- API /tac/request
  - 1. TwoStepElevationController
  - 2.TwoStepElevationServiceImpl
    - tacRequest()
  - 3. TACRequestServiceImpl
    - 预处理：
    - step1：getUserProfile 获取用户信息（包括用户的token信息）
    - step2：prepareRequestDetail 准备配置数据，同步数据
    - step3：decideAuthorizationType 判断（用户可选的）授权类型
    - step4：updateAndTrigger 请求授权和发送消息
    - 分析
  - 4. 接口TokenService
    - 4.1 INBTokenServiceImpl
    - 4.2 BVTokenServiceImpl
    - 4.3 MBSSTokenServiceImpl
    - 4.4 OSPLTokenServiceImpl
  - 5. 接口AuthorizationStrategy
  - 6. 接口SMSService
  - 7. 接口SoftTokenService
  - 8. 接口HardTokenService
  - 9. TriggerServiceImpl
    - 1. preHandle()
    - 2. triggerAuthorization()
    - 3. triggerAuthzService()
    - 4. handle()
    - 5. checkForAuthTypeChannel()
- API /tac/validate
  - 1. TACValidateServiceImpl
    - step1，查询AUTHZ_TXN表。
    - step2，根据AUTHZ_REF_CHECK配置和authz_policy表，校验随机码。
    - step3，使用TxnUtl类checkDurationFlow()
    - step4，使用了AuthorizationStrategy及其具体实现类
    - step5，根据校验结果，更新AUTHZ_TXN表。
  - 2. MessageServiceImpl
- API /otp/request
- API /otp/validate
- API /v3/access-elevation/request
  - 1. AccessElevationRequestServiceImpl
    - 预处理
    - step1，获取用户的可用token列表
    - step2：decideAuthorizationType 判断（用户可选的）授权类型
    - step3：updateAndTrigger 请求授权和发送消息
- API /v3/access-elevation/validate
  - 1. AccessElevationValidateServiceImpl
    - step1，通过请求头的auth ID，查询AUTHZ_TXN表。
    - step2，预校验。校验当前状态。
    - step3，使用了AuthorizationStrategy及其具体实现类validateAuthorization()
    - step4，校验后处理
- AuthenticationContext 上下文类
- CommonContext

梳理每个API的主要流程和架构，其中使用到的类/接口，表/字段，实体/字段等。

为什么ms-customer-authorization这么复杂?

纵向，某些接口的流程长。大步骤里面小步骤。

横向，在channel和toke type两个维度上扩展。问题一是耦合在一起，核心功能和非核心功能耦合，不同的业务耦合。问题二是抽象不够？没有把共用的功能，核心的功能抽象出来。

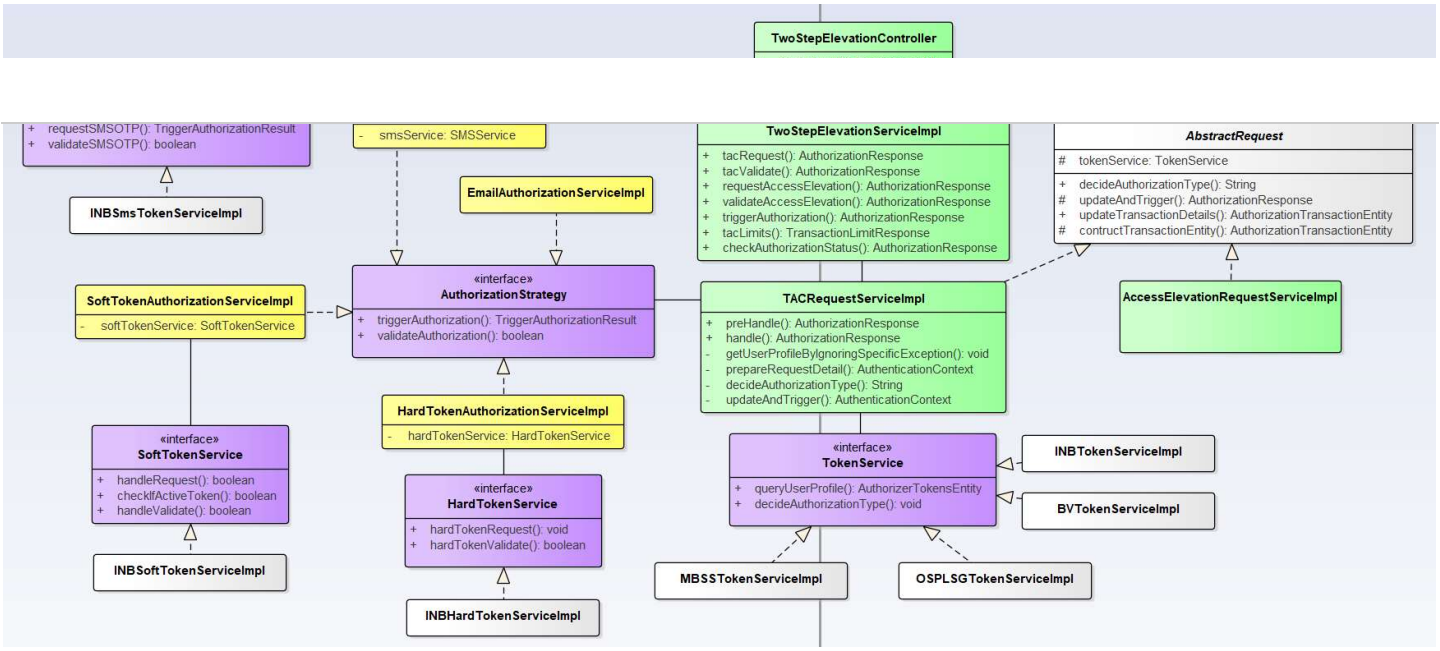
有没有合理和不合理的地方?

架构设计的角度，可以优化复杂度吗?

纵向，如果准备阶段的步骤，只和channel有关，考虑分离出去，比如放到ms-customer-authorization-inb实现，互不影响。

另外，这些步骤之间，是否有依赖关系，如果没有，可以使用异步任务实现。

横向，把共用的功能抽取出来放到抽象类实现，或者放到default具体实现类，把个性化的功能放到具体实现类。区分开共用的功能和个性化的功能，进行解耦。



API /tac/request

1. TwoStepElevationController

提供了v1/v2/v3的不同版本。

提供了所有的核心功能的接口，包括/tac/request，/tac/validate，access-elevation/request，access-elevation/validate等。

2.TwoStepElevationServiceImpl

tacRequest()

根据请求头，创建AuthenticationContext，设置tacRequest，mfaType，challengeCode，triggerOTP，transactionPayload字段。

3. TACRequestServiceImpl

主要方法handle()，包括4个步骤。

预处理：

判断是否passthrough还是standalone，是否pilot user mode。

根据请求传参，设置AuthenticationContext的**challengeCode**和**customerChallengeCode**字段。

根据channel，设置AuthenticationContext的**identity**字段（决定**AuthorizationStrategy接口的具体实现类**）。

以下步骤以INBTokenServiceImpl为例

step1：getUserProfile 获取用户信息（包括用户的token信息）

通过查询缓存或者调用SOA，获取用户信息，可用的token。并设置AuthenticationContext的**authorizerTokenEntity**字段（对应**AUTHZ_USER_PROFILE**表）。

使用了**TokenService**及其具体实现类queryUserProfile()方法。

涉及查询**AUTHZ_USER_PROFILE**表。

异步调用SOA（UAS-QueryUser）→iSprint，获取SWK信息。设置AuthorizerTokensEntity的isprintUserStatusCode，isprintLoginAccountStatusCode，isSwkSvcDown，custom（SWK_LOGIN_MODULE_ID，SWK_BAD_LOGIN_COUNT）等字段。

异步调用SOA（UAS-QueryToken），设置AuthorizerTokensEntity的softTokenStatusCode，softTokenSerialNo，deviceFingerPrint，deviceOSName，Tid，softTokenSerialNo，pkiTokenSerialNo，custom（SWK_STATUS，USP_COOLING_PERIOD_STARTDT）等字段。

异步调用SOA（2FA-QueryUser-I）→ DSSS，获取SMS/HWK信息。设置AuthorizerTokensEntity的userStatusCode，custom（DSSS_USER_STATUS，DSSS_OTP_TOKEN_FAILED_TRIES）。

异步调用SOA（2FA-GetTokenReqStatus-I）。设置AuthorizerTokensEntity的smsTokenStatusCode，hardTokenStatus，hwkSerialNo，custom（HARD_TOKEN_REQUEST_STATE，）等字段。

异步调用SOA（Auth_Inquiry_2FAMediaDestination）。设置AuthorizerTokensEntity的mobileNo字段。

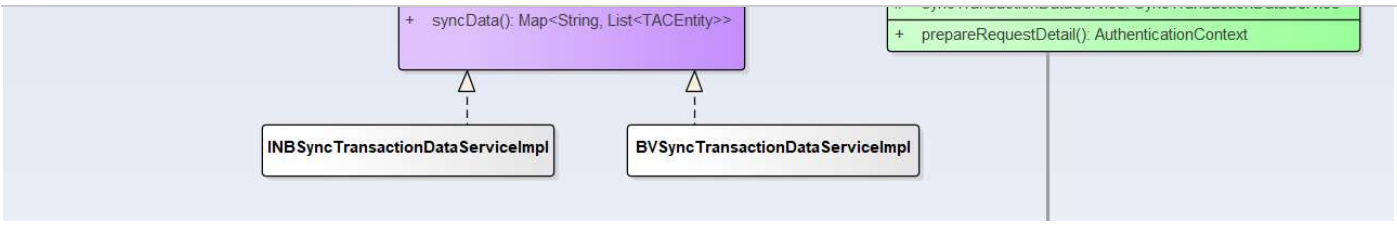
step2：prepareRequestDetail 准备配置数据，同步数据

查询缓存和**TAC**表，**customer_tac**表?

调用SyncTransactionDataService类具体实现类的sync()方法，从old car同步数据。

设置AuthorizationContext的authorizationType，tacEntity，highRiskFlag，extraAuthorizationType，mfaId，transactionType，gtSigningMode，ltSigningMode字段。

更新**AUTHZ_POLICY**表?



step3：decideAuthorizationType 判断（用户可选的）授权类型

使用了**TokenService**及其具体实现类。

设置AuthorizationContext的requireChallengeCode，availableAuthType，swkFirstLogin，finalAuthorizationType，softTokenStepUpRequired字段。

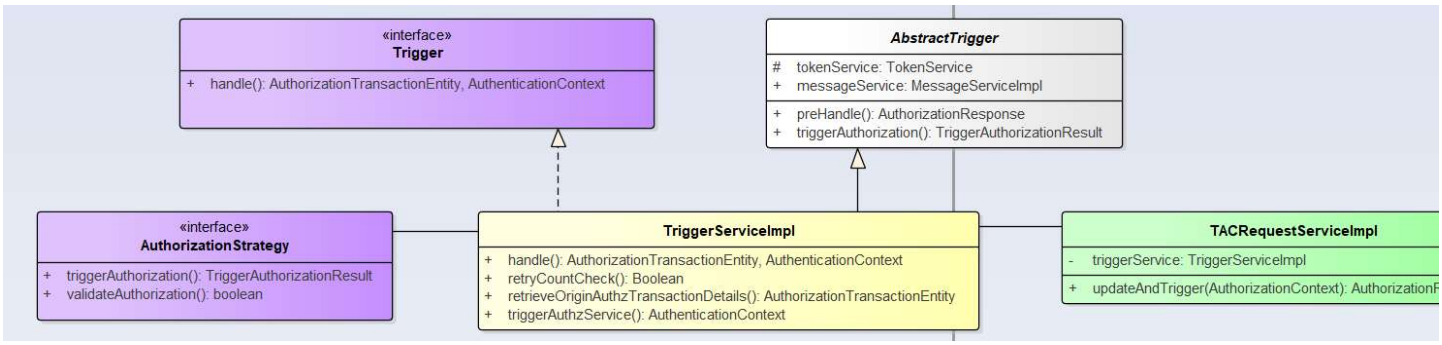
step4：updateAndTrigger 请求授权和发送消息

涉及AUTHZ_TXN表

调用**TriggerServiceImpl**类。

调用**AuthorizationStrategy**及其具体实现类，比如**INBSmsTokenServiceImpl**类。

发送Kafka消息。



分析

1. 实际上，step1/2/3都是为了step4的准备动作，而step4是核心功能：请求（发送）。

2. 因为TACRequest和2FARequest都有类似的准备动作，所以抽象出了AbstractRequest类。

3. 这些准备动作当中，比如queryUserProfile()和decideAuthorizationType()，**同具体的channel有关，和具体的token type无关。**

所以通过接口**TokenService**及其具体实现类，比如在**INBTokenServiceImpl**类等实现具体逻辑。

4. 接口TokenService

该接口提供了2个方法：**queryUserProfile**()和**decideAuthorizationType**()。

该接口有4个实现类：INBTokenServiceImpl，BVTokenServiceImpl，MBSSTokenServiceImpl，OSPLTokenServiceImpl。

注意：该接口在channel维度进行了扩展，而没有在token type维度进行扩展。

因为不同的token type，在获取用户信息和判断授权类型上没有区别?

4.1 INBTokenServiceImpl

用到4张表：

TokenPreferenceRepository，调用inb数据库的存储过程

AuthorizerTokenRepository，调用**AUTHZ_USER_PROFILE**表

TokenTypePreferenceRepository，调用cust_token_preference表

UserSmsRepository，调用USER_SMS_RECORD表

4.2 BVTokenServiceImpl

用到1张表：AuthorizerTokenRepository，调用AUTHZ_USER_PROFILE表

4.3 MBSSTokenServiceImpl

用到1张表：AuthorizerTokenRepository，调用AUTHZ_USER_PROFILE表

4.4 OSPLTokenServiceImpl

用到1张表：AuthorizerTokenRepository，调用AUTHZ_USER_PROFILE表

5. 接口AuthorizationStrategy

该接口提供了2个方法：triggerAuthorization()和validateAuthorization()。

该接口有4个实现类：SMSAuthorizationServiceImpl，EmailAuthorizationServiceImpl，SoftTokenAuthorizationServiceImpl，HardTokenAuthorizationServiceImpl。

而且，每个实现类都关联了一个接口，比如SMSAuthorizationServiceImpl关联了**SMSService**接口。

不论哪种token，都需要有请求（发送）和校验的核心功能。

通过XXXService接口，实现了channel维度的扩展。

因为任何一种token，对应到不同的channel，请求（发送）和校验的功能的具体实现又存在不同。

通过这种设计，实现了2个维度的组合。

问题：在SMSAuthorizationServiceImpl类的triggerAuthorization()方法具体逻辑，和INBSmsTokenServiceImpl类的requestSMSOTP()方法的具体逻辑，有什么不同？

SMSAuthorizationServiceImpl类的triggerAuthorization()方法的功能简单，主要是根据channel设置AuthorizationContext的identity字段，决定SMSService接口的具体实现类，比如INBSmsTokenServiceImpl类。

INBSmsTokenServiceImpl类的requestSMSOTP()方法，实现具体逻辑。

6. 接口SMSService

该接口提供了2个方法：requestSMSOTP()，validateSMSOTP()。

该接口有多个实现类：**INBSmsTokenServiceImpl**，BVSmsTokenServiceImpl，MBSSSmsTokenServiceImpl，DefaultSmsTokenServiceImpl。

INBSmsTokenServiceImpl这个类复杂吗？

7. 接口SoftTokenService

该接口提供了3个方法：handleRequest()，handleValidate()，checkIfActive()

该接口有多个实现类：INBSoftTokenServiceImpl，BVSoftTokenServiceImpl，MBSSSoftTokenServiceImpl，DefaultSoftTokenServiceImpl。

8. 接口HardTokenService

该接口提供了2个方法：hardTokenRequest()，hardTokenValidate()

该接口有多个实现类：INBHardTokenServiceImpl，BVHardTokenServiceImpl，MBSSHardTokenServiceImpl，DefaultHardTokenServiceImpl。

9. TriggerServiceImpl

主要完成request核心功能之前的准备工作，并调用**AuthorizationStrategy**的具体实现类实现核心功能。

1. preHandle()

判断是否是passthrough。

2. triggerAuthorization()

具体执行trigger服务。实际上是由**AuthorizationStrategy**的具体实现类（SMSAuthorizationServiceImpl，EmailAuthorizationServiceImpl，SoftTokenAuthorizationServiceImpl，HardTokenAuthorizationServiceImpl）的**triggerAuthorization()**方法实现。

而**triggerAuthorization()**方法调用了**SMSService**接口的具体实现类，比如**INBSMSServiceImpl**类的**requestSMSOTP()**方法。

而**requestSMSOTP()**方法调用了**NotificationService**接口的具体实现类，比如**CustomerNotificationServiceImpl**类的**send**()方法。

而**send**()方法的具体实现包括以下子方法：

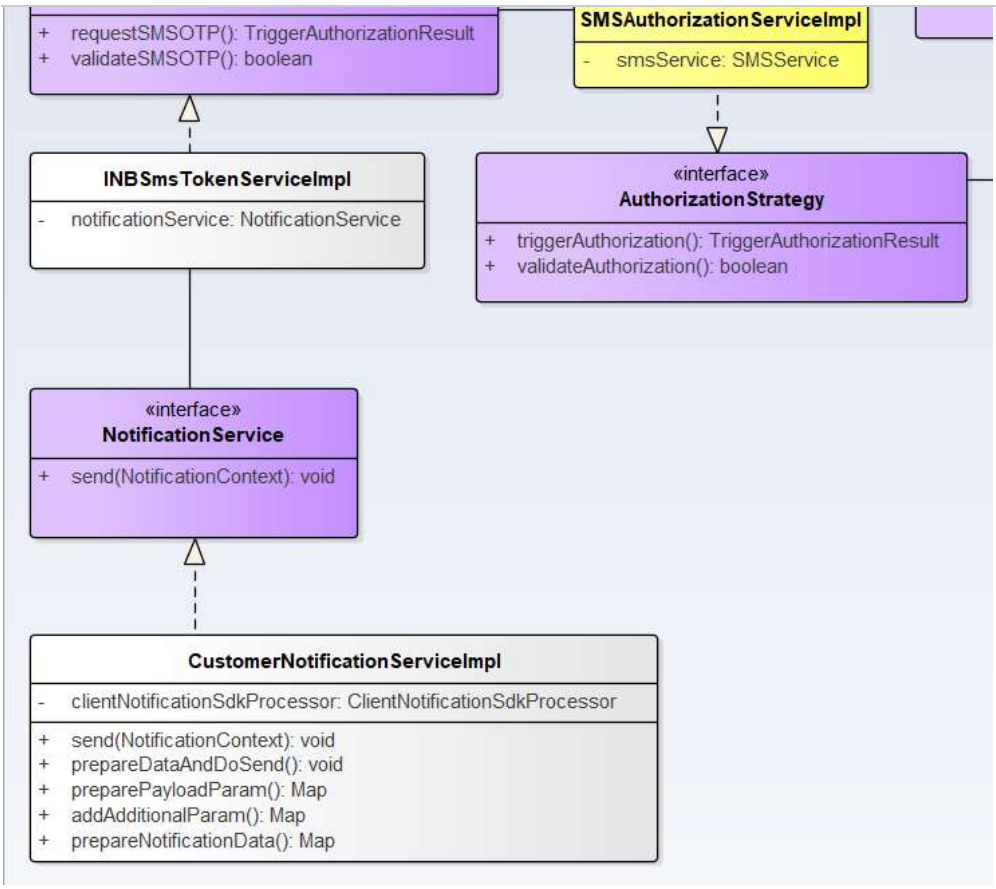
prepareDataAndDoSend()

preparePayloadParam()

addAdditionalParam()

prepareNotificationData()

最终调用sdk提供的**clientNotificationSdkProcessor**类的**processNotification**()方法发送消息。



3. triggerAuthzService()

处理发送消息给用户，包括SMS，EMAIL，SWK。实际调用抽象父类**AbstractTrigger**的triggerAuthorization()。

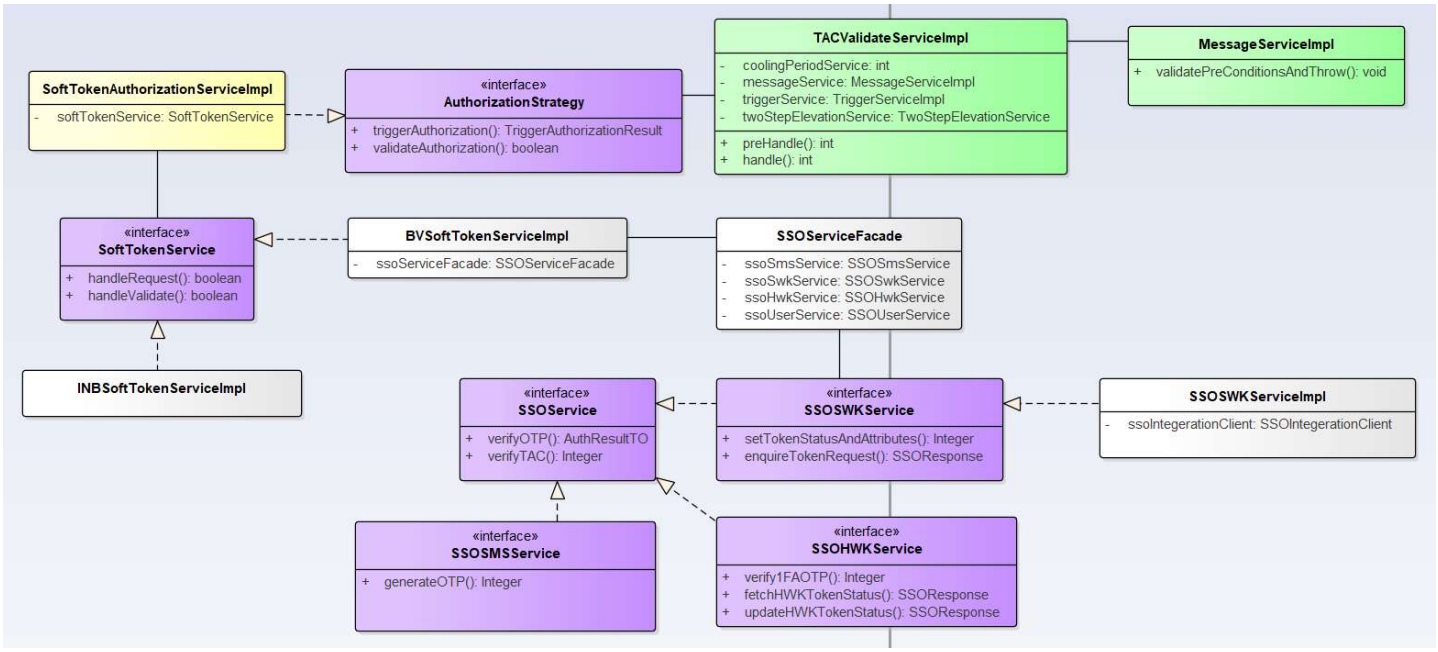
4. handle()

调用triggerAuthzService()方法。

5. checkForAuthTypeChannel()

比较authorization type，选择最优先。

API /tac/validate



1. TACValidateServiceImpl

主要方法handle()，包括3个步骤

---

step2，根据AUTHZ_REF_CHECK配置和authz_policy表，校验随机码。

step3，使用TxnUtl类checkDurationFlow()

在校验之前检查authz_status。

通过**MessageServieImpl**类的validatePreConditionsAndThrow，进行预校验。

step4，使用了AuthorizationStrategy及其具体实现类

调用validateAuthorization()方法进行校验。

比如BVSoftTokenServiceImpl具体实现类的handleValidate()方法，调用SSOServiceFacade的verifyOTPByMFAType()，调用SSOSWKService类的verifyTAC()，调用SSOIntegrationClient类的verifySoftTokenTAC()进行校验。

实际上，SSOIntegrationClient类实现了各种校验，包括SSOSWKService，SSOHWKService，SSOSMSService。

step5，根据校验结果，更新AUTHZ_TXN表。

2. MessageServiceImpl

提供了validatePreConditionsAndThrow()方法，在调用validate核心功能之前，检查状态和校验。

API /otp/request

这个API，对于不同的channel有区别吗？基本没有。

通过**GenericElevationServiceImpl**类的**genericOTPAndSendSMSorNotification**()产生OTP并发送（SMS，Email）。

step1，通过请求头判断是通过SMS还是Email发送OTP。

step2，通过**MessageServiceImpl**类**isEligibleForTriggering**()，判断当前的AUTHZ_STS_CD状态字段，是否允许发送OTP。

step3，通过**GenericUtility**类**generateOTP**()，产生OTP，并计算hash值。产生参考码。

step4，通过**AuthorizationPolicyCache**类从本地缓存或者**AUTHZ_POLICY**表查询配置，OTP有效时长。

step5，更新AUTHZ_TXN表。

step6，通过**NotificationService**类**send**()方法，发送消息。实际通过**ms-customer-notification-sdk**。

API /otp/validate

通过**GenericElevationServiceImpl**类的**genericOTPValidateV2**()校验OTP。

step1，通过请求头的auth ID，查询**AUTHZ_TXN**表。

step2，预校验。校验请求头字段和表记录是否一致，包括transactionID，参考码。

step3，通过比较请求头的otp和表记录的otp，进行hash校验。

step4，根据校验结果，更新**AUTHZ_TXN**表。

API /v3/access-elevation/request

1. AccessElevationRequestServiceImpl

主要方法**handle**()，包括?个步骤。

预处理

判断是否passthrough。

通过**AccessElevationCache**类从本地缓存或者**ACCESS_ELEVATION**表查询相关配置。

校验jwtDomain

校验是否需要提权，比较表字段ial和请求头amr字段。

step1，获取用户的可用token列表

通过TokenService具体实现类的queryUserProfile()，查询查询缓存或SOA。

step2：decideAuthorizationType 判断（用户可选的）授权类型

使用了**TokenService**及其具体实现类的decideAuthorizationType()。

通过**AuthorizationPolicyCache**类从本地缓存或者**AUTHZ_POLICY**表查询配置

step3：updateAndTrigger 请求授权和发送消息

如果triggerOTP为Y，通过TriggerServiceImpl类的triggerAuthzService()，实际上通过AuthorizationStrategy的具体实现类的triggerAuthorization()实现。

API /v3/access-elevation/validate

step2，预校验。校验当前状态。

通过**MessageServieImpl**类的validatePreConditionsAndThrow，进行预校验。

step3，使用了AuthorizationStrategy及其具体实现类validateAuthorization()

step4，校验后处理

如果校验成功，更新**AUTHZ_TXN**表状态字段，并发送kafka消息。

如果校验失败，更新**AUTHZ_TXN**表失败次数字段，并抛出异常。

AuthenticationContext 上下文类

accessElevationRequest
请求体中access-elevation/request

accessElevationValidate
请求体中access-elevation/validate

authorizationElevationRequest

authorizerTokenEntity
存储用户信息user profile，对应AUTHZ_USER_PROFILE表

challengeCode
产生的ChallengeCode，或者请求的传参。
customerChallengeCode
请求传参是否包含challengeCode。

mfaType

tacRequest
请求体

tacValidate
请求体

triggerOTP
是否触发OTP

transactionPayload
请求体

CommonContext

auditModel
channel
headers
tokenContext
存储token信息，来源于jwt

tokenContext.custom

无标签