请给出客户端注册时认证的相关代码

服务器是如何处理客户端给出的注册客户端的请求

详细讲解一下验证客户端断言的相关代码

详细讲解一下服务器如何处理客户端的注册请求

详细讲解一下服务器如何处理PAR请求

详细讲解一下整个项目的通过断言认证的流程，给出代码

断言参数中的audiance

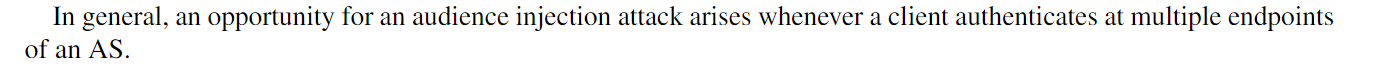
当客户端发送的断言到达服务器时，服务器是如何处理的

当客户端使用断言进行认证时，客户端和服务器如何交互的

讲解一下PAR

JWT从请求中获取audience

Assertion 定义aud是一个节点还是一个列表



当客户端的使用断言进行认证时

**客户端**

生成JWT断言

构建认证请求，发送post请求

**服务器**

（1）请求接收与初始验证

端点接收 ：JWTClientValidator` 处理请求

参数验证 ：通过 `clientAssertionParametersValidation` 方法验证：

- client\_assertion\_type 必须为 urn:ietf:params:oauth:client-assertion-type:jwt-bearer

- client\_assertion 参数必须存在且非空 （2）JWT解析与验证

- 解析JWT ：通过 `readJws` 方法提取JWT头部和载荷

- 客户端身份验证 ：验证 iss 声明与客户端ID匹配，且客户端状态正常（未禁用、未锁定）

- 签名验证 ：

- client\_secret\_jwt ：使用客户端密钥验证HMAC签名（支持密钥轮换）

- private\_key\_jwt ：使用客户端公钥验证RSA/ECDSA签名

- 声明验证 ：

- 有效期 ：检查 exp （过期时间）和 iat （签发时间）

- 受众 ：通过 `validateTokenAudience` 方法验证 aud 包含服务器预期值（如令牌端点URL）

- 重放防护 ：通过 `validateTokenReuse` 方法缓存 jti 防止重复使用

服务器响应阶段

成功响应则生成访问令牌或者刷新令牌

失败则返回标准OAuth2.0错误响应

JWTClientValidator.java

    public boolean clientAssertionParametersValidation() {

        //KEYCLOAK-19461: Needed for quarkus resteasy implementation throws exception when called with mediaType authentication/json in OpenShiftTokenReviewEndpoint

        if(!isFormDataRequest(context.getHttpRequest())) {

            Response challengeResponse = ClientAuthUtil.errorResponse(Response.Status.BAD\_REQUEST.getStatusCode(), "invalid\_client", "Parameter client\_assertion\_type is missing");

            context.challenge(challengeResponse);

            return false;

        }

        params = context.getHttpRequest().getDecodedFormParameters();

        String clientAssertionType = params.getFirst(OAuth2Constants.CLIENT\_ASSERTION\_TYPE);

        clientAssertion = params.getFirst(OAuth2Constants.CLIENT\_ASSERTION);

        if (clientAssertionType == null) {

            Response challengeResponse = ClientAuthUtil.errorResponse(Response.Status.BAD\_REQUEST.getStatusCode(), "invalid\_client", "Parameter client\_assertion\_type is missing");

            context.challenge(challengeResponse);

            return false;

        }

        if (!clientAssertionType.equals(OAuth2Constants.CLIENT\_ASSERTION\_TYPE\_JWT)) {

            Response challengeResponse = ClientAuthUtil.errorResponse(Response.Status.BAD\_REQUEST.getStatusCode(), "invalid\_client", "Parameter client\_assertion\_type has value '"

                    + clientAssertionType + "' but expected is '" + OAuth2Constants.CLIENT\_ASSERTION\_TYPE\_JWT + "'");

            context.challenge(challengeResponse);

            return false;

        }

        if (clientAssertion == null) {

            Response challengeResponse = ClientAuthUtil.errorResponse(Response.Status.BAD\_REQUEST.getStatusCode(), "invalid\_client", "client\_assertion parameter missing");

            context.failure(AuthenticationFlowError.INVALID\_CLIENT\_CREDENTIALS, challengeResponse);

            return false;

        }

        return true;

    }

从JWT中提取出audience参数

JWTClientValidator validator = new JWTClientValidator(context, getId());

JsonWebToken token = validator.getToken();

if (!signatureValid) {

                throw new RuntimeException("Signature on JWT token failed validation");

            }

            validator.validateTokenAudience(context, realm, token);

            validator.validateToken();

            validator.validateTokenReuse();

当客户端发送请求时，服务器如何处理

JWTClientAuthenticator类

public void authenticateClient(ClientAuthenticationFlowContext context) {

        JWTClientValidator validator = new JWTClientValidator(context, getId());

        if (!validator.clientAssertionParametersValidation()) return;

        try {

            validator.readJws();

            if (!validator.validateClient()) return;

            if (!validator.validateSignatureAlgorithm()) return;

            RealmModel realm = validator.getRealm();

            ClientModel client = validator.getClient();

            JWSInput jws = validator.getJws();

            JsonWebToken token = validator.getToken();

            String clientAssertion = validator.getClientAssertion();

            // Get client key and validate signature

            PublicKey clientPublicKey = getSignatureValidationKey(client, context, jws);

            if (clientPublicKey == null) {

                // Error response already set to context

                return;

            }

            boolean signatureValid;

            try {

                JsonWebToken jwt = context.getSession().tokens().decodeClientJWT(clientAssertion, client, (jose, validatedClient) -> {

                    DEFAULT\_VALIDATOR.accept(jose, validatedClient);

                    String signatureAlgorithm = jose.getHeader().getRawAlgorithm();

                    ClientSignatureVerifierProvider signatureProvider = context.getSession().getProvider(ClientSignatureVerifierProvider.class, signatureAlgorithm);

                    if (signatureProvider == null) {

                        throw new RuntimeException("Algorithm not supported");

                    }

                    if (!signatureProvider.isAsymmetricAlgorithm()) {

                        throw new RuntimeException("Algorithm is not asymmetric");

                    }

                }, JsonWebToken.class);

                signatureValid = jwt != null;

            } catch (RuntimeException e) {

                Throwable cause = e.getCause() != null ? e.getCause() : e;

                throw new RuntimeException("Signature on JWT token failed validation", cause);

            }

            if (!signatureValid) {

                throw new RuntimeException("Signature on JWT token failed validation");

            }

            validator.validateTokenAudience(context, realm, token);

            validator.validateToken();

            validator.validateTokenReuse();

            context.success();

        } catch (Exception e) {

            ServicesLogger.LOGGER.errorValidatingAssertion(e);

            Response challengeResponse = ClientAuthUtil.errorResponse(Response.Status.BAD\_REQUEST.getStatusCode(), OAuthErrorException.INVALID\_CLIENT, "Client authentication with signed JWT failed: " + e.getMessage());

            context.failure(AuthenticationFlowError.INVALID\_CLIENT\_CREDENTIALS, challengeResponse);

        }

    }

readJws()

validateClient()

    private JsonWebToken token;

读取并解析JWT readJws()

public void readJws() throws JWSInputException {

        if (clientAssertion == null) throw new IllegalStateException("Incorrect usage. Variable 'clientAssertion' is null. Need to validate clientAssertion first before read JWS");

        jws = new JWSInput(clientAssertion);

        token = jws.readJsonContent(JsonWebToken.class);

    }

        jws = new JWSInput(clientAssertion);

JWSInput（）源码

    public JWSInput(String wire) throws JWSInputException {

        try {

            this.wireString = wire;

            String[] parts = wire.split("\\.");

            if (parts.length < 2 || parts.length > 3) throw new IllegalArgumentException("Parsing error");

            encodedHeader = parts[0];

            encodedContent = parts[1];

            encodedSignatureInput = encodedHeader + '.' + encodedContent;

            content = Base64Url.decode(encodedContent);

            if (parts.length > 2) {

                encodedSignature = parts[2];

                signature = Base64Url.decode(encodedSignature);

            }

            byte[] headerBytes = Base64Url.decode(encodedHeader);

            header = JsonSerialization.readValue(headerBytes, JWSHeader.class);

        } catch (Throwable t) {

            throw new JWSInputException(t);

        }

    }

readJsonContent()源码

JWSInput类中用于将JWS载荷反序列化为指定类型

    public <T> T readJsonContent(Class<T> type) throws JWSInputException {

        try {

            return JsonSerialization.readValue(content, type);

        } catch (IOException e) {

            throw new JWSInputException(e);

        }

    }