 应用的分布式共享内存基础库。它用于在多个服务实例之间共享路由提示、分片所有权和可合并计数等协同状态，并把状态标识、类型化集合、本地操作、在线传播、冲突处理和副本修复放入统一 Runtime。业务团队因此可以复用经过定义和验证的协调机制，减少在各个服务中重复实现传播、合并与修复逻辑的工作。

这项价值有明确边界。最终收敛不提供事务一致性，副本修复不提供完整业务事件重放。订单、库存和支付仍需要各自的权威事实系统；应用团队仍需定义业务不变量、分区期间的处理方式、外部 fencing 或幂等保护，以及部署环境中的安全和容量标准。

本书使用「分布式订单履约控制面」贯穿全程：

- `route-hints` Register 保存服务路由提示。
- `shard-owners` Lease 管理分片所有权。
- `request-counter` CRDT 合并多节点请求计数。

全书分为六个连续阶段：第一篇识别适合共享的状态并建立 Runtime；第二篇依据业务不变量选择 Register、Lease 或 CRDT；第三篇展开在线传播、漏失、修复和分区边界；第四篇通过领域适配器与 Spring Boot 接入真实工程；第五篇补齐可观测性、安全、生命周期和容量责任；第六篇使用真实端口、跨集群实验和毕业项目复核整体结论。

每一章先描述问题，再引入对应机制。实验验收卡记录运行命令、可观察结果、证据等级和未覆盖范围。完成全书后，开发者应能从业务状态出发选择或拒绝 DSM，并能通过代码、故障实验和运行证据说明选择依据。这个过程把 DSM 降低一致性工程成本的价值，落实为可执行、可诊断和可复核的工程能力。

从第 1 章开始，先判断哪些状态应该进入 DSM。

LEARNING ARC

先看见共享状态

第 1–4 章

先分清业务事实、协同状态与事件历史，再启动第一个 *Runtime* 并观察第二个节点。

第 1 章——DSM 的职责与适用边界

本章目标：本章完成后，开发者能够根据数据责任和故障恢复方式，判断一个状态是否适合放入 DSM。

学习目标

1. 用一句话说明 DSM 解决的协同问题。
2. 区分业务事实、协同状态和事件历史。
3. 解释为什么「可以从同伴修复」不是「可以替代数据库」。
4. 对订单履约案例中的四类数据作出初步存储选择。

前置条件

- 能阅读基本 Java 接口。
- 知道数据库事务和服务多实例部署的基本含义。
- 本章不要求运行 DSM，也不要求理解复制协议。

案例进度

履约系统有多个 API 和 worker 实例。它们需要共享服务路由、订单分片所有权和请求计数，但订单、库存和支付已经有各自的事实系统。

本章只确定责任边界。第 2 章才会创建第一个 Runtime。

DSM 解决的协同问题

先看一个常见但危险的需求：

> 某团队认为 DSM 能把数据复制到多个节点，因此计划把订单状态迁入 DSM 并移除数据库。

问题不在于 DSM 能不能保存一个 `Order` 对象，而在于订单状态承担什么责任。订单是需要事务约束、审计历史和确定写入结果的业务事实。副本在网络恢复后得到相同可见值，不能自动补回一次丢失的扣款，也不能撤销已经返回给客户的成功响应。

DSM 适合另一类数据：体量较小、服务运行所需、能够从同伴修复或从权威来源重新生成的协同状态。

按数据责任划分存储边界

图中有三类责任：

- 事务数据库保存订单、库存和支付等业务事实。
- DSM Runtime 保存路由、租约和可合并计数等协同状态。
- 消息或事件系统保存需要可靠传递和历史重放的业务事件。

同一个应用可以同时使用三类系统。选择 DSM 不等于排斥数据库或消息系统。

从公开契约看 DSM

当前 `DsmRuntime` 是承载多个类型化集合的公开门面。它公开三种注册入口：

```
<E extends DsmEntity<E>> DsmRegister<E> register(CollectionSpec<E> spec);

<E extends LeaseEntity<E>> DsmLeaseRegister<E> leaseRegister(
        LeaseCollectionSpec<E> spec);

<E extends DsmEntity<E>, S extends MergeableState<S>>
DsmCrdtCollection<E, S> crdt(
        CrdtCollectionSpec<E, S> spec,
        StateMerger<E, S> merger);
```

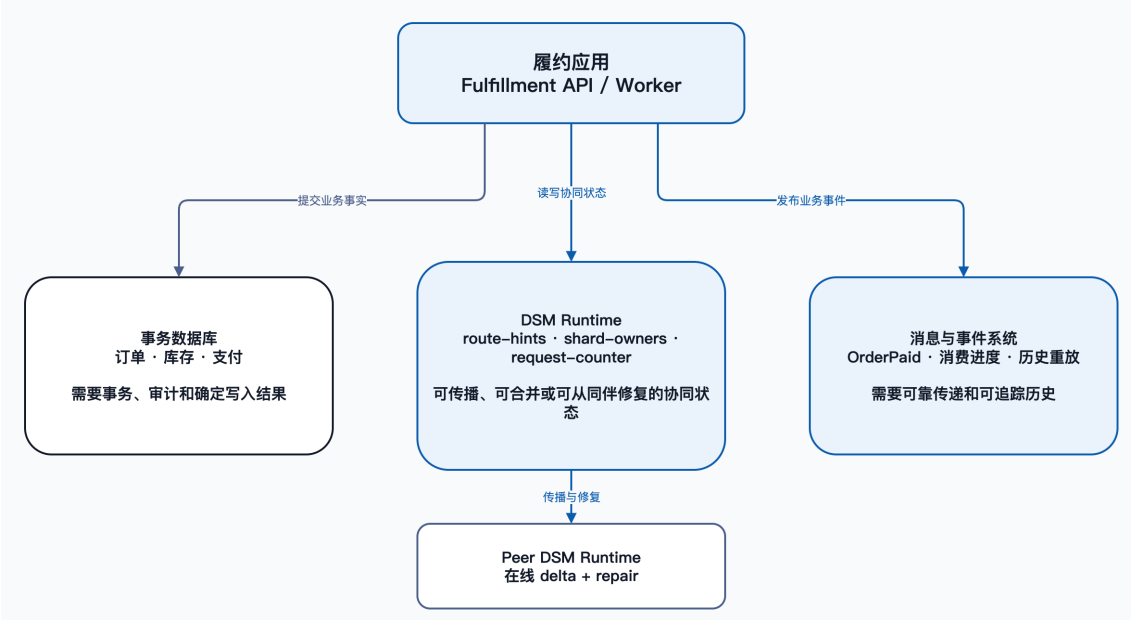

图 1: 订单履约中的数据责任边界

这三个入口代表三种不同问题，而不是三个容量档位：

问题	首选集合	本书案例
一个 key 当前应该看到哪个逻辑值	Register	`route-hints`
当前谁拥有一项有期限的工作权	Lease	`shard-owners`
多个节点的本地更新怎样合并	CRDT	`request-counter`

第 5—9 章会逐一建立这些语义。当前阶段先识别业务不变量，再选择集合。

第一个可验证增量：给状态分类

本章的资产提供了 8 个候选状态。打开 `classification-cases.json`，在查看参考答案前逐项填写：

- `transactional-fact`：需要事务与审计的业务事实。
- `coordination-state`：可修复或可重建的运行协同状态。
- `event-history`：需要可靠传递或历史重放的事件。
- `analytical-data`：面向大规模查询和分析的数据。

运行检查：

```
npm test -- --test-name-pattern="chapter 01 classification"
```

这个命令只验证分类资产和参考答案，不运行 DSM。

拆开来看

1. 业务事实的正确性依赖提交历史

订单从 `PENDING` 变成 `PAID`，通常还伴随支付记录、库存变更和审计信息。系统需要回答「哪次事务成功」「谁修改了什么」「失败后如何补偿」。DSM 当前集合 API 不是跨资源事务协议。

因此，订单主记录、库存扣减和支付账务留在事务系统。

2. 协同状态服务于运行决策

Route hint 告诉网关当前访问哪个履约实例。它可能短暂过期，也可以由健康检查重新发布。节点漏掉一次在线 delta 时，repair 可以让副本追上当前状态。

这种状态适合 DSM，但仍要写清陈旧窗口和失败后的业务行为。

3. 事件历史需要传递保证

`ChangeStream` 可以让工具或应用观察集合变化，但它不是持久消息日志。需要重放全部 `OrderPaid` 事件时，应使用消息系统、outbox 或事件存储。

4. 大规模分析需要不同的数据模型

DSM 集合以 key 和协同语义为中心，不提供面向大量数据的 join、二级索引和分析查询。高容量日志和指标应该进入相应的数据管道。

反例与故障注入

假设把库存余量放入 Register：

1. 节点 A 读取余量为 1，并返回扣减成功。
2. 网络分区期间，节点 B 也读取余量为 1，并返回扣减成功。
3. 分区恢复后，Register 通过确定性规则只保留一个可见值。

副本最后可以收敛，但两个客户已经收到成功响应。收敛没有撤销第二次销售，也没有产生一条可审计的库存事务。

这个反例说明，DSM 适用性取决于业务响应能否撤销、是否需要事务历史；对象能否序列化只是技术条件之一。

为什么会这样

Register 的责任是让候选状态得到确定的可见结果。它不协调多个业务资源，也不替调用方撤回已经产生的外部效果。

后续章节会解释本地提交、在线 delta 和 repair。现在先保留一句边界：

> repair 修复副本状态，不修复已经对外发生的业务承诺。

调整后的状态模型

把「库存余量」改成「库存服务当前首选路由」。重新判断：

1. 丢失后能否从健康检查重新生成？
2. 短时间读到旧值时，调用方能否重试或切换？
3. 是否需要查询完整修改历史？
4. 多节点并发发布时，能否接受确定性 winner？

如果答案分别是「能、能、不需要、能」，它比库存余量更接近 DSM Register 的适用条件。

常见误区

把「内存」理解成单机缓存

DSM 的共享状态由 Runtime 管理，并可通过 sync 和 repair 在节点间传播。它不是把一个 `Map` 暴露给所有 JVM，也不是简单的本地缓存封装。

把「多个副本」理解成数据永不丢失

数据安全仍取决于持久化配置、节点拓扑、故障相关性、备份和恢复流程。多个内存副本不能替代数据库备份。

先选集合，再找业务理由

「CRDT 看起来最先进」不是选择依据。没有并发本地更新和可合并状态时，CRDT 只会增加模型与验证成本。

理解检查

1. 服务发现地址适合 Register 的关键条件是什么？
2. 为什么一个最终收敛的库存值仍可能造成超卖？
3. 如果数据需要支持按客户、时间和状态组合查询，DSM 是否应该是主存储？为什么？
4. 一个 worker 的当前心跳适合 DSM，不代表 worker 完成过的全部任务历史也适合 DSM。两者的责任差异是什么？

实验

将以下状态分成业务事实、协同状态、事件历史和分析数据：

- 订单当前状态。
- 履约服务路由提示。
- 分片 owner 与租约到期时间。
- 各节点请求计数。
- `OrderPaid` 事件历史。
- 支付账务分录。
- 调用延迟直方图。
- 临时功能开关。

完成后与 参考答案 比较。参考答案不是机械分类：临时功能开关是否适合 DSM，还取决于它是否有外部事实源、陈旧窗口和回滚要求。

实验验收卡

字段	内容
运行命令	`npm test -- --test-name-pattern="chapter 01 classification"`
输入或故障	8 个候选状态及其责任说明
可观察结果	分类资产结构有效；参考答案覆盖全部候选状态
证据等级	E1：只验证书籍资产与静态责任边界
本实验未证明	DSM Runtime 可以启动、节点可以复制、任何生产容量或可用性结论

回顾

- DSM 保存分布式协同状态，不替代所有共享数据系统。
- 业务事实、协同状态、事件历史和分析数据需要不同的保证。
- Register、Lease 和 CRDT 分别回答最新值、所有权和可合并更新问题。
- repair 修复副本状态，不撤销已经发生的业务响应。

下一步

第 2 章将启动一个最小 `DsmRuntime`，创建 `route-hints` Register，并第一次写入和读取路由提示。

证据链接

- `DsmRuntime`
- `DsmRegister`
- `DsmCrdtCollection`
- DSM 项目约束
- Phase 0 基线验证

第 2 章——启动第一个 Runtime

本章目标：本章完成后，开发者能够构建一个单节点 DsmRuntime，注册 route-hints，并准确解释本地写入成功的含义。

学习目标

1. 识别 Runtime、membership 和 collection spec 的职责。
2. 按正确顺序完成 build、register、start、close。
3. 写入并读取第一个 RouteHint。
4. 区分本地 API 成功与远端可见。

前置条件

- 已完成第 1 章的数据责任分类。
- 本地有 JDK 25；示例仓库已包含 Maven Wrapper。

案例进度

履约网关现在需要发布 fulfillment-primary → 10.0.0.8:8080。本章只让一个节点保存并读取该提示，不讨论网络传播。

四步生命周期

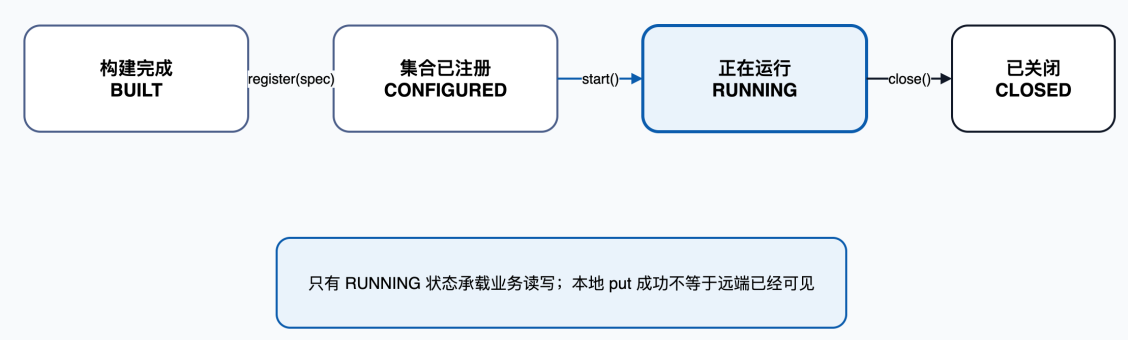

图 2: 第一个 Runtime 的生命周期

DsmRuntimeBuilder 负责组装运行时；ClusterMembership 提供节点身份和通信；CollectionSpec 给集合稳定的定位与 schema；DsmRegister 才是业务读写句柄。

```
NodeInfo self = new NodeInfo("gateway-a", "127.0.0.1", 19090);
DsmRuntime runtime = DsmRuntimeBuilder.builder()
        .clusterId("fulfillment-local")
        .serviceId("gateway")
        .membership(new StandaloneClusterMembership(self))
        .build();

CollectionSpec<RouteHint> routes =
        CollectionSpecBuilder.<RouteHint>register("demo", "gateway", "route-hints")
                .schemaId("route-hints/v1")
                .recordCodec(RouteHint.class)
                .build();
DsmRegister<RouteHint> register = runtime.register(routes);
runtime.start();
```

先注册集合再启动，使 Runtime 在进入 RUNNING 时已经知道集合 schema。使用完毕调用 close()；示例和测试应放在 try/finally 或 try-with-resources 等价结构中。

第一个本地提交

```
register.put(new RouteHint(
        "fulfillment-primary",
        EntityMetadata.empty(),
        "10.0.0.8:8080"));

String address = register.get("fulfillment-primary")
        .orElseThrow()
        .address();
```

put 返回表示当前节点已接受本次本地变更。它没有返回复制确认数量，也没有承诺另一个节点此刻已经读到。单节点 membership 更不会凭空产生远端副本。

反例与故障注入

尝试在 runtime.start() 之前把 Runtime 当成已经对外服务，或忘记 close()。前者混淆配置完成与可运行状态；后者让测试泄漏线程和资源。另一个常见错误是看到本地 get 成功就宣称「集群已经一致」。

实验

运行上游 Runtime/REPL 示例测试：

```
cd submodule/dsm-examples/basic-usage-examples
./mvnw -q -Dtest=DsmReplTest test
```

观察命令处理器可以存储、列出条目，同时保留测试中显式创建和关闭 Runtime 的边界。该命令使用受控的 fake peers，因此成功结果仅表示单进程路径可用，不构成真实网络验收。

实验验收卡

字段	内容
运行命令	`cd submodule/dsm-examples/basic-usage-examples && ./mvnw -q -Dtest=DsmReplTest test`
输入或故障	单节点读写；测试内受控 peer
可观察结果	DsmReplTest 的 3 个测试通过；Runtime 被显式关闭
证据等级	E2：运行固定源码的进程内测试
本实验未证明	真实网络、生产发现机制、远端写入确认和容量

回顾

- Runtime 是集合和节点协作的生命周期边界。
- 集合 spec 在 start() 前注册，业务通过类型化 handle 操作。
- 本地 put 成功只证明本地提交，不等于远端已可见。
- 示例需要关闭 Runtime，避免把资源泄漏藏在教程后面。

下一步

第 3 章把 RouteHint 拆成 entry identity、实体字段、metadata、codec 和 schema，让传播双方谈论同一份状态。

证据链接

- DsmRuntimeBuilder
- RuntimeExampleFactory
- DsmReplTest
- Phase 0 基线验证

第 3 章——共享状态的稳定身份

本章目标：本章完成后，开发者能够定义跨节点可识别、可编码且可演进的 `RouteHint`。

学习目标

1. 区分 collection locator 与 entry key。
2. 解释 `DsmEntity`、`EntityMetadata` 和 codec 的协作。
3. 用 `schemaId` 管理线格式兼容边界。
4. 识别身份漂移与 schema mismatch。

前置条件

- 能启动第 2 章的单节点 Runtime。
- 理解 Java record 和序列化的基本概念。

案例进度

第 2 章只写入了一个地址。本章为它补上稳定身份：集合是 `demo/gateway/route-hints`，条目 key 是服务名，实体值才是地址和健康提示。

五层身份

图 3: 共享状态的稳定身份

```
public record RouteHint(
        String entryKey,
        EntityMetadata metadata,
        String address)
        implements DsmEntity<RouteHint> {

    @Override
    public RouteHint withMetadata(EntityMetadata next) {
        return new RouteHint(entryKey, next, address);
```

```
    }
}
```

`entryKey` 是集合内部身份，需要在实体内容变化时保持不变。`metadata` 保存 lineage、HLC、epoch、fencing token 和可选 expiry 等运行时信息。业务不应自行伪造这些字段来赢得冲突。

集合 locator 由 `tenantId/applicationId/collectionId` 组成。同名 key 放在不同 locator 下，是两个相互隔离的条目。`clusterId/serviceId` 属于运行与通信域，不能替代集合 locator；第 17 章会验证这两层隔离。

codec 与 schema 不是装饰

两端需要把相同字节解释成相同实体。`recordCodec(RouteHint.class)` 提供记录编码，而 `schemaId("route-hints/v1")` 把线格式意图带入集合契约。若一端把 `address` 当字符串，另一端悄悄改成完全不同的结构，正确行为是拒绝不兼容数据，而不是猜测转换。

安全演进通常遵循：先发布能读取旧/新格式的版本，再切换写格式，最后清理旧读取路径。仅修改 Java 类名不等于完成分布式 schema 迁移。

反例与故障注入

准备两个同 locator、同 `schemaId`，但 codec 含义不同的节点。即使注册成功，也可能在远端 decode 时失败。反过来，schema 指纹不一致被拒绝，是保护而不是可用性缺陷：它阻止错误数据进入可见状态。

另一个错误是用随机 UUID 作为每次发布的 `entryKey`。这样每次健康更新都会新增条目，Register 永远没有机会替换同一个逻辑服务。

实验

检查本章身份卡，并运行固定的 schema mismatch 集成测试：

```
node --test tests/chapter-assets.test.mjs
cd submodule/dsm
mvn -q -pl dsm-integration-test -am \
  -Dtest=TwoNodeIntegrationTest#schemaFingerPrintMismatchIsRejectedGracefullyBetweenTwoNodes \
  -Dsurefire.failIfNoSpecifiedTests=false test
```

实验先证明书中 locator/key/schema 卡片完整，再证明两个节点对不兼容 schema 采取显式拒绝。

实验验收卡

字段	内容
运行命令	`node --test tests/chapter-assets.test.mjs`；聚焦 Maven 命令见上文
输入或故障	同一路由集合上的 schema fingerprint 不匹配
可观察结果	章节资产有效；不兼容复制被拒绝且测试通过
证据等级	E3：双节点受控集成测试
本实验未证明	自动 schema 迁移、跨版本滚动升级和生产数据修复

回顾

- locator 识别集合，entry key 识别集合中的逻辑条目。
- 实体业务字段与 Runtime metadata 各负其责。
- codec 决定字节含义，schemaId/fingerprint 保护兼容边界。
- 身份漂移会制造新状态，schema 漂移应 fail closed。

下一步

第 4 章连接第二个节点，观察同一个 `RouteHint` 从本地提交变成远端可见。

证据链接

- `DsmEntity`

- `EntityMetadata`
- `CollectionSpecBuilder`
- `TwoNodeIntegrationTest`

第 4 章——从本地提交到远端可见

本章目标：本章完成后，开发者能够运行双节点复制实验，并用事件来源证明远端可见发生在本地提交之后。

学习目标

1. 连接两个 membership 并为两端注册相同 spec。
2. 跟踪 local put、delta 和 remote apply。
3. 用等待条件代替固定 `sleep` 验证最终可见。
4. 说明进程内双节点测试的证据边界。

前置条件

- 已完成第 2—3 章。
- 能区分 locator、entry key 和 schema。

案例进度

网关节点 A 发布 `fulfillment-primary`；节点 B 在自己的 Register 中最终读到相同地址。订单业务事实仍未进入 DSM。

从本地提交到远端应用

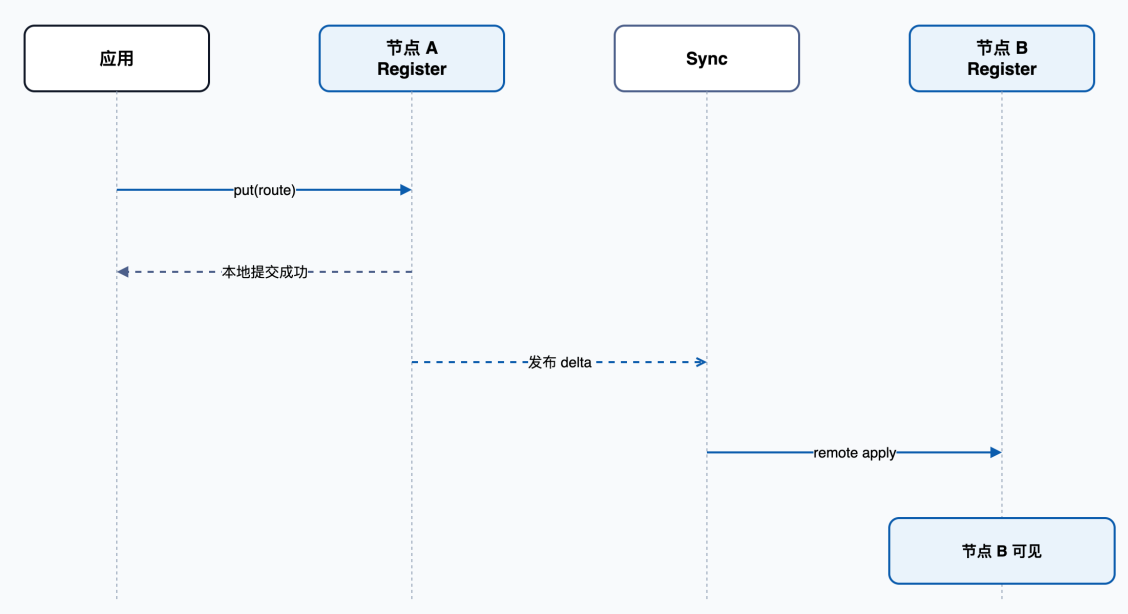

图 4: 一次更新到达第二个节点

两端需要相同的集群/服务通信域、相同的集合 locator、兼容 schema，并让 sync service 把集合挂到路由表。顺序很重要：本地 `put` 可以先返回，远端稍后才处理 `REMOTE_DELTA`。

```
registerA.put(new RouteHint("fulfillment-primary", metadata, "10.0.0.8:8080"));
await(() -> registerB.get("fulfillment-primary").isPresent(), 2_000);
assertEquals("10.0.0.8:8080",
        registerB.get("fulfillment-primary").orElseThrow().address());
```

等待条件表达实际验收目标。固定 `Thread.sleep(500)` 既可能太短导致偶发失败，也可能掩盖复制已经异常缓慢。

让事件说明来源

`ChangeStream` 的 `ChangeOrigin.REMOTE_DELTA` 能区分远端应用与本地写入。测试同时断言可见值和来源，比只轮询 `get` 更接近协议事实。但 ChangeStream 不是持久消息日志；订阅者离线期间的完整历史不能靠它保证。

反例与故障注入

只在节点 A 注册集合，或让节点 B 使用不同 `schemaId`。此时「membership 已连接」不代表数据会被正确应用。复制路径包含集合路由与 schema 检查，任何一环缺失都应显式暴露，而不是把超时归咎于「最终一致就是慢」。

实验

运行聚焦双节点测试：

```
cd submodule/dsm
mvn -q -pl dsm-integration-test -am \
  -Dtest=TwoNodeIntegrationTest#runtimeFirstRegisterDeltasReplicateAcrossTwoNodes \
  -Dsurefire.failIfNoSpecifiedTests=false test
```

该测试覆盖 A→B put、A→B remove 和 B→A put，并断言远端事件来源。

实验验收卡

字段	内容
运行命令	上述 `TwoNodeIntegrationTest` 聚焦命令
输入或故障	两个 fake membership 节点；双向写入和删除
可观察结果	对端在 2 秒条件窗口内可见；事件为 `REMOTE_DELTA`
证据等级	E3：受控双节点协议集成测试
本实验未证明	跨进程网络、丢包修复、生产发现和线性一致

回顾

- 本地提交与远端可见是两个时刻。
- 相同 membership 还不够，集合路由和 schema 也需要匹配。
- 最终条件应通过轮询目标状态验证，不使用拍脑袋 `sleep`。
- `REMOTE_DELTA` 证明在线传播来源，但不提供事件日志保证。

下一步

第 5 章回到 Register 本身，建立 put、get、all、remove 和本地可见性的完整契约。

证据链接

- `TwoNodeIntegrationTest`
- `RuntimePlatformSyncService`
- `ChangeOrigin`

第一篇回顾——先看见共享状态（第 1-4 章）

第一篇从业务责任边界推进到双节点可见，尚未覆盖并发冲突、repair 或生产网络。

本篇形成的认识

- 第 1 章先把订单、支付和事件历史留在各自事实源，只把可修复协同状态交给 DSM。
- 第 2 章建立 Runtime 生命周期，并把本地 `put` 成功限定为本地提交。
- 第 3 章用 locator、entry key、entity、metadata、codec 和 schema 建立稳定身份。
- 第 4 章观察在线 delta，让节点 B 在本地提交之后最终读到节点 A 的 RouteHint。

可验证的学习结果

- 判断一个状态是否具备「可从同伴修复或从权威源重建」的前提。
- 构建单节点 Runtime，注册并关闭集合。
- 为 RouteHint 选择稳定 key 和明确 schema。
- 用条件等待和 `REMOTE_DELTA` 证明受控双节点传播。

本篇如何兑现 DSM 价值

第一篇把“共享状态”落实为可检查的责任边界、稳定身份、Runtime 生命周期和远端可见性。业务服务可以复用集合注册、状态编码和基础传播入口，同时仍需决定哪些状态适合共享，以及本地成功后何时可以依赖远端结果。

四个容易混淆的时刻

时刻	已经证明	还没有证明
`runtime.start()` 返回	本地 Runtime 进入运行生命周期	peer 已发现、集合已同步
`register.put()` 返回	本地接受变更	任一远端已应用
节点 B `get()` 命中	B 当前可见该值	所有节点相同、不会再冲突
双节点测试通过	固定源码在受控 fake membership 下满足断言	真实网络、生产容量、故障修复

常见错误

错误	纠正
把 DSM 当事务数据库	从不可撤销业务承诺和审计责任重新分类
每次发布用新 entry key	用稳定业务身份，让 Register 更新同一逻辑条目
用 `sleep` 验证复制	等待目标条件并设置明确超时
测试通过就宣称生产可用	写出测试拓扑、通信实现和未覆盖边界

迁移练习

给「临时功能开关」写两份设计：一份有权威配置中心，DSM 只保存可重建 hint；另一份没有权威源、开关直接控制资金动作。解释为什么前者可能进入 DSM，而后者需要更强的事实与审计系统。

下一步

进入第二篇，用 Register、Lease 和 CRDT 分别回答当前值、所有权和可合并更新。

LEARNING ARC

选择正确的一致性语义

第 5–9 章

从业务不变量出发，分别理解 *Register*、*Lease* 与 *CRDT* 的冲突和合并语义。

第 5 章——Register：当前值与陈旧读取

本章目标：本章完成后，开发者能够用 Register 表达一个 key 的当前逻辑值，并避免把本地查询误当成集群强一致读取。

学习目标

1. 掌握 `put/get/all/remove/size` 的本地契约。
2. 区分更新同一 key 与创建新 key。
3. 解释 remove 也需要传播和冲突排序。
4. 为 route hint 写出陈旧读取策略。

前置条件

- 已观察第 4 章的双节点 delta。
- 理解 locator 与 entry key。

案例进度

履约服务从 `10.0.0.1` 滚动到 `10.0.0.2`。网关更新同一个 `fulfillment-primary`，旧节点退役后删除提示。

一个 key 的可见值

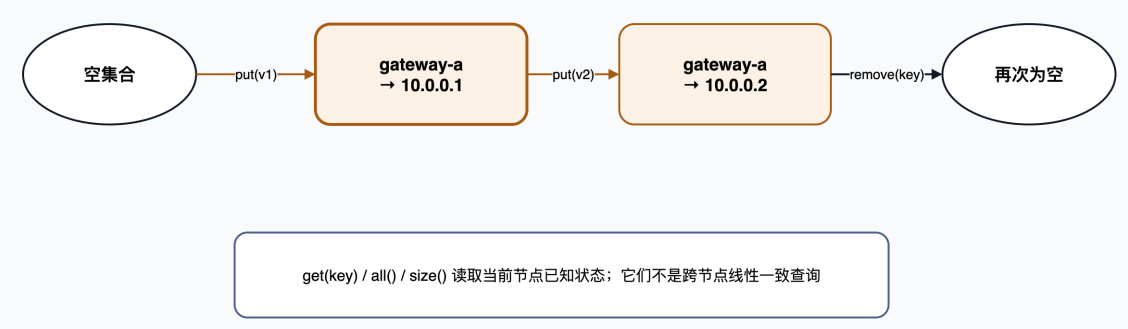

图 5: Register 的可见值操作

`put(v2)` 提交一个带 lineage metadata 的新候选值，没有数据库行锁保护的覆盖写语义。单节点立即看到 v2；其他节点通过在线 delta 或后续 repair 追上。

```
register.put(route("fulfillment-primary", "10.0.0.1:8080"));
register.put(route("fulfillment-primary", "10.0.0.2:8080"));
Optional<RouteHint> current = register.get("fulfillment-primary");
Optional<RouteHint> removed = register.remove("fulfillment-primary");
```

`all()`、`size()` 和 `get()` 都观察当前节点已知状态。它们适合本地路由决策和诊断，不是跨节点快照，也没有读 quorum。

业务侧的陈旧读取策略

网关不能只写「从 Register 读取」。它还要回答：条目缺失时是否回源？连接失败时是否尝试次选地址？多长时间后把 hint 视为过期？如果路由错误会产生不可撤销效果，DSM 就不应独自承担决定。

本案例采用：route hint 失败时重新查询权威服务目录并重试一次；订单提交本身仍由履约服务的事务接口保证幂等。

反例与故障注入

把服务每次健康探测结果写成随机 key，然后调用 `all().get(0)` 取一个地址。条目会无限增长，顺序也没有业务含义。正确模型是稳定服务 key 对应当前 hint；需要保留历史时，把历史写入可审计系统。

另一个错误是 remove 后立刻断言所有节点都为空。remove 与 put 一样需要传播，并可能遇到并发候选值。

实验

运行双节点 put/remove/反向 put 测试，并记录三个观察点：本地提交、远端 put、远端 remove。

```
cd submodule/dsm
mvn -q -pl dsm-integration-test -am \
  -Dtest=TwoNodeIntegrationTest#runtimeFirstRegisterDeltasReplicateAcrossTwoNodes \
  -Dsurefire.failIfNoSpecifiedTests=false test
```

实验验收卡

字段	内容
运行命令	上述聚焦 Maven 命令
输入或故障	同一 key 的 put、remove；反向节点写入
可观察结果	对端当前可见值按操作变化；远端 remove 有来源事件
证据等级	E3：双节点受控集成测试
本实验未证明	全局强一致读取、历史查询、事务条件更新

回顾

- Register 表达一个 key 的当前逻辑值，不表达完整历史。
- 所有查询都是当前节点视角。
- remove 也是需要排序、传播和修复的状态变化。
- 业务需要定义缺失、陈旧和调用失败时的回源或降级策略。

下一步

第 6 章让 A、B 在分区窗口同时更新同一个 key，解释它们为何需要选出同一个 winner。

证据链接

- `DsmRegister`
- `RegisterLineageOrder`
- `TwoNodeIntegrationTest`

第 6 章——Register 的并发写入与确定性收敛

本章目标：本章完成后，开发者能够解释并发候选值怎样确定性收敛，并验证自定义 resolver 是否满足交换律。

学习目标

1. 区分墙上时钟、HLC 与 lineage total order。
2. 说明默认 last-write-wins 的确定性来源。
3. 写出 resolver 的交换律检查。
4. 识别「最终同值」不能修复的外部副作用。

前置条件

- 已掌握 Register 本地视角和双节点传播。
- 知道网络分区期间两端可能各自接受本地写入。

案例进度

节点 A 把 `fulfillment-primary` 指向 east，节点 B 同时指向 west。恢复后两端需要选择同一个可见 hint，否则请求会永久分裂。

确定性比“谁更快”重要

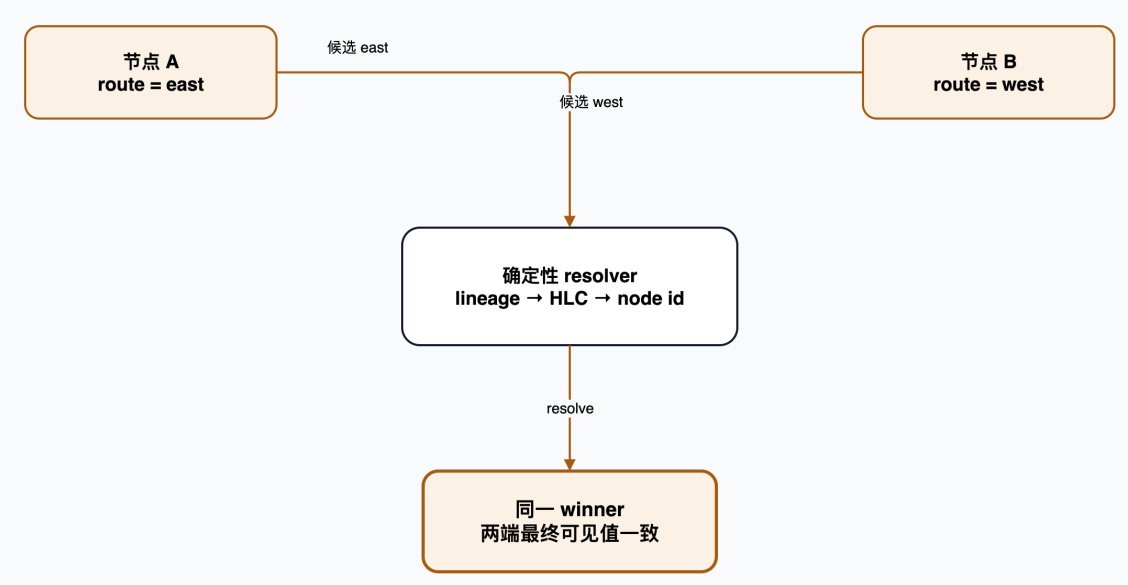

图 6: 并发候选值的确定性裁决

默认 `ConflictResolver.lastWriteWins()` 使用 `RegisterLineageOrder` 比较 metadata。排序包含 lineage/HLC 信息，并以稳定节点标识打破平局。正确性取决于所有节点对同一对候选值执行相同 total order，不能依赖某台机器对本地时间的判断。

数学上，至少要满足：

```
resolve(A, B).winner == resolve(B, A).winner
```

如果 resolver 按第一个参数优先，A 节点可能保留 east，B 节点可能保留 west。消息重复、顺序交换后仍无法收敛。

自定义 resolver 的安全入口

`ConflictResolver.verified(delegate)` 会分别用 `(local, remote)` 与 `(remote, local)` 调用 resolver，并比较获胜实体及 outcome family。不满足交换律时抛出 `CONFLICT_RESOLVER_NOT_COMMUTATIVE`，适合测试、预发或受控上线。

这项检查只覆盖实际输入对，也会让 resolver 执行两次，因此不能视为形式化证明。resolver 仍需保持纯、确定且无外部副作用。

反例与故障注入

定义 `resolve(local, remote) -> localWins(local)`。在两个节点分别以自己值为 local 的情况下，结果永久分裂。再定义一个读取当前时间或随机数的 resolver，即使偶尔交换后同值，也无法对重放保持确定。

并发库存扣减即使最终选出同一个 winner，也已经可能向两个客户返回成功。这再次证明冲突裁决只解决副本可见值，不撤销外部承诺。

实验

运行自定义 resolver 双节点测试：

```
cd submodule/dsm
mvn -q -pl dsm-integration-test -am \
 -Dtest=TwoNodeIntegrationTest#customConflictResolversConvergeMergedRegisterValuesAcrossTwoNodes \
  -Dsurefire.failIfNoSpecifiedTests=false test
```

然后阅读本章 `resolver-cases.json`，对四个候选策略判断是否纯、交换、确定。

实验验收卡

字段	内容
运行命令	上述 Maven 命令；`node --test tests/chapter-assets.test.mjs`
输入或故障	两端同 key 并发候选值；输入顺序反转
可观察结果	合并结果在两个节点相同；错误策略在资产检查中被标注
证据等级	E3：受控双节点冲突测试
本实验未证明	所有输入域上的形式化正确性、跨资源事务或外部副作用撤销

回顾

- 并发写要求所有节点使用同一个 total order 或可交换合并规则。
- HLC 提供逻辑顺序，不是全局同步时钟。
- 自定义 resolver 需要纯、确定且满足交换律。
- 收敛修复副本，不修复已经对外发生的业务效果。

下一步

第 7 章转向有限期所有权：分片处理关注谁在限定时间内有权操作，而非哪个候选值更新。

证据链接

- `ConflictResolver`
- `RegisterLineageOrder`
- `TwoNodeIntegrationTest`

第 7 章——Lease：有期限的所有权

本章目标：本章完成后，开发者能够用 acquire、renew、transfer、release 与 fencing token 表达有期限的分片所有权。

学习目标

1. 区分「当前值」和「当前有权执行者」。
2. 理解 TTL、epoch 与 fencing token 的不同职责。
3. 解释续租、转移、释放和过期。
4. 让外部资源拒绝旧持有者。

前置条件

- 已理解 Register 冲突裁决。
- 知道进程暂停、网络分区和时钟推进会造成旧 owner 继续运行。

案例进度

履约 worker 需要领取 `orders-17` 分片。只有有效持有者可以提交该分片的调度结果；所有权需要过期，转移后旧 worker 需要被挡在外部写入之前。

生命周期与护栏

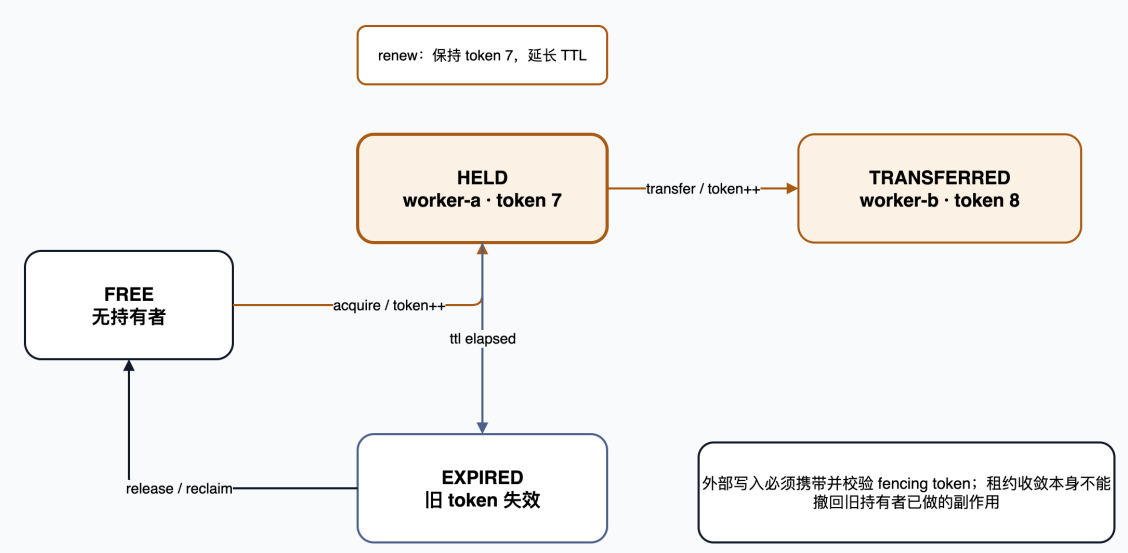

图 7: 分片所有权的租约生命周期

```
lease.acquire("orders-17", acquireOptions).join();
lease.renew("orders-17", renewOptions).join();
lease.transfer("orders-17", "worker-b", transferOptions).join();
lease.release("orders-17", releaseOptions).join();
```

TTL 回答「所有权何时失效」，fencing token 回答「外部系统如何识别旧持有者」。一次成功转移产生更高 token；数据库、对象存储或下游执行器需要保存最后接受的 token，并拒绝更小的值。

如果外部资源完全忽略 token，旧 worker 即使租约已过期仍可能写入。这不是 DSM 多复制一次就能修复的问题。

稳定成员视图为什么重要

Lease 是高风险协调动作。成员视图不稳定时，盲目重新分配可能让两个 worker 各自相信自己拥有分片。当前实现和测试的具体拒绝/模式边界应按 API 与运行信号判断，业务层不能只看一个本地 boolean。

反例与故障注入

让 worker A 获得 token 7 后暂停；租约转给 B，token 变成 8；A 恢复并继续写。如果外部执行器不比较 token，它会接受旧 owner。正确实验应同时验证租约状态和 `FencingExecutor` 对 token 7 的拒绝。

实验

```
cd submodule/dsm
mvn -q -pl dsm-integration-test -am \
  -Dtest=RuntimeIntegrationTest#fencingExecutorRejectsStaleHolderAfterLeaseTransfer \
  -Dsurefire.failIfNoSpecifiedTests=false test
```

测试除检查 holder 名称外，还要在转移后让旧 holder 带旧 token 执行，以确认外部边界能够拒绝旧持有者。

实验验收卡

字段	内容
运行命令	上述 `RuntimeIntegrationTest` 聚焦命令
输入或故障	A 持有后转移给 B；A 用旧 token 再执行
可观察结果	B 获得更高 token；旧 token 被 fencing executor 拒绝
证据等级	E2：单 Runtime 的集成行为测试
本实验未证明	真实成员分裂、跨进程暂停、所有外部存储都正确校验 token

回顾

- Lease 表达有期限的执行权，不是普通最新值。
- TTL 终止时间窗口，fencing token 保护外部副作用边界。
- 转移需要让 token 单调前进；旧持有者可能仍活着。
- 没有外部 token 校验，就没有完整的 fencing 保证。

下一步

第 8 章允许所有节点都本地更新计数，不选唯一 owner，而是设计可以安全合并的状态。

证据链接

- `DsmLeaseRegister`
- `LeaseState`
- `RuntimeIntegrationTest`

第 8 章——CRDT：多节点本地更新与合并

本章目标：本章完成后，开发者能够用 PN Counter 表达多节点本地增减，并说明合并函数需要满足的代数性质。

学习目标

1. 解释 state、update、version vector 和 delta。
2. 区分 CRDT 合并与 Register 选 winner。
3. 说明交换律、结合律和幂等为何必要。
4. 判断哪些业务量不能只靠计数器表达。

前置条件

- 已理解第 6 章的并发候选值。
- 知道请求计数允许短暂差异，但需要避免重复合并导致翻倍。

案例进度

east 节点本地记录 3 次请求，west 节点记录 2 次。网络恢复后两端都应得到 5，而不需要为每次请求争夺中心锁。

合并状态，不挑一个 winner

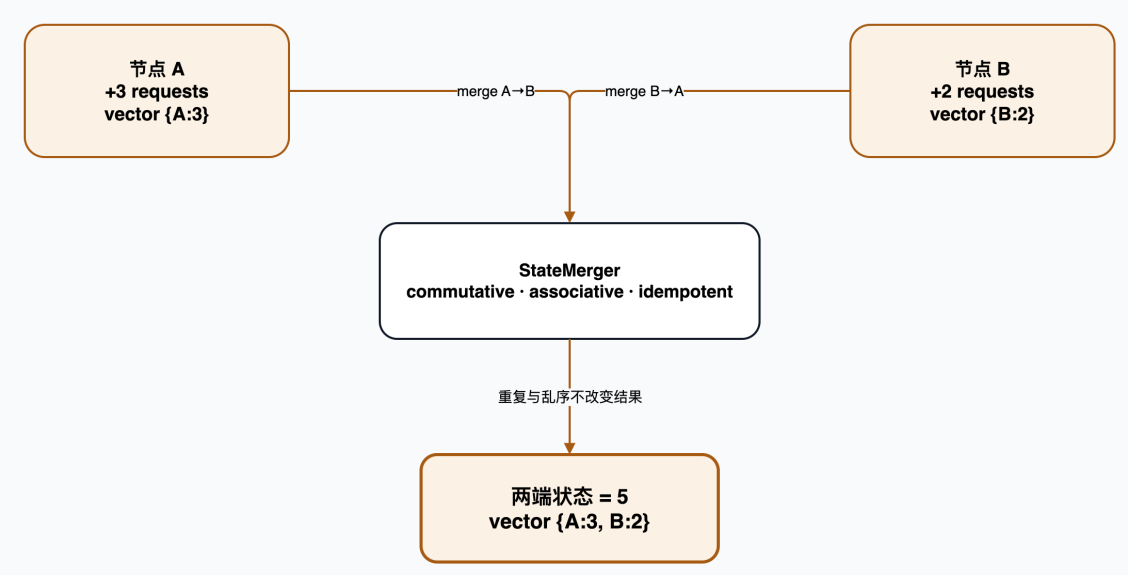

图 8: 各节点本地计数与合并

`DsmCrdtCollection` 的核心操作是 `apply(entity)` 与 `state()`。`versionVector()` 描述已观察的节点进度，`deltaSince(vector)` 生成给落后 peer 的状态增量。

```
counter.apply(new PnCounterUpdate("east-1", metadata, "east", 3, 0));
counter.apply(new PnCounterUpdate("west-1", metadata, "west", 2, 0));
long total = counter.state().value();
```

具体构造参数应以固定源码为准；这里的重点是每个节点贡献独立分量，merge 对每个分量取单调信息，再计算总值。重复收到同一状态不会再次加 3。

三条不可协商的性质

- 交换律：`merge(A, B) = merge(B, A)`，消息顺序不影响结果。
- 结合律：`merge(merge(A, B), C) = merge(A, merge(B, C))`，分批修复不影响结果。

- 幂等：`merge(A, A) = A`，重试和重复投递不放大结果。

这三条描述 merge，不等于所有业务操作都可交换。余额扣减、库存约束和「不超过一次」的外部效果不能因为用了 CRDT 就自动成立。

反例与故障注入

实现 `merge(left, right) = left.total + right.total` 并把总数当作新状态。相同快照被重复发送时，总数不断翻倍。正确 PN Counter 保存每个节点的正/负分量或等价单调结构，而不是反复相加已经聚合的总数。

实验

```
cd submodule/dsm
mvn -q -pl dsm-integration-test -am \
  -Dtest=RuntimeIntegrationTest#oneRuntimeMergesCrdtCounterUpdatesThroughTheRuntimeFacade \
  -Dsurefire.failIfNoSpecifiedTests=false test
```

再用 `crdt-merge-cases.json` 手算顺序交换、分组交换和重复输入三组结果。

实验验收卡

字段	内容
运行命令	上述 Maven 命令；`node --test tests/chapter-assets.test.mjs`
输入或故障	两个节点分量；顺序变化和重复输入
可观察结果	Runtime facade 得到合并计数；三条性质的期望结果完整
证据等级	E2：Runtime 内合并测试 + 书籍代数样例
本实验未证明	双节点网络传播、业务去重、精确一次计费或生产容量

回顾

- CRDT 合并多方状态，不从候选值中只留一个。
- version vector 描述已观察进度，delta 可针对落后 peer。
- merge 需要可交换、可结合、幂等。
- 技术可合并不代表业务约束可合并。

下一步

第 9 章把 Register、Lease、CRDT 放进同一张业务不变量决策树，完成第一次设计评审。

证据链接

- `DsmCrdtCollection`
- `PnCounterStateMerger`
- `RuntimeIntegrationTest`

第 9 章——从业务不变量选择集合

本章目标：本章完成后，开发者能够从业务不变量出发，为 Register、Lease 或 CRDT 给出可反驳、可验证的选择理由。

学习目标

1. 先判断状态是否适合 DSM，再选择集合。
2. 把不变量映射到集合语义和失败边界。
3. 为每个选择写出最小验证实验。
4. 识别应该退回数据库或消息系统的状态。

前置条件

- 已完成第 5—8 章的三类集合实验。
- 能说出本地成功、远端可见和最终收敛的差异。

案例进度

团队对 route hint、shard owner 和 request counter 分别开展设计评审，逐项写出采用条件、失败行为和拒绝理由，不预设所有状态都进入 DSM。

决策顺序

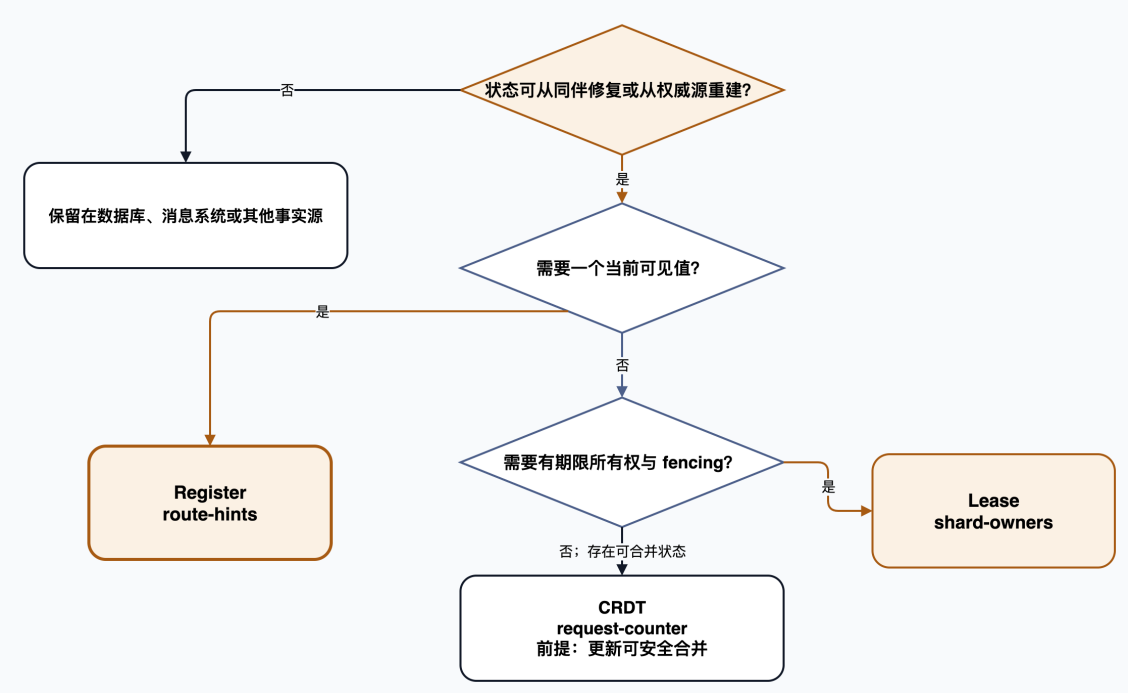

图 9: 从业务不变量选择集合

第一问始终是：状态丢失后能否从同伴修复或从权威源重建？若不能，或一次局部成功会直接形成不可撤销业务承诺，就先保留在事务/事件事实源。

状态	关键不变量	选择	分区窗口	验证重点
route hint	一个 key 有当前可见提示	Register	两端可短暂不同	同 key 确定性收敛、陈旧回源
shard owner	有期限所有权，旧 owner 不能写	Lease	旧 holder 可能仍运行	转移后 fencing token 拒绝旧写

状态	关键不变量	选择	分区窗口	验证重点
request counter	各节点可本地更新并安全合并	CRDT	局部计数短暂不完整	重复、乱序、分组合并结果不变
order/payment	事务事实与审计历史	不采用 DSM 主存储	双成功不可撤销	事务、幂等、对账、审计

选择是一个可检验假设

写「使用 Lease，因为它支持所有权」还不够。更完整的决策是：

> 对 `orders-17`，任一时刻只有持有最新 fencing token 的 worker 可提交调度结果；租约转移后，执行器拒绝旧 token。用转移测试和旧 token 反例验证。

这句话同时包含主体、状态、失败窗口、外部责任和证据。

反例与故障注入

- 用 Register 保存 shard owner：可以选出最新值，却没有 TTL/续租/转移/fencing 合同。
- 用 Lease 保存请求总数：强迫所有更新围绕唯一 owner，丢失本地可写优势。
- 用 CRDT 保存订单状态：可合并不等于合法状态迁移，也不提供跨资源事务和审计历史。

实验

打开 `collection-review.json`，为每个场景填写 `decision`、`invariant`、`failureWindow` 和 `evidenceCommand`。然后运行：

```
node --test tests/chapter-assets.test.mjs
```

参考答案要求每个「采用」和「拒绝」都有业务理由，不能只填集合名称。

实验验收卡

字段	内容
运行命令	`node --test tests/chapter-assets.test.mjs`
输入或故障	7 个业务状态及其不可变量、失败窗口
可观察结果	三类 DSM 集合和拒绝项均被覆盖；字段完整
证据等级	E1：设计评审资产检查，引用前章 E2/E3 实验
本实验未证明	新业务场景自动适配、生产容量或组织接受度

回顾

- 先决定 DSM 是否适合，再在三类集合之间选择。
- Register 解决当前值，Lease 解决有期限所有权，CRDT 解决可合并本地更新。
- 每个决定需要同时写出失败窗口与验证命令。
- 订单、库存和支付继续由事务事实源承担。

下一步

第 10 章进入第三篇：在讨论 delta 和 repair 前，先弄清谁在集群里、谁只是被怀疑离线。

证据链接

- `DsmRegister`
- `DsmLeaseRegister`
- `DsmCrdtCollection`
- 第 1 章责任边界

第二篇回顾——选择正确的一致性语义（第 5–9 章）

第二篇建立了以业务不变量、失败窗口和验证方法为依据的集合选择过程。

本篇形成的认识

- Register 保留一个 key 的当前逻辑可见值；查询是本地视角。
- 并发 Register 候选值需要以确定性 total order 或满足交换律的 resolver 收敛。
- Lease 用 TTL 和 fencing token 表达有期限所有权，并要求外部系统拒绝旧 token。
- CRDT 允许节点本地更新，但 merge 需要可交换、可结合、幂等。
- 不满足可修复/可重建前提的状态，不应因为 API 方便而进入 DSM。

能力压缩矩阵

业务不变量	集合	分区期间	恢复后	验证重点
一个当前 hint	Register	可能看到不同值	确定性 winner	顺序交换仍同值；陈旧回源
有期限的唯一执行权	Lease	旧 holder 可能仍运行	新 token 成为有效 owner	外部资源拒绝旧 token
多节点本地更新可安全合并	CRDT	局部状态不完整	merge 后同状态	交换、结合、幂等
事务事实或可靠历史	不采用 DSM 主存储	局部成功可能不可撤销	收敛无法补偿副作用	事务、幂等、审计、重放

本篇如何兑现 DSM 价值

第二篇用 Register、Lease 和 CRDT 承担三类常见协调语义，使业务团队可以复用已定义的冲突裁决、租期和合并机制。价值取决于集合语义与业务不变量是否匹配；业务团队仍负责写明分区后果、fencing 和权威事实来源。

评审时的五个问题

1. 这个状态的权威来源是谁？
2. 丢失时能否从 peer 或权威源恢复？
3. 分区期间允许两个节点各自成功吗？
4. 外部副作用如何被 fencing、幂等或事务保护？
5. 哪条实际命令能够证明选定语义，而不是只证明代码可编译？

常见错误

错误	纠正
用 Register 模拟 Lease	把 TTL、transfer 与外部 fencing 写进契约
用 CRDT 包装任意对象	先证明业务 merge 的代数性质
resolver 读取时间或随机数	保持纯、确定，并用输入反转检查交换律
只测试 happy path	注入旧 token、重复 delta、顺序变化和分区候选值

迁移练习

为优惠券剩余量、批处理 worker 所有权和页面浏览计数分别选择存储语义。若选择 DSM，写出失败窗口；若拒绝 DSM，说明需要的事务或历史保证。没有「三题都选 DSM」的加分项。

下一步

进入第三篇：把「最终一致」拆成成员视图、在线传播、漏失检测和 repair。

LEARNING ARC

理解复制、分区与修复

第 10–14 章

沿着成员视图、*delta*、*repair* 与 *change stream*，建立可观察的一致性边界。

第 10 章——成员视图与稳定性判断

本章目标：本章完成后，开发者能够区分成员事件、当前可见成员和稳定成员视图，并解释为什么高风险 Lease 不能对一次成员观察立即作决定。

学习目标

1. 说清成员发现、故障怀疑、离开与恢复分别代表什么。
2. 区分 `activeMembers()`、`clusterSize()` 与稳定集群大小。
3. 解释 `serviceId` 如何阻止不同逻辑集群互相发现。
4. 用成员稳定窗口保护 QUORUM Lease。

前置条件

- 已完成第 4 章的双节点复制实验。
- 已理解第 7 章 Lease 的 fencing 与 QUORUM 模式。
- 不再把「网络暂时没回应」直接等同于「节点永久离开」。

案例进度

履约集群加入 `node-b` 后，`node-a` 先看见成员事件，随后才连续看到相同的成员数量。就在这段抖动窗口里，worker 想领取 `orders-17`。团队需要决定：用哪一个成员视图计算多数派？

一次观察不是全局事实

图 10: 从发现成员到稳定成员视图

`ClusterMembership` 同时承担两类责任：提供成员观察，也提供 DSM 控制面和数据面通信通道。它公开 `self()`、`activeMembers()`、`clusterSize()`、`membershipRoundInterval()`，以及成员和数据监听器。但这些方法给出的是**当前节点的本地观察**，不是线性一致的集群名册。

信号	能说明什么	不能说明什么
`NodeJoined`	本节点收到加入信息	所有节点已形成相同视图
`NodeSuspected`	故障检测开始怀疑某节点	该节点已经永久失败
`NodeLeft`	本节点确认离开或收到离开信息	分区另一侧也已确认
`activeMembers()`	当前可通信的成员集合	原始集群总规模已经改变
`clusterSize()`	实现认为的集群规模	这个规模已连续稳定多轮

测试替身故意把 `clusterSize()` 与 `activeMembers()` 分开：发生分区时，原始集群规模保持不变，而当前可见

成员减少。这样 QUORUM 判断不会把单个节点的局部视图误判为完整集群，并据此错误放行。

稳定窗口为何存在

`MembershipStabilityTracker` 连续观察成员规模。只有达到 `quorumStabilityRounds`，`stableClusterSize()` 才从 `-1` 变成可用值；成员规模一旦变化，计数立即重置。

假设配置需要连续 3 轮稳定：

轮次	观察规模	稳定计数	`stableClusterSize()`
1	2	1	-1
2	2	2	-1
3	2	3	2
4	3	0	-1
5—7	3	1—3	最终为 3

QUORUM Lease 在稳定大小未知时返回 `membership-unstable`；视图稳定但可见成员不足多数时返回 `quorum-unavailable`。这是 fail closed：宁可暂时不发新租约，也不用漂移的分母计算多数派。

`serviceId` 是第一道隔离线

同一网络中可能同时运行开发、测试和生产集群。传输地址相同不代表它们属于同一逻辑服务。成员实现通过 `serviceId` 隔离发现与消息；不同 `serviceId` 的节点不应出现在彼此的成员列表中。

把 `serviceId` 当业务租户号也不合适。它定义的是集群级广播域；第 17 章会再加入集合 locator 的租户/应用隔离。

反例与故障注入

1. 让三节点集群中的两个节点互相分区。
2. 在孤立节点上读取 `activeMembers()`，它可能只看到自己。
3. 若错误地用可见数 `1` 当总规模，则多数派门槛变成 `1`，孤立节点会误判可写。
4. 正确实现保留集群规模或等待稳定视图，因此 QUORUM Lease 拒绝领取。

实验

先验证成员故障模型，再验证 Lease 门禁：

```
cd submodule/dsm
mvn -q -pl dsm-test-support,dsm-runtime -am \
 -Dtest=FakeClusterMembershipChaosTest,MembershipStabilityTrackerTest,InMemoryDsmLeaseRegisterTest \
  -Dsurefire.failIfNoSpecifiedTests=false test
```

阅读 `membership-observations.json`，逐项判断它属于事件、可见成员、总规模还是稳定视图。任何单个事件都不足以支持“全局已经一致”的结论。

实验验收卡

字段	内容
运行命令	上述 Maven 聚焦测试；`node --test tests/chapter-assets.test.mjs`
输入或故障	丢包、延迟、网络分区、怀疑、崩溃与恢复；成员规模变化
可观察结果	当前可见成员可减少；稳定规模经过连续轮次才生效；QUORUM Lease 在不稳定时拒绝
证据等级	E2：单进程成员与 Lease 行为测试
本实验未证明	跨主机网络质量、真实组播可达性、生产故障检测阈值合适

回顾

- 成员视图是本地观察，不是瞬时全局事实。
- `activeMembers()` 与原始集群规模承担不同角色。
- 高风险多数派判断需要等待连续稳定轮次。
- `serviceId` 隔离逻辑集群，但不替代集合级租户边界。

下一步

成员终于稳定了。第 11 章沿着一次 `route-hints.put`，追踪从本地提交到远端可见的完整 delta 路径。

证据链接

- `ClusterMembership`
- `MembershipStabilityTracker`
- `FakeClusterMembershipChaosTest`
- `InMemoryDsmLeaseRegisterTest`

第 11 章——一次写入的远端传播路径

本章目标：本章完成后，开发者能够把本地提交、delta 编码、传输、远端校验与应用分开观察，并知道在线传播为什么不能替代 repair。

学习目标

1. 跟踪 Register 更新从本地 mutation 到远端应用的五个阶段。
2. 解释 collection locator、schema 与 delta envelope 的作用。
3. 区分控制面消息和数据面传输。
4. 为传播链路上的漏失设计可修复边界。

前置条件

- 已完成第 5 章 Register 操作。
- 能区分第 10 章的成员视图与数据内容。
- 接受本地 `put` 成功不包含远端确认。

案例进度

Route Publisher 在 `node-a` 把 `payment` 的健康路由更新为 `10.0.0.8`。API 立即看到本地值，但 `node-b` 仍短暂保留旧地址。团队沿传播链路逐段定位，而不是把整个现象笼统称为「同步慢」。

五个阶段，五个观察点

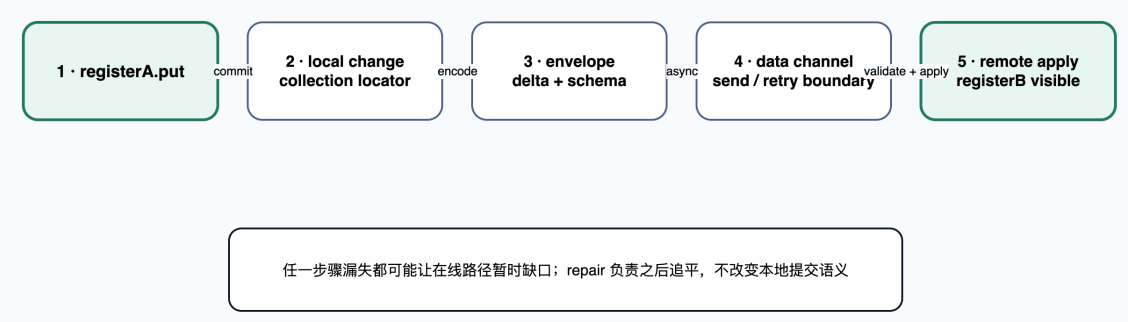

图 11: delta 从本地提交到远端应用

1. **本地提交**：集合完成 resolver/metadata 处理，当前节点可读。
2. **集合定位**：变更携带 tenant、application、collection 身份，不能只靠 Java 类型猜目标。
3. **封装编码**：delta 与 schema/capability 信息进入 wire envelope。
4. **数据通道**：发送、分帧、重试或丢失发生在传输边界。
5. **远端应用**：远端校验目标与能力，再把 REMOTE_DELTA 应用到对应集合。

这条链最重要的结论是：第 1 步完成就可以返回本地成功，而第 5 步可能更晚，甚至在在线路径上永远没有发生。

身份比载荷更先决定正确性

两个集合都保存 `RouteHint`，不代表它们能互相接收 delta。`RuntimePlatformSyncService` 按集合 route/locator 寻址，并在能力协商后路由 Register、Lease 或 CRDT 的同步消息。

传播契约至少要回答：

- 这是哪个逻辑服务发来的消息？
- 目标 collection locator 是什么？
- 载荷属于 Register、Lease 还是 CRDT？
- schema/codec 是否兼容？
- 这是在线 delta、replay 还是 snapshot 片段？

任何一项含糊，都可能把「消息送到了」误写成「状态正确应用了」。

在线 delta 的边界

在线 delta 优化正常情况下的低延迟传播。它不是可靠业务事件日志：

情况	在线路径可能发生什么	后续责任
接收节点暂时离线	没有实时应用	恢复后比较 digest 并 repair
控制面消息丢失	未及时发现差异	anti-entropy sweep 主动检测
envelope 不兼容	拒绝应用	记录拒绝原因，修复版本/能力
重复到达	需要幂等或按元数据裁决	集合 merge/resolver 处理
乱序到达	旧值不能覆盖新值	lineage、HLC 或集合语义处理

因此，ChangeStream 消费者不能假设事件历史完整。它用于观察集合变化，不提供消息队列式的审计日志。

反例与故障注入

将 `node-a` 到 `node-b` 的数据丢弃，再执行本地 `put`：

- `node-a.get(key)` 返回新路由；
- `node-b.get(key)` 仍是旧路由；
- 本地成功并没有被撤销；
- 网络恢复后，repair 才补齐缺口。

若测试只断言 `node-a`，它只能证明集合的本地语义。若只等待 `node-b` 结果却不记录 origin，也无法区分在线 delta 与后来 repair 的结果。

实验

运行两节点传播与故障恢复测试：

```
cd submodule/dsm
mvn -q -pl dsm-integration-test -am \
  -Dtest=TwoNodeIntegrationTest,ChaosIntegrationTest \
  -Dsurefire.failIfNoSpecifiedTests=false test
```

再打开 `delta-trace.json`，检查每个阶段都有输入、输出、可观察信号和失败后的接续动作。

实验验收卡

字段	内容
运行命令	上述 Maven 聚焦测试；`node --test tests/chapter-assets.test.mjs`
输入或故障	本地更新、远端节点、在线消息漏失与恢复
可观察结果	本地先成功；正常链路远端可见；漏失后由 repair 追平
证据等级	E3：进程内多节点与 chaos 集成测试
本实验未证明	跨主机吞吐、真实网络重传上限、业务事件 exactly-once

回顾

- 本地提交和远端可见是两个不同完成点。
- locator、集合类型和 schema 共同约束远端应用目标。
- 在线 delta 面向低延迟，不保证完整历史。
- 漏失是设计输入，repair 提供持续的差异检测与修复过程。

下一步

第 12 章让迟到的 `node-b` 主动比较 digest，并在 replay 与 snapshot 之间做可解释选择。

证据链接

- `RuntimePlatformSyncService`
- `RegisterCollectionRoute`
- `TwoNodeIntegrationTest`
- `ChaosIntegrationTest`

第 12 章——基于摘要的差异检测与副本修复

本章目标：本章完成后，开发者能够用 digest 发现差异，解释 replay 与 snapshot 的选择条件，并拒绝会让请求方倒退的 repair。

学习目标

1. 解释 digest、anti-entropy 与 repair 的关系。
2. 根据保留窗口和估算成本选择 replay 或 snapshot。
3. 识别「请求方比来源更新」和「同序列但更完整」的倒退风险。
4. 区分当前状态修复与业务事件历史重放。

前置条件

- 已完成第 11 章的 delta 路径。
- 理解在线消息可以丢失，但集合最终需要重新收敛。
- 理解 repair 是后台修复过程，不属于本地写入事务。

案例进度

`node-b` 离线期间错过了 10 条 route hint 更新。恢复后，它没有要求 `node-a` 重发「所有业务事件」，而是先报告自己的 collection digest，由 source 判断从保留 delta 继续还是传输当前 snapshot。

从差异到修复计划

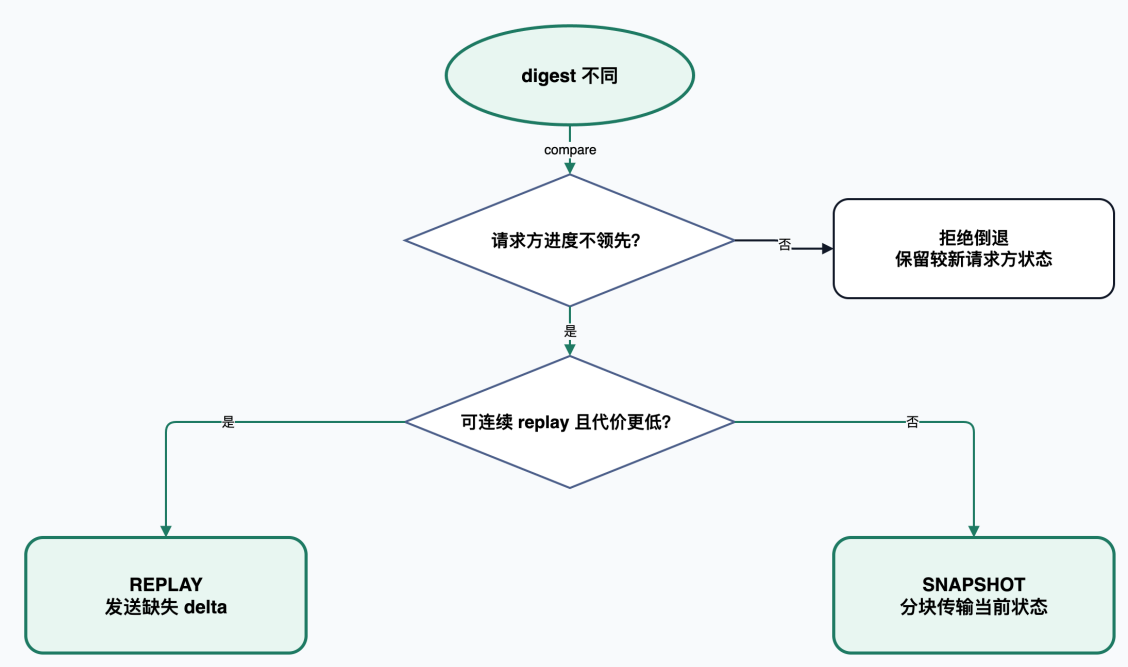

图 12: repair 对 replay、snapshot 与拒绝倒退的选择

digest 是状态摘要：它让双方判断 observed sequence、entry 数量或分桶 hash 是否存在差异。它不需要携带完整值，也不保证自己就是权威；它只是下一步协商的输入。

`AdaptiveRepairPlanner` 的决策边界可以压缩成四条：

1. 没有差异时返回 `NONE`。
2. 请求方进度领先，或同 sequence 下内容更完整时，不向它发送较旧 snapshot。
3. 请求方进度仍在 source 的 replay 保留窗口内时，比较 replay 与 snapshot 的估算成本。
4. replay 已不可用时，使用 snapshot 补齐当前状态。

Replay 与 Snapshot 不只是大小差异

维度	Replay	Snapshot
内容	从指定 sequence 开始的保留 delta	当前集合状态分块
适用	缺口连续且仍在保留窗口内	缺口过旧、过大或 snapshot 更便宜
优点	增量小，保留演进顺序	不依赖完整增量历史
风险	日志窗口不足；大量小消息	大传输；接收端需安全组装和切换
不是	业务事件审计重放	数据库备份或跨资源事务恢复

自适应选择依赖 peer profile 的延迟、吞吐和历史样本。没有 profile 时，planner 回退到固定策略；这比用一次偶发慢请求立即改策略更可预测。

绝不能修复成更旧

repair 失败会留下显式差异；更隐蔽的风险是 repair 返回“成功”却把请求方覆盖成旧状态。例如，请求方 sequence 为 101，而 source 只有 100。此时即使 hash 不同，也不能把 source snapshot 当作权威状态倒灌。

同样，在 sequence 相同的情况下，请求方拥有更多 entry 时，source 不应把较少 entry 的 snapshot 发过去。`AdaptiveRepairPlannerTest` 把这两种情况都固定为 `RepairMode.NONE`。

Anti-entropy 是持续控制环

一次在线 delta 漏失后，系统需要周期性 sweep：

交换摘要 → 判断差异 → 生成 `repair plan` → 传输 → 应用 → 再比较

控制面消息也可能丢。`ChaosIntegrationTest` 注入 50% 控制面丢失，验证后续 sweep 仍能发现并修复。一次 repair 请求成功只能说明单次操作完成；修复过程还需通过最终摘要一致来确认。

反例与故障注入

- 将 requester sequence 设置为 source 之前，观察 planner 拒绝倒退。
- 将 requester 放到 replay 窗口之外，观察计划切换为 snapshot。
- 使用 profile 让 snapshot 估算更快，观察即使 replay 可用也选择 snapshot。
- 丢掉第一轮控制面消息，观察后续 anti-entropy sweep 补偿。

实验

```
cd submodule/dsm
mvn -q -pl dsm-sync,dsm-integration-test -am \
  -Dtest=AdaptiveRepairPlannerTest,ChaosIntegrationTest \
  -Dsurefire.failIfNoSpecifiedTests=false test
```

在 `repair-scenarios.json` 中，为每个 requester/source 组合选择 `NONE`、`REPLAY` 或 `SNAPSHOT`，并写出拒绝倒退的理由。

实验验收卡

字段	内容
运行命令	上述 Maven 聚焦测试；`node --test tests/chapter-assets.test.mjs`
输入或故障	replay 窗口内/外、成本样本、请求方领先、控制面丢失
可观察结果	planner 选择可解释；请求方不倒退；后续 sweep 最终修复
证据等级	E2 planner 行为 + E3 chaos repair
本实验未证明	生产数据规模下的最佳阈值、跨区域带宽成本、业务历史完整性

回顾

- digest 发现差异，repair plan 决定如何补齐。
- replay 可用不代表一定更便宜，snapshot 也不天然更权威。
- 请求方领先或更完整时需要拒绝倒退。
- anti-entropy 是重复控制环，当前状态收敛不等于业务事件重放。

下一步

第 13 章把注意力转向观察者：即使集合能收敛，慢订阅者也可能让本地变更缓冲溢出。

证据链接

- `AdaptiveRepairPlanner`
- `AdaptiveRepairPlannerTest`
- `CollectionDigest`
- `ChaosIntegrationTest`

第 13 章——ChangeStream 的背压与溢出策略

本章目标：本章完成后，开发者能够为 ChangeStream 显式选择容量、事件来源和溢出策略，并把丢弃、阻塞与回调超时变成可观察结果。

学习目标

1. 区分集合收敛与 ChangeStream 事件投递。
2. 解释有界缓冲为何是稳定性边界。
3. 比较 `DROP_OLDEST`、`DROP_NEWEST`、`BLOCK` 和 `ERROR`。
4. 根据消费者用途选择本地/远端事件来源与失败策略。

前置条件

- 已完成第 5 章的 Register 变更流实验。
- 理解第 11—12 章的 delta 与 repair。
- 不把 ChangeStream 当可靠业务事件日志。

案例进度

Metrics Adapter 订阅 `route-hints`，把变化投给一个变慢的诊断面板。集合本身仍在正常更新，但消费者速度低于生产速度。团队需要选择：丢旧事件、丢新事件、短暂阻塞，还是明确报错。

缓冲区满时的策略选择

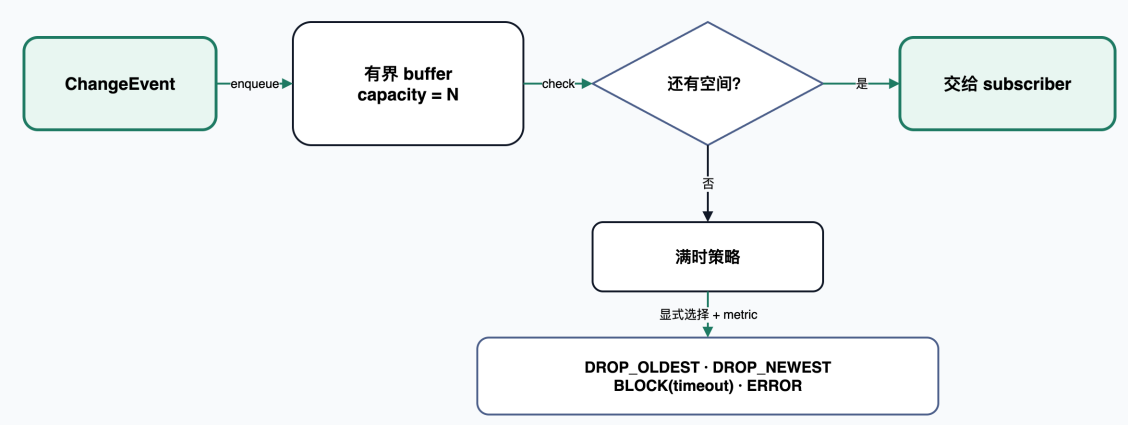

图 13: ChangeStream 的有界缓冲与溢出策略

`ChangeStreamOptions` 默认包含本地和远端事件，缓冲容量为 1024，溢出策略为 `DROP_OLDEST`，阻塞上限 1 秒，订阅回调上限 5 秒。默认值使流保持有界，但默认值并不替代业务选择。

策略	满时行为	更适合	主要代价
`DROP_OLDEST`	丢最旧事件，接纳最新事件	UI 刷新、当前状态提示	中间变化不完整
`DROP_NEWEST`	保留已有队列，丢新事件	先到先处理的短队列	当前状态可能更陈旧
`BLOCK`	等待消费者腾位直到 timeout	可承受短抖动的受控消费者	生产线程等待；超时仍会丢
`ERROR`	抛结构化溢出异常	不允许静默损失的验证/控制路径	调用方需要处理失败

这些策略改变的是观察流，不是已经提交的集合值。ChangeStream 丢一条事件后，`register.get(key)` 仍可能返回最新值；消费者若需要当前状态，应重新读取集合，而不是假设事件序列完整。

来源过滤也是契约

事件 origin 区分本地操作与远端 delta。只观察 `includeLocal=true` 适合确认本节点动作；同时观察 local/remote 适合诊断收敛；两者都关闭没有意义，因此构造器会拒绝。

对贯穿案例：

- 诊断面板同时看 local 与 remote，标注 origin。
- 本地发布审计辅助可以只看 local，但它仍不是交易审计源。
- repair 验证需要结合最终集合读取，不能只数 ChangeEvent。

慢回调不能无限占住推送线程

push subscriber 的回调也有超时。超过 `subscriberCallbackTimeout` 时，流通过 `onError` 暴露失败。把用户代码放在无限制回调中，会把「一个面板慢」扩散成整个本地更新路径的资源问题。

稳定做法是：回调只做快速转交；耗时 IO 进入独立有界执行器；同时记录 overflow 和 callback timeout 的低基数指标。

反例与故障注入

把容量设为 1，连续发布两个变化但不消费：

- `DROP_OLDEST` 最终留下第二条；
- `DROP_NEWEST` 最终留下第一条；
- `BLOCK` 等消费者腾位，超时后保留已有事件并记录 overflow；
- `ERROR` 抛出结构化异常并记录指标。

这个实验刻意不讨论哪一个「总是正确」。正确答案由消费者是否需要最新视图、顺序前缀还是显式失败决定。

实验

```
cd submodule/dsm
mvn -q -pl dsm-runtime -am \
  -Dtest=BufferedChangeStreamTest,InMemoryDsmRegisterTest \
  -Dsurefire.failIfNoSpecifiedTests=false test
```

然后完成 `overflow-cases.json`：为四种策略写出 retained event、producer outcome 和 required metric。

实验验收卡

字段	内容
运行命令	上述 Maven 聚焦测试；`node --test tests/chapter-assets.test.mjs`
输入或故障	容量 1、两条事件、慢消费者和超时回调
可观察结果	四种策略保留/拒绝行为不同；overflow 与 callback timeout 可见
证据等级	E2：集合和缓冲区行为测试
本实验未证明	ChangeStream 可靠历史投递、跨进程 exactly-once、生产容量足够

回顾

- 集合状态正确不代表每个订阅者都收到每条事件。
- 有界容量需要配套显式 overflow policy。
- 阻塞是有上限的策略，不是无限可靠性。
- 观察流漏事件后，应按用途重读当前状态或转向可靠事件系统。

下一步

第 14 章把所有失败窗口叠在一起：网络分区时两个节点都可能返回局部成功，而 repair 不能撤销已经发生的业务承诺。

证据链接

- `ChangeStreamOptions`
- `OverflowPolicy`
- `BufferedChangeStreamTest`
- `InMemoryDsmRegisterTest`

第 14 章——收敛边界与已经发生的业务响应

本章目标：本章完成后，开发者能够画出分区窗口中的局部成功与恢复后收敛，并判断哪些业务承诺不能交给最终一致状态处理。

学习目标

1. 区分响应语义、当前可见状态和最终收敛状态。
2. 解释 autonomous 与 quorum 协调在分区中的取舍。
3. 识别 repair 无法撤销的外部副作用。
4. 为 route hint、Lease、计数器和订单事实划定一致性边界。

前置条件

- 已完成第 6 章并发冲突与第 7 章 Lease。
- 已理解在线 delta、repair 和 ChangeStream 漏失。
- 能把「最后收敛为一个值」与「过程中只成功一次」分开。

案例进度

网络把 `node-a` 与 `node-b` 隔开。两端都更新 route hint，并分别向调用方返回 OK。恢复后 resolver 选出一个 winner。运维看到副本一致了，却发现两个调用方都已经基于各自的 OK 执行后续动作。

收敛不等于撤销过去

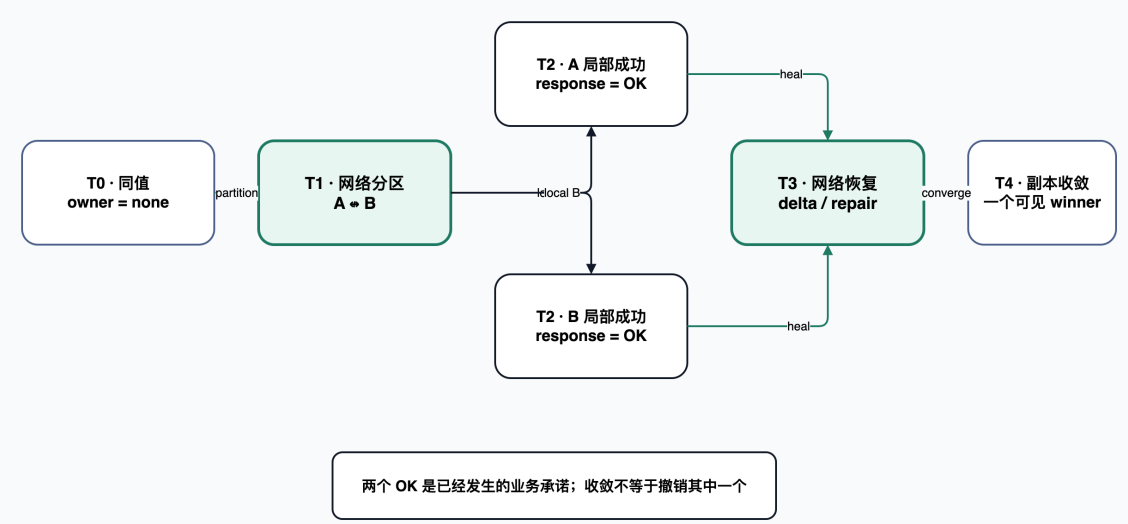

图 14: 网络分区中的两个局部成功与恢复后收敛

在 T2，两个局部成功都是真实发生过的响应。T3 的 delta/repair 可以让 T4 的当前可见值收敛，却不能让客户端忘记其中一个 OK，更不能自动补偿已经触发的邮件、扣款、发货或外部写入。

因此要分别写三份合同：

合同	要回答的问题	DSM 能否独立保证
本地操作合同	`put/acquire/increment` 在何时返回	能说明本地集合结果
收敛合同	分区恢复后副本如何得到兼容状态	Register/Lease/CRDT 各自负责
业务承诺合同	已返回的成功能否重试、撤销、对账	需要事务、幂等、fencing 或补偿

三类集合在分区中的不同表现

Register：两个 OK，一个最终 winner

Register 允许两个节点各自接受候选值，恢复后用元数据和 resolver 选择一个当前值。它适合 route hint，因为失败的候选可以被覆盖且权威健康源可重新发布；它不适合把两个订单状态写入后假设 loser 从未发生。

Lease：availability 与独占性的明确取舍

AUTONOMOUS 模式可能在分区两侧产生 holder，依赖外部 fencing token 拒绝旧 owner。QUORUM 模式在成员不稳定或多数不可见时拒绝 acquire，从而用可用性换取更强的发放约束。无论哪种模式，下游都需要校验 token；内存中的 holder 判断不能约束已经离线的旧进程。

CRDT：合并安全不等于业务合法

请求计数可在两端本地增加，恢复后合并贡献。这个性质适合指标，却不能证明库存扣减不会超卖：数学上可合并的两个 decrement，不一定满足库存业务下限和审计要求。

一张拒绝清单

以下状态不能因为最终会收敛就直接放入 DSM 作为权威事实：

- 订单状态迁移和不可逆审批；
- 库存扣减与额度消耗；
- 支付、退款与账务分录；
- 需要完整、不可遗漏历史的业务事件；
- 需要给调用方单一确定结果的跨资源事务。

DSM 可以缓存或协调这些系统周围的可重建状态，但权威结果仍由事务事实源、幂等协议和审计系统承担。

反例与故障注入

`ChaosIntegrationTest` 同时展示两种 Lease 选择：

- autonomous Lease 在分区中可能出现 dual holders；
- quorum Lease 在失去多数时拒绝 acquire，恢复并重新稳定后才允许。

这个对照没有宣布一种模式普遍优于另一种。它迫使业务 Owner 回答：暂停工作和短暂双执行，哪个风险更可接受？外部副作用是否有 fencing/幂等保护？

实验

```
cd submodule/dsm
mvn -q -pl dsm-integration-test -am \
  -Dtest=ChaosIntegrationTest \
  -Dsurefire.failIfNoSpecifiedTests=false test
```

再审阅 `partition-contract.json`。每个状态都需要写 `localResponse`、`partitionBehavior`、`healedState` 与 `externalGuard`；任何一栏写「由最终一致保证」都不合格。

实验验收卡

字段	内容
运行命令	上述 Maven chaos 测试；`node --test tests/chapter-assets.test.mjs`
输入或故障	网络分区、双端本地操作、网络恢复与 anti-entropy
可观察结果	Register/repair 最终收敛；autonomous Lease 可双 holder；quorum Lease fail closed
证据等级	E3：多节点 chaos 行为测试
本实验未证明	业务副作用已经被补偿、跨资源事务成立、生产网络参数合适

回顾

- 最终一个 winner 不会撤销分区期间已经返回的两个 OK。
- 集合语义解决当前协同状态，不替代业务承诺协议。
- Lease 模式是在可用性与发放约束之间做显式选择。
- 订单、库存、支付与事件历史继续留在各自权威系统中。

下一步

第四篇开始把这些边界带进真实工程。第 15 章先建立领域端口，让业务代码说「发布路由、领取分片、记录请求」，而不是散落 DSM API。

证据链接

- `ChaosIntegrationTest`
- `ConflictResolver`
- `LeaseMode`
- 贯穿案例设计

第三篇回顾——复制、分区与修复的可观察过程（第 10–14 章）

第三篇把笼统的“网络恢复后会一致”展开为可观察过程：漏失发生在哪一段、由谁检测、如何修复，以及已发生的业务响应为何仍需外部保护。

从成员到收敛的完整过程

成员观察稳定
- → 本地 `mutation` 产生 `delta`
- → 在线传播尝试远端应用
- → `digest/anti-entropy` 发现缺口
- → `replay` 或 `snapshot` 修复
- → 再次比较并确认收敛

ChangeStream 是这个过程的观察面之一，但它自己的有界缓冲也可能丢事件。最终判断需要回到集合状态、digest 与业务事实源。

五个边界

边界	常见误解	正确合同
成员	一次 join/leave 就是全局事实	连续稳定视图后再做高风险 quorum 决策
传播	本地成功等于远端已写	分开记录 local commit 与 remote apply
修复	replay/snapshot 越多越安全	不倒退；在可用窗口和成本之间选择
订阅	ChangeStream 是可靠事件日志	有界缓冲、显式 overflow、必要时重读状态
响应	最终一个 winner 撤销 loser 的 OK	事务、幂等、fencing 或补偿保护副作用

本篇如何兑现 DSM 价值

第三篇说明 Runtime 如何处理成员变化、在线 delta、差异检测和副本修复。这些通用机制减少了各业务服务自行编写传播与补差流程的工作。DSM 负责当前协同状态的恢复；调用方仍负责解释局部成功、保护外部副作用，并为 ChangeStream 选择可接受的丢弃或失败策略。

故障到证据的映射

故障	最小证据	仍未证明
成员抖动	稳定轮次重置；Lease 返回 `membership-unstable`	真实网络阈值合适
在线 delta 漏失	本地先成功；恢复后 repair 追平	exactly-once 历史投递
requester 落后	planner 选择 replay/snapshot；不倒退	大规模最优成本
慢订阅者	四种 overflow 行为与指标	无损业务事件流
网络分区	autonomous/quorum Lease 对照	外部副作用已被保护

评审时的六个问题

1. 这里使用的是当前可见成员、原始集群规模还是稳定规模?
2. 本地返回成功时，远端到了哪一个阶段?

3. 谁会发现在线传播漏掉的 delta?
4. replay 不可用或更贵时，snapshot 如何安全接管?
5. 订阅缓冲满时，选择丢、等还是报错，指标是什么?
6. 分区中的局部成功是否触发了不可撤销业务副作用?

迁移练习

把贯穿案例的 route hint 换成「灰度开关」。分别写出：权威来源、locator、分区期间两个本地 OK 的业务影响、repair 方式、ChangeStream overflow 策略，以及不允许放入 DSM 的开关类型。

下一步

进入第四篇：把集合和失败边界封装为业务端口，接入 Spring Boot，并用两层身份隔离与多层测试证据守住工程边界。

LEARNING ARC

接入真实业务工程

第 15–18 章

把 *DSM* 收进领域端口、*Spring* 装配、隔离规则与分层测试，避免散落在业务代码里。

第 15 章——通过领域适配器隔离 DSM 细节

本章目标：本章完成后，开发者能够用领域端口封装 DSM 三类 typed handles，把 locator、生命周期和失败翻译集中在适配器边界内。

学习目标

1. 识别 DSM API 泄漏进业务层的维护代价。
2. 设计只使用业务语言的 `FulfillmentCoordination` 端口。
3. 把 Register、Lease、CRDT handle 与 locator 集中到 composition root。
4. 分开验证领域决策和真实 Runtime 行为。

前置条件

- 已完成第 9 章的集合选择。
- 已理解第 14 章的失败与业务承诺边界。
- 能运行 `examples/order-fulfillment-control-plane` Maven 示例。

案例进度

前三篇直接操作集合是为了学习语义。现在履约服务要进入真实工程：业务用例只说「发布路由」「领取分片」「记录请求」，不再知道 `shared/gateway/route-hints` 或 `LeaseAcquireOptions`。

业务端口拥有语言，适配器拥有技术细节

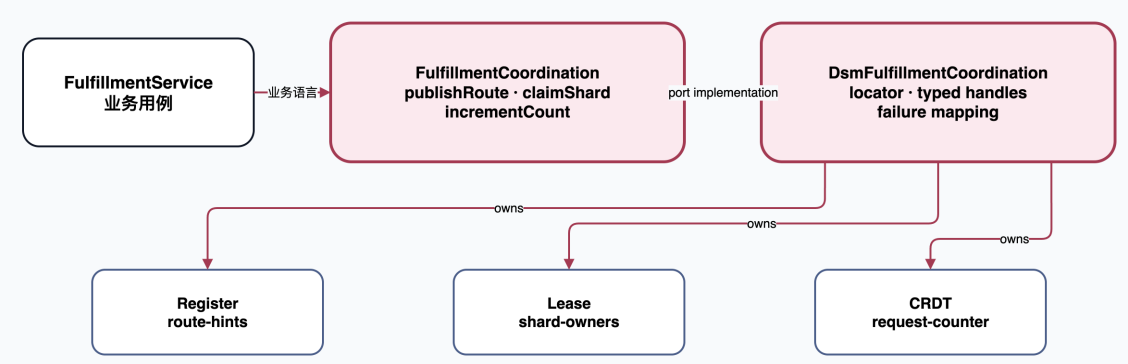

图 15: 领域端口与 DSM 适配器边界

书籍示例定义的端口很小：

```
public interface FulfillmentCoordination {
    void publishRoute(String serviceKey, String address);
    Optional<String> routeFor(String serviceKey);
    ShardClaim claimShard(String shardId);
    long incrementRequestCount();
}
```

端口不返回 `DsmRegister`、`LeaseSnapshot`、`PnCounterState` 或 `CollectionLocator`。业务层因此能用 fake 直接测试；DSM 适配器负责把结果翻译成 `ShardClaim(granted, uncertain, fencingToken, reason)`。

Composition root 是唯一装配点

`DsmFulfillmentModule` 集中拥有：

- `DsmRuntime` 的创建、启动和关闭；
- `clusterId` 与 `serviceId`；
- 三个稳定 locator 与 schema id；

- Register、Lease、CRDT 的 entity/codec/spec；
- 三个 typed handles 到领域端口的映射。

这不是为了隐藏所有 DSM 概念。locator 与 schema 仍是重要合同，只是由适配器拥有，而不是散落到 controller、service 和 job 中。

失败翻译不能抹掉不确定性

Lease acquire 有 `granted`、`uncertain`、snapshot 和 reason。适配器若只返回 `boolean`，调用方无法区分明确拒绝与结果不确定。领域结果应保留业务决策实际需要的信息，同时阻止调用方任意操作底层 handle。

同样，fencing token 需要穿过端口进入受保护的下游写入；把 token 藏掉会让第 7 章的安全边界失效。示例当前返回 token，但没有实现真实下游 guard，因此验收卡明确不声称已完成外部 fencing。

两类测试，各自只证明一层

示例有两个测试：

1. 用 `RecordingCoordination` 验证 `FulfillmentService` 只依赖领域端口。这是 E1。
2. 用真实 `DsmRuntime` 和三类集合验证适配器。这是单进程 E2。

如果只写第二类测试，业务决策和 DSM 装配会粘在一起；如果只写第一类，fake 全绿也不能证明真实 spec、codec 或 Lease 能运行。

反例与故障注入

- 把 locator 字符串复制到 `FulfillmentService`，然后改变 collectionId，观察需要修改多少业务文件。
- 把 `LeaseAcquireResult` 压成 boolean，尝试表达 `uncertain=true`。
- 忘记 `runtime.start()`，检查生命周期错误落在哪一层。
- 让业务层直接调用 `requestCounter.state()`，观察领域测试如何被 CRDT 类型污染。

实验

```
cd examples/order-fulfillment-control-plane
mvn clean test
```

阅读 `port-contract.json`，确认每个业务动作都有领域输入、领域输出、DSM handle 和未被适配器承担的外部责任。

实验验收卡

字段	内容
运行命令	`cd examples/order-fulfillment-control-plane && mvn clean test`
输入或故障	fake 端口；真实单节点 Runtime；三类 typed handles
可观察结果	领域服务不导入 DSM 类型；真实适配器完成路由、租约与计数操作
证据等级	E1 领域单元测试 + E2 单进程 Runtime 行为
本实验未证明	多节点传播、repair、真实网络、下游 fencing 或生产装配

回顾

- 领域端口拥有业务语言，适配器拥有 DSM 细节。
- locator、schema、codec 和生命周期应集中在 composition root。
- 失败翻译要保留不确定性和 fencing token。
- fake 与真实 Runtime 测试分别回答不同问题。

下一步

第 16 章把手工 composition root 切换为 Spring Boot 属性和条件 bean，并验证实际上下文能启动。

证据链接

- DSM examples：Runtime 装配入口
- DSM examples：Spring Boot 装配入口
- DSM examples：完整 Basic Usage 示例
- 当前业务集成指南

第 16 章——使用 Spring Boot 装配 Runtime

本章目标：本章完成后，开发者能够用 `dsm.*` 属性创建 Runtime 和命名集合 bean，理解自动装配的失败门禁，并验证应用上下文和生命周期。

学习目标

1. 说明 `DsmProperties`、`DsmAutoConfiguration` 与集合 bean 的装配顺序。
2. 配置 `clusterId`、`serviceId`、membership 和三类集合。
3. 通过 bean name 注入 typed collection handle 或领域适配器。
4. 让错误配置在启动期 fail fast。

前置条件

- 已完成第 15 章的领域端口与 composition root。
- 熟悉 Spring Boot configuration properties 和 bean 生命周期。
- 能区分「上下文启动成功」与「多节点网络已验证」。

案例进度

履约应用不再手写所有 builder。`application.yml` 描述 Runtime 与集合，auto-configuration 先创建 membership 和 Runtime，再注册命名集合 bean，最后把它们注入 DSM 领域适配器。

从配置到业务端口

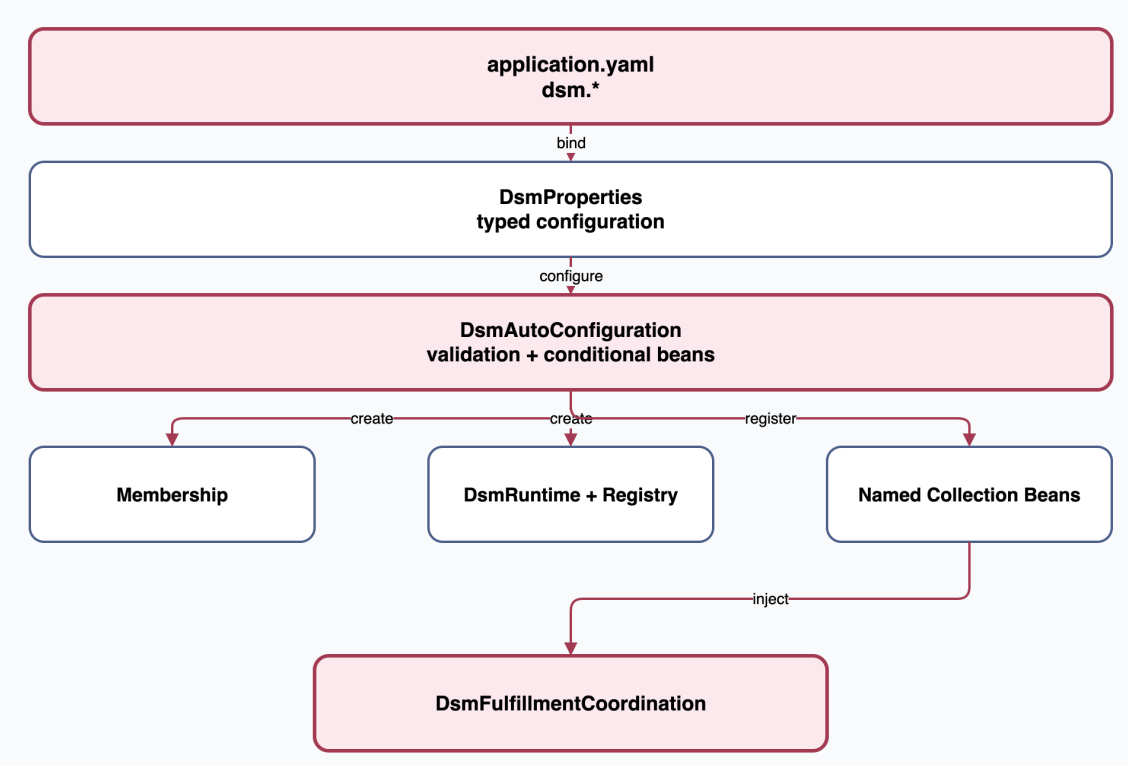

图 16: Spring Boot 从属性到领域适配器的装配

核心顺序不能倒置：集合注册需要已存在的 Runtime；领域适配器需要已注册的 handles；上下文关闭时 lifecycle 需要停止 Runtime。

最小属性合同

第 16 章资产给出三类集合的完整配置。关键字段如下：

```
dsm:
  cluster-id: fulfillment
  service-id: fulfillment-control-plane
  cluster:
    mode: standalone
  collections:
    - bean-name: routeHintsCollection
      tenant-id: shared
      application-id: gateway
      collection-id: route-hints
      schema-id: route-hints/v1
      type: REGISTER
      consistency-tier: REGISTER
      entity-type: com.example.RouteHint
```

真实类名和 codec/merger/entity factory 需要存在。完整资产使用占位的 `com.example.*` 教学类型，因此用于配置评审，不直接冒充可启动应用；实际启动证据来自固定源码中的 `DsmAutoConfigurationTest` 与 `SpringBootDsmApplicationTest`。

Bean 名称有两种

auto-configuration 为 locator 建立规范名，例如：

```
dsmCollection:shared/gateway/route-hints
```

配置中的 `bean-name` 再提供业务友好别名，例如 `routeHintsCollection`。两者指向同一个注册结果。业务适配器应使用显式 qualifier/bean name，避免容器仅凭原始泛型推断目标 bean。

Record codec 的便利与边界

设置 `entity-type` 且类型是 record 时，自动装配可以派生 `RecordCodec`。非 record 实体没有显式 `codec-bean` 会启动失败。CRDT 还需要提供 state codec、initial state 与 merger；Lease 需要提供 entity factory。

这种 fail fast 很重要：配置缺项若等到第一条远端消息才暴露，故障位置已经离装配错误太远。

启动门禁

`DsmAutoConfigurationTest` 固定了这些失败或成功行为：

配置	结果
缺少 `cluster-id`	启动失败
locator 重复	启动失败
non-record 缺 codec	启动失败
CRDT 缺 merger	启动失败
Lease 参数非法	启动失败
自定义 `ClusterMembership` bean	不被默认实现覆盖
上下文关闭	Runtime lifecycle 停止

反例与故障注入

复制第 16 章配置资产，删除 CRDT merger bean 或让两个集合使用相同 locator。预期结果是上下文直接拒绝启动并指出配置边界，错误不应延迟到业务功能执行阶段。

实验

```
cd submodule/dsm
mvn -q -pl dsm-spring-boot-autoconfigure -am \
  -Dtest=DsmAutoConfigurationTest,DsmSpringBootSmokeTest \
```

```
  -Dsurefire.failIfNoSpecifiedTests=false test
```

再运行真实示例应用上下文：

```
cd submodule/dsm-examples/basic-usage-examples
./mvnw -q -Dtest=SpringBootDsmApplicationTest test
```

实验验收卡

字段	内容
运行命令	上述 auto-configuration 与 Basic Usage 聚焦测试
输入或故障	三类集合属性、缺失 codec/merger、重复 locator、上下文关闭
可观察结果	Runtime 启动；规范名与别名 bean 可取；错误配置 fail fast；关闭时停止
证据等级	E2：真实 Spring ApplicationContext 与单进程 Runtime
本实验未证明	多实例发现、真实网络复制、生产密钥或部署就绪

回顾

- Spring Boot 只替代装配，不改变集合语义和失败边界。
- Runtime 需要先于集合，集合需要先于领域适配器。
- locator 规范名和业务别名都应明确、稳定、可测试。
- 配置错误应在启动期失败，而不是运行中静默降级。

下一步

第 17 章拆开两层身份：`clusterId/serviceId` 控制通信域，locator 控制同一 Runtime 内的集合域。

证据链接

- `DsmProperties`
- `DsmAutoConfiguration`
- `DsmAutoConfigurationTest`
- `SpringBootDsmApplicationTest`

第 17 章——通信域与集合域的隔离

本章目标：本章完成后，开发者能够分别配置通信域和集合域，并用同名 entry key 的反例证明两层身份没有混写。

学习目标

1. 区分 `clusterId`、`serviceId` 与 collection locator。
2. 解释消息为何要同时校验集群、服务和集合身份。
3. 证明不同 locator 下的同名 entry key 是不同状态。
4. 避免把 Spring bean name 当成 wire identity。

前置条件

- 已完成第 3 章 locator 与第 10 章 `serviceId`。
- 已完成第 16 章 Spring bean 注册。
- 能区分 Java 容器标识与分布式协议标识。

案例进度

网关和 worker 都使用 entry key `fulfillment-primary`。它们运行在同一履约 Runtime 中，却分别属于 `route-hints` 与 `shard-owners`。与此同时，另一个测试环境使用相同 locator，但不同 cluster/service 身份，也不能收到生产消息。

两层边界，不是一串可互换的名字

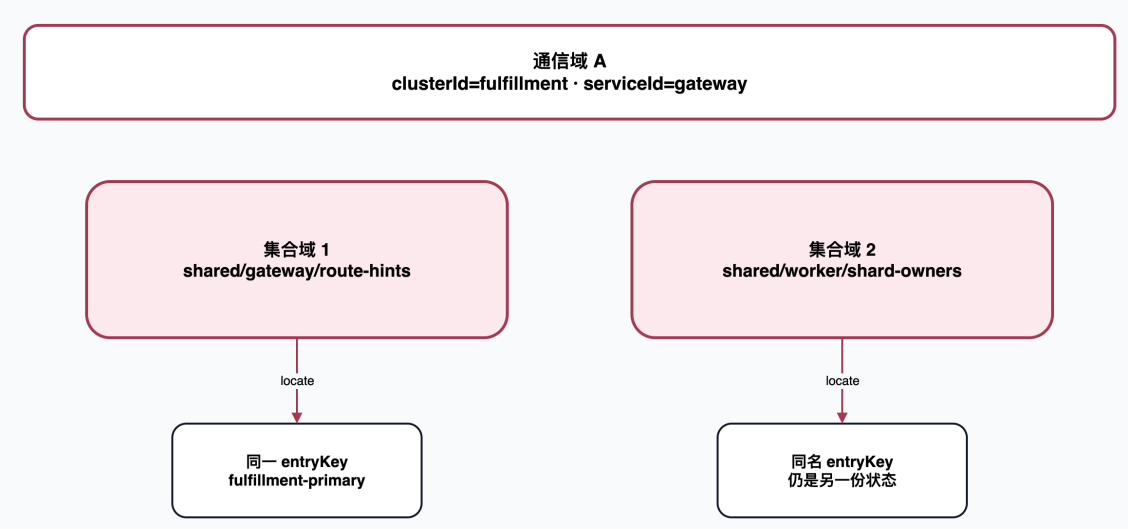

图 17: 通信域与集合域的两层隔离

标识	作用范围	解决的问题
`clusterId`	Runtime/sync envelope	消息是否属于同一个部署集群
`serviceId`	membership 与 sync envelope	哪些服务实例能发现并交换 DSM 消息
`tenantId`	collection locator	状态归属的租户/命名空间
`applicationId`	collection locator	哪个应用子域拥有集合
`collectionId`	collection locator	具体集合的稳定身份
Spring `bean-name`	单个应用上下文	代码如何选择一个已注册 handle
`entryKey`	单个集合内部	集合中的具体实体

entryKey 只有和完整 locator 放在一起才有意义。相同 entryKey 出现在不同 collectionId 或 applicationId 下，不会自动表示同一业务对象。

消息到达还要过两次身份检查

成员传输会拒绝不同 serviceId 的发现或消息；RuntimePlatformSyncService 还会校验 envelope 中的 clusterId 和 serviceId。通过通信域之后，collection route 再按 locator 找具体 handle。

因此，隔离不是只靠一个字段：

网络可达
→ clusterId/serviceId 匹配
→ 集合 descriptor/locator 匹配
→ schema/capability 匹配
→ entryKey 在目标集合内应用

每一层拒绝都应有可观察原因；不能把所有「看不到数据」都归因于 locator。

RuntimeIntegrationTest 的同名 key 反例

测试在一个 Runtime 中注册：

- shared/gateway/route-hints
- shared/worker/route-hints

两个 Register 都写入 same-key，一个值是 gateway-value，另一个是 worker-value。最终两边 size 都是 1，并各自读回自己的值。这直接证明 locator，而不是 Java entity 类型或 entry key，决定集合归属。

改名是协议迁移，不是代码重构

修改 bean alias 只影响当前 Spring 上下文；修改 locator、clusterId 或 serviceId 会改变分布式通信与状态身份。后者需要双写/迁移、兼容窗口或明确冷启动策略，不能作为普通 rename 直接上线。

反例与故障注入

- 两个集合配置相同 locator，Spring 启动应拒绝重复注册。
- 两个 Runtime 使用不同 serviceId，期望它们不形成同一成员视图。
- 同一 Runtime 用不同 applicationId 写同名 key，期望值互不覆盖。
- 只改 bean alias，确认 locator 与 wire identity 保持不变。

实验

```
cd submodule/dsm
mvn -q -pl dsm-integration-test,dsm-cluster -am \
  -Dtest=RuntimeIntegrationTest,MulticastClusterMembershipTest \
  -Dsurefire.failIfNoSpecifiedTests=false test
```

再检查 isolation-cases.json：每个场景都应明确变化的是通信域、集合域还是应用上下文名。

实验验收卡

字段	内容
运行命令	上述 Runtime 与 membership 聚焦测试；书籍资产测试
输入或故障	同名 key、不同 locator、不同 serviceId、重复 locator
可观察结果	locator 隔离值；serviceId 隔离成员；重复 locator fail fast
证据等级	E2 单 Runtime 隔离 + E3 受控多节点 membership
本实验未证明	操作系统/网络层强隔离、访问控制授权或租户数据合规

回顾

- `clusterId/serviceId` 控制通信域，locator 控制集合域。
- bean name 只服务于依赖注入，不是 wire identity。
- 相同 entity 类型和 entry key 不会跨 locator 自动合并。
- 分布式身份改名需要迁移计划。

下一步

第 18 章把前四篇的测试整理成证据阶梯，明确每一层证明了什么、没有证明什么。

证据链接

- `DsmRuntimeBuilder`
- `RuntimePlatformSyncService`
- `MulticastClusterMembership`
- `RuntimeIntegrationTest`

第 18 章——多层测试证据的适用范围

本章目标：本章完成后，开发者能够为领域端口、真实 Runtime、多节点协议和网络黑盒分别选择证据，并拒绝用较低层绿灯替代较高层验收。

学习目标

1. 建立 E1—E4 测试证据阶梯。
2. 为每层写出环境、固定 revision、已证明与未证明事项。
3. 区分真实集合实现、受控 fake membership 和真实网络端口。
4. 为三类集合安排最小但不失真的验证组合。

前置条件

- 已完成第 15 章书籍案例与第 16—17 章装配/隔离。
- 已运行第 14 章 chaos 测试。
- 不把 test count 当作生产就绪分数。

案例进度

履约团队已有领域测试、单 Runtime 测试和 chaos 测试。发布评审追问：「这些绿灯是否证明真实端口、客户端协议与进程隔离?」团队不再回答笼统的「测试覆盖了」，而是按证据层逐项陈述。

证据强度是适用范围，不是积分

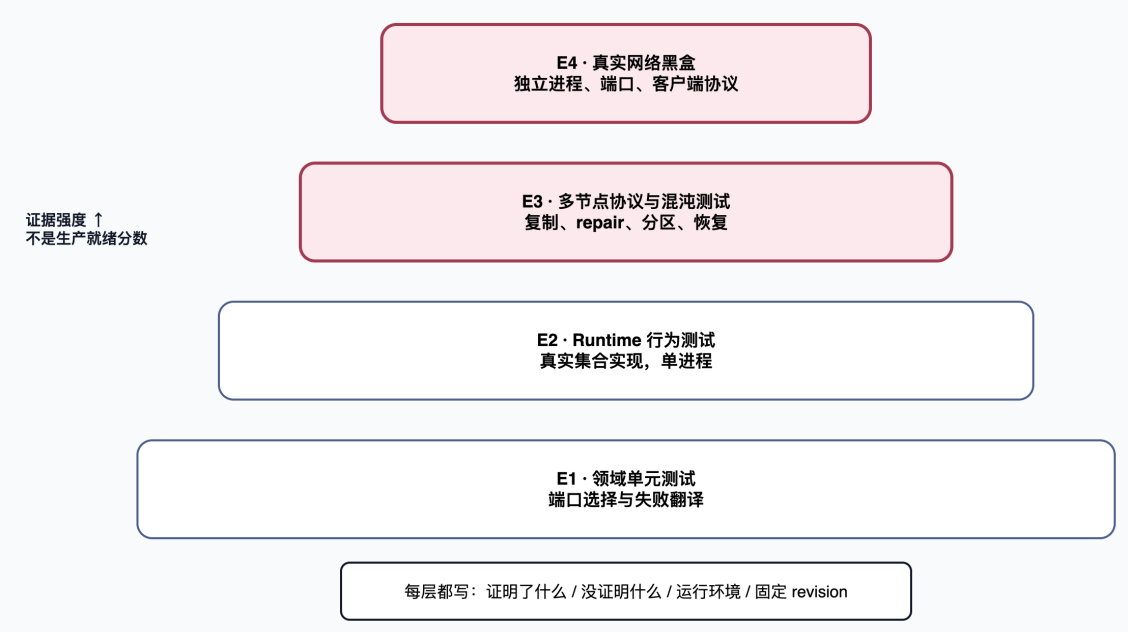

图 18: 从领域合同到真实网络黑盒的证据阶梯

等级	执行面	最适合证明	不能替代
E1	领域 fake / 静态资产	端口语言、决策表、失败翻译	真实集合与协议行为
E2	真实 Runtime，单进程	spec、codec、typed handle、生命周期	多节点传播和分区
E3	多节点协议/受控 chaos	delta、repair、分区、恢复、成员事件	独立进程和真实端口

等级	执行面	最适合证明	不能替代
E4	独立进程、真实端口、客户端协议	进程边界、协议兼容、网络可达	生产容量与故障域全貌

E4 比 E3 更接近部署现实，但依然不是生产就绪分数。生产网络拓扑、容量、权限、告警与运维流程仍要单独验收。

三类集合的最小证据组合

Register

- E1：业务把 route hint 当可重建提示，不当订单事实。
- E2：put/get/remove、resolver、ChangeStream overflow。
- E3：双节点 delta、tombstone、schema mismatch、repair。
- E4：第 23 章 Redis-shaped lab 通过真实 TCP 端口观察复制窗口。

Lease

- E1：领域结果保留 uncertain、reason 与 fencing token。
- E2：acquire/transfer 后旧 token 被 guard 拒绝。
- E3：autonomous dual holder 与 quorum fail closed。
- E4：外部受保护资源实际校验 token；本书只提供边界，不冒充现成业务系统。

CRDT

- E1：计数只用于运行观察，不作为计费事实。
- E2：单 Runtime merge 与代数性质样例。
- E3：多节点本地更新、重复/乱序与 repair 收敛。
- E4：客户端协议是否暴露该状态，取决于具体应用，不由 CRDT 单元测试证明。

证据卡需要固定四类信息

每次记录至少包含：

1. **环境与 revision**：运行在哪个源码与 JDK/Maven 环境。
2. **输入/故障**：实际注入了什么，不用「模拟异常」含糊带过。
3. **观察结果**：test 数、失败、错误、跳过及关键行为。
4. **未证明事项**：下一层仍要回答什么。

`Skipped` 和 `Tests run: 0` 不是通过；只编译成功也不是行为证据。

反例与故障注入

- 给 Maven 指定不存在的 test，并关闭 `failIfNoSpecifiedTests` 门禁，观察「构建成功但 0 test」的假绿。
- 用 fake port 的 E1 结果宣称网络 repair 通过，指出缺少的执行面。
- 用受控多节点 E3 结果宣称真实 TCP 客户端协议兼容，指出进程/端口证据缺口。
- 只写「全部通过」但不记录 revision，观察结论如何无法复现。

实验

先运行书籍自有 E1/E2：

```
cd examples/order-fulfillment-control-plane
mvn clean test
```

再运行固定源码 E2/E3：

```
cd submodule/dsm
mvn -q -pl dsm-integration-test -am \
  -Dtest=RuntimeIntegrationTest,TwoNodeIntegrationTest,ChaosIntegrationTest \
  -Dsurefire.failIfNoSpecifiedTests=false test
```

用 `evidence-ladder.json` 核对每一层的 `proves` 与 `doesNotProve` 都非空。

实验验收卡

字段	内容
运行命令	书籍示例 2 test；Runtime/TwoNode/Chaos 聚焦测试
输入或故障	fake port、真实单 Runtime、双节点、消息漏失和网络分区
可观察结果	各层绿灯与 test count 可独立复核；没有 0 test 或 skipped 假绿
证据等级	E1、E2、E3；E4 留给第 23 章真实端口实验
本实验未证明	生产容量、跨主机部署、真实外部 fencing、完整安全配置

回顾

- 测试等级描述执行面，不是质量积分。
- 每层都需要写已证明和未证明事项。
- fake、单 Runtime、多节点 chaos、真实端口不能互相替代。
- 先固定 revision 和失败输入，再谈绿色结果。

下一步

第五篇进入运行责任：第 19 章先建立 diagnostics、metrics 和 trace 观察面，让分布式差异可以被定位。

证据链接

- DSM examples：Spring Boot 应用测试
- `RuntimeIntegrationTest`
- `TwoNodeIntegrationTest`
- `ChaosIntegrationTest`

第四篇回顾——从集合 API 到业务工程（第 15–18 章）

业务层现在只依赖履约协调端口；DSM 的 locator、typed handles、Spring 装配与测试证据各自有明确归属。

工程边界

```
FulfillmentService
  → FulfillmentCoordination（业务端口）
    → DSM adapter（失败翻译）
      → composition root / Spring auto-configuration
        → Runtime + named typed handles
```

业务层可以用 fake 做快速决策测试，适配器仍需要在真实 Runtime 上验证。Spring Boot 只是另一种 composition root，不会改变 Register、Lease、CRDT 的语义。

四个不混写

不应混写	左侧职责	右侧职责
领域端口 / DSM handle	业务动作与结果	集合操作和技术失败
bean name / locator	当前容器选择 bean	分布式集合身份
clusterId/serviceId / locator	通信与同步域	集合与状态域
测试通过 / 生产就绪	固定执行面内的证据	部署、容量、安全与运维验收

本篇如何兑现 DSM 价值

第四篇把 DSM 的复用价值带入业务代码：领域端口隔离集合 API，composition root 集中 locator、codec 和生命周期，Spring Boot 负责可验证装配，多层测试限定结论范围。这样可以降低接入与演进成本，并避免 DSM 类型、身份和失败语义扩散到业务层。

可迁移检查表

1. 业务 service 是否导入了 `com.leanowtech.dsm.*`？若是，先判断它是不是 composition root 或 adapter。
2. locator 是否只有一个事实源，并带 schema id？
3. Lease 结果是否保留 uncertain、reason 和 fencing token？
4. Spring 是否对重复 locator、缺 codec/merger 和非法 Lease 配置 fail fast？
5. 集群/服务身份变更是否被当作协议迁移？
6. 每层测试是否写出 `proves` 与 `doesNotProve`？

本篇证据

- 书籍自有示例提供 E1 领域端口和 E2 真实 Runtime 测试。
- Spring auto-configuration 固定 Runtime、三类集合 bean、配置失败和生命周期。
- Runtime integration 固定 locator 隔离、CRDT 和 Lease fencing。
- TwoNode/Chaos 固定 E3 复制、repair 与分区行为。

下一步

第五篇不再扩展业务功能，而是补齐生产运行合同：可观测性、安全、生命周期和容量测量。

LEARNING ARC

让系统可以运行和排查

第 19–22 章

把诊断、安全、生命周期和容量验证写成可以检查的运行合同。

第 19 章——通过可观测信号定位副本差异

本章目标：本章完成后，开发者能够把 diagnostics、低基数 metrics 和 trace 组合成可证伪的排障假设，而不是只盯着一个「同步失败」计数器。

学习目标

1. 从 `RuntimeDiagnostics` 读取身份、状态、成员视图与集合摘要。
2. 用 outcome/reason 指标区分成员、传播、repair、Lease 和消费者问题。
3. 用 trace context 关联一次跨节点传播。
4. 避免高基数标签与无边界的诊断快照。

前置条件

- 已完成第 12 章 repair 和第 13 章背压。
- 已完成第 18 章证据分层。
- 理解诊断数据是观察，不是新的权威事实源。

案例进度

`node-b` 的 route hint 比 `node-a` 陈旧。团队不再直接重启节点，而是先确认 Runtime 是否 RUNNING、成员是否可见、目标 locator 是否注册、是否有 message drop/repair outcome，再用 trace 关联该次传播。

三个观察面组成诊断循环

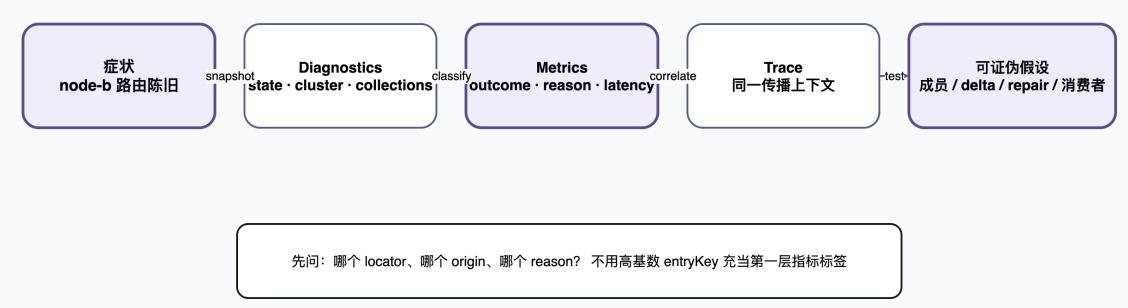

图 19: 从症状到可证伪假设的诊断循环

Diagnostics：现在是什么状态

`runtime.diagnostics()` 返回不可变快照，包括 `RuntimeInfo`、生命周期状态、cluster view、总条目数、各集合诊断、Lease 专属诊断和最后错误。它适合回答「当前这个节点看见什么」，不提供历史趋势。

Metrics：哪类结果正在累积

`DsmMetrics` 记录 sync latency、cluster size、认证失败、消息丢弃、repair outcome、Lease acquire/renew/transfer/release、fencing reject、ChangeStream overflow 等。reason 与 outcome 应保持低基数，才能用于聚合和告警。

Trace：这一次动作经过哪里

`TraceContextHolder` 让 outbound platform envelope 携带当前 trace context；两节点集成测试验证 register delta 能传播 trace。Trace 用于关联一次路径，不替代 metrics 的趋势或 diagnostics 的当前快照。

一条可执行排障路径

步骤	问题	首选信号
1	Runtime 是否准备好	`state`、`isReady`、`lastErrorMessage`

步骤	问题	首选信号
2	节点是否在正确通信域	cluster view、service/cluster mismatch metric
3	locator 是否注册且 schema 匹配	collection diagnostics、message dropped reason
4	在线 delta 是否到达	sync latency、REMOTE_DELTA、trace
5	repair 是否发现并处理差异	digest/repair outcome、replay/snapshot reason
6	只是观察者落后吗	ChangeStream overflow/callback timeout

每一步都能生成下一条可证伪假设。例如「成员正常、delta drop 增加、后续 repair success」支持在线漏失而非 locator 配错。

高基数是另一种故障

把任意 `entryKey`、异常全文或 traceId 放进 metrics tag，会让时间序列数量随业务数据增长。`DsmMetrics` 的文档明确要求稳定 reason/outcome；具体 key 和异常上下文进入结构化日志、审计或 trace，而不是第一层指标维度。

反例与故障注入

- 让 ChangeStream 容量溢出，确认集合 diagnostics 仍可显示当前条目，而 overflow metric 增长。
- 注入 50% 控制面丢失，确认后续 repair outcome 可见。
- 在 trace context 下执行 register put，确认 outbound envelope 带 trace。
- 把随机 entryKey 作为 tag 写进练习资产，观察 cardinality 审查拒绝。

实验

```
cd submodule/dsm
mvn -q -pl dsm-runtime,dsm-sync,dsm-metrics-micrometer,dsm-integration-test -am \
 -Dtest=DefaultDsmRuntimeTest,MicrometerDsmMetricsTest,TraceContextHolderTest,RuntimePlatformSyncServiceTest
 -Dsurefire.failIfNoSpecifiedTests=false test
```

使用 `diagnostic-runbook.json` 为四类症状填写 snapshot、metric、trace 与下一条假设。

实验验收卡

字段	内容
运行命令	上述 Runtime/metrics/trace/chaos 聚焦测试；书籍资产测试
输入或故障	陈旧副本、消息丢失、repair、overflow、trace context
可观察结果	当前状态、结果趋势和单次链路能互相印证但不互相替代
证据等级	E2 观察面实现 + E3 受控多节点 trace/chaos
本实验未证明	指标后端容量、告警阈值、生产 trace 采样率或值班流程有效

回顾

- Diagnostics 回答当前快照，metrics 回答趋势，trace 回答单次路径。
- 排障从 identity/locator/reason 开始，不从重启开始。
- 指标使用低基数 outcome/reason，具体 key 留给日志和 trace。
- 观察一致不等于业务事实正确。

下一步

第 20 章让每条远端消息在进入 sync/collection route 前经过准入、验签、解密、防重放和限流。

证据链接

- RuntimeDiagnostics
- DsmMetrics
- MicrometerDsmMetrics
- RuntimePlatformSyncServiceTest

第 20 章——消息安全与收敛前提

本章目标：本章完成后，开发者能够区分集群准入、真实性、机密性、防重放、滥用控制与审计，并说明密钥轮换期间的兼容窗口。

学习目标

1. 为远端 envelope 建立顺序明确的安全门。
2. 区分 HMAC 验签、载荷加密和 nonce 防重放。
3. 解释 key version、minimum acceptable version 与轮换窗口。
4. 把每类拒绝落到结构化错误、指标和审计。

前置条件

- 已完成第 11 章 wire envelope 与第 19 章观察面。
- 理解 TLS/网络隔离不自动替代消息级身份与重放保护。
- 知道安全配置错误应 fail closed。

案例进度

履约集群开始接收跨节点 delta。团队需要防止错误 token 的节点加入、载荷被篡改、旧签名长期有效、同一合法消息被重复播放，以及连续失败发送方消耗验证资源。

五道门各自回答一个问题

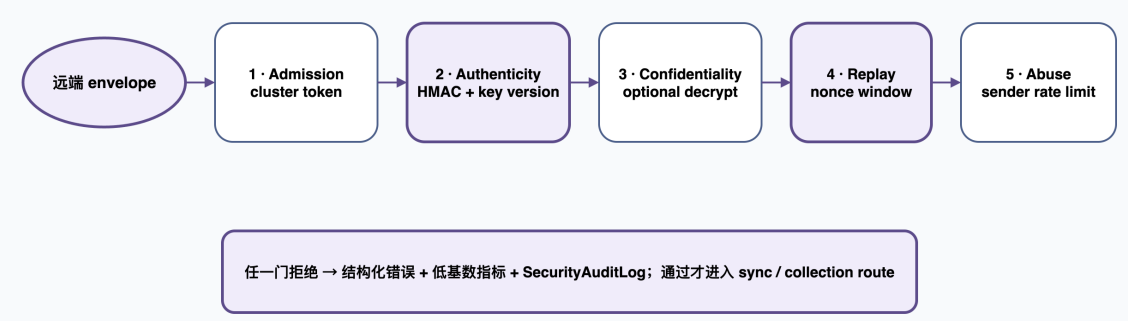

图 20: 远端消息进入集合前的安全门

门	问题	典型组件
Admission	节点是否持有集群准入凭据	`SharedKeyClusterTokenValidator`
Authenticity	消息是否由可信 key 签名且未被篡改	`HmacMessageAuthenticator`
Confidentiality	载荷是否需要并成功解密	AES-GCM / ChaCha20 encryptor
Replay	nonce 是否新鲜且未重复	`SlidingWindowNonceTracker`
Abuse	连续失败发送方是否应暂时封禁	`SenderRateLimiter`

加密不证明发送者身份，验签也不阻止合法包重放。把五类能力合并成「security enabled」会让故障和配置审查失去精度。

密钥轮换是一个窗口

安全集成测试验证 rolling migration：新旧签名可以在过渡期并存。接收方用 key version 选择验证 key；当所有发送方升级后，提高 `min-acceptable-key-version`，旧版本 envelope 才被明确拒绝。

```
阶段 A：active=1，accept >=1
阶段 B：发布 key 2，发送 active=2，接收仍 accept >=1
```

阶段 C：确认所有节点切换后，accept >=2

若一上来只保留新 key，尚未滚动到的节点会整体失联；若永不提高最低版本，旧 key 泄露风险长期存在。

Nonce 窗口与内存上限

`SlidingWindowNonceTracker` 接受递增或窗口内未见 nonce，拒绝重复、过旧和 0。它还限制 `maxTrackedSenders` 并按 LRU 淘汰，以免未知发送方无限占用内存。这个上限是安全与容量共同参数，过小会频繁遗忘发送者，过大则扩大内存面。

审计不是普通 debug log

`SecurityAuditLog` 记录 token mismatch、签名失败、重放、challenge timeout、sender ban、fencing reject 等安全结果。审计事件应包含稳定 reason、发送者和时间，不泄露 secret、明文 key 或完整敏感载荷。

反例与故障注入

- 篡改已签名 payload，预期验签失败并产生审计。
- 第二次提交同一 nonce，预期 replay rejection。
- 让接收方要求加密，再发送 plaintext，预期明确拒绝。
- 连续发送篡改消息达到阈值，预期 sender 被封禁；到期后恢复。
- 将最低 key version 提到 2，再提交 version 1 envelope，预期拒绝。

实验

```
cd submodule/dsm
mvn -q -pl dsm-security,dsm-integration-test -am \
  -Dtest='HmacMessageAuthenticatorTest,SlidingWindowNonceTrackerTest,'\
'SharedKeyClusterTokenValidatorTest,SenderRateLimiterTest,'\
'BuiltInMessageEncryptorTest,SecurityIntegrationTest' \
  -Dsurefire.failIfNoSpecifiedTests=false test
```

使用 `security-gates.json` 核对每一道门的输入、拒绝原因、指标和审计结果；不得出现 secret 值。

实验验收卡

字段	内容
运行命令	上述 security unit + integration tests；书籍资产测试
输入或故障	token mismatch、篡改、明密文不兼容、nonce replay、sender ban、旧 key
可观察结果	消息 fail closed；结构化失败、指标和审计一致；滚动轮换可兼容
证据等级	E2 密码/状态组件行为 + E3 安全集成测试
本实验未证明	密钥托管、证书体系、生产轮换流程、合规留存或网络边界已通过审计

回顾

- 准入、验签、加密、防重放、限流和审计是不同责任。
- 密钥轮换需要双版本兼容窗口与最终最低版本提升。
- nonce tracker 既要防重放，也要有发送者容量上限。
- 安全失败需要 fail closed，但不能把 secret 写入观测数据。

下一步

第 21 章把启动、注册冻结、准备就绪和关闭写成生命周期合同，防止部署系统把 STARTING 当 RUNNING。

证据链接

- `HmacMessageAuthenticator`

- `SlidingWindowNonceTracker`
- `SecurityAuditLog`
- `SecurityIntegrationTest`

第 21 章——Runtime 生命周期与流量准入

本章目标：本章完成后，开发者能够用 Runtime 状态、registry freeze、readiness 和关闭顺序定义部署生命周期，不把 Spring Context 存在当成 DSM 已就绪。

学习目标

1. 解释 `CREATED → STARTING → RUNNING → STOPPING → CLOSED`。
2. 知道集合为何在启动时冻结注册。
3. 区分 liveness、readiness 与 diagnostics。
4. 设计关闭入口、后台任务、membership 和资源释放顺序。

前置条件

- 已完成第 2 章 Runtime 生命周期和第 16 章 Spring 装配。
- 已理解第 19 章 diagnostics 与最后错误。
- 部署平台可以读取 readiness，而不只是进程存活。

案例进度

履约服务开始滚动发布。旧实例需要先停止接收新请求，再释放本地资源；新实例只有在集合冻结、membership/sync 启动并进入 RUNNING 后才接流量。否则一次部署会制造「应用已启动但协调面未准备好」的窗口。

状态机是调用许可表

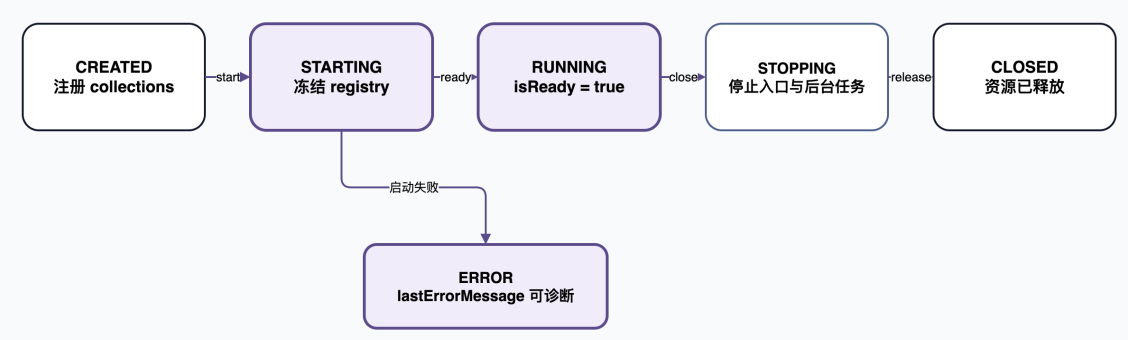

图 21: Runtime 生命周期与注册冻结

状态	允许的主要动作	部署含义
`CREATED`	注册集合、读取静态 identity	还不能接业务流量
`STARTING`	冻结 registry，启动生命周期组件	readiness 为 false
`RUNNING`	服务本地集合操作与后台协作	`isReady()` 为 true
`STOPPING`	停止入口与后台任务	从负载均衡摘除
`CLOSED`	不再使用 Runtime	资源已经释放
`ERROR`	读取 `lastErrorMessage`，执行恢复策略	不接流量，不能伪装就绪

为什么启动时冻结 registry

collection descriptor 参与 capability/descriptor 广告和 sync route。节点已经开始通信后再悄悄增加集合，会产生「部分 peer 已知、部分 peer 未知」的动态契约窗口。默认 registry 在 start 时 freeze，后续注册被拒绝；Spring 生命周期测试也确认启动后 Runtime 已 RUNNING。

若未来需要动态注册，应设计显式迁移协议，而不是绕过 freeze。

Liveness 与 Readiness 不同

- **Liveness**：进程和主线程仍活着。
- **Readiness**：Runtime 为 RUNNING，关键配置和集合已注册，可以承担本地工作。
- **Diagnostics**：解释为何不 ready，例如 ERROR 与 last error。

Spring Context 创建中、Runtime STARTING 或 ERROR 都不应对外报告 ready。`DsmRuntimeLifecycle` 在 context close 时停止 Runtime，但部署平台仍要先摘流量，不能把 JVM 关闭钩子当完整优雅下线协议。

关闭顺序的业务含义

推荐顺序：

1. readiness 置 false，阻止新流量。
2. 等待本地业务请求和短任务排空。
3. 停止订阅消费者、repair/sync scheduler 等后台入口。
4. 关闭 Runtime/membership/传输资源。
5. 记录未完成、uncertain 或需补偿的业务动作。

DSM close 释放本地资源，不会自动回滚已经提交的订单事务或外部副作用。

反例与故障注入

- Runtime start 后再次注册 collection，预期 registry freeze 拒绝。
- 在 STARTING 时把 HTTP readiness 返回 200，指出可能接收到的错误流量。
- 关闭 Spring Context，确认 Runtime lifecycle 不再 RUNNING。
- 启动失败后只检查进程 liveness，观察 ERROR 节点如何被错误纳入流量。

实验

```
cd submodule/dsm
mvn -q -pl dsm-runtime,dsm-spring-boot-autoconfigure -am \
 -Dtest=DefaultDsmRuntimeTest,DefaultDsmCollectionRegistryTest,DsmAutoConfigurationTest,DsmSpringBootSmokeTe
 -Dsurefire.failIfNoSpecifiedTests=false test
```

完成 `lifecycle-drill.json`，为每个状态写允许动作、readiness 和失败处理。

实验验收卡

字段	内容
运行命令	上述 Runtime registry 与 Spring lifecycle 聚焦测试
输入或故障	启动、冻结后注册、上下文关闭、启动失败
可观察结果	状态转换明确；late registration 被拒绝；close 后 lifecycle 停止
证据等级	E2：真实 Runtime 与 Spring ApplicationContext 生命周期
本实验未证明	部署平台 drain、真实负载均衡摘除、长事务补偿或容器强杀行为

回顾

- CREATED/STARTING 不等于可接流量，只有 RUNNING 才 ready。
- registry freeze 固定启动后的集合协议面。
- ERROR 需要 diagnostics，不应通过 liveness 混入服务池。
- 优雅关闭还需要业务入口与部署平台配合。

下一步

第 22 章建立容量测量合同：用 JMH 回答一个具体热路径问题，而不是把一次短测数字写成生产 SLA。

证据链接

- `DsmRuntimeState`
- `DsmRuntime`
- `DefaultDsmCollectionRegistryTest`
- `DsmRuntimeLifecycle`

第 22 章——从微基准到生产容量证据

本章目标：本章完成后，开发者能够选择一个具体 DSM 热路径，固定 JMH 环境和参数，保存原始 JSON，并只在可比条件下形成容量判断。

学习目标

1. 区分 benchmark harness smoke、微基准结果和生产容量。
2. 为 Register、Lease、CRDT、ChangeStream 与 security 选择对应 fixture。
3. 记录 JDK、CPU、参数、单位和原始结果。
4. 识别吞吐、延迟分布和分布式端到端性能的不同问题。

前置条件

- 已完成第 13 章背压和第 19 章 metrics。
- 能运行 Maven/JMH。
- 接受微基准不包含生产网络、GC 配置、业务载荷和依赖系统。

案例进度

履约团队想知道「DSM 能不能扛住流量」。这个问题太大，无法直接测。团队把它拆成：RecordCodec 编解码延迟、Register get、Lease renew、CRDT apply/merge、ChangeStream overflow、加密 envelope 开销等独立热路径。

容量结论的证据链

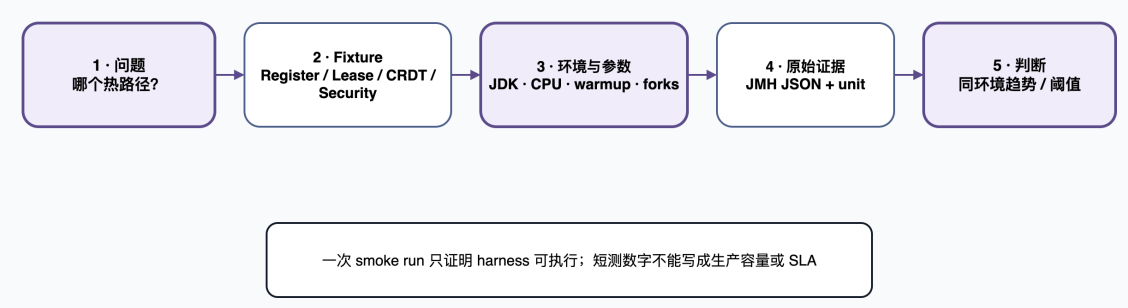

图 22: 从热路径问题到可复跑容量证据

仓库中的 benchmark module 包含：

- `RecordCodecLatency`、`ProtobufSerializationLatency`
- `RegisterGetLatency`
- `LeaseRenewThroughput`
- `CrdtApplyThroughput`、`CrdtMergeThroughput`
- `ChangeStreamOverflowPolicyThroughput`
- `SecureEnvelopeEncryptionOverhead`
- HLC、striped lock、QoS scheduler 等组件热路径

每个 fixture 只回答它命名的局部问题。

Smoke run 与测量 run

`BenchmarkHarnessTest` 只验证 request 默认值、参数校验和 JSON 输出结构。一个很短的 JMH run 可以证明 harness 可执行，但 warmup、fork、测量时间不足时，结果波动不能用于容量承诺。

可复制的 smoke 命令：

```
cd submodule/dsm
mvn -q -pl dsm-benchmark -am test
```

```
mvn -q -pl dsm-benchmark -DskipTests exec:exec \
  -Dbenchmark.include=RegisterGetLatency \
  -Dbenchmark.warmupIterations=0 \
  -Dbenchmark.measurementIterations=1 \
  -Dbenchmark.measurementTimeMs=100 \
  -Dbenchmark.forks=0
```

这条命令是 CI/教学 smoke，不是报告数字的推荐测量配置。

结果卡的必要字段

字段	示例
revision	DSM `505e7557`
环境	macOS、JDK 25、CPU/内存、功耗模式
benchmark	完整类/方法名
fixture 参数	payload、条目数、线程等
JMH 参数	warmup、measurement、forks
mode / unit	throughput 或 sample time；ops/s、ns/op 等
原始证据	JSON 文件路径与 SHA-256
解释边界	未包含网络、业务 IO、集群竞争等

只抄平均值会丢失单位、误差、分位数和运行条件。

从微基准到容量计划

微基准帮助发现热路径回归。生产容量还要叠加：实体大小与 key 分布、节点数、网络延迟/丢包、repair 流量、GC、线程竞争、安全开销、观察后端以及业务峰谷。最终压力测试需要在接近生产的独立进程环境中运行。

反例与故障注入

- 把 `forks=0`、100 ms 的 smoke 数字写成 SLA，指出缺失条件。
- 比较不同 JDK/CPU 的两个结果但不记录环境，观察结论不可复核。
- 用 `RegisterGetLatency` 推断跨节点 repair 吞吐，指出执行面不匹配。
- 只保留 Markdown 数字、删除 JMH JSON，观察原始证据丢失。

实验

运行 harness test 和一个短 smoke；将环境与参数写入 `benchmark-run-card.json`。书稿不固化短测分数，只校验结果卡字段完整。

实验验收卡

字段	内容
运行命令	`BenchmarkHarnessTest` + 指定 fixture 的 JMH smoke
输入或故障	明确 include、warmup、measurement、forks 和环境
可观察结果	harness 生成 JSON；结果带 mode/unit；命令可复跑
证据等级	E2：组件微基准 harness 与本机 smoke
本实验未证明	生产吞吐、P99 SLA、跨节点容量、峰值 repair 或成本预算

回顾

- 先问具体热路径，再选择 fixture。
- smoke 证明可执行，不证明容量。
- 结果需要绑定环境、参数、单位、原始 JSON 和 revision。
- 生产容量需要独立的端到端负载与故障测试。

下一步

第 23 章进入第六篇：用真实 TCP 端口和 Redis 客户端观察 DSM 的复制、repair 与协议边界，补上 E4 证据。

证据链接

- `BenchmarkMain`
- `BenchmarkHarnessTest`
- `RegisterGetLatency`
- `SecureEnvelopeEncryptionOverhead`

第五篇回顾——把运行责任写成合同（第 19–22 章）

可观测性、安全、生命周期和容量共同构成 DSM 的运行责任，并直接影响故障发现、隔离与恢复。

四份运行合同

合同	正确问题	最危险的替代说法
Observability	哪个 identity/locator/origin/reason 出现差异	「同步有问题，先重启」
Security	哪一道准入/验签/加密/重放/限流门拒绝	「开了 security」
Lifecycle	哪个状态允许接流量，何时冻结和关闭	「Spring Context 已起来」
Capacity	哪个 fixture、环境和参数支持这个趋势	「本机跑到一个很大的数」

本篇如何兑现 DSM 价值

第五篇把共享状态能力转化为可运行系统。统一 diagnostics、metrics、trace、安全门和生命周期减少了各服务独立建设基础运行面的工作，容量基准为热路径回归提供可比较证据。目标环境仍需确定告警阈值、密钥托管、流量排空和生产容量。

一条发布前路径

1. 检查 Runtime RUNNING、集合注册与 readiness。
2. 让 diagnostics、metrics、trace 能定位 membership/delta/repair/consumer 差异。
3. 验证 token、签名、加密策略、nonce、限流和审计结果。
4. 演练 readiness 下线、排空、Runtime close 与未完成动作记录。
5. 用固定 revision 和环境运行目标微基准；保留 JSON，不把 smoke 写成 SLA。

仍然没有证明的事

- 指标后端、告警阈值和值班流程能在真实事故中工作。
- 密钥托管、生产轮换与审计留存满足组织要求。
- 部署平台能完成真实流量排空和强杀恢复。
- 微基准能代表生产网络、repair 峰值和业务负载。

这些缺口将作为第六篇 E4 实验与最终发布判断的输入。

下一步

进入第六篇：先用 Redis-shaped 黑盒补足真实进程/端口证据，再讨论 Federation、替代方案和毕业项目。

LEARNING ARC

证明边界并完成毕业项目

第 23–26 章

用真实端口、跨集群边界、反选决策和完整证据链完成毕业项目。

第 23 章——通过真实端口观察一致性边界

本章目标：本章完成后，开发者能够用两个独立 RESP 客户端、真实 TCP/UDP 端口和 repair 诊断复现实例间传播，并解释为什么收敛不能撤回分区期间已经返回的成功。

学习目标

1. 区分 RESP 命令层、本地 projection、DSM mutation batch 和复制/修复层。
2. 用真实 socket 观察在线 delta、晚加入节点和 TCP repair。
3. 复现两个 `SET NX` 成功与重复 List pop。
4. 把 E4 黑盒证据与生产部署验收分开。

前置条件

- 已完成第 14 章分区窗口与第 18 章证据阶梯。
- 能使用 Java 25 和 Maven；手工演示可选安装 `redis-cli`。
- 接受这个示例实现 Redis-shaped 协议，不是 Redis 兼容产品。

案例进度

履约团队需要一个业务方也能看懂的黑盒实验。Node A 监听 RESP `6380`，Node B 监听 `6381`；客户端只看到 Redis 风格命令和返回值，底层通过 UDP live delta 与 TCP repair 传播 mutation batch。

一张图看懂实验边界

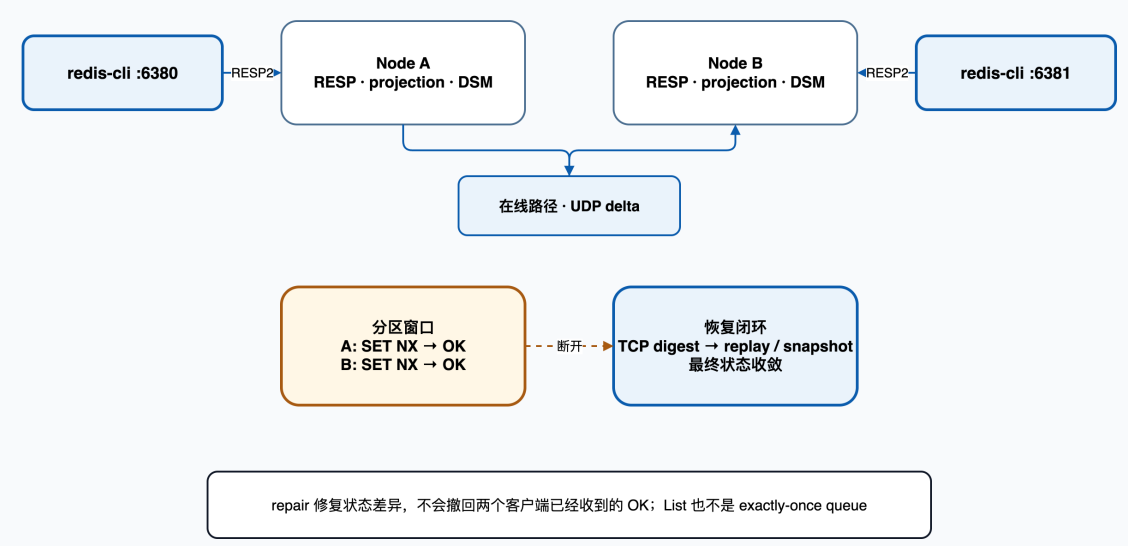

图 23: 真实 RESP 端口、分区窗口与 repair 过程

协议层接受 RESP2 bulk-string array，并限制 transaction queue 与 mutation batch 大小。projection 为 String、Hash、List、Set、TTL 和 counter 提供本地视图；DSM 复制的是确定性 mutation batch，不是把 Redis 服务器伪装成共享内存。

在线路径与修复路径

节点都在线时，后续写入通过 UDP live delta 传播。晚加入或漏掉在线消息时，节点通过 TCP data plane 交换 digest，并按可用窗口选择 replay 或 snapshot。`INFO DSM` 暴露本地 `dsm_repair_state`、最近模式、时间、peer 和错误。

这些字段是当前节点的诊断，不是全局一致查询。`digest_match` 只说明这次比较未发现差异。

分区时两个 OK 都是真的

`SET key value NX` 只检查入口节点当前 projection。隔离 A 与 B 后，两边都可能看见 key 不存在并返回 OK。恢复和 repair 会确定性收敛到一个状态，但另一个客户端已经收到的成功不能被撤回。

List 的 `LPOP/RPOP` 也依据本地 projection；分区期间两个节点可能弹出同一个元素。因此这个示例不能作为 exactly-once queue、全局 CAS 或库存扣减系统。

反例与故障注入

- 断开 live delta，在 A 与 B 对同一 key 执行 `SET NX`，记录两个返回值。
- 让两端对同一 List 元素执行 pop，确认两个客户端都可能获得该元素。
- 在 B 离线时向 A 写入 String 与 Hash，再启动 B，观察 bootstrap repair。
- 发送超过 32 KiB 的 mutation batch，确认整批拒绝而非部分应用。

实验

完整自动黑盒验证：

```
cd submodule/dsm-examples/dsm-redis-server
./mvnw -q clean verify
```

手工边界演示：

```
cd submodule/dsm-examples/dsm-redis-server
./scripts/run-consistency-boundary-demo.sh
```

使用 `redis-boundary-observations.json` 按「客户端返回、节点本地视图、repair 结果、不能推出的结论」记录观察。

实验验收卡

字段	内容
运行命令	Redis-shaped example `clean verify`；可选边界演示脚本
输入或故障	真实 RESP socket、UDP live delta、节点晚加入、分区、TCP repair
可观察结果	命令可达；在线传播与 repair 可见；两个局部成功最终收敛
证据等级	E4：独立 socket/真实端口/客户端协议黑盒；仍在本机 loopback
本实验未证明	Redis 完整兼容、跨主机网络、生产安全、持久化、全局 CAS 或 exactly-once

回顾

- 真实端口提高执行面证据强度，但不会改变集合语义。
- online delta 与 repair 是两条互补路径。
- repair 让状态重新收敛，不会撤回已返回的业务成功。
- familiar protocol 只是观察入口，不能偷换成 Redis 产品承诺。

下一步

第 24 章把范围扩大到两个 cluster，并逐一说明 Register、Lease 和 CRDT 穿过 federation 边界时保留了什么、放弃了什么。

证据链接

- Redis-shaped 示例边界
- `RealNetworkBlackBoxTest`
- `DsmRedisNodeIntegrationTest`

- 一致性边界演示脚本

第 24 章——跨集群传播的集合语义

本章目标：本章完成后，开发者能够为 Register、Lease 和 CRDT 分别描述 federation 的传播、恢复与所有权边界，并拒绝「跨区域 DSM 自动成为全局强一致」的说法。

学习目标

1. 用 `FederationTarget` 明确 remote cluster、service 与 locator。
2. 区分三类集合的 federation 语义。
3. 理解 relay health 与 binding status。
4. 识别 in-process baseline 与生产跨区域 transport 的差距。

前置条件

- 已完成第 9 章集合选型、第 12 章 repair 和第 20 章安全门。
- 理解 cluster 内复制与 federation 是不同故障域。
- 已接受跨集群链路需要独立安全、容量和恢复验证。

案例进度

履约团队在两个区域运行独立 cluster。route hint 需要跨区可见，shard owner 只能跨区观察，request counter 可以合并。团队不能用一套模糊的「同步策略」覆盖三种不变量。

三类集合穿过同一边界，语义不同

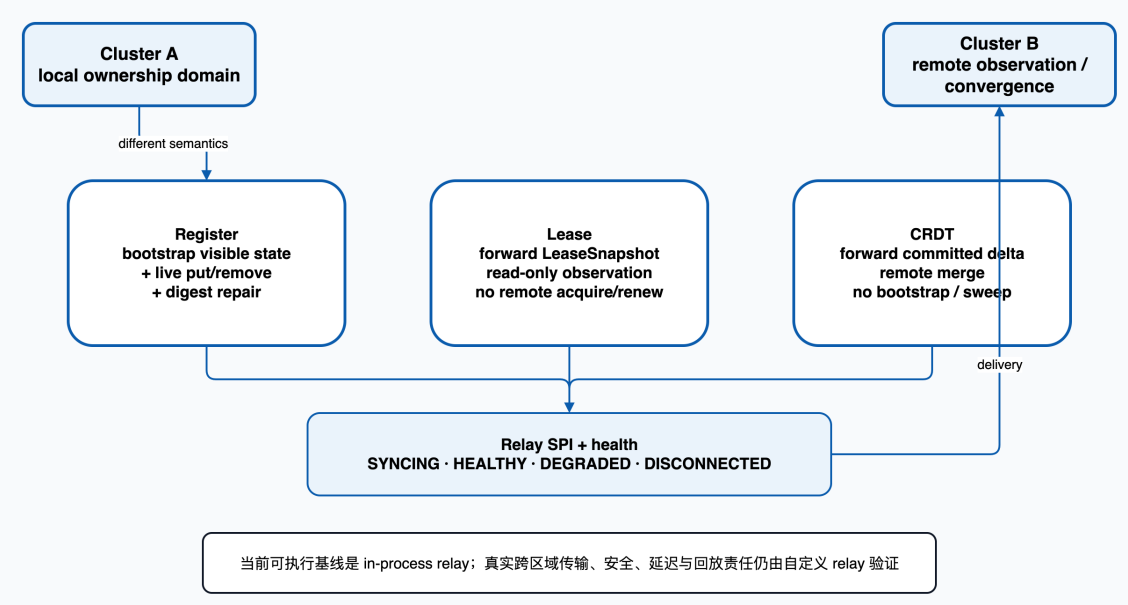

图 24: Register、Lease 与 CRDT 的 federation 语义

类型	初始状态	实时变化	补差路径	跨区边界
Register	bootstrap 当前可见快照	forward put/remove	周期 digest repair；必要时重新 bootstrap	最终收敛，不是跨区事务
Lease	无 bootstrap	forward `LeaseSnapshot`	无 anti-entropy sweep	远端只读观察，不能 acquire/renew
CRDT	无 bootstrap	forward committed delta	无 federation sweep	依赖 delta 可合并、幂等与最终送达

Lease 尤其不能写成「跨区所有权」。owner 与 fencing 的权威域仍在原 cluster；远端 snapshot 适合路由或运维观察，不是外部副作用许可。

Target 是完整身份，不是一个 URL

`FederationTarget` 由 `remoteClusterId`、`remoteServiceId` 和 `remoteLocator` 组成。relay 按 cluster/service 找 runtime，再按 locator 找 collection。任何一层配错都应显式失败，不能把同名 collection 当作同一状态。

Health 决定传播信心

binding 状态为 `SYNCING`、`HEALTHY`、`DEGRADED` 或 `DISCONNECTED`。relay failure 时停止转发；恢复后 Register 触发 catch-up bootstrap。这个状态描述 relay/binding 健康，不代表两地业务事实已经完成全局确认。

当前实现最重要的边界

`InMemoryFederationRelay` 在一个进程内连接独立 Runtime。它能执行真实 bridge 语义并验证三类集合，却没有 gRPC/HTTP、跨主机链路、身份认证、重试预算或生产延迟。生产 relay 需要实现 bootstrap、put/remove、repair、Lease snapshot 与 CRDT delta，并单独验证第 20–22 章的安全和容量合同。

反例与故障注入

- 将 target 的 serviceId 写错，确认找不到 remote runtime/collection。
- 在 relay failed 时写 Register，恢复后确认排队或 bootstrap 补齐。
- 尝试从远端 snapshot 推断 Lease acquire 权限，指出违反只读边界。
- 丢弃 CRDT delta，说明当前 federation 无 CRDT anti-entropy，不能假定自动补齐。

实验

```
cd submodule/dsm
mvn -q -pl dsm-federation,dsm-integration-test -am \
  -Dtest=InMemoryFederationBridgeTest,FederationIntegrationTest \
  -Dsurefire.failIfNoSpecifiedTests=false test
```

使用 `federation-contract.json` 核对每种集合的 bootstrap、live delivery、repair 与 ownership 边界。

实验验收卡

字段	内容
运行命令	federation unit + integration tests；书籍资产测试
输入或故障	bootstrap、live mutation、relay failure/recovery、Lease snapshot、CRDT delta
可观察结果	Register 补差；Lease 只读观察；CRDT 合并；status 随 relay 改变
证据等级	E2/E3：两个独立 Runtime + in-process relay
本实验未证明	真实跨区域 transport、安全、延迟、带宽、完整历史或全局事务

回顾

- Federation 是跨故障域复制合同，不是把 cluster 拉长。
- Register、Lease、CRDT 需要保留各自语义。
- remote target 身份包含 cluster、service 和 locator。
- 当前 relay 是可执行基线，不是生产网络实现。

下一步

第 25 章反过来问：哪些状态不应进入 DSM，以及数据库、缓存、事件日志和共识服务何时更合适。

证据链接

- `dsm-federation` 边界
- `FederationBridge`
- `InMemoryFederationBridgeTest`
- `FederationIntegrationTest`

第 25 章——识别不适合 DSM 的状态

本章目标：本章完成后，开发者能够从业务不变量、失败后果和证据要求反选技术，不因低延迟、分布式或「共享」三个词默认选择 DSM。

学习目标

1. 先判断状态是否可修复，再讨论集合类型。
2. 区分事实源、缓存、历史日志、共识协调与 DSM。
3. 为采用或拒绝 DSM 写出可审查理由。
4. 识别双写、无限状态和不可观测修复三类反模式。

前置条件

- 已完成三类集合、分区边界、容量和 federation。
- 能写出状态丢失、重复、乱序和分歧的业务后果。
- 愿意把「不用 DSM」视为成功的设计结论。

案例进度

履约团队重新评审订单、库存扣减、route hint、shard owner、request counter 和审计事件。评审目标是确认每种状态的正确责任边界，不以扩大 DSM 覆盖范围为成果。

先反选，再选集合

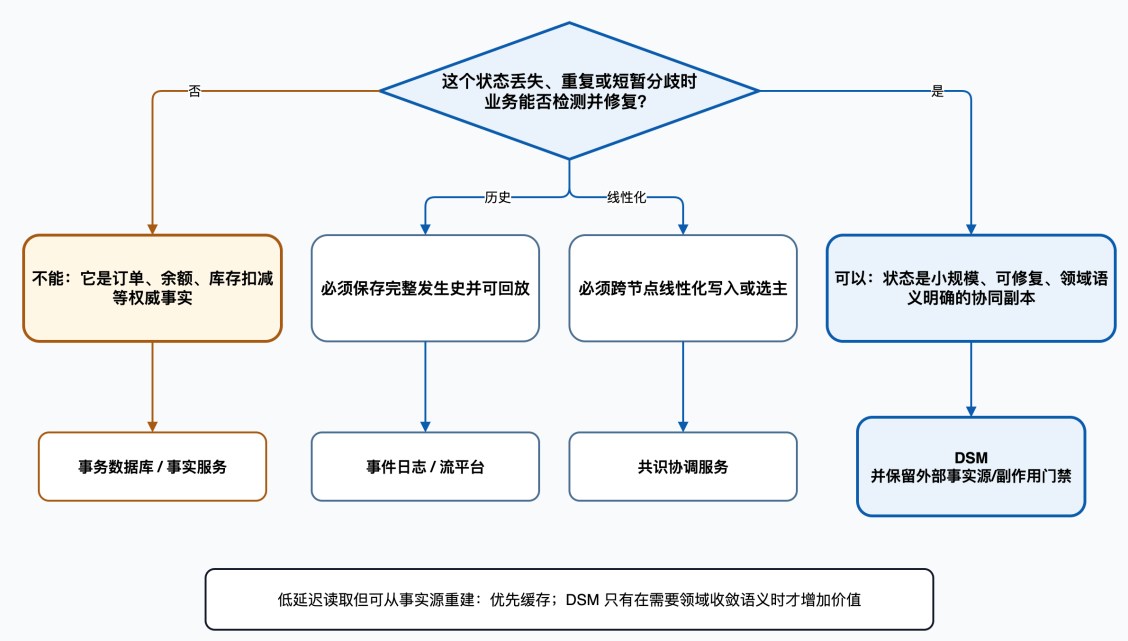

图 25: DSM 反选决策树

需求	首选责任	为什么不是 DSM 的主责任
订单、余额、库存扣减等权威事实	事务数据库或事实服务	已确认事务不能被最终收敛重写
完整、可重放的发生历史	事件日志或流平台	DSM 以当前协同状态为中心，不承诺完整历史
跨节点线性化配置、选主或强 CAS	共识协调服务	DSM 分区下可能允许局部成功
只需低延迟读且可从事实源重建	缓存	不需要引入领域合并、repair 与协议身份

需求	首选责任	为什么不是 DSM 的主责任
小规模、可修复、领域语义明确的协同状态	DSM 候选	可用 Register、Lease、CRDT 表达不变量

「候选」还不是批准。仍需回答状态规模、TTL/清理、schema 演进、失败可观测性、repair 成本、安全、容量和团队运维能力。

三个应当直接拒绝的信号

1. **把 DSM 和数据库都当 writer**：没有单一权威和补偿协议的双写，只会制造另一层分歧。
2. **状态无界增长**：把完整订单历史、日志或分析明细塞进内存集合，生命周期与容量不可控。
3. **无法解释失败窗口**：只说 eventual consistency，却说不出两个 OK、旧 token、丢 delta 或慢消费者的业务处理。

贯穿案例的最终边界

- `route-hints`：Register 候选；事实来源仍是健康检查/服务管理系统。
- `shard-owners`：Lease 候选；外部写入需要校验 fencing token。
- `request-counter`：PN-Counter 候选；用于可合并观测，不用于财务结算。
- 订单与库存：拒绝 DSM 作为事实源。
- 审计事件：进入不可变历史系统，不能只依赖 ChangeStream。

反例与故障注入

- 把库存余额改为 Register，并制造分区双写；说明收敛后的 winner 仍会丢失业务扣减。
- 把 ChangeStream 当审计日志，触发 overflow；指出观察事件可丢与历史完整性的冲突。
- 把 CRDT counter 当账单金额，说明重复来源、撤销和法律记录无法由局部 merge 自动解决。
- 为纯缓存数据增加自定义 resolver，比较它与从事实源重建的复杂度。

实验

```
npm test -- --test-name-pattern="phase 6 assets"
```

在 `technology-selection-review.json` 中对八类状态逐项写出 invariant、failure cost、source of truth、选择与反证条件。

实验验收卡

字段	内容
运行命令	书籍静态决策资产测试
输入或故障	事实、历史、强协调、缓存、可修复协同状态
可观察结果	每项都有采用/拒绝理由和改变决定的反事实条件
证据等级	E1：设计评审；需与目标系统运行证据结合
本实验未证明	某个替代产品已经满足 SLA、安全、成本或团队能力

回顾

- 先问失败是否可修复，再问用哪种集合。
- DSM 不拥有订单事实、完整历史或全局线性化承诺。
- 可从事实源重建的低延迟读取问题通常先按缓存需求评估，不能仅因涉及多个实例就引入一致性框架。
- 拒绝 DSM 需要像采用一样给出可复核理由。

下一步

第 26 章把全书方法收敛成一个可运行、可故障注入、可观察且边界诚实的毕业项目。

证据链接

- 贯穿案例设计
- 集合选型章节
- 一致性承诺边界
- 容量证据合同

第 26 章——交付可复核的履约控制面

本章目标：本章完成后，开发者能够运行书籍自有履约控制面，串起三类集合、故障实验、观察信号与证据等级，并作出带边界的发布判断。

学习目标

1. 从业务合同追到 DSM adapter，而不是让业务层依赖集合 API。
2. 同时验收 Register、Lease、CRDT 的正常路径和失败窗口。
3. 用一条脚本复跑书籍、示例、观测、安全、生命周期、federation、真实端口与出版证据。
4. 输出「可以发布什么」和「仍需部署验证什么」。

前置条件

- 已完成第 1–25 章。
- Java 25、Maven/Wrapper 与 Node.js 22+ 可用。
- DSM `0.1.2` 已安装到本地 Maven repository，或先执行完整 DSM 构建。

案例进度

履约控制面拥有三个业务动作：发布健康 route、领取 shard、累计接受请求。业务服务只依赖 `FulfillmentCoordination`；`DsmFulfillmentModule` 在 composition root 内拥有 Runtime、locator、codec 和 typed handles。

验收的对象是一条链

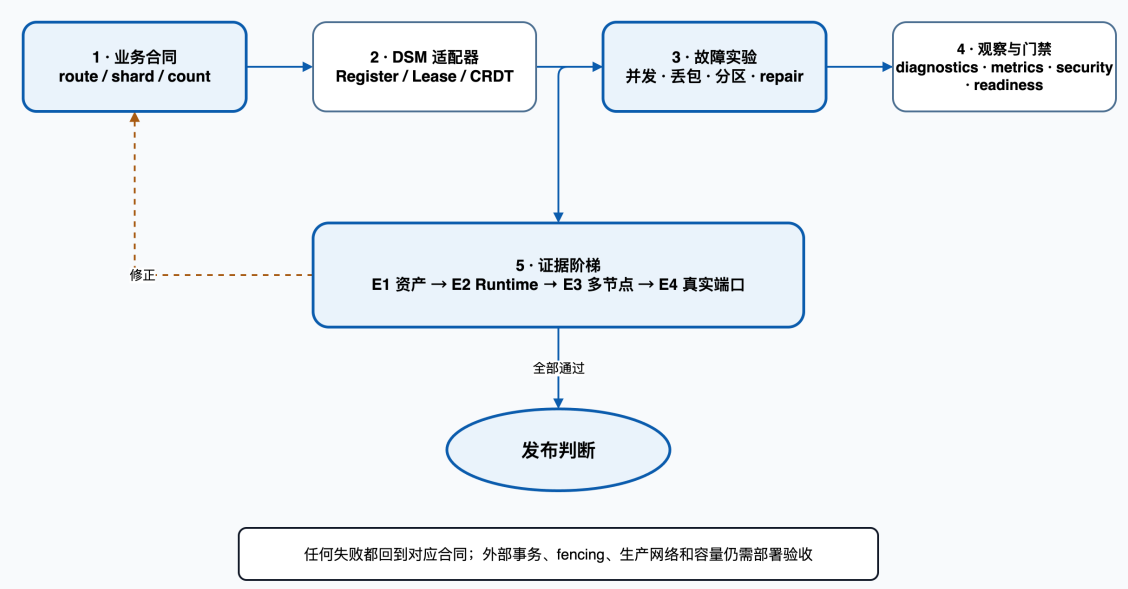

图 26: 毕业项目的合同、故障与证据链

层	验收问题	失败时回到哪里
业务合同	调用者是否只看到 route、claim、count 与 uncertain/fencing	领域端口与业务不变量
集合适配	Register/Lease/CRDT 是否各自表达正确状态	locator、codec、集合选择
故障实验	并发、分区、漏失、repair 的业务结果是否写清	失败合同与外部门禁
运行责任	能否诊断、拒绝不可信消息、正确 ready/stop	Arc 5 运行合同

层	验收问题	失败时回到哪里
证据等级	每个声明在哪个执行面被验证	追踪矩阵与验收命令

DSM 价值的最终兑现

毕业项目把全书价值压缩为一条可运行路径：业务层通过领域端口表达发布路由、领取分片和累计请求，DSM adapter 复用 Register、Lease、CRDT、传播与修复能力，测试和观察面说明这些机制在何种执行环境下成立。业务团队可以减少在各个服务中重复实现同类协调基础设施的工作，同时继续负责订单事实、外部 fencing、分区期间的响应策略以及目标环境验收。

这条路径也是全书学习结果的判据。开发者需要能够从业务不变量选择或拒绝 DSM，运行正常与故障实验，解释观察结果，并把未覆盖范围转成后续部署任务。

正常路径不是最终验收

书籍 Maven 示例验证领域 fake 与真实 standalone Runtime；DSM 集成测试验证多节点传播、repair、安全和 federation；Redis-shaped 示例验证真实端口。三层证据组合后仍不能宣称生产网络、外部事务 fencing、部署排空或容量 SLA 已完成。

一键验收

```
node scripts/verify-capstone.mjs
```

脚本串行运行，避免多个 Maven reactor 共享 `target` 造成无效并发失败。它会停止在第一条失败命令，并打印每个证据层的名称；最后重新导出 Draw.io 图、构建在线书和中文 PDF，确保内容与出版物属于同一候选状态。

发布判断模板

可以确认：书籍固定 revision 上，业务端口隔离成立；三类 collection 能运行；受控多节点传播/repair、安全/federation 测试与 loopback RESP 黑盒通过；章节资产、链接、图稿和出版构建可复跑。

仍需部署验收：生产 topology、跨主机网络、外部存储 fencing、指标后端和值班、密钥托管、真实 drain、数据规模、峰值 repair、容量与灾难恢复。

任何一项「仍需」都需要有 owner、环境、命令/探针、阈值与失败处置，不能用本书绿灯代替。

反例与故障注入

- 修改 asset 却不更新 manifest，或让业务层直接返回 `DsmRegister`，确认门禁分别拒绝证据漂移与技术类型泄漏。
- 把 Redis loopback 写成生产跨主机结论，或并行运行共享 `target` 的 Maven reactor，确认边界审查与串行门禁能阻止假绿。

实验

1. 运行 `node scripts/verify-capstone.mjs`。
2. 对照 `capstone-acceptance.json` 检查每项 claim、evidence、boundary。
3. 为目标部署另建 acceptance card；书籍验收卡不能替代生产完成证明。

实验验收卡

字段	内容
运行命令	`node scripts/verify-capstone.mjs`
输入或故障	三类集合、领域端口、repair/security/lifecycle/federation、真实 RESP socket 与出版链路
可观察结果	所有固定证据串行通过；失败命令非零退出；边界卡完整
证据等级	E1–E4 组合，按声明选择，不以最高等级覆盖全部结论
本实验未证明	目标组织和生产环境已经完成上线批准

回顾

- 交付物是业务合同、运行实现、故障实验、证据和边界组成的链。
- 一键脚本提高可复核性，不自动提高证据等级。
- 任何发布判断都需要绑定 revision、环境和未证明事项。
- 全书完成的判据包括模块可构建，以及开发者能够独立复现实验并作出边界一致的判断。

下一步

使用附录快速查询 API、Spring 属性、排障路径、术语和证据；当 DSM revision 改变时，从附录 E 反向定位需要复核的章节。

证据链接

- DSM examples：Basic Usage 示例
- Capstone 验收脚本
- 内容事实追踪矩阵
- 可运行示例矩阵

第六篇回顾——用边界和证据毕业（第 23–26 章）

DSM 的价值在于为适合共享的协同状态提供明确语义、恢复路径和证据边界；权威业务事实仍由相应系统负责。

四个毕业判断

1. **协议判断**：真实端口证明路径可达，但 Redis-shaped 不等于 Redis 产品，也不改变分区语义。
2. **范围判断**：federation 为三类集合保留不同合同，不创造全局事务或跨区 Lease 所有权。
3. **反选判断**：权威事实、完整历史、强协调和纯缓存分别回到更合适的系统。
4. **发布判断**：证据链按 claim 选等级，明确当前通过与部署待验收。

本篇如何兑现 DSM 价值

第六篇通过真实端口、federation、技术反选和毕业项目检验 DSM 的价值边界。完成这些实验后，团队可以说明哪些一致性工程已经由 DSM 复用，哪些业务与生产责任仍在外部，并据此作出采用、拒绝或继续验证的决定。

贯穿案例最终映射

状态	最终选择	保留的外部责任
route hint	Register	健康事实来源、TTL/清理、错误路由处置
shard owner	Lease	外部 fencing、uncertain 处理、分区策略
request counter	PN-Counter	仅作可合并观测，不作结算
订单与库存	拒绝 DSM 为事实源	事务、审计、补偿
跨区协同	按集合选择 federation	真实 relay、安全、容量与恢复

可验证的全书学习结果

- 能从业务不变量选择 Register、Lease、CRDT 或拒绝。
- 能解释 local commit、remote visibility、repair 与业务成功的差异。
- 能用 diagnostics/metrics/trace 定位问题，并控制指标基数。
- 能写出 security、lifecycle、capacity 与 federation 的诚实边界。
- 能运行 capstone，并把本书证据与目标部署验收分开。

全书最后一句

分布式共享内存不把远端访问伪装成本地访问。DSM 通过明确的集合语义、恢复过程和证据边界，使不可避免的分歧可以被业务理解、由工程系统处理，并通过实验复核。

附录 A ——Runtime 与集合 API 速查

本附录用于定位入口，不替代各章的语义、失败窗口和证据边界。API 以 DSM `505e7557` / `0.1.2` 为准。

Runtime

动作	入口	关键约束
构建	`DsmRuntimeBuilder.builder()`	固定 `clusterId`、`serviceId`、membership 与安全/观测扩展
注册 Register	`runtime.register(spec)`	在 `start()` 前完成；locator 不重复
注册 Lease	`runtime.leaseRegister(spec)`	需要 codec、entity factory 与 Lease policy
注册 CRDT	`runtime.crdt(spec, merger)`	需要 update/state codec、initial state 与 merger
启动	`runtime.start()`	registry freeze；成功后进入 `RUNNING`
诊断	`runtime.diagnostics()`	节点本地不可变快照，不是全局一致视图
就绪	`runtime.isReady()`	只有 `RUNNING` 为 true
关闭	`runtime.close()`	释放本地资源，不回滚业务副作用

集合公共身份

```
CollectionLocator = tenantId / applicationId / collectionId
CollectionSpec = locator + schemaId/fingerprint + codec + consistency tier
```

同名 `entryKey` 只有在相同通信域与 locator 内才属于同一逻辑集合。schema 不兼容应拒绝，而不是尝试猜测解码。

Register

操作	含义	不保证
`put(entity)`	本地提交当前候选值	返回时远端已可见
`get(key)`	当前节点可见值	线性化全局读取
`remove(key)`	本地提交删除	所有 peer 同步删除完成
`all()` / `size()`	本地 projection 查询	全局精确快照
`changes(options)`	有界观察流	不丢事件的历史日志

冲突 resolver 需要确定；若依赖输入顺序，应明确拒绝或证明交换律。

Lease

操作	关注结果
`acquire`	`granted`、`uncertain`、holder、expiry、fencing token、reason
`renew`	token/holder 仍有效，成员与模式允许
`transfer`	新 owner 与更高 fencing token
`release`	释放本地协调状态，不撤销已发生外部写入

外部资源需要校验 fencing token。AUTONOMOUS 分区模式可能出现临时双 holder；QUORUM 模式在成员不稳定或 quorum 不足时应 fail closed。

CRDT

动作	含义
`apply(update)`	把本地 update 合入 state 并生成可传播 delta
`state()`	当前节点的合并状态

merger 应满足交换、结合和幂等。PN-Counter 适合可合并运行计数，不自动成为账务事实。

常用构建器

- `CollectionSpecBuilder.register(...)`
- `LeaseCollectionSpecBuilder.lease(...)`
- `CrdtCollectionSpecBuilder.crdt(...)`

先设置稳定 locator 和 `schemaId`，再提供 typed codec；spec 由集中装配边界统一构建，避免散落在业务服务中。

源码入口

- `DsmRuntime`
- `CollectionLocator`
- `DsmRegister`
- `DsmLeaseRegister`
- `DsmCrdtCollection`

附录 B ——Spring Boot 配置速查

属性入口为 `dsm.*`，绑定类是 `DsmProperties`。默认值用于本地起步，不是生产推荐值；部署前需要结合网络、安全、容量和故障域复核。

顶层身份

属性	作用	门禁
`dsm.cluster-id`	集群身份	必填；空值启动失败
`dsm.service-id`	服务家族/通信隔离	与 peer 保持一致；不承担租户 ID 的职责
`dsm.node.id`	节点身份	默认随机 UUID；稳定诊断场景建议显式配置
`dsm.node.host` / `port`	广告地址	需要可被目标网络中的 peer 访问

Membership

属性组	关键项	说明
`dsm.cluster.mode`	`STANDALONE` / `MULTICAST` / `UNICAST`	决定 membership 实现
`dsm.cluster.multicast.*`	group、port、heartbeat、failure threshold	受网络 multicast 能力约束
`dsm.cluster.unicast.*`	gossip port/interval、seeds/DNS、failure threshold	seed 只是发现入口，不是永久 leader

高风险 Lease 决策还要检查 quorum 与 stability rounds；节点数量本身不足以支持所有权判断。

Runtime 与 Sync

`dsm.runtime.*` 管理生命周期/线程等 Runtime 行为，`dsm.sync.*` 管理传播、repair、batch/QoS 等同步参数。修改 timeout、queue、batch 或 repair 周期时，同时更新容量实验和故障演练，不能只改 YAML。

Security

属性面	要回答的问题
token/challenge	谁能加入通信域
authentication/key version	谁签名，接受的最低版本是什么
encryption	接收方是否强制密文，使用什么算法/key version
nonce window/senders	重放窗口与 sender 内存上限
rate limit/ban	连续失败如何限流与恢复

secret 应来自部署密钥系统，不进入书稿、日志、metric tag 或仓库配置。

Collections

每个 `dsm.collections[]` 至少需要：

```
- bean-name: routeHintsCollection
  tenant-id: shared
  application-id: gateway
  collection-id: route-hints
  schema-id: route-hints/v1
  type: REGISTER
```

```
  consistency-tier: REGISTER
  entity-type: com.example.RouteHint
```

类型	额外要求
Register	record 可派生 codec；非 record 需要 `codec-bean`
Lease	entity factory、Lease mode/TTL/quorum 配置
CRDT	update codec、state codec、initial state 与 merger bean

规范 bean 名为 `dsmCollection:<tenant>/<application>/<collection>`；`bean-name` 是额外业务别名。重复 locator、缺 codec/merger/factory 或非法范围会 fail fast。

验证入口

```
cd submodule/dsm
mvn -q -pl dsm-spring-boot-autoconfigure -am \
  -Dtest=DsmAutoConfigurationTest,DsmSpringBootSmokeTest \
  -Dsurefire.failIfNoSpecifiedTests=false test
```

- `DsmProperties`
- 第 16 章配置资产

附录 C ——排障决策树

先保留现场，再按 identity → lifecycle → membership → route → delta → repair → observer 的顺序缩小范围。重启会改变现场状态，不适合作为第一条诊断动作。

入口决策

症状	先看	下一条假设
应用活着但请求失败	`state`、`isReady`、`lastErrorMessage`	Runtime 未 RUNNING 或启动配置失败
peer 完全不可见	cluster/service、membership view、端口	通信域/seed/网络不一致
只有某集合不同步	locator、schema、collection diagnostics	locator 或 fingerprint 不匹配
在线更新偶尔丢失	message dropped reason、sync trace	delta 路径丢失，等待/触发 repair
repair 后仍不同	digest、replay window、snapshot result	来源选择、倒退保护或 handler 失败
Lease 拒绝	mode、membership stability、quorum、token	成员不稳定、quorum 不足或旧 token
订阅者缺事件	overflow/callback timeout	观察者落后，不一定是集合状态缺失
远端消息被拒绝	admission/auth/decrypt/replay/rate limit	对应安全门配置或攻击行为
Federation 不健康	target identity、relay status、binding status	remote locator 缺失或 relay 不可达

快速检查顺序

1. 记录 revision、节点 identity、时间窗口和具体 locator。
2. 读取 diagnostics 快照；在保存证据前不清空错误或重启。
3. 检查低基数 metrics 的 operation/outcome/reason。
4. 用 trace 关联一条具体操作；traceId 进入日志或 trace，不进入 metric tag。
5. 复核安全审计，但不得输出 token、key 或敏感 payload。
6. 对照 peer capability/schema 后再决定 replay、snapshot 或配置修正。
7. 写明恢复后哪些业务成功无法撤回、哪些外部副作用需要补偿。

常见假绿

- `Tests run: 0`。
- 测试被 skipped，但汇总只看退出码。
- compile-only 被写成行为验证。
- in-process fake 被写成真实网络。
- 本地 `get` 成功被写成远端可见。
- 收敛被写成分区期间没有双成功。
- JMH smoke 数字被写成生产 SLA。

最小证据记录

```
revision:
environment/topology:
claim:
command/probe:
observed result:
does not prove:
next action / owner:
```

关联章节

- 第 12 章：repair
- 第 19 章：诊断循环
- 第 20 章：安全门
- 第 21 章：生命周期
- 第 23 章：真实端口实验

附录 D ——术语与易混边界

术语	本书含义	不应混写为
DSM	面向可修复协同状态的分布式共享内存基础库	分布式数据库替代品
Runtime	拥有 identity、registry、lifecycle 与协作组件的 `DsmRuntime`	JVM 进程本身
locator	`tenantId / applicationId / collectionId`	只有 collection name
Register	每个 key 的当前可见值，经确定性 resolver 收敛	线性化 KV
Lease	有期限所有权、模式和 fencing token	分布式事务锁
CRDT	依赖交换、结合、幂等 merge 的可合并状态	自动正确的任意业务数据
local commit	本地路径接受并更新 projection	已在所有 peer 可见
live delta	正常在线传播路径	实时强同步
repair	digest 后用 replay/snapshot 修复当前副本	完整业务历史重放
convergence	恢复后副本得到等价可见状态	分区期间只有一个成功
membership	当前节点观察到的成员与状态	全局瞬时真相
fencing token	外部资源拒绝旧 owner 的单调令牌	DSM 自动阻止全部外部写入
ChangeStream	有界状态变更观察流	不丢失的审计日志
diagnostics	当前节点的不可变运行快照	业务权威事实
Federation	跨 cluster 的集合语义传播	全局事务或跨区单一 Lease owner
E1–E4	从静态资产到真实端口的执行面等级	质量分或生产成熟度等级

五组关键边界

- `put` 返回确认本地提交；远端可见需要独立证据。
- repair 重新建立状态收敛；它不撤回已返回的 OK。
- Lease 提供 token；外部资源负责校验 token。
- 测试证明给定 revision、环境、拓扑和断言；它不自动证明生产就绪。
- Federation 传播集合语义；它不创造跨区域事务。

完整编辑术语规范见 项目术语表。

附录 E ——能力与证据索引

固定 revision

来源	revision	基线验证
DSM	`505e7557`	完整 Maven suite
DSM Examples	`0eca40d`	Basic 与 Redis-shaped suite

表中使用短 revision；完整 SHA 见内容事实追踪矩阵。revision 改变后，本索引进入待复核；不得沿用旧测试数量或结论。

能力索引

能力	章节	运行证据	等级与开放边界
Runtime	2–3、21	Runtime tests	E2 单进程
两节点 Register	4–6	TwoNode test	E3 受控多节点
Lease/fencing	7、14	Lease/chaos	E2/E3；外部 fencing
CRDT	8	Runtime integration	E2；传播见后文
membership	10、17	cluster/Multicast	E2/E3；实验网络
delta/repair	11–12	sync/planner/chaos	E3 repair
ChangeStream	13	buffer/Register tests	E2；非审计历史
Spring	16	auto-config/Basic	E2 app context
观察面	19	metrics/trace/chaos	E2/E3；后端开放
安全	20	security tests	E2/E3；密钥托管
benchmark	22	JMH smoke	E2；非容量 SLA
RESP 黑盒	23	Redis `clean verify`	E4 loopback 端口
Federation	24	bridge tests	E2/E3；进程内
业务适配器	15、26	book example	E1/E2

本索引使用短证据标签控制出版宽度；完整测试类与命令保留在内容事实追踪矩阵和对应章节中。

一键入口

执行发布门禁：

```
node scripts/verify-capstone.mjs
```

它串行验证书籍合同、书籍自有示例、Federation 与 Redis-shaped 黑盒。完整发布候选还需执行根项目最终审查中列出的 DSM、Basic、Redis、图稿、在线书和 PDF 全量门禁。

控制面链接

- 内容事实追踪矩阵
- 可运行示例矩阵
- 质量差距账本
- 逐章目标地图