外部用户

模型概念

1. ExternalEntity 外部实体
    a. 给外部实体一个domain内的id，并记录原始数据
        i. 避免domain外name变更，导致本地数据不可追溯的问题
    b. type：实体类型，有identity，groups、tags、organizations、tags
    c. parent_name：支持父子关系，类型为当前type
    d. relations: 支持从属关系：groups、tags、organizations、tags
    e. resource_urn：支持链接到其他实体
        i. linked_urn?
    f. union_id：支持跨source关联identity
2. Account：本地用户
    a. external_id：记录外部实体id
3. ExternalIdentity：临时用户
    a. 用于弱验证场景
    b. 与Account同级，为临时用户分配一个本地身份
    c. 无法绑定多因子
    d. external_id：记录外部实体id
    e. *实际上也可以用Account+temporary字段来描述
        i. 但是为了避免开发出问题，暂时先不改
4. 用户组管理
    a. external_id：记录外部实体id
    b. 不支持EI，因为EI仅用于弱认证
5. source 认证来源
    a. 有两种使用场景
        i. VALIDATE 确认提供的Account的身份
        ii. IDENTIFY 认证并获取外部身份
    b. 数据结构：
        i. register_account：是否为弱认证，认证结果是 EI 还是 A

ii. adapter：指定adapter的代码和参数

iii. mapper：返回的数据如何映射到本地身份 （考虑移除）

c. 认证结果有3种

i. 弱认证(allow_external=True) => ExternalIdentity

ii. Account

1. 认证时创建了account 或 认证结果对应到了现有account

iii. None，MappedEntity

1. reverse-enroll，当前认证的结果没有对应的Account

设计内容

1. 原有 source.independent 设计解释
   a. 原有设计 wechat can id, but may not independent
   b. 以VALIDATE / IDENTIFY来区分更直接和明确
      i. independent是在IDENTITY场景中区分身份是临时的EI还是本地的A
2. 外部实体EE，链接A / EI
   a. 场景1：第一次认证，创建账号时
      i. authn_result - > register_account=True -> ensure_account
   b. 场景2：enroll_externally - 先登录系统，再绑定新的 **[外部]** 认证方式
      i. 即：添加 [外部身份] 作为 [本地用户的认证方式]
      ii. authn_result - > EE -> make_link
   c. 场景3：**enroll-reversed
      i. 微信上下文认证后 - [禁止EI+禁止注册A] - 没有对应的A - 返回EE -
      ii. AuthnManager发现没有A - 保存EE.id - 继续触发认证 - 直到能获得到A
      iii. 认证成功后，提示用户是否关联之前认证的信息
   d. 场景4：弱认证
      i. authn_result - > allow_external=True, register_account=False
      ii. -> ensure_external_identity -> EI
3. 落 account 还是 EI
   a. 对用户的含义：是否可以绑定其他认证方式
   b. 数据所有权的区别，一个是外部所有，一个是本地优先
   c. EI的意义：account的翻版，临时身份，例如弱验证

4. 后续account更新
    a. 只有account.source_urn和external的source匹配才能更新
5. 定向邀请：先创建Account，邀请特定source的EE来链接
    a. unique_binding { name=source/<id>:name, key=<name>, identity_urn=account::urn }
6. union_id：跨source关联EE-A
    a. mapper生成EE.union_id – 尝试通过binding匹配union_id –
        i. 匹配上则直接关联Account
        ii. 如果无匹配 – 新建Account – 创建binding
    b. unique_binding { name=union_id, key=<union_id>, identity_urn=account::urn }

特殊场景处理

1. 换绑CAS带来过来的手机号，如何避免覆盖?
    a. source_id=<0-console>
    b. 后续数据覆盖时，由于source_id不一致，无法覆盖
2. 换绑创建时来源source，导致数据错乱怎么办?
    a. 例如：企业微信用户，
    b. 不得换绑来源source
    c. 诸如微信扫码、手机号等容易换绑的，禁止register_account
        i. 隐含了一个EI-name唯一性和unique-binding的重复问题
3. 绑定关系冲突解决
    a. 例如本地有个admin，手机号是M1，外部用户也有个M1
        i. 这样就会导致外部用户未预期的和本地用户合并
    b. 本source内自动协调，超过source范畴不予覆盖
    c. **[P2]** [bool] 仅同步增量绑定关系?
4. 外部source内name重复
    a. 无法处理，无法预见和分离该场景
    b. 后来的数据会覆盖之前的
5. 外部source内name变更
    a. 例如本科生毕业，研究生入学，学号变了，想继承原EE/A的关系
    b. 必要信息：谁和谁有关系，old-name => new-name
    c. migrate:

    i. old.EE { resource_urn = account }
    ii. old.EE { resource_urn = new.EE }, new.EE { resource_urn = account }
6. [P2] 如果同时接入企业微信和钉钉，怎么关联A-A?
    a. 添加old-account => new-account的链接关系
    b. 数据是否merge，如何merge?
7. [P2] stale-registered-account from CAS
    a. CAS已经删了这个用户，本地绑定了其他认证方式，还能登录
    b. 解决方案：定期要求CAS认证一次
    c. **refresh_token 后台刷新

TBD 待设计

1. 管理站本身的身份使用需求
    a. 例如改密码
    b. 例如绑定新的登陆方式时
2. 用户组的权限透出
3. EI / A的关系问题
    a. 如果企业微信的用户register_account了，数据所有权是谁的?
4. 性能
    a. phase-lize，如何延迟加载关联组件（非一次性返回的）
    b. performance，一次登录要多少次sql?
5. union_id场景如何做属性覆盖？如何更新？如果>2个数据源呢?
6. 弱认证可以记住用户名?

source的设计和分类

1. facade层
    a. can_id：是否可以作为第一认证方式，可能会将identity检索出来提供给adapter层
    b. challenge_type: LOCAL | EXTERNAL，本地认证优先，但可以指定外部认证方式
    c. can(identity)：当前身份是否支持认证
    d. enroll(...)：待设计
2. adapter层
    a. require_identity：是否需要有现成的身份才能认证；如何为False，隐含了只能支持Account

b. identity_location: ACCOUNT / EXTERNAL 毫无意义，是否can_id，也就是，是否能产生EI/A才是区别

3. source层
   a. register_account: 是否注册为Account，隐含了can_id=yes，否则没有创建EI的能力
4. requirement层
   a. 站点：需要的身份源
5. **binding:MOBILE，EI:NAME？ binding自动化的问题，mapped.bindings**
6. identity
   a. EI
      i. 暂无场景，因为EI产生后，直接输出，不支持MFA
      ii. **弱认证可以记住用户名?**
   b. A
      i. 由last_login或其他认证步骤(MFA)提供
7. can_id=no, identity=A，register_account=no
   a. 一般用于需要提前注册（例如公钥）的场景
   b. 既然不能can_id，则一定不能register_account
   c. 确认A的某种认证方式，例如password，sm2_key，FIDO2，wechat_shared_code，soter（依赖于wechat_shared_code(有openid且关注推送公众号）通过推送消息来认证，也可以改造为扫码认证，那就是can_id了）
   d. 由于不能can_id，结果只能是输出A本身
8. can_id=yes，identity=A | None，register_account=yes
   a. 最常见的认证方式，来源一般为独立权威身份源
   b. last_login + CAS | 企业微信 | LDAP | radius等
   c. 注意：认证结果的A和last_login不一定一致
9. can_id=yes，identity=A | None，register_account=no
   a. 一般作为辅助认证方式
   b. 原始出的是EI，可能关联到其他A，最终出EI | A
   c. last_login + SMS | wechat | sms_weak（**adapter层面其实可以共享？**）
   d. 注意：认证结果的A和输入的A不一定一致

回归VXP全景

1. 作为用户身份，account / EI等价

    a. 实际根据站点需求提供，要MFA的，认可某种认证方式的
2. account
    a. 允许绑定其他身份源的外部用户身份
    b. 普通本地用户
    c. 管理员
3. EI
    a. 临时用户

场景分析

1. CAS + 本地sms认证 （VAC不信任cas的权威性）

1. 开启account_register
2. 添加二次认证要求
3. 注册：提前写name到unique_binding：
    a. key=source/CAS:name, value=<CAS:name>, identity_urn=<account-urn>
    b. 引导CAS用户认证时映射到对应的account_id

2. CAS + 携带了手机号，本地改绑手机号

1. unique_binding的source_id从<CAS>改为<0-本地>
2. 后续更新不会覆盖这些修改

3. CAS + 携带了手机号，但和本地的发生了冲突

1. 效果：不覆盖，仅在source内允许覆盖
2. unique_binding里的手机号，source_id是否与当前认证来源一致，不一致则不允许重新绑定

4. [P2] enroll-reversed – 先登录上下文微信 – 再认证account

1. 数据表现：requirements = { methods=[wechat, CAS], source=CAS }
    a. source<CAS>.register_account=True
    b. source<wechat> { register_account=False }
    c. adapter写死: { allow_external=False }
2. 微信上下文认证后 – [禁止EI+禁止注册A] – 没有对应的A – 返回EE –
3. 发现没有A – 保存EE.id – 继续触发认证 – 直到能获得到A

4. LoginState.binding_ids – 保存当前要绑定的EE，登陆成功后提示绑定

5. 弱验证：SMS / 微信扫码

1. 禁止account_register
2. EI直接访问业务

6. 通达OA

1. 提前创建 account
    a. 添加tongda_oa:mobile属性，避免用户能独立登录系统
2. 提前创建 binding: key=source/tongda_oa/name, source_id=0
3. 通达OA访问的时候自动创建EI，关联到Account，获取手机号
4. 每个认证步骤入库

7. 博达站群

1. 提前创建 account
    a. 根据需求添加boda_manage:sm2_key属性
2. 提前创建 binding: key=boda_manage/name, value=sn，source_id=0
3. 用户认证时，自动创建EI，关联到Account，获取sm2_key信息
4. 每个认证步骤入库

8. 企业微信认证+钉钉认证同时接入

1. 如何双向匹配entity
    a. 两边都用 union_id 做 binding
    b. 基本的数据：基于什么逻辑做匹配+配套数据，进mapper
    c. 常见的是使用其中一个claim作为union_id
2. **特殊场景：**如果union_id在domain内重复：算法有错误，或数据异常
    a. 检索domain内是否有冲突的union_id
        i. 如有，则不做链接，创建新的account，等待后续的数据清理
    b. 算法变化导致union_id变化?
        i. source_id/name 已存在时更新union_id，并记录
        ii. 否则检索domain内是否有冲突的union_id
            1. 如有，则不做链接，创建新的account，等待后续的数据清理

    c. 非identity如何支持？暂不支持，一般非identity不会这么复杂
3. TBD：如何做属性覆盖？如何更新？

9. –

TODO

1. 移除source.independent，添加了__identity_location__用来标注是本地用户还是外部 **done**
    a. __identity_location__ 好像意义不大
2. 移除EI之间互相链接的能力 **done**
3. ~~修复method和source adapter分离的问题~~
    a. 移除~~__require_identity__~~
4. 同步至account的功能 **done**
5. 追随预分配的binding **done**
6. 划分unique_binding的realm **done**
    a. modify_account 里面的3个binding，应有binding才创建binding **done**
7. EI的安全功能，锁定等（无法设置密码，无法MFA）FE–work @旭煌 **done**
8. 创建管理员 / 不同角色的权限 FE–work @旭煌 **done**
9. 创建普通用户 FE–work @旭煌 **done**
10. union_id可见，可编辑
11. 同步groups **done**
12. enroll – 先登录系统，再绑定新的认证方式 **done**
    a. 确保同source无其他同name链接到account **无需，name+identity已经唯一**
13. portal的认证方式 @**旭煌**
14. sm2_key改为不需要id **@王乾**
15. 如果account / external disabled @**旭煌**
16. 纠正：facade/application不能出现在domain **@旭煌**
17. **微信上下文认证 **@boyxuper**
18. **定期要求CAS认证一次，避免stale **延后**
19. **enroll–reversed – 先登录简单认证方式 – 再认证account **延后**
20. me
    a. unique_binding的identity_urn改account_id **延后**
    b.

完整同步

1. 场景
    a. 定期同步
    b. 数据迁移
2. –