外部⽤户

模型概念

位置：vac/domain/directory/entity

1. ExternalEntity 外部实体
   1. 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

* 1. type：实体类型，有identity，groups、tags、organizations、tags
  2. parent\_name：⽀持⽗⼦关系，类型为当前type
  3. relations: ⽀持从属关系：groups、tags、organizations、tags
  4. resource\_urn：⽀持链接到其他实体

**ⅰ.** linked\_urn？

* 1. union\_id：⽀持跨source关联identity

1. Account：本地⽤户
   1. external\_id：记录外部实体id
2. ExternalIdentity：临时⽤户
   1. ⽤于弱验证场景
   2. 与Account同级，为临时⽤户分配⼀个本地身份
   3. ⽆法绑定多因⼦
   4. external\_id：记录外部实体id
   5. \*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

1. ⽤户组管理
   1. external\_id：记录外部实体id
   2. 不⽀持EI，因为EI仅⽤于弱认证 无法使用用户组管理临时用户
2. source 认证来源
   1. 有两种使⽤场景

ⅰ. VALIDATE 确认提供的Account的身份

确认用户是否有效

ⅱ. IDENTIFY 认证并获取外部身份

完整的认证流程并获取外部身份信息

* 1. 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 怎么发起微信登录，结果

ⅲ. mapper：返回的数据如何映射到本地身份 （考虑移除） 原始数据清洗

* 1. 认证结果有3种

ⅰ. 弱认证(allow\_external=True) => ExternalIdentity

返回外部身份，证明该认证是弱认证

ⅱ. Account

* + 1. 认证时创建了account 或 认证结果对应到了现有account

若验证成功，返回account。或者新创建account

ⅲ. None，MappedEntity

1. reverse-enroll，当前认证的结果没有对应的Account

未找到account，则返回MappedEntity表示无匹配项

设计内容

1. 原有 source.independent 设计解释
   1. 原有设计 wechat can id, but may not independent
   2. 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

1. 外部实体EE，链接A / EI
   1. 场景1：第⼀次认证，创建账号时

ⅰ. authn\_result - > register\_account=True -> ensure\_account

第一次认证时，如果验证结果为可以创建账号则，创建个本地用户account

* 1. 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

用户在登录系统时可以进行绑定外部认证方式，例如vac的绑定页。

* 1. 场景3：\*\*enroll-reversed（反向注册）

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

* 1. 场景4：弱认证

ⅰ. authn\_result - > allow\_external=True, register\_account=False

ⅱ. -> ensure\_external\_identity -> EI

1. 落 account 还是 EI
   1. 对⽤户的含义：是否可以绑定其他认证⽅式

落A可以绑定多因子

* 1. 数据所有权的区别，⼀个是外部所有，⼀个是本地优先
  2. EI的意义：account的翻版，临时身份，例如弱验证

1. 后续account更新
   1. 只有account.source\_urn和external的source匹配才能更新
2. 定向邀请：先创建Account，邀请特定source的EE来链接
   1. unique\_binding { name=source/<id>:name, key=<name>, identity\_urn=account::urn }
3. union\_id：跨source关联EE-A
   1. 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

* 1. 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？
   1. source\_id=<0-console>

Source\_id是用来区别认证来源 <0-console>是什么？

* 1. 后续数据覆盖时，由于source\_id不⼀致，⽆法覆盖

比如

认证来源是微信，source\_Id为1，绑定的手机号为A

认证来源是钉钉，source\_id为2，绑定的手机号为B

就不需要覆盖，反而要避免覆盖，因为不是一个数据来源

1. 换绑创建时来源source，导致数据错乱怎么办？
   1. 例如：企业微信⽤户，
   2. 不得换绑来源source

同1.b

* 1. 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

外部用户name可能会产生数据合并覆盖情况，所以要unique。

来源相同，身份想同就会产生unique\_binding重复

1. 绑定关系冲突解决
   1. 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

可以本地用户优先，外部用户手机号置空

或者在调用用户信息时优先调用本地用户

或者可以使用其他标识符证明身份

* 1. 本source内⾃动协调，超过source范畴不予覆盖
  2. [P2] [bool] 仅同步增量绑定关系？

同步外部用户的绑定关系时，同步那些新增进来的关系

1. 外部source内name重复
   1. ⽆法处理，⽆法预⻅和分离该场景
   2. 后来的数据会覆盖之前的

可能会有相同来源，相同身份的用户，起了相同的名字，可以在绑定时在名字后面加一个id，这样就不会第二个名字重复进来时覆盖掉第一个

1. 外部source内name变更
   1. 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

比如本科学号为 no1 研究生学号为no2， 可以保存no1和no2的映射关系,

例如：

{

old\_num: no1,

New\_num:no2,

}

* 1. 必要信息：谁和谁有关系，old-name => new-name
  2. migrate:

ⅰ. old.EE { resource\_urn = account }

ⅱ. old.EE { resource\_urn = new.EE }, new.EE { resource\_urn = account }

不太理解

1. [P2] 如果同时接⼊企业微信和钉钉，怎么关联A-A？
   1. 添加old-account => new-account的链接关系

也可以使用

{

Wx\_num: no1,

Dd\_num:no2,

}

来保存用户信息

* 1. 数据是否merge，如何merge？

1. [P2] stale-registered-account from CAS
   1. CAS已经删了这个⽤户，本地绑定了其他认证⽅式，还能登录

比如在CAS中删除了majunhong的A，但是我在本地系统绑定了手机号，我可以通过手机号登录系统

* 1. 解决⽅案：定期要求CAS认证⼀次

CAS认证时，如果出现7.a的问题，就可以直接拒绝登录请求

* 1. \*\*refresh\_token 后台刷新

可以将本地信息跟CAS同步，这样就不会出现该问题

TBD 待设计

1. 管理站本身的身份使⽤需求
   1. 例如改密码
   2. 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
   1. 如果企业微信的⽤户register\_account了，数据所有权是谁的？
4. 性能
   1. phase-lize，如何延迟加载关联组件（⾮⼀次性返回的）
   2. 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
   1. 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
   2. challenge\_type: LOCAL | EXTERNAL，本地认证优先，但可以指定外部认证⽅式
   3. can(identity)：当前身份是否⽀持认证 方法
   4. enroll(...)：待设计
2. adapter层
   1. 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
   2. 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
   1. 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
   1. 站点：需要的身份源
5. binding:MOBILE，EI:NAME？ binding⾃动化的问题，mapped.bindings
6. identity
   1. EI
   2. A

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？

ⅰ. 由last\_login或其他认证步骤(MFA)提供

1. can\_id=no, identity=A，register\_account=no
   1. ⼀般⽤于需要提前注册（例如公钥）的场景
   2. 既然不能can\_id，则⼀定不能register\_account
   3. 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)
   4. 由于不能can\_id，结果只能是输出A本身
2. can\_id=yes，identity=A | None，register\_account=yes
   1. 最常⻅的认证⽅式，来源⼀般为独⽴权威身份源
   2. last\_login + CAS | 企业微信 | LDAP | radius等
   3. 注意：认证结果的A和last\_login不⼀定⼀致
3. can\_id=yes，identity=A | None，register\_account=no
   1. ⼀般作为辅助认证⽅式
   2. 原始出的是EI，可能关联到其他A，最终出EI | A
   3. last\_login + SMS | wechat | sms\_weak（adapter层⾯其实可以共享？）
   4. 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
   1. 实际根据站点需求提供，要MFA的，认可某种认证⽅式的
2. account
   1. 允许绑定其他身份源的外部⽤户身份
   2. 普通本地⽤户
   3. 管理员
3. EI
   1. 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：
      1. key=source/CAS:name, value=<CAS:name>, identity\_urn=<account-urn>
      2. 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
3. 效果：不覆盖，仅在source内允许覆盖
4. 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
   1. [P2] enroll-reversed - 先登录上下⽂微信 - 再认证account
5. 数据表现：requirements = { methods=[ wechat, CAS ], source=CAS }
   1. source<CAS>.register\_account=True
   2. source<wechat> { register\_account=False }
   3. adapter写死: { allow\_external=False }
6. 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 -
7. 发现没有A - 保存EE.id - 继续触发认证 - 直到能获得到A
8. LoginState.binding\_ids - 保存当前要绑定的EE，登陆成功后提示绑定
9. 弱验证：SMS / 微信扫码
10. 禁⽌account\_register
11. EI直接访问业务
12. 通达OA
13. 提前创建 account

a. 添加tongda\_oa:mobile属性，避免⽤户能独⽴登录系统

1. 提前创建 binding: key=source/tongda\_oa/name, source\_id=0
2. 通达OA访问的时候⾃动创建EI，关联到Account，获取⼿机号
3. 每个认证步骤⼊库
4. 博达站群
5. 提前创建 account

a. 根据需求添加boda\_manage:sm2\_key属性

1. 提前创建 binding: key=boda\_manage/name, value=sn，source\_id=0
2. ⽤户认证时，⾃动创建EI，关联到Account，获取sm2\_key信息
3. 每个认证步骤⼊库
4. 企业微信认证+钉钉认证同时接⼊
5. 如何双向匹配entity
   1. 两边都⽤ union\_id 做 binding
   2. 基本的数据：基于什么逻辑做匹配+配套数据，进mapper
   3. 常⻅的是使⽤其中⼀个claim作为union\_id
6. 特殊场景：如果union\_id在domain内重复：算法有错误，或数据异常

比如a微信和a钉钉公用union\_id1 b微信和b钉钉公用union\_id2 但是union\_id1和union\_id2重复了，导致数据错乱。

* 1. 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

* 1. 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
  1. ⾮identity如何⽀持？暂不⽀持，⼀般⾮identity不会这么复杂

1. 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
   1. \_\_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~~
   1. ~~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
   1. 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 / 不同⻆⾊的权限 FE-work @旭煌 done
9. 创建普通⽤户 FE-work @旭煌 done
10. union\_id可⻅，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
    1. 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
    1. unique\_binding的identity\_urn改account\_id 延后

b.

完整同步

1. 场景
   1. 定期同步
   2. 数据迁移

2. -