用户会话Client\_session： 表示用户登录状态，管理用户的认证会话信息，每个用户登录后便有一个用户会话对象。记录用户登录后的各种上下文信息

客户端会话：表示用户与特定客户端之间认证状态，记录用户与客户端交互的关键信息。

Client\_id定义的语句

Keycloak\keycloak\core\src\main\java\org\keycloak\representations\idm\ClientRepresentation.java 此文件定义了客户端的一些基本属性和访问接口

public class ClientRepresentation {

    protected String id;

    protected String clientId;

    protected String name;

传入客户端的注册请求rep，构造ClientModel对象，rep中包含client\_id

ClientModel clientModel = ClientManager.createClient(session, realm, rep);

RepresentationToModel类中 使用resourceRep中的client\_id构造客户端对象(CLientModel)，写入数据库中

 ClientModel client = resourceRep.getId() != null ? realm.addClient(resourceRep.getId(), resourceRep.getClientId()) : realm.addClient(resourceRep.getClientId());

JpaRealmProvider类 将客户端信息存储到数据库中

public ClientModel addClient(RealmModel realm, String id, String clientId) {

        ClientModel resource;

        if (id == null) {

            id = KeycloakModelUtils.generateId();

        } else if (id.length() > ClientEntity.ID\_MAX\_LENGTH){

            throw new ModelValidationException("Client ID must not exceed 36 characters");

        }

        if (clientId == null) {

            clientId = id;

        }

        logger.tracef("addClient(%s, %s, %s)%s", realm, id, clientId, getShortStackTrace());

        ClientEntity entity = new ClientEntity();

        entity.setId(id);

        entity.setClientId(clientId);

        entity.setEnabled(true);

        entity.setStandardFlowEnabled(true);

        entity.setRealmId(realm.getId());

        em.persist(entity); //写入数据库

        resource = toClientModel(realm, entity);

        session.getKeycloakSessionFactory().publish((ClientModel.ClientCreationEvent) () -> resource);

        return resource;

    }

该方法在ClientAdapter.java中实现，从ClientModel中的缓存中读取

public String getClientId() {

        if (isUpdated()) return updated.getClientId();

        return cached.getClientId();

    }

Cache是从数据库中提取client\_id

客户端会话对象（ClientSession）初始化时会绑定client\_id和（通过在初始化对象时传入ClientModel类型的对象client进行绑定，）

public AuthenticatedClientSessionAdapter(KeycloakSession kcSession,

                                             AuthenticatedClientSessionEntity entity, ClientModel client, UserSessionModel userSession,

                                             SessionsChangelogBasedTransaction<UUID, AuthenticatedClientSessionEntity> clientSessionUpdateTx, boolean offline) {

        if (userSession == null) {

            throw new NullPointerException("userSession must not be null");

        }

        this.kcSession = kcSession;

        this.entity = entity;

        this.userSession = userSession;

        this.client = client;

        this.clientSessionUpdateTx = clientSessionUpdateTx;

        this.offline = offline;

    }

ClientModel接口中定义了一个方法getClientId()用于获取Client\_id

接下来去找将client\_id与code绑定时的代码，

 // Standard or hybrid flow

        String code = null;

        if (responseType.hasResponseType(OIDCResponseType.CODE)) {

            OAuth2Code codeData = new OAuth2Code(UUID.randomUUID().toString(),

                Time.currentTime() + userSession.getRealm().getAccessCodeLifespan(),

                nonce,

                authSession.getClientNote(OAuth2Constants.SCOPE),

                authSession.getClientNote(OIDCLoginProtocol.REDIRECT\_URI\_PARAM),

                authSession.getClientNote(OIDCLoginProtocol.CODE\_CHALLENGE\_PARAM),

                authSession.getClientNote(OIDCLoginProtocol.CODE\_CHALLENGE\_METHOD\_PARAM),

                authSession.getClientNote(OIDCLoginProtocol.DPOP\_JKT),

                userSession.getId());

            code = OAuth2CodeParser.persistCode(session, clientSession, codeData);

            redirectUri.addParam(OAuth2Constants.CODE, code);

        }

生成一个code对象，调用Oauth2CodeParser.persistCode绑定住clientSession，ClientSession中包含

public static String persistCode(KeycloakSession session, AuthenticatedClientSessionModel clientSession, OAuth2Code codeData) {

        SingleUseObjectProvider codeStore = session.singleUseObjects();

        String key = codeData.getId();

        if (key == null) {

            throw new IllegalStateException("ID not present in the data");

        }

        Map<String, String> serialized = codeData.serializeCode();

        codeStore.put(key, clientSession.getUserSession().getRealm().getAccessCodeLifespan(), serialized);

        return key + "." + clientSession.getUserSession().getId() + "." + clientSession.getClient().getId();

    }

codeData.getId() 授权码的唯一标识符（随机生成）

clientSession.getUserSession().getId() 用户会话的唯一标识符

clientSession.getClient().getId() Client\_id

if (!userSessionId.equals(persistedUserSessionId)) {

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

    return result.illegalCode();

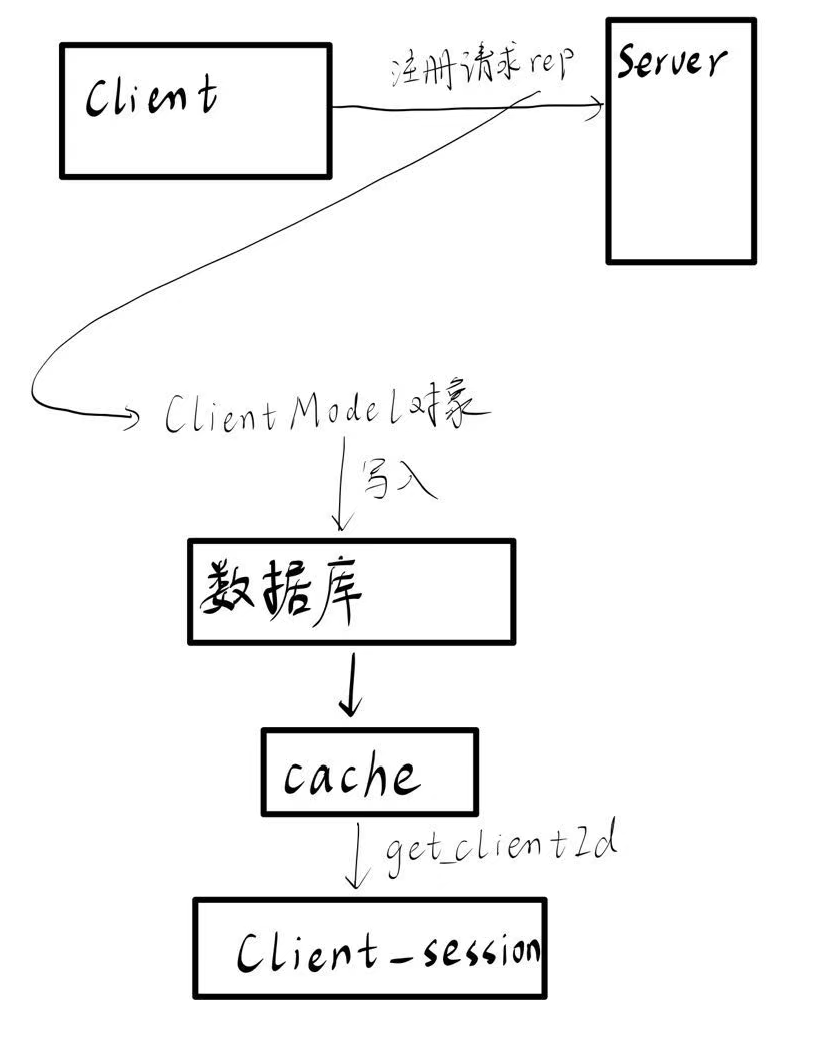
}

这部分代码通过以下步骤验证客户端身份匹配： equals()

将授权码分割为三部分： codeUUID 、 userSessionId 和 clientUUID

使用 userSession.getAuthenticatedClientSessionByClient(clientUUID) 获取客户端会话

验证解析出的 userSessionId 与存储在授权码数据中的 persistedUserSessionId 是否匹配



授权码单次使用

AuthorizationCodeGrantType类中 isIllegalCode()判断是否是非法Code

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

        if (parseResult.isIllegalCode()) {

            AuthenticatedClientSessionModel clientSession = parseResult.getClientSession();

            // Attempt to use same code twice should invalidate existing clientSession

            if (clientSession != null) {

                clientSession.detachFromUserSession();

            }

            event.error(Errors.INVALID\_CODE);

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

        }

找到isIllegalCode()定义 属于OAuth2CodeParser类

public boolean isIllegalCode() {

            return isIllegalCode;

        }

将Code设置成非法状态 调用其即可

private ParseResult illegalCode() {

            this.isIllegalCode = true;

            return this;

        }

检查授权码是否已经使用 若已经使用则调用illegalCode（）

SingleUseObjectProvider codeStore = session.singleUseObjects();

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

        // Either code not available or was already used

        if (codeData == null) {

            logger.warnf("Code '%s' already used for userSession '%s' and client '%s'.", codeUUID, userSessionId, clientUUID);

            return result.illegalCode();

        }

SingleUseObjectProvider接口

是 Keycloak 中用于管理单次使用对象的核心接口，定义了安全存储和检索只能使用一次的敏感数据的标准方法

remove方法：查找并删除codeUUID对应的code，如果不存在则返回null

1