Token申请请求的处理

TokenEndpoint类 processGrantRequest

    public Response processGrantRequest() {

        cors = Cors.builder().auth().allowedMethods("POST").auth().exposedHeaders(Cors.ACCESS\_CONTROL\_ALLOW\_METHODS);

        MultivaluedMap<String, String> formParameters = request.getDecodedFormParameters();

        if (formParameters == null) {

            formParameters = new MultivaluedHashMap<>();

        }

        formParams = formParameters;

        grantType = formParams.getFirst(OIDCLoginProtocol.GRANT\_TYPE\_PARAM);

        httpResponse.setHeader("Cache-Control", "no-store");

        httpResponse.setHeader("Pragma", "no-cache");

        checkSsl();

        checkRealm();

        checkGrantType();

        if (!grantType.equals(OAuth2Constants.UMA\_GRANT\_TYPE)

                // pre-authorized grants are not necessarily used by known clients.

                && !grantType.equals(PreAuthorizedCodeGrantTypeFactory.GRANT\_TYPE)) {

            checkClient();

            checkParameterDuplicated();

        }

        DPoPUtil.retrieveDPoPHeaderIfPresent(session, clientConfig, event, cors).ifPresent(dPoP -> {

            session.setAttribute(DPoPUtil.DPOP\_SESSION\_ATTRIBUTE, dPoP);

        });

        OAuth2GrantType.Context context = new OAuth2GrantType.Context(session, clientConfig, clientAuthAttributes,

                                                                      formParams, event, cors, tokenManager);

        return grant.process(context);

    }

关键代码

checkGrantType();

checkClient();

return grant.process(context);

checkGrantType: 检查授权许可类型

checkClient: 验证客户端身份

grant.process: 执行申请token的流程

checkClient源码

    private void checkClient() {

        AuthorizeClientUtil.ClientAuthResult clientAuth = AuthorizeClientUtil.authorizeClient(session, event, cors);

        client = clientAuth.getClient();

        clientAuthAttributes = clientAuth.getClientAuthAttributes();

        clientConfig = OIDCAdvancedConfigWrapper.fromClientModel(client);

        cors.allowedOrigins(session, client);

        if (client.isBearerOnly()) {

            throw new CorsErrorResponseException(cors, OAuthErrorException.INVALID\_CLIENT, "Bearer-only not allowed", Response.Status.BAD\_REQUEST);

        }

    }

通过认证器的具体类实现了好几种认证的方式

最为关注的通过secret实现的认证 关键代码

        if (clientSecret == null) {

            Response challengeResponse = ClientAuthUtil.errorResponse(Response.Status.UNAUTHORIZED.getStatusCode(), "unauthorized\_client", "Invalid client or Invalid client credentials");

            context.challenge(challengeResponse);

            return;

        }

        if (client.getSecret() == null) {

            reportFailedAuth(context);

            return;

        }

        OIDCClientSecretConfigWrapper wrapper = OIDCClientSecretConfigWrapper.fromClientModel(client);

        if (!client.validateSecret(clientSecret)) {

            if (!wrapper.validateRotatedSecret(clientSecret)){

                reportFailedAuth(context);

                return;

            }

        }

validateSecret 通过数据库比对

    public boolean validateSecret(String secret) {

        return MessageDigest.isEqual(secret.getBytes(), entity.getSecret().getBytes());

    }

process函数 AuthorizationCodeGrantType

    public Response process(Context context) {

        setContext(context);

        String code = formParams.getFirst(OAuth2Constants.CODE);

        if (code == null) {

            String errorMessage = "Missing parameter: " + OAuth2Constants.CODE;

            event.detail(Details.REASON, errorMessage);

            event.error(Errors.INVALID\_CODE);

            throw new CorsErrorResponseException(cors, OAuthErrorException.INVALID\_REQUEST, errorMessage, Response.Status.BAD\_REQUEST);

        }

        OAuth2CodeParser.ParseResult parseResult = OAuth2CodeParser.parseCode(session, code, realm, event);

        if (parseResult.isIllegalCode()) {

            AuthenticatedClientSessionModel clientSession = parseResult.getClientSession();

                        if (clientSession != null) {

                clientSession.detachFromUserSession();

            }

            event.error(Errors.INVALID\_CODE);

            throw new CorsErrorResponseException(cors, OAuthErrorException.INVALID\_GRANT, "Code not valid", Response.Status.BAD\_REQUEST);

        }

        AuthenticatedClientSessionModel clientSession = parseResult.getClientSession();

        if (parseResult.isExpiredCode()) {

            event.error(Errors.EXPIRED\_CODE);

            throw new CorsErrorResponseException(cors, OAuthErrorException.INVALID\_GRANT, "Code is expired", Response.Status.BAD\_REQUEST);

        }

        UserSessionModel userSession = null;

        if (clientSession != null) {

            userSession = clientSession.getUserSession();

        }

        if (userSession == null) {

            event.error(Errors.USER\_SESSION\_NOT\_FOUND);

throw new CorsErrorResponseException(cors, OAuthErrorException.INVALID\_GRANT, "User session not found", Response.Status.BAD\_REQUEST);

        }

        UserModel user = userSession.getUser();

        if (user == null) {

            event.error(Errors.USER\_NOT\_FOUND);

            throw new CorsErrorResponseException(cors, OAuthErrorException.INVALID\_GRANT, "User not found", Response.Status.BAD\_REQUEST);

        }

        event.user(userSession.getUser());

        if (!user.isEnabled()) {

            event.error(Errors.USER\_DISABLED);

            throw new CorsErrorResponseException(cors, OAuthErrorException.INVALID\_GRANT, "User disabled", Response.Status.BAD\_REQUEST);

        }

        OAuth2Code codeData = parseResult.getCodeData();

        String redirectUri = codeData.getRedirectUriParam();

        String redirectUriParam = formParams.getFirst(OAuth2Constants.REDIRECT\_URI);

        if (redirectUriParam != null && redirectUriParam.contains("session\_state=") && !redirectUri.contains("session\_state=")) {

            redirectUriParam = KeycloakUriBuilder.fromUri(redirectUriParam)

                    .replaceQueryParam(OAuth2Constants.SESSION\_STATE, null)

                    .build().toString();

        }

        if (redirectUri != null && !redirectUri.equals(redirectUriParam)) {

            String errorMessage = "Parameter 'redirect\_uri' did not match originally saved redirect URI used in initial OIDC request. Saved redirectUri: %s, redirectUri parameter: %s";

            event.detail(Details.REASON, String.format(errorMessage, redirectUri, redirectUriParam));

            event.error(Errors.INVALID\_REDIRECT\_URI);

            logger.tracef(errorMessage, redirectUri, redirectUriParam);

            throw new CorsErrorResponseException(cors, OAuthErrorException.INVALID\_GRANT, "Incorrect redirect\_uri", Response.Status.BAD\_REQUEST);

        }

        if (!client.getClientId().equals(clientSession.getClient().getClientId())) {

            String errorMessage = "Auth error: Found different client\_id in clientSession";

            event.detail(Details.REASON, errorMessage);

            event.error(Errors.INVALID\_CLIENT);

            throw new CorsErrorResponseException(cors, OAuthErrorException.INVALID\_GRANT, errorMessage, Response.Status.BAD\_REQUEST);

        }

        if (!client.isStandardFlowEnabled()) {

            String errorMessage = "Client not allowed to exchange code";

            event.detail(Details.REASON, errorMessage);

            event.error(Errors.NOT\_ALLOWED);

            throw new CorsErrorResponseException(cors, OAuthErrorException.INVALID\_GRANT, errorMessage, Response.Status.BAD\_REQUEST);

        }

        if (!AuthenticationManager.isSessionValid(realm, userSession)) {

            String errorMessage = "Session not active";

            event.detail(Details.REASON, errorMessage);

            event.error(Errors.USER\_SESSION\_NOT\_FOUND);

            throw new CorsErrorResponseException(cors, OAuthErrorException.INVALID\_GRANT, errorMessage, Response.Status.BAD\_REQUEST);

        }

        String codeVerifier = formParams.getFirst(OAuth2Constants.CODE\_VERIFIER);

        String codeChallenge = codeData.getCodeChallenge();

        String codeChallengeMethod = codeData.getCodeChallengeMethod();

        String authUserId = user.getId();

        String authUsername = user.getUsername();

        if (authUserId == null) {

            authUserId = "unknown";

        }

        if (authUsername == null) {

            authUsername = "unknown";

        }

        if (codeChallengeMethod != null && !codeChallengeMethod.isEmpty()) {

            PkceUtils.checkParamsForPkceEnforcedClient(codeVerifier, codeChallenge, codeChallengeMethod, authUserId, authUsername, event, cors);

        } else {

            // PKCE Activation is OFF, execute the codes implemented in KEYCLOAK-2604

            PkceUtils.checkParamsForPkceNotEnforcedClient(codeVerifier, codeChallenge, codeChallengeMethod, authUserId, authUsername, event, cors);

        }

        DPoPUtil.validateDPoPJkt(codeData.getDpopJkt(), session, event, cors);

        try {

            session.clientPolicy().triggerOnEvent(new TokenRequestContext(formParams, parseResult));

        } catch (ClientPolicyException cpe) {

            event.detail(Details.REASON, Details.CLIENT\_POLICY\_ERROR);

            event.detail(Details.CLIENT\_POLICY\_ERROR, cpe.getError());

            event.detail(Details.CLIENT\_POLICY\_ERROR\_DETAIL, cpe.getErrorDetail());

            event.error(cpe.getError());

            throw new CorsErrorResponseException(cors, OAuthErrorException.INVALID\_GRANT, cpe.getErrorDetail(), Response.Status.BAD\_REQUEST);

        }

        updateClientSession(clientSession);

        updateUserSessionFromClientAuth(userSession);

        String scopeParam = codeData.getScope();

        Supplier<Stream<ClientScopeModel>> clientScopesSupplier = () -> TokenManager.getRequestedClientScopes(session, scopeParam, client, user);

        if (!TokenManager.verifyConsentStillAvailable(session, user, client, clientScopesSupplier.get())) {

            String errorMessage = "Client no longer has requested consent from user";

            event.detail(Details.REASON, errorMessage);

            event.error(Errors.NOT\_ALLOWED);

            throw new CorsErrorResponseException(cors, OAuthErrorException.INVALID\_SCOPE, errorMessage, Response.Status.BAD\_REQUEST);

        }

        ClientSessionContext clientSessionCtx = DefaultClientSessionContext.fromClientSessionAndScopeParameter(clientSession, scopeParam, session);

        clientSessionCtx.setAttribute(OIDCLoginProtocol.NONCE\_PARAM, codeData.getNonce());

        return createTokenResponse(user, userSession, clientSessionCtx, scopeParam, true, s -> {return new TokenResponseContext(formParams, parseResult, clientSessionCtx, s);});

    }

关键代码

        OAuth2CodeParser.ParseResult parseResult = OAuth2CodeParser.parseCode(session, code, realm, event);

ParseCode方法关键代码

检查授权码是否使用

SingleUseObjectProvider codeStore = session.singleUseObjects();

        Map<String, String> codeData = codeStore.remove(codeUUID);

检查传入的Code中的client\_id与存储的Code绑定的client\_id是否一样

String persistedUserSessionId = result.codeData.getUserSessionId();

if (!userSessionId.equals(persistedUserSessionId)) {

            logger.warnf("Code '%s' is bound to a different session", codeUUID);

            return result.illegalCode();}

UserSessionId：从请求中传入的code中读取的userSessionId

PersistedUserSessionId: 根据传入的code中读取codeUUID，然后根据codeUUID去服务器存储的code库中提取出对应的code，读取其中的UserSessionId

检查授权码是否过期

int currentTime = Time.currentTime();

        if (currentTime > result.codeData.getExpiration()) {

            return result.expiredCode();

        }

验证传入请求中的client\_id参数与传入的code中绑定的client\_id是否相等，确保只有code中绑定的客户端才能使用该code

        if (!client.getClientId().equals(clientSession.getClient().getClientId())) {

            String errorMessage = "Auth error: Found different client\_id in clientSession";

            event.detail(Details.REASON, errorMessage);

            event.error(Errors.INVALID\_CLIENT);

            throw new CorsErrorResponseException(cors, OAuthErrorException.INVALID\_GRANT, errorMessage, Response.Status.BAD\_REQUEST);

        }

