【SSO 重构】数据层实现现状调研

实现现状细节调研：

repository 层既读又写的方法

其中 **47%** 的接口可能需要在方法内部进行细粒度地控制读写，需要评估：

- 【复杂业务逻辑】在 repository 引入了较为复杂的业务逻辑从而导致既读又写的操作

可能需要作为特例单独处理

- 这三个方法存在互相调用，需要一起处理，这里直接把逻辑全部抽到 proxy 层重新实现，细粒度地控制读写

- com.xiaohongshu.fls.skywalker.account.cas.gateways.persistence.ConfigRepositoryImpl#findConfigByType

com.xiaohongshu.fls.skywalker.account.cas.gateways.persistence.ConfigRepositoryImpl#add()

com.xiaohongshu.fls.skywalker.account.cas.gateways.persistence.ConfigRepositoryImpl#add(java.util.List<com.xiaohongshu.fls.skywalker.account.cas.gateways.persistence.entity.Config>)

- 这个方法调了 service 层，仍然按照【无完整数据】的方法进行处理了，对 service 层的调用最终会回到 adaptor 去控制读写源，所以这里不需要抽出

com.xiaohongshu.fls.skywalker.account.cas.gateways.persistence.AccountLockdownRepositoryImpl#unlockAccountByToken

- 这个方法按照【无完整数据】的方法进行处理即可

com.xiaohongshu.fls.skywalker.account.cas.gateways.persistence.AccountLockdownRepositoryImpl#resendUnlockToken

- 这是个 private 方法，被大部分 user repo 中的方法依赖，其中的写逻辑是 —— 对 created_at 不存在的记录更新 created_at 到当前时间，在 mysql 的逻辑中不需要实现。

com.xiaohongshu.fls.skywalker.account.user.gateways.persistence.UserRepositoryImpl#query

- 同上

com.xiaohongshu.fls.skywalker.account.user.gateways.persistence.UserRepositoryImpl#batchQuery

- 【无完整数据】传入部分字段作为查询条件进行增删改的操作

proxy 层迁移中依赖新接口，将这种接口转换为【多余操作】的接口，约定迁移中完整数据由 proxy 传入，存量数据同步完成后可以考虑切回老接口，以提早发现问题，下掉 proxy 后调用方实际调用的接口逻辑会自然切到老接口上。

在存量数据不完整时 mysql 侧双写删正常，但增改操作由于没有完整数据，无法直接插入，需要方案

com.xiaohongshu.fls.skywalker.account.user.gateways.persistence.UserRepositoryImpl#updateAvatar(java.lang.String, java.lang.String, java.lang.String)

com.xiaohongshu.fls.skywalker.account.user.gateways.persistence.UserRepositoryImpl#updateHasQywx

com.xiaohongshu.fls.skywalker.account.user.gateways.persistence.UserRepositoryImpl#updateEmail

com.xiaohongshu.fls.skywalker.account.user.gateways.persistence.UserRepositoryImpl#destroyAuthToken

com.xiaohongshu.fls.skywalker.account.user.gateways.persistence.UserRepositoryImpl#updateLdapPasswd

com.xiaohongshu.fls.skywalker.account.user.gateways.persistence.UserRepositoryImpl#generateAndGetJwtSerect

com.xiaohongshu.fls.skywalker.account.user.gateways.persistence.UserRepositoryImpl#removeClientId

com.xiaohongshu.fls.skywalker.account.permission.gateway.persistence.SubsystemRepositoryImpl#removeAccountNo

com.xiaohongshu.fls.skywalker.account.permission.gateway.persistence.SubsystemRepositoryImpl#addAccountNo

剩下 **53%** 的接口目前看来没有问题，可以在对应 mysql db 操作实现中合并为一次写操作。

- 【多余操作】本质是更新的操作
- 存在多余查询和写入操作，可在 mysql 相关操作实现中使用 update 直接代替
- 传入了单个记录的完整数据，可以在 mysql 存量数据缺失时进行 insert

TODO: 还需考察来自调用方传入的记录是否确实完整(数据本身不完整，或 UserEntity -> User 的封装缺失了信息)

com.xiaohongshu.fls.skywalker.account.permission.gateway.persistence.PermissionRepositoryImpl#edit

com.xiaohongshu.fls.skywalker.account.permission.gateway.persistence.ResourceRepositoryImpl#edit

com.xiaohongshu.fls.skywalker.account.user.gateways.persistence.SmsCodeRepositoryImpl#updateStatus

com.xiaohongshu.fls.skywalker.account.user.gateways.persistence.UserRepositoryImpl#addClientId

com.xiaohongshu.fls.skywalker.account.user.gateways.persistence.UserRepositoryImpl#blindMobile

com.xiaohongshu.fls.skywalker.account.user.gateways.persistence.UserRepositoryImpl#unBlindMobile

com.xiaohongshu.fls.skywalker.account.user.gateways.persistence.UserRepositoryImpl#blindSnsUserId

com.xiaohongshu.fls.skywalker.account.user.gateways.persistence.UserRepositoryImpl#savePasswdHash

com.xiaohongshu.fls.skywalker.account.user.gateways.persistence.UserRepositoryImpl#deactivate

com.xiaohongshu.fls.skywalker.account.user.gateways.persistence.UserRepositoryImpl#updatePasswd

com.xiaohongshu.fls.skywalker.account.user.gateways.persistence.UserRepositoryImpl#resetPswd

com.xiaohongshu.fls.skywalker.account.user.gateways.persistence.UserRepositoryImpl#bindAlias

com.xiaohongshu.fls.skywalker.account.user.gateways.persistence.UserRepositoryImpl#updateSnsUserId

com.xiaohongshu.fls.skywalker.account.user.gateways.persistence.UserRepositoryImpl#updateSnsUserIdAndUserName

com.xiaohongshu.fls.skywalker.account.user.gateways.persistence.UserRepositoryImpl#updateName

com.xiaohongshu.fls.skywalker.account.user.gateways.persistence.UserRepositoryImpl#updateAvatar(com.xiaohongshu.fls.skywalker.account.user.domain.User, java.lang.String)

com.xiaohongshu.fls.skywalker.account.user.gateways.persistence.UserRepositoryImpl#activate

com.xiaohongshu.fls.skywalker.account.user.gateways.persistence.UserRepositoryImpl#updatePrimaryAccount

com.xiaohongshu.fls.skywalker.account.user.gateways.persistence.PasswordResetRepositoryImpl#updateStatus

com.xiaohongshu.fls.skywalker.account.permission.gateway.persistence.RoleRepositoryImpl#addAccountNo

com.xiaohongshu.fls.skywalker.account.permission.gateway.persistence.RoleRepositoryImpl#deleteAccountNo

com.xiaohongshu.fls.skywalker.account.permission.gateway.persistence.RoleRepositoryImpl#deletePermission

com.xiaohongshu.fls.skywalker.account.permission.gateway.persistence.RoleRepositoryImpl#addPermission

com.xiaohongshu.fls.skywalker.account.permission.gateway.persistence.RoleRepositoryImpl#edit

com.xiaohongshu.fls.skywalker.account.permission.gateway.persistence.RoleGroupRepositoryImpl#addAccountNo

com.xiaohongshu.fls.skywalker.account.permission.gateway.persistence.RoleGroupRepositoryImpl#addRoleAlias

com.xiaohongshu.fls.skywalker.account.permission.gateway.persistence.RoleGroupRepositoryImpl#deleteRoleAlias

com.xiaohongshu.fls.skywalker.account.permission.gateway.persistence.RoleGroupRepositoryImpl#deleteAccountNo

com.xiaohongshu.fls.skywalker.account.permission.gateway.persistence.RoleGroupRepositoryImpl#edit

com.xiaohongshu.fls.skywalker.account.permission.gateway.persistence.RoleAssignmentRepositoryImpl#updateActivated

com.xiaohongshu.fls.skywalker.account.permission.gateway.persistence.RoleAssignmentRepositoryImpl#delete

mysql 耦合 mongo repo 层的情况

- UserRepository 接口返回了很多 mongo 的 entity
  - 如果仅使用了 entity 的数据，没有问题（目前来看都是这样的）

##

迁移逻辑实现方案梳理

迁移节点：

存量同步过程 -> 观察随时回滚过程

- mongo 读写，mysql 写（存量开始同步）
- mongo 写，mysql 读写（存量同步结束，观察随时回滚）
- mongo -，mysql 读写（结束，观察无问题，停写 mongo）

迁移中的读写开关状态：

- mongo mysql 读互斥
- mongo mysql 写在前两阶段同时存在

最少需要三个开关：

- mongo/mysql 读操作切换开关
- mongo 写开关
- mysql 写开关

实现现状：

- xxxRepositoryImpl 是一系列直接操作 db 的实现（下文称 db 操作实现类），对应 xxxRepository 是一系列接口（下文称 db 操作接口）。
- 实现中小部分方法存在业务逻辑，不是纯净的合适粒度的 db 操作。
- 实现中小部分方法既读又写。

问题：

- 双写中遇到的问题
  - 对增删操作，分情况论证了双写对操作是完全透明的，预期无任何问题。
  - 对改操作，遵循"若记录存在则更新，不存在则插入"规则时，若满足下列期望，则预期无任何问题。

■ ✓ 期望 dts 发现某存量数据已存在时跳过。

✓ 需要讨论"存在"是什么含义。

✓ 需要参考 dts 平台提供哪些能力。

● 存量数据 mysql 写 oid，需要方案

■ 期望 mongodb db 操作实现类不存在对"mongodb 中不存在的记录"的修改操作。

排查后发现没有用到 mong 的 update，忽略

若 mongodb 更新了本不存在于 mongodb 中的数据，这条数据就不会在 mongodb 中出现，但 mysql 发现不存在时会插入，造成增量数据不一致。

规则 "若记录存在则更新，不存在则插入" 看似是必须的，因为我们无法确定 mysql 中数据未存在是因为 dts 未同步到，还是 mongo 中本就不存在。

可能的解决方案是，在 mysql 的修改操作执行前查询 mongodb 相关记录，来确认上述情况到底是哪一种，但这样会引入一定复杂度，且 mongodb 对"mongodb中不存在的记录"的修改操作的情况应该很少或根本不存在，这里需要评估。

解决方案：只能引入复杂度

把操作提升到 proxy 里

mongo -> mysql

● 对既读又写的方法，由于已经是最接近 db 的抽象层，在不改动调用方逻辑（或调用方依赖的 db 操作接口）的前提下，由于这些读写操作是绑定在一起的，所以无法在这个粒度上切换读写源。

####

实现方案讨论：

● 方案一，修改原 db 操作实现类（细粒度，但不优雅，且在撤掉 mongo 代码时引入修改复杂度）

直接在原有使用 mongo 的 db 操作实现类中添加使用 mysql 进行增删改查的实现，通过配置来细粒度地控制读写操作源。

● √ 方案二，引入透明代理（优雅但存在粒度不足的问题）

额外实现一系列与 mongo db 操作实现类功能一致的 mysql db 操作实现类，在调用方与每个 db 操作接口之间引入一透明代理，在代理中手动区分和维护读接口与写接口，并根据读写开关配置分发和调用到对应的 db 操作实现类中。

根据上方对"双写遇到的问题"的讨论，对实现类中纯净的增删改查方法，只要符合上述讨论中的所有期望，预期不会有任何问题。

纯净的方法指不引入任何业务逻辑，且仅包含增删改查之一操作的方法。

但在实现类中存在一些可能引入副作用的方法：

● 引入业务逻辑的方法

暂时未发现引入业务逻辑带来的可能影响数据库迁移的副作用（除同时读写），待 review。。。

● 同时引入读（查）写（增删改）操作的方法

从迁移的第一阶段开始到第二阶段开始前的过程，是逐渐同步存量数据的过程，即存量数据在 mysql 中可能不完整。

读源的数据完整性决定了读出的数据的正确性，从而决定此后代码随该数据的控制流，从而决定了其后的写操作的数据和行为。

解决方案：

在保证 mysql db 操作实现类的逻辑与 mongo 一致的情况下，为了保证数据同步，那么必然要求此时读源为 mongo，来保证得到正确的数据，从而保证其后的控制流流向也一致，进而保证其后对 mysql 的写操作的数据和行为在语义上（指屏蔽不同数据库的数据类型的差异后，例如 TINYINT 和 bool）也一致。

需要讨论这里是否存在例外情况

使用这一解决方案的思路，可能在实现上要引入与方案一同样的复杂度。

这是一个从整体的抽象的概念出发去解决问题的方案，复杂但有效。

若具体问题具体分析，对特殊 case 进行特殊处理，会存在一些问题，下面给出一个具体分析的例子：

[com.xiaohongshu.fls.skywalker.account.user.gateways.persistence.UserRepositoryImpl#destroyAuthToken]

```
@Override
public void destroyAuthToken(String userId) {
    UserEntity user = queryById(userId);
    user.setAuthToken(null);
    mongoTemplate.save(user);
}

@Override
public UserEntity queryById(String id) {
    Query query = Query.query(Criteria.where("_id").is(id));
    return this.query(query);
}
```

该方法（destroyAuthToken）先后进行了读和写操作，其最终目的是更新 user.auth_token 为 null，但由于传入的参数是 userId，所以首先进行了额外的查询。

若不加拆分，不引入细粒度的控制，可能带来的问题：

- 在迁移阶段一，若某次查询该 user 的存量数据未同步到 mysql，双写时直接使用该方法对 mysql 进行操作，则由于无法查询到该 user，故也无法在其后更新该 user。

- 如果能够细粒度地控制读源来自 mongo，这里可能还要引入一个转换层，来将从 mongo 中读到的实体转换为 mysql 的实体，然后进入 mysql 的写相关逻辑，进行写操作，对所有写操作（增删改），只要满足上方 "双写中遇到的问题" 中的期望，则将完成一次正确的写逻辑，保证增量数据语义上一致。

在不改动调用方的逻辑的前提下，到 mysql 的实现这里，若不在 mysql 实现中引入 mongo 读源，是无法遵循 "若记录存在则更新，不存在则插入" 这一规则的，因为该方法无从查询 mongo 中该记录的数据，所以无法做到 "不存在则插入" 这一操作，也许唯一的解决方案就是从 mongo 中读，然后转换为 mysql 实体，然后进行其后的 mysql 实现中的写逻辑。（需要讨论）

```
INSERT ... ON DUPLICATE KEY UPDATE Statement
```

在不改动调用方业务逻辑的前提下，读写开关的开发方案的核心问题，在于直接面向 db 的 repository 层存在既读又写的方法，下面列举了 account 中 repository 层所有实现类既读又写的方法清单，及其分类。

db repo 层 diff 方案

对比单个 entity 的开销比开线程小得多，除非可以开本线程的无栈协程，否则得不偿失，直接同步处理即可。

对比数组的开销需要考虑，在 m1 下最坏情况平均每个元素需要花费 0.00003 ms，当数组长度数量级达到 1w - 10w 时，对比的时间开销会达到磁盘 IO 的毫秒级，从而影响接口的周转时间，这时候可能就有必要开线程了。

对第二阶段（mysql 读）需要额外查询 Mongodb 时，开线程跑 IO 的逻辑。

db 层开发的良好实践

见SSO Mongo2Mysql 迁移方案、开发方案复盘 & 总结